47th International Colloquium on Automata, Languages, and Programming

ICALP 2020, July 8–11, 2020, Saarbrücken, Germany
(Virtual Conference)

Edited by

Artur Czumaj
Anuj Dawar
Emanuela Merelli

LIPIcs – Vol. 168 – ICALP 2020 www.dagstuhl.de/lipics

Editors

Artur Czumaj
University of Warwick, UK
A.Czumaj@warwick.ac.uk

Anuj Dawar
University of Cambridge, UK
anuj.dawar@cl.cam.ac.uk

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

ACM Classification 2012
Theory of Computation

ISBN 978-3-95977-138-2

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at https://www.dagstuhl.de/dagpub/978-3-95977-138-2.

Publication date
June, 2020

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

Digital Object Identifier: 10.4230/LIPIcs.ICALP.2020.0

ISBN 978-3-95977-138-2 **ISSN 1868-8969** **https://www.dagstuhl.de/lipics**

LIPIcs – Leibniz International Proceedings in Informatics

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

ISSN 1868-8969

https://www.dagstuhl.de/lipics

Contents

Invited Talks

Track A: Algorithms, Complexity and Games

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

Preface

This volume contains the papers presented at the *47th International Colloquium on Automata, Languages and Programming (**ICALP 2020**)*, held *virtually*, hosted by the Saarland Informatics Campus in Saarbrücken, Germany, during July 8–11, 2020. ICALP is a series of annual conferences of the *European Association for Theoretical Computer Science (EATCS)*, which first took place in 1972. ICALP 2020 was co-located with the 35th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2020).

The conference was affected by the outbreak of COVID-19, which had an enormous impact across the world and the ICALP community was no exception. The original plan had been to hold ICALP 2020, in conjunction with LICS 2020 at Peking University in Beijing, China. When it became clear that this would not be possible due to travel restrictions that were being imposed, and after intensive discussions between the ICALP and LICS steering committees, together with the PC chairs and conference chairs, it was decided to re-locate the conferences to Saarbrücken. Eventually, keeping the safety, health and well-being of ICALP participants as a top priority, it was decided to hold ICALP 2020 entirely online. We are very grateful to the organizers in Beijing and Saarbrücken and to all members of the theoretical computer science community for their flexibility and adaptability in this difficult situation. This first online ICALP is an experiment forced on us by the situation and will no doubt offer many lessons for the future.

For 15 years, the ICALP conference ran with three tracks. This has been the subject of much deliberation in the ICALP community in recent years and the decision was taken, with ICALP 2020, to return to a two-track format. Topics previously included in Track C have been incorporated into Track A. This year, the ICALP program consisted of the following two tracks:

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

In response to the call for papers, a total of 470 submissions were received: 347 for Track A and 123 for Track B. Each submission was assigned to at least three Program Committee members, aided by 857 external subreviewers. The committees decided to accept 138 papers for inclusion in the scientific program: 102 papers for Track A and 36 for Track B. The selection was made by the Program Committees based on originality, quality, and relevance to theoretical computer science. The quality of the manuscripts was very high indeed, and many deserving papers could not be selected.

The EATCS sponsored awards for both a best paper and a best student paper in each of the two tracks, selected by the Program Committees.
The **best paper awards** were given to the following papers:

Track A: Paweł Gawrychowski, Shay Mozes, and Oren Weimann. *Minimum cut in* $O(m \log^2 n)$ *time.*

Track B: David Barozzini, Lorenzo Clemente, Thomas Colcombet and Paweł Parys. *Cost automata, safe schemes, and downward closures.*

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

Track A: Aditya Potukuchi. *A spectral bound on hypergraph discrepancy.*

Track B: Erik Paul. *Finite sequentiality of finitely ambiguous max-plus tree automata.*

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Apart from the contributed talks, ICALP 2020 included invited presentations by Stefan Kiefer (Oxford University), Robert Krauthgamer (The Weizmann Institute of Science) and Virginia Vassilevska Williams (MIT). There were also two invited talks joint with LICS 2020: by Jerôme Leroux (Bordeaux University) and Andrew Chi-Chih Yao (Tsinghua University). This volume contains all the contributed papers presented at the conference, papers that accompany the invited talks of Andrew Yao and Stefan Kiefer, and an abstract of the invited presentation of Robert Krauthgamer.

The program of ICALP 2020 also included presentations of the EATCS Award 2020 to Mihalis Yannakakis, the Gödel Prize 2020 to Robin A. Moser and Gábor Tardos, the Presburger Award 2020 to Dmitriy Zhuk, and the EATCS Distinguished Dissertation Awards to Josh Alman, Sándor Kisfaludi-Bak, and Jakub Tarnawski.

The following workshops were held as satellite events of ICALP and LICS 2020 on July 6–7, 2020:

- Algorithmic Aspects of Temporal Graphs (AATG),
- Fine-Grained and Parameterized Approximation Algorithms (FG-PAAW),
- Verification of Infinite-State Systems (INFINITY),
- Logic and Computational Complexity Workshop (LCC 2020),
- Logic Mentoring Workshop (LMW),
- Programming Research in Mainstream Languages (PRiML).

We wish to thank all authors who submitted extended abstracts for consideration, the Program Committees for their scholarly effort, and all the referees who assisted the Program Committees in the evaluation process. We are also grateful to the Conference Co-Chairs Xiaotie Deng and Holger Hermanns and all the support staff of the Organizing Committee for organizing ICALP 2020: to our colleagues in Peking University, led by Xiaotie Deng, for their organizational efforts in the originally planned location in Beijing, and to the colleagues from Saarbrücken, led by Holger Hermanns, who generously accepted the challenging task of organizing the conference at short notice, and who were then ready to run the conference online. Finally, we are grateful to all members of the TCS community who offered their support in this difficult situation.

We wish to thank CPEC — Center for Perspicuous Computing and Saarland University in Saarbrücken for their generous support for the conference.

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

July 2020

Artur Czumaj
Anuj Dawar
Emanuela Merelli

Organization

Program Committee

Track A

Andris Ambainis	University of Latvia, Lativia
Sepehr Assadi	Rutgers University, United States
Andrej Bogdanov	The Chinese University of Hong Kong, Hong Kong
Sebastian Brandt	ETH Zürich, Switzerland
Vladimir Braverman	Johns Hopkins University, United States
Keren Censor-Hillel	Technion, Israel
Flavio Chierichetti	Sapienza University of Rome, Italy
Artur Czumaj	University of Warwick, United Kingdom, Chair
Holger Dell	IT University of Copenhagen, Denmark
Jelena Diakonikolas	University of Wisconsin-Madison, United States
Matthias Englert	University of Warwick, United Kingdom
Piotr Faliszewski	AGH University of Science and Technology, Poland
Martin Grohe	RWTH Aachen University, Germany
Martin Hoefer	Goethe University, Frankfurt am Main, Germany
John Iacono	Université libre de Bruxelles, Belgium
Gautam Kamath	University of Waterloo, Canada
Christian Konrad	University of Bristol, United Kingdom
Erik Jan van Leeuwen	Utrecht University, Netherlands
Pinyan Lu	Shanghai University of Finance and Economics, China
Seffi Naor	Technion, Israel
Jakob Nordström	University of Copenhagen, Denmark
Giuseppe Persiano	Università di Salerno, Italy
Jeff M. Phillips	University of Utah, United States
Michał Pilipczuk	University of Warsaw, Poland
Thomas Sauerwald	University of Cambridge, United Kingdom
Christian Scheideler	Paderborn University, Germany
Stefan Schmid	University of Vienna, Austria
Melanie Schmidt	University of Cologne, Germany
Tselil Schramm	Stanford University, United States
Shay Solomon	Tel Aviv University, Israel
Tatiana Starikovskaya	École Normale Supérieure, Paris, France
Cliff Stein	Columbia University, United States
Inbal Talgam-Cohen	Technion, Israel
Luca Trevisan	Bocconi University, Italy
Eric Vigoda	Georgia Institute of Technology, United States
Jens Vygen	University of Bonn, Germany
Mihalis Yannakakis	Columbia University, United States
Yuichi Yoshida	National Institute of Informatics, Tokyo, Japan
Morteza Zadimoghaddam	Google Research Zürich, Switzerland
Shengyu Zhang	Tencent, China

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Track B

Luca Aceto	Reykjavik University, Iceland and Gran Sasso Science Institute, Italy
Manuel Bodirsky	TU Dresden, Germany
Patricia Bouyer	CNRS, France
Supratik Chakraborty	IIT Bombay, India
Anuj Dawar	University of Cambridge, United Kingdom, Chair
Volker Diekert	University of Stuttgart, Germany
Kord Eickmeyer	TU Darmstadt, Germany
Kousha Etessami	University of Edinburgh, United Kingdom
Claudia Faggian	CNRS, France
Mai Gehrke	CNRS and Université Côte d' Azur, France
Juha Kontinen	University of Helsinki, Finland
Tony Kucera	Masaryk University, Czech Republic
Marta Kwiatkowska	University of Oxford, United Kingdom
Kim G. Larsen	Aalborg University, Denmark
Ranko Lazic	University of Warwick, United Kingdom
Fenrong Liu	Tsinghua University, China and The University of Amsterdam, Netherlands
Valeria de Paiva	Samsung Research America, USA
Luc Segoufin	INRIA and ENS Paris, France
Peter Selinger	Dalhousie University, Canada
Mahsa Shirmohammadi	CNRS and IRIF, Université de Paris, France
Szymon Torunczyk	University of Warsaw, Poland
Marc Zeitoun	University of Bordeaux, France
Martin Ziegler	KAIST, South Korea

Organizing Committee

Holger Hermanns	Saarbrücken, Germany
Xiaotie Deng	Beijing, China

Steering Committee

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

Financial Sponsors

CPEC – Center for Perspicuous Computing
Saarland University in Saarbrücken

Additional Reviewers

Amir Abboud
Andreas Abels
Antonis Achilleos
Marek Adamczyk
Henry Adams
Assale Adje
Naman Agarwal
Akanksha Agrawal
Shweta Agrawal
Saba Ahmadi
Saeed Akhoondian Amiri
S. Akshay
Jonathan Allcock
Eric Allender
Andreas Alpers
Yackolley Amoussou-Guenou
Ellie Anastasiadi
Alexandr Andoni
Haris Angelidakis
Anurag Anshu
Antonios Antoniadis
Benny Applebaum
Elena Arseneva
Anna Arutyunova
Mohamed Faouzi Atig
Martin Aumüller
Martin Avanzini
Pranjal Awasthi
Yossi Azar
Maxim Babenko
Giorgio Bacci
Miriam Backens
Arturs Backurs
Uwe Baier
Sebastian Bala
Nikhil Balaji
A.R. Balasubramanian
Eric Balkanski
Nikhil Bansal
Suguman Bansal
Pablo Barceló
Siddharth Barman
Libor Barto
Nicolas Basset
Julien Baste
Jatin Batra
Soheil Behnezhad
Xiaohui Bei
Amos Beimel
Paul Bell
Rémy Belmonte
Aleksandrs Belovs
Amir Ben-Amram
Fabrice Benhamouda
Matthias Bentert
Johan van Benthem
Kristóf Bérczi
Benno van den Berg
Christoph Berkholz
Sebastian Berndt
Aaron Bernstein
Nathalie Bertrand
Dietmar Berwanger
René van Bevern
Siddharth Bhandari
Amey Bhangale
Binay Bhattacharya
Sayan Bhattacharya
Vijay Bhattiprolu
Marcin Bienkowski
Daniel Bienstock
Davide Bilò
Vittorio Bilò
Alexander Birx
Arpita Biswas
Markus Bläser
Thomas Bläsius
Michael Blondin
Hans L. Bodlaender
Greg Bodwin
Mikolaj Bojanczyk
Udi Boker
Benedikt Bollig
Eduardo Bonelli
Frederik M. Bønneland
Edouard Bonnet
Michele Boreale
Johannes Borgström
Zarathustra Brady
Joshua Brakensiek
Matthew de Brecht
Robert Bredereck
Karl Bringmann
Guido Brückner
Peter Buergisser
Andrei Bulatov
Elisabet Burjons
Jaroslaw Byrka
Michaël Cadilhac
Leran Cai
Nairen Cao
Yixin Cao
Silvio Capobianco
Arnaud Carayol
Marco Carmosino
Olivier Carton
Valentina Castiglioni
Alonso Castillo-Ramirez
Matteo Ceccarello
Rohit Chadha
André Chailloux
Deeparnab Chakrabarty
Diptarka Chakraborty
Parinya Chalermsook
Jérémie Chalopin
Hubert Chan
Siu On Chan
Timothy M. Chan
Yi-Jun Chang
Bernadette Charron-Bost
Arkadev Chattopadhyay
Edgar Chavez
Hubie Chen
Taolue Chen
Xi Chen
Yu Chen
Zongchen Chen
Siu-Wing Cheng
Yu Cheng
Yun Kuen Cheung
Nai-Hui Chia
Rayan Chikhi
Ashish Chiplunkar
Rajesh Chitnis
Christian Choffrut
Arka Rai Choudhuri
Rezaul Chowdhury
Tobias Christiani
George Christodoulou
Michele Ciampi
Lorenzo Clemente

Christian Coester
Vincent Cohen-Addad
Amin Coja-Oghlan
Alexander Conway
University Copenhagen
Emilio Cruciani
Szabo Csaba
Felipe Cucker
James Currie
Radu Curticapean
Wojciech Czerwiński
Daniel Dadush
Fredrik Dahlqvist
Mina Dalirrooyfard
Thao Dang
Loris D'Antoni
Jacques Dark
Luc Dartois
Debarati Das
Laure Daviaud
Sami Davies
Niel De Beaudrap
Yuan Deng
Jyotirmoy Deshmukh
Henry DeYoung
Pietro Di Gianantonio
Marco Di Summa
Ilias Diakonikolas
Martin Dietzfelbinger
Hu Ding
Michael Dinitz
Simon Docherty
Henk Don
Dean Doron
Laurent Doyen
Lukas Drexler
Anne Driemel
Ran Duan
Clemens Dubslaff
Guillaume Ducoffe
Aditi Dudeja
Bartlomiej Dudek
Szymon Dudycz
Ingo van Duijn
Stephane Durocher
Christoph Dürr
Sven Dziadek
Harley Eades Iii
Eduard Eiben
Hicham El-Zein
David Eppstein
Thomas Erlebach
Javier Esparza
Chaim Even-Zohar
Yuri Faenza
Rolf Fagerberg
Brandon Fain
Yaron Fairstein
Angelo Fanelli
Alireza Farhadi
Syyeda Zainab Fatmi
Ansgar Fehnker
Sándor Fekete
Hugo Férée
Jiri Fiala
Santiago Figueira
Nathanaël Fijalkow
Yuval Filmus
Aris Filos-Ratsikas
Omrit Filtser
Dario Fiore
Manuela Fischer
Orr Fischer
Lukas Fleischer
Noah Fleming
Till Fluschnik
Fedor Fomin
Sebastian Forster
Henry Förster
Klaus-Tycho Förster
Kyle Fox
Pierre Fraigniaud
Adrian Francalanza
Daniel Freund
Dror Fried
Tom Friedetzky
Zachary Friggstad
Vincent Froese
Christiane Frougny
Yuxi Fu
Matthias Függer
Frank Fuhlbrück
Kaito Fujii
Takuro Fukunaga
Harold Gabow
Travis Gagie
Martin Gairing
Jakub Gajarský
Andreas Galanis
Pietro Galliani
Robert Ganian
Pierre Ganty
Pu Gao
Hector J. Garcia
Richard Garner
Leszek Gasieniec
Paul Gastin
Olivier Gauwin
Francesco Gavazzo
Dmitry Gavinsky
Pawel Gawrychowski
Samir Genaim
Blaise Genest
Stefan Gerhold
Sevag Gharibian
Vlad Gheorghiu
Khalil Ghorbal
Suprovat Ghoshal
George Giakkoupis
Matt Gibson
Stephan Gocht
Aarushi Goel
Alexander Göke
Paul Goldberg
Stefan Göller
Petr Golovach
Senén González
Mayank Goswami
Thorsten Götte
Erich Grädel
Fabrizio Grandoni
Joshua Grochow
Allan Grønlund
Christoph Grunau
Yan Gu
Heng Guo
Anupam Gupta
Tom Gur
Rohit Gurjar
Frank Gurski
Grzegorz Guspiel
Anselm Haak
Christoph Haase
Amar Hadzihasanovic
Torben Hagerup
Matthew Hague
Vesa Halava

Magnús M. Halldórsson
Tingting Han
Miika Hannula
Helle Hvid Hansen
Kristoffer Arnsfelt Hansen
Tero Harju
Marcel Hark
Nathan Harms
Sariel Har-Peled
Herman Haverkort
Tom Hayes
Carmit Hazay
Klaus Heeger
Matthias Heizmann
Lauri Hella
Jelle Hellings
Loic Helouet
D. Ellis Hershkowitz
Hiroshi Hirai
Juho Hirvonen
Hsi-Ming Ho
Piotr Hofman
Jana Hofmann
Lukáš Holík
Jacob Holm
Markus Holzer
Hendrik Jan Hoogeboom
Max Hopkins
Samuel Hopkins
Levin Hornischer
Pavel Hrubes
Chien-Chung Huang
Hao Huang
Lingxiao Huang
Shang-En Huang
Jan Hubička
Christopher Hugenroth
Christoph Hunkenschröder
Christian Ikenmeyer
Catalin-Andrei Ilie
Fotis Iliopoulos
Sungjin Im
Shunsuke Inenaga
Jānis Iraids
Sergey Ivanov
Yuni Iwamasa
Yoichi Iwata
Taisuke Izumi
Stefan Jaax
Marcel Jackson
Manfred Jaeger
Radha Jagadeesan
Sidharth Jaggi
Aayush Jain
Rahul Jain
Arun Jambulapati
Bart M. P. Jansen
Klaus Jansen
Jesper Jansson
Ismaël Jecker
Mathias Claus Jensen
Peter Gjøl Jensen
Łukasz Jeż
Zhengfeng Ji
Shaofeng H.-C. Jiang
Zhengzhong Jin
Mitchell Jones
Hossein Jowhari
Amir Kafshdar Goharshady
Naonori Kakimura
Sagar Kale
Shahin Kamali
Naoyuki Kamiyama
Frank Kammer
Michael Kapralov
Bruce Kapron
Jarkko Kari
Hrishikesh Karmarkar
Karthik C. S.
Akitoshi Kawamura
Ehsan Kazemi
Dominik Kempa
George Kenison
Thomas Kesselheim
Sanjeev Khanna
Stefan Kiefer
Emanuel Kieronski
Zachary Kincaid
Kohei Kishida
Aleks Kissinger
Susumu Kiyoshima
David Klaška
Hartmut Klauck
Pieter Kleer
Dominik Klein
Michal Kleinbort
Hans Kleine Büning
Bartek Klin
Peter Kling
Simon Knäuer
Dušan Knop
Tomohiro Koana
Stephen Kobourov
Chris Köcher
Tomasz Kociumaka
Dax Koh
Gillat Kol
Christina Kolb
Ilan Komargodski
Michael Kompatscher
Guy Kortsarz
Peter Kostolányi
Martin Koutecky
Annamaria Kovacs
Marcin Kozik
Laszlo Kozma
Robert Krauthgamer
Martin S. Krejca
Jan Kretinsky
Ravishankar Krishnaswamy
Jari J.H. de Kroon
Piotr Krysta
Manfred Kufleitner
Ariel Kulik
Alexander Kulikov
Ravi Kumar
Marvin Künnemann
Denis Kuperberg
Orna Kupferman
William Kuszmaul
O-Joung Kwon
Rasmus Kyng
Bundit Laekhanukit
Guillaume Lagarde
Victor Lagerkvist
Russell W. F. Lai
Patrick Landwehr
Julien Lange
Stefan Langerman
Zach Langley
John Lapinskas
Sławomir Lasota
Lap Chi Lau
Massimo Lauria
Francois Le Gall
Stephane Le Roux
Euiwoong Lee
Jasper C.H. Lee
Engel Lefaucheux
Karoliina Lehtinen

Dean Leitersdorf
Aurélien Lemay
Amit Levi
Asaf Levin
Mateusz Lewandowski
Bo Li
Lvzhou Li
Shi Li
Shuai Li
Tongyang Li
Yi Li
Yinan Li
Yingkai Li
Chao Liao
Nutan Limaye
Jiabao Lin
Wei-Kai Lin
Andrea Lincoln
Jingcheng Liu
Qipeng Liu
Quanquan Liu
Wanwei Liu
Yang Liu
Christof Löding
Maarten Löffler
Daniel Lokshtanov
Sylvain Lombardy
Brendan Lucier
Martin Lück
Gerald Luettgen
Thodoris Lykouris
Sepideh Mahabadi
Kenji Maillard
Anirban Majumdar
Visu Makam
Johann Makowsky
Nikolaos Makriyannis
Marcello Mamino
Florin Manea
Sebastian Maneth
Amaldev Manuel
Pasin Manurangsi
Jan Marcinkowski
Nicolas Markey
Barnaby Martin
Caroline Mattes
Jannik Matuschke
Yannic Maus
Peter Mayr
Filip Mazowiecki
Samuel McCauley
Dylan McDermott
Andrew McGregor
Conor McMeel
Kitty Meeks
Nicole Megow
Abbas Mehrabian
Arne Meier
Hammurabi Mendes
George Mertzios
Julian Mestre
Marc Mezzarobba
Alexei Miasnikov
Othon Michail
Ivan Mikhailin
Majid Mirzanezhad
Tushant Mittal
Atsushi Miyauchi
Matthias Mnich
Ankur Moitra
Marco Molinaro
Hendrik Molter
Tobias Mömke
Benjamin Monmege
Pat Morin
Christopher Morris
Benjamin Moseley
Georg Moser
Antoine Mottet
Shay Mozes
Marcin Mucha
Sagnik Mukhopadhyay
Henry Martyn Mulder
David Müller
Wolfgang Mulzer
Cameron Musco
Dimitrios Myrisiotis
Wojciech Nadara
Viswanath Nagarajan
Vasileios Nakos
Danupon Nanongkai
Daniele Nantes-Sobrinho
David Naori
Amir Nayyeri
Jesper Nederlof
Maryam Negahbani
Yakov Nekrich
Daniel Neuen
Eike Neumann
Stefan Neumann
Ilan Newman
Eric Neyman
Huy Nguyen
André Nichterlein
Andrey Nikolaev
Aleksandar Nikolov
Ryo Nishimaki
Martin Nöllenburg
Gethin Norman
Ashkan Norouzi Fard
Beth Novick
Petr Novotný
Krzysztof Nowicki
Dirk Nowotka
André Nusser
Zeev Nutov
Jan Obdrzalek
Pascal Ochem
Timm Oertel
Eunjin Oh
Pierre Ohlmann
Isabel Oitavem
Karolina Okrasa
Neil Olver
Krzysztof Onak
Tim Oosterwijk
Jakub Opršal
Sebastian Ordyniak
Rotem Oshman
Rafi Ostrovsky
Dominik Pajak
Catuscia Palamidessi
Katarzyna Paluch
Paritosh Pandya
Debmalya Panigrahi
Denis Pankratov
Fahad Panolan
Kunsoo Park
Sewon Park
Francesco Pasquale
Mohnish Pattathurajan
Arno Pauly
Romain Péchoux
Mathias Ruggaard Pedersen

Edita Pelantová
Vincent Penelle
Pan Peng
Simon Perdrix
Will Perkins
Enoch Peserico
David Phillips
Astrid Pieterse
Krzysztof Pietrzak
Giovanni Pighizzini
Marcin Pilipczuk
Veronika Pillwein
Michael Pinsker
Wojciech Plandowski
Vladimir Podolskii
Igor Potapov
Danny Bøgsted Poulsen
Amaury Pouly
Emmanouil Pountourakis
Pavithra Prabhakar
Gautam Prakriya
Nicola Prezza
Eric Price
Kirk Pruhs
Ioannis Psarros
Alexandros Psomas
Simon Puglisi
Gabriele Puppis
Manish Purohit
David Purser
Youming Qiao
Karin Quaas
Thomas Quinn-Gregson
Mikaël Rabie
Roman Rabinovich
Harald Räcke
Sharath Raghvendra
Rajmohan Rajaraman
Govind Ramnarayan
Robert Rand
Martin Raussen
Ilya Razenshteyn
Sarah Rees
Guus Regts
Vojtech Rehak
Pierre-Alain Reynier
Alireza Rezaei
Andrii Riazanov
Colin Riba
Michel Rigo
Kilian Risse
Nicolás Rivera
Peter Robinson
Liam Roditty
Marcel Roeloffzen
Heiko Röglin
Will Rosenbaum
Matthieu Rosenfeld
Clemens Rösner
Benjamin Rossman
Jurriaan Rot
Ralf Rothenberger
Thomas Rothvoss
Natan Rubin
Aviad Rubinstein
Alexander Russell
Jakub Rydval
Paweł Rzążewski
Aleksi Saarela
Barna Saha
Prakash Saivasan
Mohammad Salavatipour
R.B. Sandeep
Bryce Sandlund
Arnaud Sangnier
Laura Sanita
Sriram Sankaranarayanan
Piotr Sankowski
Ocan Sankur
Richard Santiago
Gabriel Santos
Hayk Saribekyan
Jayalal Sarma
David Saulpic
Saket Saurabh
Alceste Scalas
Luke Schaeffer
Patrizia Schalk
Michael Schapira
Kevin Schewior
Martin Schirneck
Daniel R. Schmidt
Ulrike Schmidt-Kraepelin
Sylvain Schmitz
Martin Schneider
Patrick Schnider
Philippe Schnoebelen
Peter Schrammel
Matthias Schroeder
Steffen Schuldenzucker
Gregory Schwartzman
Pascal Schweitzer
Chris Schwiegelshohn
Diego Seco
Saeed Seddighin
Jeremy Seligman
Svetlana Selivanova
Pavel Semukhin
Geraud Senizergues
Alexander Setzer
Chung-chieh Shan
Don Sheehy
Yuping Shen
Suhail Sherif
Thomas Shermer
Elaine Shi
Takeharu Shiraga
Aaron Sidford
Anastasios Sidiropoulos
Hans Simon
Mohit Singh
Sahil Singla
Luisa Siniscalchi
Fiona Skerman
Mateusz Skomra
Alexander Skopalik
Michał Skrzypczak
Friedrich Slivovsky
Andrew Sogokon
Christian Sohler
Dmitry Sokolov
Tasuku Soma
Juraj Somorovsky
Fang Song
Zhao Song
Krzysztof Sornat
José A. Soto
Frits Spieksma
Paul Spirakis
Jiri Srba
A V Sreejith
Florian Starke
Rob van Stee
Rafał Stefański

Benjamin Steinberg
Thomas Steinke
Noah Stephens-Davidowitz
Noah Streib
Venkata Subrahmanyam
Ondrej Suchy
Nike Sun
Xiaoming Sun
Xiaorui Sun
Jukka Suomela
Grégoire Sutre
Ola Svensson
Chaitanya Swamy
Joseph Swernofsky
John Sylvester
Prafullkumar Tale
Ohad Talmon
Zihan Tan
Ewin Tang
Zhihao Gavin Tang
Jakub Tarnawski
Nikolaj Tatti
K S Thejaswini
Thomas Thierauf
Daniel Thoma
Declan Thompson
Mikkel Thorup
Kevin Tian
Tigran Tonoyan
Hazem Torfah
Csaba Toth
Patrick Totzke
Dave Touchette
Ohad Trabelsi
Long Tran-Thanh
Amitabh Trehan
Chittaranjan Tripathy
Ashutosh Trivedi
Konstantinos Tsakalidis
Max Tschaikowski
Leonidas Tsepenekas
Takeshi Tsukada
Emilio Tuosto
Paxton Turner
Andrea Turrini
Nikos Tzevelekos
Kei Uchizawa
Ryuhei Uehara
Jara Uitto
Henning Urbat
Jouko Vaananen
Ali Vakilian
Jan Van den Bussche
Virginia Vassilevska Williams
Sergei Vassilvitskii
Prashant Vasudevan
Dominik Velan
Daniele Venturi
Oleg Verbitsky
José Verschae
Pavel Veselý
Jamie Vicary
Thomas Vidick
Cosimo Vinci
Jonni Virtema
Ivan Visconti
Mahesh Viswanathan
Jan Philipp Wächter
Tal Wagner
Magnus Wahlström
Erik Waingarten
David Wajc
Chen Wang
Chunhao Wang
Zihe Wang
Justin Ward
Julian Wargalla
Daniel Warner
Shun Watanabe
Karol Węgrzycki
Pascal Weil
Oren Weimann
Nicole Wein
S. Matthew Weinberg
Armin Weiss
Mor Weiss
Jake Wellens
Benjamin Wesolowski
Daniel Wichs
Daniel Wiebking
Udi Wieder
Sebastian Wiederrecht
Andreas Wiese
Max Willert
David Williamson
Karl Wimmer
Dominik Wojtczak
Petra Wolf
James Worrell
Marcin Wrochna
Kevin Wu
Xuan Wu
Zhilin Wu
Mingji Xia
Mingyu Xiao
Abuzer Yakaryilmaz
Takashi Yamakawa
Guang Yang
Lin Yang
Sheng Yang
Jonathan Yaniv
Penghui Yao
Kevin Yeo
Fang-Yi Yu
Huacheng Yu
Chenyang Yuan
Ilias Zadik
Viktor Zamaraev
Emmanouil Zampetakis
Meirav Zehavi
Thomas Zeume
Mark Zhandry
Chihao Zhang
Hanrui Zhang
Jialin Zhang
Binhai Zhu
Chunjiang Zhu
Wieslaw Zielonka
Johannes Zink
Philipp Zschoche
Anna Zych

List of Authors

Amir Abboud (4, 5)
IBM Almaden Research Center,
San Jose, CA, USA

Mahmoud Abo Khamis (106)
relationalAI, Berkeley, CA, USA

Naor Alaluf (6)
Department of Mathematics and Computer
Science, Open University of Israel,
Ra'anana, Israel

Susanne Albers (68)
Department of Computer Science,
Technical University of Munich, Germany

Dan Alistarh (7)
IST Austria, Klosterneuburg, Austria

Shaull Almagor (107)
Department of Computer Science,
Technion, Haifa, Israel

Maryam Bahrani (8)
Princeton University, NJ, USA

Christel Baier (138)
Technische Universität Dresden, Germany

Boaz Barak (108)
Harvard University, Cambridge, MA, USA

David Barozzini (109)
Institute of Informatics,
University of Warsaw, Poland

Libor Barto (110)
Department of Algebra, Faculty of
Mathematics and Physics, Charles University,
Praha 8, Czech Republic

Andrew Bassilakis (9)
Stanford University, CA, USA

Pascal Baumann (111)
Max Planck Institute for Software Systems,
Kaiserslautern, Germany

Michael Benedikt (112)
University of Oxford, UK

Suman K. Bera (11)
University of California at Santa Cruz, CA, USA

Aaron Bernstein (12)
Rutgers University, Department of Computer
Science, New Brunswick, NJ, USA

Marcin Bienkowski (13)
Institute of Computer Science,
University of Wrocław, Poland

Philip Bille (14)
DTU Compute, Technical University of
Denmark, Lyngby, Denmark

Greg Bodwin (15)
Georgia Tech, Atlanta, GA, USA

Mikołaj Bojańczyk (113)
Institute of Informatics,
University of Warsaw, Poland

Alin Bostan (114)
INRIA Saclay Île-de-France, Palaiseau, France

Marin Bougeret (16)
LIRMM, Université de Montpellier, CNRS,
France

Alex Brandts (17)
Department of Computer Science,
University of Oxford, UK

Milutin Brankovic (18)
University of Sydney, Australia

Karl Bringmann (4, 19)
Saarland University, Saarland Informatics
Campus (SIC), Saarbrücken, Germany;
Max Planck Institute for Informatics,
Saarland Informatics Campus (SIC),
Saarbrücken, Germany

Kevin Buchin (20)
Department of Mathematics and Computer
Science, TU Eindhoven, Netherlands

Andrei A. Bulatov (21)
School of Computing Science,
Simon Fraser University, Burnaby, Canada

Georgina Bumpus (115)
University of Oxford, UK

Laurine Bénéteau (10)
Aix Marseille Univ, Université de Toulon, CNRS,
LIS, Marseille, France

Michaël Cadilhac (116, 117)
DePaul University, Chicago, IL, USA

Jin-Yi Cai (22, 23, 66)
Department of Computer Sciences, University of
Wisconsin-Madison, Madison, WI, USA

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Arnaud Carayol (114)
LIGM, Univ. Gustave Eiffel, CNRS,
Marne-la-Vallée, France

Titouan Carette (118)
Université de Lorraine, CNRS, Inria, LORIA,
Nancy, France

Ruoxu Cen (24)
Institute for Interdisciplinary Information
Sciences, Tsinghua University, Beijing, China

Amit Chakrabarti (11)
Dartmouth College, Hanover, NH, USA

Diptarka Chakraborty (25)
National University of Singapore, Singapore

Jérémie Chalopin (10)
Aix Marseille Univ, Université de Toulon, CNRS,
LIS, Marseille, France

Timothy F. N. Chan (26)
School of Mathematical Sciences, Monash
University, Melbourne, Australia;
Mathematics Institute and DIMAP,
University of Warwick, Coventry, UK

Panagiotis Charalampopoulos (27)
Department of Informatics, King's College
London, UK;
Institute of Informatics,
University of Warsaw, Poland

Rohit Chatterjee (28)
Stony Brook University, NY, USA

Shiri Chechik (29, 81)
Blavatnik School of Computer Science,
Tel Aviv University, Israel

Yu Chen (30)
Department of Computer and Information
Science, University of Pennsylvania,
Philadelphia, PA, USA

Victor Chepoi (10)
Aix Marseille Univ, Université de Toulon, CNRS,
LIS, Marseille, France

Dmitry Chistikov (116, 119)
Centre for Discrete Mathematics and its
Applications (DIMAP) & Department of
Computer Science, University of Warwick,
Coventry, UK

Man-Kwun Chiu (31)
Institut für Informatik, Freie Universität Berlin,
Germany

Aruni Choudhary (31)
Institut für Informatik, Freie Universität Berlin,
Germany

Keerti Choudhary (15, 25)
Tel Aviv University, Israel

George Christodoulou (32)
Department of Computer Science,
University of Liverpool, UK

Julia Chuzhoy (33)
Toyota Technological Institute at Chicago,
IL, USA

Laura Ciobanu (120)
Heriot-Watt University, Edinburgh,
Scotland, UK

Lorenzo Clemente (109, 121)
Institute of Informatics,
University of Warsaw, Poland

Ilan Cohen (34)
Jether Energy Ltd, Tel Aviv, Israel

Thomas Colcombet (109)
IRIF-CNRS-Université de Paris, France

Jacob W. Cooper (26)
Faculty of Informatics, Masaryk University,
Brno, Czech Republic

Raphaëlle Crubillé (108)
IMDEA Software Institute, Madrid, Spain;
University of Paris, IRIF, France

Amineh Dadsetan (21)
School of Computing Science,
Simon Fraser University, Burnaby, Canada

Ugo Dal Lago (108)
University of Bologna, Italy;
INRIA, Sophia Antipolis, France

Mina Dalirrooyfard (35)
MIT, Cambridge, MA, USA

Samir Datta (122)
Chennai Mathematical Institute, India

Laure Daviaud (123)
CitAI, Department of Computer Science, City,
University of London, UK

Anuj Dawar (36)
Department of Computer Science and
Technology, University of Cambridge, UK

Joel D. Day (124)
Loughborough University, UK

Bart de Keijzer (71)
King's College London, UK

Anindya De (37)
Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA, USA

Argyrios Deligkas (38)
Royal Holloway University of London, UK

Alberto Dennunzio (125)
Dipartimento di Informatica, Sistemistica e Comunicazione, Università degli Studi di Milano-Bicocca, Milano, Italy

Dean Doron (39)
Department of Computer Science, Stanford University, CA, USA

Jan Dreier (40)
Department of Computer Science, RWTH Aachen University, Germany

Andrew Drucker (9)
University of Chicago, IL, USA

Ran Duan (24, 41)
Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China

Shaddin Dughmi (42)
Department of Computer Science, University of Southern California, Los Angeles, CA, USA

Martin Dyer (66)
School of Computing, University of Leeds, UK

Eduard Eiben (43)
Department of Computer Science, Royal Holloway, University of London, Egham, United Kingdom

Jonas Ellert (14)
Department of Computer Science, Technical University of Dortmund, Germany

Alina Ene (6)
Department of Computer Science, Boston University, MA, USA

Rogers Epstein (98)
Massachusetts Institute of Technology, Cambridge, MA, USA

Chenglin Fan (20)
Department of Computer Science, University of Texas at Dallas, Richardson, TX, USA

John Fearnley (38)
University of Liverpool, UK

Uriel Feige (44)
The Weizmann Institute, Rehovot, Israel

Moran Feldman (6)
Department of Computer Science, University of Haifa, Israel

Shon Feller (5)
University of Haifa, Israel

Hendrik Fichtenberger (45)
Department of Computer Science, TU Dortmund, Germany

Andrés Fielbaum (46)
Department of Cognitive Robotics, Faculty of Mechanical, Maritime and Materials Engineering, TU Delft, The Netherlands

Michael Figelius (126)
Universität Siegen, Germany

Emmanuel Filiot (127)
Université libre de Bruxelles (ULB), Belgium

Arnold Filtser (47, 48)
Department of Computer Science, Columbia University, New York, NY, USA

Omrit Filtser (48)
Department of Applied Mathematics and Statistics, Stony Brook University, NY, USA

Johannes Fischer (14)
Department of Computer Science, Technical University of Dortmund, Germany

Nick Fischer (19)
Saarland University, Saarland Informatics Campus (SIC), Saarbrücken, Germany; Max Planck Institute for Informatics, Saarland Informatics Campus (SIC), Saarbrücken, Germany

Fedor V. Fomin (49)
University of Bergen, Norway

Enrico Formenti (125)
Université Côte d'Azur, CNRS, I3S, Nice, France

Dimitris Fotakis (50, 51)
National Technical University of Athens, Greece

Pierre Fraigniaud (128)
Institut de Recherche en Informatique Fondamentale, CNRS, Université de Paris, France

Zhiguo Fu (22)
School of Information Science and Technology
and KLAS, Northeast Normal University,
Changchun, China

Martin Fürer (52)
Pennsylvania State University,
University Park, PA, USA

Marco Gaboardi (129)
Boston University, MA, USA

Martin Gairing (32)
Department of Computer Science,
University of Liverpool, UK

Andreas Galanis (53)
Department of Computer Science,
University of Oxford, UK

Moses Ganardi (126)
Universität Siegen, Germany

Arun Ganesh (54)
Department of Electrical Engineering and
Computer Sciences, University of California at
Berkeley, CA, USA

Chaya Ganesh (55)
Department of Computer Science and
Automation, Indian Institute of Science,
Bangalore, India

Robert Ganian (43)
Algorithms and Complexity Group,
TU Wien, Austria

Mingze Gao (45)
Department of Computer Science, University of
Sheffield, UK

Paritosh Garg (56)
EPFL, Lausanne, Switzerland

Paweł Gawrychowski (27, 57)
Institute of Computer Science,
University of Wrocław, Poland

Maciej Gazda (130)
Department of Computer Science,
University of Sheffield, UK

Raffaella Gentilini (127)
University of Perugia, Italy

Prantar Ghosh (11)
Dartmouth College, Hanover, NH, USA

Yiannis Giannakopoulos (32)
Operations Research Group,
TU Munich, Germany

Anna Gilbert (58)
Department of Mathematics, Yale University,
New Haven, CT, USA

Pierre Gillibert (131)
Institut für Diskrete Mathematik und Geometrie,
Technische Universität Wien, Austria

Leslie Ann Goldberg (53)
Department of Computer Science,
University of Oxford, UK

Artem Govorov (66)
Department of Computer Sciences, University of
Wisconsin-Madison, Madison, WI, USA

Petr Gregor (60)
Department of Theoretical Computer Science
and Mathematical Logic, Charles University,
Prague, Czech Republic

Darij Grinberg (125)
Mathematisches Forschungsinstitut Oberwolfach,
Oberwolfach-Walke, Germany

Vadim Grinberg (44)
Toyota Technological Institute at Chicago,
IL, USA

Nikola Grujic (18)
University of Sydney, Australia

Albert Gu (58)
Department of Computer Science,
Stanford University, CA, USA

Yong Gu (24)
Institute for Interdisciplinary Information
Sciences, Tsinghua University, Beijing, China

Heng Guo (53)
School of informatics, University of Edinburgh,
UK

Rohit Gurjar (61)
Indian Institute of Technology Bombay, India

Venkatesan Guruswami (62)
Carnegie Mellon University, Pittsburgh,
PA, USA

Alexander Göke (59)
Technische Universität Hamburg, Germany

Mika Göös (9)
Stanford University, CA, USA

Inge Li Gørtz (14)
DTU Compute, Technical University of
Denmark, Lyngby, Denmark

Christoph Haase (115, 119)
Department of Computer Science,
University College London, UK

Tuomas Hakoniemi (63)
Universitat Politècnica de Catalunya,
Barcelona, Spain

Thekla Hamm (43)
Algorithms and Complexity Group,
TU Wien, Austria

Sariel Har-Peled (64)
Department of Computer Science, University of
Illinois at Urbana-Champaign, IL, USA

Haoqing He (41)
Institute for Interdisciplinary Information
Sciences, Tsinghua University, Beijing, China

Danny Hermelin (4, 19)
Department of Industrial Engineering and
Management, Ben-Gurion University of the
Negev, Beersheba, Israel

Hiroshi Hirai (65)
Department of Mathematical Informatics,
Graduate School of Information Science and
Technology, The University of Tokyo, Japan

Carlos Hoppen (52)
Universidade Federal do Rio Grande do Sul,
Porto Alegre, Brazil

Mathieu Hoyrup (132)
Université de Lorraine, CNRS, Inria, LORIA,
Nancy, France

Lunjia Hu (9)
Stanford University, CA, USA

Motoki Ikeda (65)
Department of Mathematical Informatics,
Graduate School of Information Science and
Technology, The University of Tokyo, Japan

Sungjin Im (34)
Department of Computer Science and
Engineering, University of California Merced,
CA, USA

Nicole Immorlica (8)
Microsoft Research, Cambridge, MA, USA

Taisuke Izumi (67)
Nagoya Institute of Technology, Japan

Maximilian Janke (68)
Department of Computer Science,
Technical University of Munich, Germany

Bart M. P. Jansen (16)
Eindhoven University of Technology,
The Netherlands

Emmanuel Jeandel (118)
Université de Lorraine, CNRS, Inria, LORIA,
Nancy, France

Zhihao Jiang (69)
Tsinghua University, Beijing, China

Mitchell Jones (64)
Department of Computer Science, University of
Illinois at Urbana-Champaign, IL, USA

Julius Jonušas (131)
Institut für Diskrete Mathematik und Geometrie,
Technische Universität Wien, Austria

Marcin Jurdziński (123)
Department of Computer Science,
University of Warwick, Coventry, UK

Sagar Kale (56)
University of Vienna, Austria

Vardis Kandiros (50)
Massachusetts Institute of Technology,
Cambridge, MA, USA

Sampath Kannan (30)
Department of Computer and Information
Science, University of Pennsylvania,
Philadelphia, PA, USA

Matthew J. Katz (48)
Department of Computer Science, Ben-Gurion
University of the Negev, Beer Sheva, Israel

Telikepalli Kavitha (70)
Tata Institute of Fundamental Research,
Mumbai, India

Loukas Kavouras (51)
National Technical University of Athens, Greece

Edon Kelmendi (107)
Department of Computer Science,
Oxford University, UK

Thomas Kesselheim (72)
University of Bonn, Germany

Sanjeev Khanna (30, 37)
Department of Computer and Information
Science, University of Pennsylvania,
Philadelphia, PA, USA

Sandra Kiefer (73)
RWTH Aachen University, Germany

Stefan Kiefer (3, 115)
Department of Computer Science, University of Oxford, United Kingdom

Fabian Klute (43)
Algorithms and Complexity Group,
TU Wien, Austria

Florent Koechlin (114)
LIGM, Univ. Gustave Eiffel, CNRS,
Marne-la-Vallée, France

Phokion G. Kolaitis (106)
University of California, Santa Cruz, CA, USA;
IBM Research - Almaden, CA, USA

Michael Kompatscher (131)
Department of Algebra, Faculty of Mathematics and Physics, Charles University,
Prague, Czech Republic

Tsvi Kopelowitz (74)
Department of Computer Science,
Bar-Ilan University, Ramat Gan, Israel

Matias Korman (75)
Department of Computer Science,
Tufts University, Medford, MA, USA

Egor V. Kostylev (112)
University of Oxford, UK

Grigorios Koumoutsos (51)
Université libre de Bruxelles, Belgium

Martin Koutecký (26)
Computer Science Institute, Charles University,
Prague, Czech Republic

Marcin Kozik (110)
Theoretical Computer Science, Faculty of Mathematics and Computer Science,
Jagiellonian University, Kraków, Poland

Robert Krauthgamer (2)
Weizmann Institute of Science, Rehovot, Israel

Daniel Král' (26)
Faculty of Informatics, Masaryk University,
Brno, Czech Republic;
Mathematics Institute, DIMAP and Department of Computer Science, University of Warwick,
Coventry, UK

Pankaj Kumar (122)
Chennai Mathematical Institute, India;
Department of Applied Mathematics,
Charles University, Prague, Czech Republic

Florian Kurpicz (14)
Department of Computer Science,
Technical University of Dortmund, Germany

Maria Kyropoulou (71)
University of Essex, UK

Thijs Laarhoven (76)
Eindhoven University of Technology,
The Netherlands

Sławomir Lasota (121)
University of Warsaw, Poland

Huan Li (37)
Department of Computer and Information Science, University of Pennsylvania,
Philadelphia, PA, USA

Yi Li (77)
Nanyang Technological University, Singapore,
Singapore

Thanasis Lianeas (50)
National Technical University of Athens, Greece

Xiao Liang (28)
Stony Brook University, NY, USA

Andrea Lincoln (78)
MIT, Cambridge, MA, USA

Mingmou Liu (79)
State Key Laboratory for Novel Software Technology, Nanjing University, China

Tianyu Liu (23)
University of Wisconsin-Madison,
Madison, WI, USA

Alan D. Logan (120)
Heriot-Watt University, Edinburgh,
Scotland, UK

Markus Lohrey (126)
Universität Siegen, Germany

Daniel Lokshtanov (49, 80)
University of California, Santa Barbara,
CA, USA

Huan Long (141)
BASICS, Shanghai Jiao Tong University,
Shanghai, China

Henri Lotze (40)
Department of Computer Science,
RWTH Aachen University, Germany

Maarten Löffler (20)
Department of Information and Computing Sciences, Utrecht University, Netherlands

Weiyun Ma (9)
Stanford University, CA, USA

Ofer Magen (81)
Blavatnik School of Computer Science,
Tel Aviv University, Israel

Bruce M. Maggs (54)
Department of Computer Science, Duke
University, Durham, NC, USA;
Emerald Innovations, Cambridge, MA, USA

Frédéric Magniez (82)
Université de Paris, IRIF, CNRS, France

Bernardo Magri (55)
Department of Computer Science,
Aarhus University, Denmark

Mohammad Mahmoody (83)
University of Virginia, Charlottesville, VA, USA

Rupak Majumdar (111, 133)
Max Planck Institute for Software Systems,
Kaiserslautern, Germany

Florin Manea (124)
Georg-August Universität, Göttingen, Germany

Sebastian Maneth (134)
Universität Bremen, Germany

Luciano Margara (125)
Department of Computer Science and
Engineering, University of Bologna,
Cesena, Italy

Dániel Marx (59)
Max Planck Institut für Informatik, Saarland
Informatics Campus, Saarbrücken, Germany

Richard Mayr (3)
School of Informatics, University of Edinburgh,
United Kingdom

Filip Mazowiecki (117)
Max Planck Institute for Software Systems,
Saarland Informatics Campus,
Saarbücken, Germany

Brendan D. McKay (73)
Australian National University,
Canberra, Australia

Arturo Merino (84)
Technische Universität Berlin, Germany

Evi Micha (85)
University of Toronto, Canada

Ivan Mihajlin (49)
University of California, San Diego, CA, USA

Pranabendu Misra (80)
Max Planck Institut für Informatik, Saarland
Informatics Campus, Saarbrücken, Germany

Ondřej Mička (60)
Department of Theoretical Computer Science
and Mathematical Logic, Charles University,
Prague, Czech Republic

Matthias Mnich (59)
Technische Universität Hamburg, Germany

Divyarthi Mohan (8)
Princeton University, NJ, USA

Marco Molinaro (72)
PUC-Rio, Rio de Janeiro, Brazil

Ignacio Morales (46)
Departamento de Ingeniería Industrial, Escuela
de Ingeniería, Pontificia Universidad Católica,
Santiago, Chile

Antoine Mottet (131)
Department of Algebra, Faculty of Mathematics
and Physics, Charles University,
Prague, Czech Republic

Hamoon Mousavi (87)
Department of Computer Science,
University of Toronto, Canada

Mohammad Reza Mousavi (130)
School of Informatics, University of Leicester,
UK

Nikos Mouzakis (50)
National Technical University of Athens, Greece

Shay Mozes (57)
The Interdisciplinary Center Herzliya, Israel

Anish Mukherjee (122)
Institute of Informatics, University of Warsaw,
Poland

Wolfgang Mulzer (31)
Institut für Informatik, Freie Universität Berlin,
Germany

J. Ian Munro (14)
Cheriton School of Computer Science,
University of Waterloo, Canada

Jack Murtagh (39)
School of Engineering and Applied Sciences,
Harvard University, Cambridge, MA, USA

Tobias Mömke (86)
Saarland University, Saarland Informatics
Campus, Saarbrücken, Germany

Torsten Mütze (60)
Department of Computer Science, University of Warwick, Coventry, UK;
Department of Theoretical Computer Science and Mathematical Logic, Charles University, Prague, Czech Republic

Giorgi Nadiradze (7)
IST Austria, Klosterneuburg, Austria

Vasileios Nakos (77)
Universität des Saarlandes, Saarbrücken, Germany;
Max Planck Institut für Informatik, Saarland Informatics Campus, Saarbrücken, Germany

Ashwin Nayak (82)
Dept. of Combinatorics and Optimization, University of Waterloo, Canada;
IQC, University of Waterloo, Canada

Moran Nechushtan (29)
Blavatnik School of Computer Science, Tel Aviv University, Israel

Daniel Neuen (88)
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

Seyed Sajjad Nezhadi (87)
Department of Computer Science, University of Toronto, Canada

Hung Q. Ngo (106)
relationalAI, Berkeley, CA, USA

Huy L. Nguyen (6)
Khoury College of Computer and Information Science, Northeastern University, Boston, MA, USA

Lê Thành Dũng Nguyễn (135)
Laboratoire d'informatique de Paris Nord, Villetaneuse, France;
Laboratoire Cogitamus (http://www.cogitamus.fr/indexen.html)

Cyril Nicaud (114)
LIGM, Univ. Gustave Eiffel, CNRS, Marne-la-Vallée, France

Hesam Nikpey (37)
Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA, USA

Kobbi Nissim (129)
Georgetown University, Washington, DC, USA

Damian Niwiński (136)
Institute of Informatics, University of Warsaw, Poland

Martin Nöllenburg (43)
Algorithms and Complexity Group, TU Wien, Austria

Taihei Oki (89)
Department of Mathematical Informatics, Graduate School of Information Science and Technology, University of Tokyo, Japan

Yota Otachi (67)
Nagoya University, Japan

Joël Ouaknine (107)
Max Planck Institute for Software Systems, Saarland Informatics Campus, Saarbrücken, Germany;
Department of Computer Science, Oxford University, UK

Maciej Pacut (13)
Faculty of Computer Science, University of Vienna, Austria

Omkant Pandey (28)
Stony Brook University, NY, USA

Debmalya Panigrahi (34, 54, 69)
Department of Computer Science, Duke University, Durham, NC, USA

Fahad Panolan (80)
IIT Hyderabad, India

Charles Paperman (117)
Université de Lille, Villeneuve d'Ascq, France

Pál András Papp (90, 91)
ETH Zürich, Switzerland

Merav Parter (15, 33)
The Weizmann Institute of Science, Rehovot, Israel

Paweł Parys (109)
Institute of Informatics, University of Warsaw, Poland

Panagiotis Patsilinakos (50)
National Technical University of Athens, Greece

Erik Paul (137)
Institute of Computer Science, Leipzig University, Germany

Ami Paz (128)
Faculty of Computer Science, Universität Wien, Austria

Kristýna Pekárková (26)
Faculty of Informatics, Masaryk University,
Brno, Czech Republic

Pan Peng (45)
Department of Computer Science,
University of Sheffield, UK

Geevarghese Philip (80)
Chennai Mathematical Institute, UMI ReLaX,
Chennai, India

Krzysztof Piecuch (13)
Institute of Computer Science,
University of Wrocław, Poland

Michał Pilipczuk (117)
University of Warsaw, Poland

Michael Pinsker (131)
Institut für Diskrete Mathematik und Geometrie,
Technische Universität Wien, Austria;
Department of Algebra, Faculty of Mathematics
and Physics, Charles University, Czech Republic

Jakob Piribauer (138)
Technische Universität Dresden, Germany

Toniann Pitassi (92)
University of Toronto, Canada;
Institute for Advanced Study,
Princeton, NJ, USA

Radosław Piórkowski (121)
University of Warsaw, Poland

Karol Pokorski (27)
Institute of Computer Science,
University of Wrocław, Poland

Aleksandr Popov (20)
Department of Mathematics and Computer
Science, TU Eindhoven, Netherlands

Aditya Potukuchi (93)
Department of Computer Science,
Rutgers University, New Brunswick, NJ, USA

Diogo Poças (32)
Operations Research Group,
TU Munich, Germany

Pierre Pradic (135)
Department of Computer Science,
University of Oxford, UK

Marcin Przybyłko (136)
Fachbereich Informatik,
University of Bremen, Germany

David Purser (129)
University of Warwick, Coventry, UK;
Max Planck Institute for Software Systems,
Saarbrücken, Germany

Saladi Rahul (64)
Dept. of Computer Science and Automation,
Indian Institute of Science, Bangalore, India

Benjamin Raichel (20)
Department of Computer Science,
University of Texas at Dallas, Richardson,
TX, USA

Jean-François Raskin (127)
Université libre de Bruxelles (ULB), Belgium

Rajat Rathi (61)
Indian Institute of Technology Bombay, India

Zachary Remscrim (139)
Department of Mathematics, MIT, Cambridge,
MA, USA

Marcel Roeloffzen (20, 75)
Department of Mathematics and Computer
Science, Eindhoven University of Technology,
The Netherlands

Lars Rohwedder (56)
EPFL, Lausanne, Switzerland

Peter Rossmanith (40)
Department of Computer Science,
RWTH Aachen University, Germany

Eva Rotenberg (14)
DTU Compute, Technical University of
Denmark, Lyngby, Denmark

Atri Rudra (58)
Department of Computer Science and
Engineering, University at Buffalo, NY, USA

Christopher Ré (58)
Department of Computer Science,
Stanford University, CA, USA

Aleksi Saarela (140)
Department of Mathematics and Statistics,
University of Turku, Finland

Amirmojtaba Sabour (7)
IST Austria, Klosterneuburg, Austria

Mahmoud Salamati (133)
Max-Planck Institute for Software Systems,
Kaiserslautern, Germany

Sai Sandeep (62)
Carnegie Mellon University, Pittsburgh,
PA, USA

Bryce Sandlund (94)
Cheriton School of Computer Science,
University of Waterloo, Canada

Ignasi Sau (16, 95)
LIRMM, Université de Montpellier, CNRS,
France

Saket Saurabh (49, 80)
Department of Informatics,
University of Bergen, Norway;
The Institute of Mathematical Sciences,
Chennai, India

Rahul Savani (38)
University of Liverpool, UK

Helmut Seidl (134)
TU München, Germany

Martin P. Seybold (18)
University of Sydney, Australia

Dvir Shabtay (4, 19)
Department of Industrial Engineering and
Management, Ben-Gurion University of the
Negev, Beersheba, Israel

Nisarg Shah (85)
University of Toronto, Canada

Noa Shahar (15)
The Weizmann Institute of Science,
Rehovot, Israel

Shuai Shao (22, 96)
Department of Computer Sciences,
University of Wisconsin-Madison,
Madison, WI, USA

Nobutaka Shimizu (97)
The University of Tokyo, Japan

Takeharu Shiraga (97)
Chuo University, Tokyo, Japan

Morgan Shirley (92)
University of Toronto, Canada

Mahsa Shirmohammadi (3)
CNRS & IRIF, Université de Paris, France

Sandeep Silwal (98)
Massachusetts Institute of Technology,
Cambridge, MA, USA

Stratis Skoulakis (50, 51)
Singapore University of Technology and Design,
Singapore

Michał Skrzypczak (136)
Institute of Informatics,
University of Warsaw, Poland

Caleb Smith (83)
University of Virginia, Charlottesville, VA, USA

Sadegh Soudjani (133)
Newcastle University, Newcastle upon Tyne, UK

Frank Staals (75)
Department of Information and Computing
Sciences, Utrecht University, The Netherlands

Giannos Stamoulis (95)
Department of Informatics and
Telecommunications, National and Kapodistrian
University of Athens, Greece;
Inter-university Postgraduate Programme
“Algorithms, Logic, and Discrete Mathematics”
(ALMA), Athens, Greece

Rafał Stefański (113)
Institute of Informatics,
University of Warsaw, Poland

Paul-Ioan Stoienescu (115)
University of Oxford, UK

Hsin-Hao Su (99)
Boston College, MA, USA

Dan Suciu (106)
University of Washington, Seattle, WA, USA

Andrew Suh (6)
Department of Computer Science,
Boston University, MA, USA

Kevin Sun (69)
Duke University, Durham, NC, USA

Xiaoming Sun (100)
Institute of Computing Technology,
Chinese Academy of Sciences, Beijing, China

Yuan Sun (100)
Institute of Computing Technology,
Chinese Academy of Sciences, Beijing, China

Yuxin Sun (96)
Department of Computer Sciences,
University of Wisconsin-Madison,
Madison, WI, USA

Ola Svensson (56)
EPFL, Lausanne, Switzerland

Géraud Sénizergues (117)
Université de Bordeaux, France

Johnson Tan (110)
Department of Mathematics, University of
Illinois, Urbana-Champaign, Urbana, IL, USA

Li-Yang Tan (9)
Stanford University, CA, USA

Tony Tan (112)
National Taiwan University, Taipei, Taiwan

Zihan Tan (33)
Computer Science Department,
University of Chicago, IL, USA

Jonathan Tanner (115)
University of Oxford, UK

Anuj Tawari (122)
Chennai Mathematical Institute, India

K. S. Thejaswini (123)
Department of Computer Science,
University of Warwick, Coventry, UK

Dimitrios M. Thilikos (95)
LIRMM, Université de Montpellier, CNRS,
France

Ramanathan S. Thinniyam (111)
Max Planck Institute for Software Systems,
Kaiserslautern, Germany

Patrick Totzke (3)
Department of Computer Science,
University of Liverpool, United Kingdom

Vilmar Trevisan (52)
Universidade Federal do Rio Grande do Sul,
Porto Alegre, Brazil

Salil Vadhan (39)
School of Engineering and Applied Sciences,
Harvard University, Cambridge, MA, USA

Matt Valeriote (110)
Department of Mathematics and Statistics,
McMaster University, Hamilton,
Ontario, Canada

André van Renssen (18, 75)
School of Computer Science,
University of Sydney, Australia

Manolis Vardas (51)
ETH Zurich, Switzerland

Virginia Vassilevska Williams (35, 74)
EECS Department, MIT, Cambridge, MA, USA

Yann Vaxès (10)
Aix Marseille Univ, Université de Toulon, CNRS,
LIS, Marseille, France

Carmine Ventre (71)
King's College London, UK

Daniele Venturi (55)
Department of Computer Science,
Sapienza University of Rome, Italy

José Verschae (46)
Instituto de Ingeniería Matemática y
Computacional, Facultad de Matemáticas y
Escuela de Ingeniería,
Pontificia Universidad Católica, Santiago, Chile

Nils Vortmeier (122)
TU Dortmund, Germany

Magnus Wahlström (101)
Department of Computer Science,
Royal Holloway, University of London, UK

Clara Waldmann (32)
Operations Research Group,
TU Munich, Germany

Jiaheng Wang (100)
School of Electronics Engineering and Computer
Science, Peking University, Beijing, China

Thomas Watson (92)
University of Memphis, TN, USA

Roger Wattenhofer (90, 91)
ETH Zürich, Switzerland

Oren Weimann (5, 57)
University of Haifa, Israel

Nicole Wein (99)
Massachusetts Institute of Technology,
Cambridge, MA, USA

S. Matthew Weinberg (8)
Princeton University, NJ, USA

Armin Weiß (102)
Universität Stuttgart, Institut für Formale
Methoden der Informatik (FMI), Germany

Philip Wellnitz (19)
Max Planck Institute for Informatics,
Saarland Informatics Campus (SIC),
Saarbrücken, Germany

Daniel Wiebking (103)
RWTH Aachen University, Germany

Andreas Wiese (84, 86)
Department of Industrial Engineering,
Universidad de Chile, Santiago, Chile

Gregory Wilsenach (36)
Department of Computer Science and Technology, University of Cambridge, UK

Dominik Wojtczak (3)
Department of Computer Science, University of Liverpool, United Kingdom

Mary Wootters (58)
Departments of Computer Science and Electrical Engineering, Stanford University, CA, USA

James Worrell (107)
Department of Computer Science, Oxford University, UK

Marcin Wrochna (17)
Department of Computer Science, University of Oxford, UK

David J. Wu (83)
University of Virginia, Charlottesville, VA, USA

Hongxun Wu (105)
Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China

Kewen Wu (100)
School of Electronics Engineering and Computer Science, Peking University, Beijing, China

Michał Włodarczyk (104)
Eindhoven University of Technology, The Netherlands

Zhiyu Xia (100)
Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China

Xian Xu (141)
East China University of Science and Technology, Shanghai, China

Yinzhan Xu (94)
CSAIL, Massachusetts Institute of Technology, Cambridge, MA, USA

Kuan Yang (53)
Department of Computer Science, University of Oxford, UK

Andrew Chi chih Yao (1)
Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China; Shanghai Qi Zhi Institute, China

Adam Yedidia (78)
MIT, Cambridge, MA, USA

Qiang Yin (141)
Alibaba Group, Shanghai, China

Yitong Yin (79)
State Key Laboratory for Novel Software Technology, Nanjing University, China

Huacheng Yu (79)
Princeton University, NJ, USA

Henry Yuen (87)
Department of Computer Science and Department of Mathematics, University of Toronto, Canada

Meirav Zehavi (49)
Ben-Gurion University of the Negev, Beer-Sheva, Israel

Georg Zetzsche (111, 116, 126)
Max Planck Institute for Software Systems (MPI-SWS), Kaiserslautern, Germany

Thomas Zeume (122)
Ruhr-Universität Bochum, Germany

Tianyi Zhang (41)
Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China

Wenbo Zhang (141)
BASICS, Shanghai Jiao Tong University, Shanghai, China

Yufan Zheng (100)
Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China

David Zuckerman (39)
Department of Computer Science, University of Texas at Austin, TX, USA

Stanislav Živný (17)
Department of Computer Science, University of Oxford, UK

An Incentive Analysis of Some Bitcoin Fee Designs

Andrew Chi chih Yao
Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China
Shanghai Qi Zhi Institute, China
https://iiis.tsinghua.edu.cn/en/yao

Abstract

In the Bitcoin system, miners are incentivized to join the system and validate transactions through fees paid by the users. A simple "pay your bid" auction has been employed to determine the transaction fees. Recently, Lavi, Sattath and Zohar [8] proposed an alternative fee design, called the *monopolistic price* (MP) mechanism, aimed at improving the revenue for the miners. Although MP is not strictly incentive compatible (IC), they studied how close to IC the mechanism is for iid distributions, and conjectured that it is nearly IC asymptotically based on extensive simulations and some analysis. In this paper, we prove that the MP mechanism is nearly incentive compatible for any iid distribution as the number of users grows large. This holds true with respect to other attacks such as splitting bids. We also prove a conjecture in [8] that MP dominates the RSOP auction in revenue (originally defined in Goldberg et al. [5] for digital goods). These results lend support to MP as a Bitcoin fee design candidate. Additionally, we explore some possible intrinsic correlations between incentive compatibility and revenue in general.

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

Keywords and phrases Bitcoin, blockchain, incentive compatibility, maximum revenue, mechanism design

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.1

Category Invited Talk

Related Version https://arxiv.org/abs/1811.02351

1 Introduction

Bitcoin, and more broadly *blockchain* systems, rely on the willingness of honest players to participate in the system. A good blockchain system should have simple, practical designs with suitable security guarantees against cheating. The *incentive compatibility* (IC) concept which seeks to incentivize the participants to be truthful is playing an increasingly central role in the design of distributed financial systems.

Recently, Lavi, Sattath and Zohar [8] started a study of the subject of *Bitcoin Fee Market* design. In this market, there are two kinds of players: the *users* who have transaction records that need to be certified and registered in the bitcoin system, and the *miners* who create new blocks to include the transactions and get them certified. Each user declares the maximal amount she is willing to pay for her transaction, and the miners use a mechanism to decide which transactions to include and how much fee to charge each user. A primary focus of their study is the *Monopolistic Price (MP)* mechanism, which is a natural and practical mechanism, although not IC in the strict sense (see Section 2 below). Their extensive simulations indicate that the mechanism does not deviate too much from being IC for most iii distributions, as the number of users n grows large. An analysis was given for the special case of discrete distributions of finite size. They suggest that MP might be a good alternative to the "pay your bid" auction, which is subject to low bids and revenue. It is posed as a conjecture that the MP mechanism is nearly-IC for general iid distributions.

We will prove that MP is nearly-IC for any iid distribution as n grows large, thus mathematically validating the strong simulation results obtained in [8]. Note that the standard IC criterion (in auction market) only addresses one kind of attack, namely, the reporting of an untruthful bid value. There are other possible attacks by Bitcoin users: for example, in the *multiple strategic bids* (MSB) attack discussed in [8], a user could gain advantage by splitting his transaction into several transactions and bidding separately. This strategy could even enable a losing transaction to be included in the block. We will also prove the nearly-IC conjecture for MP with respect to the MSB attack.

We consider another mechanism, called the RSOP (Random Sampling Optimal Price) auction, which was first defined by Goldberg et al. [5] in the digital goods context and shown to be truthful. We prove that the RSOP revenue is always dominated by the MP revenue, as conjectured in [8]. In fact, the revenue difference can be arbitrarily large for some distributions, prompting us to look more deeply into possible correlation between IC and revenue in general (Theorem 6-8).

The contributions of the present paper are two-fold. First, we prove the Monopolistic Price mechanism to be nearly-IC, confirming the previous strong experimental data. This holds true against the MSB attack as well. We also show MP to dominate the RSOP auction in revenue. These results lend support to MP as a Bitcoin fee design candidate. Secondly, the methodology used in our proofs involves sophisticated mathematical analysis. It demonstrates that theoretical computer science can provide powerful tools to complement system design for blockchains. Finally, we believe that the emerging area of incentive compatible blockchain design is an exciting research area with many intriguing problems to solve, for theorists and system designers alike.

Related Work. The basic model for Bitcoin fee market introduced in [8] in fact resembles the maximum revenue problem for *Digital Goods* as considered by Goldberg et al. [4][5]. The MP mechanism is similar to the optimal omniscient auction in [5]. However, The Bitcoin fee market differs from digital goods in its additional features: such as the auctioneer may delete or insert bids, or the users may split bids. This makes the Bitcoin fee design a rich and relevant new research subject for auction theory and mechanism design. Among those work closely related to the current subject include Babaioff et al. [1], Kroll et al. [7], Carlsten et al. [3], Bonneau [2], Huberman et al. [6]. To formulate meaningful incentive models, there is much research work in Bitcoin which provides important ingredients for consideration. A more complete survey of related work can be found in [8, Section 1.3].

2 Review of the Models and Known Results

For background, we first review some Bitcoin fee models and mechanisms previously considered. A *miner* acts as a monopolist who offers n users to include their transactions in the miner's next block for a fee. Each user i has one transaction that needs this service and is willing to pay a fee up to some value v_i. The miner's problem is to design a mechanism to extract good revenue. The standard Bitcoin mechanism in use is a pay-your-bid system, where the miner simply takes the highest bids to fill the capacity of the block. This mechanism may not receive good revenue, since some bidders may not reveal the true value of the fees they are willing to pay. In view of this, some alternative mechanisms are proposed in [8] and their security properties considered, which we will review below.

2.1 The Monopolistic Price (MP) Mechanism

Suppose user i bids v_i for $1 \leq i \leq n$. Sort $v_1, \cdots, v_n$ into a decreasing sequence $b_1 \geq b_2 \geq \cdots \geq b_n$. Consider the expression $k \cdot b_k$ and let $k^*(v)$ be the k maximizing $k \cdot b_k$ over all k (in case of ties, k^* is taken to be the maximal one). The miner will include all users with the highest bids $b_1, b_2, \cdots b_{k^*(v)}$ and simply charge each one of them the same fee $b_{k^*(v)}$. Call this the *monopolistic price* $p^{mono}(v) = b_{k^*(v)}$. This mechanism gives the miner revenue $MP(v) = k^* \cdot b_{k^*(v)}$, which is obviously the maximum revenue obtainable if all accepted bids must be charged a single price.

The above MP Mechanism is not truthful. To analyze how serious the non-truthfulness can be, consider the following model. For any user i with bid v_i and the vector of bids $v_{-i} = (v_1, \cdots, v_{i-1}, v_{i+1}, \cdots v_n)$ of the other users, let

$$p^{honest}(v_i, v_{-i}) = p^{mono}(v_i, v_{-i}), \text{ and}$$
$$p^{strategic}(v_{-i}) = \min\{v_i \,|p^{mono}(v_i, v_{-i}) \leq v_i\}.$$

The temptation for user i to *shade* her bids can be measured by the *discount ratio* δ_i defined as:

$$\delta_i(v_i, v_{-i}) = \begin{cases} 1 - \frac{p^{strategic}(v_{-i})}{p^{honest}(v_i, v_{-i})} & \text{if } v_i \geq p^{strategic}(v_{-i}), \\ 0 & \text{otherwise.} \end{cases}$$

This ratio captures the gain a user can obtain by bidding strategically (instead of truthfully). Assume all true values v_i are drawn iid from some distribution F on $[0, \infty)$. Two measures are defined: the worst-case measure and the average case one. For the former, let

$$\delta_{max}(v) = \max_i \delta_i(v_i, v_{-i}),$$
$$\Delta_n^{max} = E_{(v_1, \cdots, v_n) \sim F}[\delta_{max}(v)].$$

For the latter, let

$$\Delta_n^{average} = E_{(v_1, \cdots, v_n) \sim F}[\delta_1(v_1, v_{-1})].$$

Clearly, for every F and n, we have $\Delta_n^{max} \geq \Delta_n^{average}$, since $\delta_{max}(v) \geq \delta_i(v_i, v_{-i})$ for any v. For the special case of discrete distributions of finite size, the following is known.

▶ **Theorem A** ([8, Theorem 2.3]). *For any distribution F with a finite support size, $\lim_{n\to\infty} \Delta_n^{max}(F) = 0$. (This implies also $\lim_{n\to\infty} \Delta_n^{average}(F) = 0$ for such F.)*

Based on extensive simulations done for a variety of distributions, it was conjectured that MP is nearly IC for general iid distributions as n gets large.

Nearly IC Conjecture for MP

1. For any distribution F, $\lim_{n\to\infty} \Delta_n^{average}(F) = 0$. Specifically $\Delta_n^{average}(F) = O(\frac{1}{n})$.
2. If F has a bounded support, $\lim_{n\to\infty} \Delta_n^{max}(F) = 0$. Specifically $\Delta_n^{max}(F) = O(\frac{1}{n})$.
3. There exists a distribution F with an unbounded support such that $\lim_{n\to\infty} \Delta_n^{max}(F) > 0$.

In the O-notation above, the constants may depend on F. The main purpose of our paper is to settle the Nearly IC Conjecture for MP in the positive (see Theorem 1–3).

A stronger attack, called the **Multiple Strategic Bids** (MSB) attack was also raised in [8]. They observed that a user could gain advantage by splitting his bid into several transactions with separate bids; this strategy could enable a losing transaction to be included in the block. We note that, under this strategy,

$$p^{multi}(v_{-i}) = \min\left\{u \cdot v_i^{(u)} \,|\, v_i^{(1)} \geq \cdots \geq v_i^{(u)} \geq p^{mono}(v_i^{(1)}, \cdots, v_i^{(u)}, v_{-i})\right\},$$

$$\delta_i(v_i, v_{-i}) = \begin{cases} 1 - \frac{p^{multi}(v_{-i})}{p^{honest}(v_i, v_{-i})} & \text{if } \ v_i \geq p^{multi}(v_{-i}), \\ 0 & \text{otherwise.} \end{cases}$$

We show that the Nearly IC Conjecture for MP remains true under the MSB attack (see Theorem 4) as conjectured in [8].

2.2 The Random Sampling Optimal Price (RSOP) Auction

We consider another mechanism, called the RSOP (Random Sampling Optimal Price) auction, first defined by Goldberg et al. [5] in the digital goods context.

▶ **Definition** (RSOP auction). *Upon receiving n bids $v = (v_1, \cdots, v_n)$, the auctioneer randomly partitions the bids into two disjoint sets A and B, and computes the monopolistic price for each set: P_A^{mono}, P_B^{mono} (with the monopolistic price for an empty set being set to 0). Finally, the set of winning bids is $A' \cup B'$, where $A' = \{i \in A : v_i \geq P_B^{mono}\}$ and $B' = \{i \in B : v_i \geq P_A^{mono}\}$. The bidders in A' each pays P_B^{mono}, and the bidders in B' each pays P_A^{mono}.*

Note the revenue obtained in this auction is

$$RSOP(v) = |A'| \cdot P_B^{mono} + |B'| \cdot P_A^{mono}.$$

▶ **Theorem B** (Goldberg et al. [5]). *The RSOP auction is truthful. For any $v = (v_1, \cdots, v_n)$ with $v_i \in [1, D]$ (where D is a constant) for all i, we have*

$$\lim_{n\to\infty} \max_{v} \frac{MP(v)}{RSOP(v)} = 1.$$

In [8], several variants of RSOP were examined and simulation carried out which led to the following conjecture.

Dominance Conjecture of MP over RSOP. [8, Conjecture 5.4]
For any v and all choices of A and B, the RSOP revenue is at most the monopolistic price revenue. That is, $RSOP(v) \leq MP(v)$.

The MP Dominance Conjecture has relevance to the robustness of RSOP against adding false bids or deleting bids by the auctioneer (see discussions in [8]). In the present paper we prove the MP Dominance Conjecture to be true.

3 Main Results

In this paper we settle the Nearly-IC Conjecture (even allowing for the MSB attack) and the MP Dominance Conjecture mentioned above: the former in Theorem 1–4, and the latter in Theorem 5. Additionally, we investigate the possible correlation between incentive compatibility and revenue. In this regard, we demonstrate that distributions with unbounded support can exhibit different characteristics from the bounded ones, and these findings will be presented in Theorems 6–8.

3.1 Nearly Incentive Compatibility of MP

We prove the Nearly-IC Conjecture, part (1)-(3), in Theorems 1-3 respectively. Let the *Inverse* distribution *Inv* be defined as $Pr_{Inv}\{X > x\} = \frac{1}{x}$ for $x \in [1, \infty)$.

▶ **Theorem 1.** *For any distribution F on bounded support, $\lim_{n\to\infty} \Delta_n^{max}(F) = 0$.*

▶ **Theorem 2.** *For any distribution F, $\lim_{n\to\infty} \Delta_n^{average}(F) = 0$.*

▶ **Theorem 3.** *For $F = Inv$, $\lim_{n\to\infty} \Delta_n^{max}(F) > c$ for all n, where $c > 0$ is some absolute constant.*

▶ **Theorem 4.** *With respect to the MSB attack, the MP mechanism is nearly incentive compatible, i.e., Theorems 1-2 are still valid.*

▶ Remark. The proof of Theorem 4 uses similar techniques as Theorems 1-2 and will be omitted in this paper. We remark that Theorem 1 can be refined to show that $\Delta_n^{max}(F) = O(\frac{1}{n^\beta})$ where $\beta > 0$ is a constant independent of F, while the constant in the O-notation is F-dependent. Similarly, Theorem 2 can be strengthened to $\Delta_n^{average}(F) = O(\frac{1}{n^\beta})$ when F satisfies $\sup_x x(1 - F(x)) < \infty$. The analysis follows the same outline as the proofs for Theorems 1, 2 above but the details are more complicated. They will be left for a later version of the paper.

3.2 The Effect of IC on Revenue

The following theorem settles the Dominance Conjecture of MP over RSOP.

▶ **Theorem 5.** *For any v, $RSOP(v) \le MP(v)$.*

Theorem B of Goldberg et al. [5] says that, RSOP yields asymptotically the same revenue as the MP mechanism when the distribution F has a bounded support $[1, D]$. We point out that this is not always true when F has infinite support. For the distribution $F = Inv$, MP can extract infinite revenue versus a finite amount by RSOP.

▶ **Theorem 6.** *For $F = Inv$,*

$$\lim_{n\to\infty} \frac{1}{n} E_{v_1,\cdots,v_n\sim F}[MP(v)] = \infty, \text{ \textit{while}}$$
$$\lim_{n\to\infty} \frac{1}{n} E_{v_1,\cdots,v_n\sim F}[RSOP(v)] = 1.$$

One may ask, could the revenue gap between MP and RSOP be attributable to the fact that RSOP is IC while MP drastically deviates from IC (as demonstrated in Theorem 3)? In other words, could the property derived in Theorem 3, $\lim_{n\to\infty} \Delta_n^{max}(F) > c$, be a necessary condition for any mechanism to achieve infinite revenue on $F = Inv$? The following theorem shows that this is not the case.

Let $n \ge 1$ and M be a fee mechanism for n bidders. For a bid vector $v = (v_1, \cdots, v_n)$, let $\delta_{max}(M, v)$ be the maximum of $\delta_i(v_i, v_{-i})$ for any i when M is applied to v. Let $\Delta_n^{max}(M, F) = E_{v_1,\cdots,v_n\sim F}[\delta_{max}(M, v)]$. Also, let $M(v)$ denote the revenue collected by M.

▶ **Theorem 7.** *For any n and $\epsilon > 0$, there exists a fee mechanism M for n users such that $\Delta_n^{max}(M, Inv) \le \epsilon$ and $E_{v_1,\cdots,v_n\sim Inv}[M(v)] = \infty$.*

Note that δ_{max} and Δ_n^{max} are not the only ways to quantify a mechanism's closeness to being incentive compatible. Two standard ways to define being *ϵ-close* to IC (or more generally, to Nash equilibrium) are

(1) Additively ϵ-close: $p(v_i, v_{-i}) - p_i^*(v_{-i}) \leq \epsilon$,

(2) Multiplicatively ϵ-close: $p(v_i, v_{-i}) - p_i^*(v_{-i}) \leq \epsilon(v_i - p(v_i, v_{-i}))$,

where $p_i^*(v_{-i})$ is defined as the *minimum* bid value u for user i that would allow her to be among the winners when the bid vector is $v = (u, v_{-i})$.

We will show that, adopting the "multiplicatively ϵ-close" definition, one can obtain nearly IC mechanisms that derive infinite revenue under the distribution $F = Inv$.

Let M be any IC mechanism (such as RSOP). For any bid vector v, let $p_i(v)$ be the fee paid by user i (if i is a winner), and $u_i(v) = v_i - p_i(v)$ be the *utility.* For any $0 \leq \epsilon < 1$, let M_ϵ be the mechanism that uses the same allocation rule as M, with the fee modified to be $p_i'(v) = p_i(v) + \frac{\epsilon}{1+\epsilon} u_i(v)$. The following theorem is easy to prove.

▶ **Theorem 8.**

(a) *M_ϵ is multiplicatively ϵ-close to IC;*

(b) $\lim_{n\to\infty} \frac{1}{n} E_{v\sim F}[R_{M_\epsilon}(v)] = \lim_{n\to\infty} \frac{1}{n} E_{v\sim F}[R_M(v)] + \frac{\epsilon}{1+\epsilon} \lim_{n\to\infty} \frac{1}{n} E_{v\sim F}(\sum_i u_i(v))$.

Theorem 8 implies that RSOP can be easily modified to become a multiplicatively ϵ-close-to-IC mechanism such that, like MP, its revenue is infinite under $F = Inv$.

We will prove Theorems 1-3, 5 in the following sections. The proof for the Multiple Strategic Bids model of Theorem 4 is similar in essence to the basic model, and hence will be omitted. We also leave out the proofs of Theorem 6-8.

4 An Overview of the Proof for Theorem 1

Theorem 1 is the most difficult to prove. In this section we give some intuition and an overview of the proof.

Let F be a distribution over $[0, D]$. Let $v_1, \cdots, v_n$ be generated according to iid F, and denote by $b_1 \geq b_2 \geq \cdots \geq b_n$ the sorted list of the $v_i's$. By Claim A9 in [8], $\delta_{max}(b) = \delta_1(b_1, b_{-1})$, i.e. the maximum discount ratio is achieved by the user with the highest bid. This leads immediately to a necessary condition on w, the optimal strategic bid by the highest bidder, as we state below.

▶ **Lemma 1** (Optimal Strategic Bid (OSB) Condition). *Let $k^* = k^*(b)$. If $\delta_{max}(b) \geq \eta$, where $0 < \eta \leq 1$, then there exists $w \in [0, (1-\eta)b_{k^*}]$ such that $i_w \cdot w \geq k^* \cdot b_{k^*} - D$, where i_w is defined by $b_{i_w} \geq w > b_{i_w+1}$.*

Lemma 1 states that, in order to have a sizable η, there has to exist a w some distance away from b_{k^*} such that $i_w \cdot w$ is only a constant D smaller than the sampled maximum $R(b) = k^* \cdot b_{k^*}$. We will prove Theorem 1 by showing that a random b is stochastically unlikely to satisfy the OSB necessary condition.

As a start, we prove Theorem 1 when the distribution F has a unique $\alpha_0 > 0$ where $A = \sup_\alpha \alpha(1 - F(\alpha))$ is achieved. The law of large number implies that, for large n, every $w \leq \alpha_0(1 - \frac{1}{2}\eta)$ satisfies $i_w \cdot w < (A - \rho)n$ (where ρ is some fixed constant) with overwhelming probability. Coupled with the fact $b_{k^*} - \alpha_0 = O(\frac{1}{\sqrt{n}})$ and $k^* \cdot b_{k^*} = A \cdot n + O(\sqrt{n})$ probabilistically, we see that the OSB condition in Lemma 1 cannot hold. Hence we have shown Theorem 1 for the case when $\sup_\alpha \alpha(1 - F(\alpha))$ has a maximum achieved at a unique point $\alpha = \alpha_0$.

The above argument does not apply when $\sup_\alpha \alpha(1-F(\alpha))$ achieves maximum value at multiple points. As an extreme case, consider the Inverse distribution modified as follows:

$$Inv^{(D)}(x) = \begin{cases} 1-\frac{1}{x} & \text{for } 1 \leq x < D, \\ 1 & \text{for } x = D. \end{cases}$$

In this case, $x(1 - Inv^{(D)}(x)) = 1$ for all $1 \leq x < D$.

Hence the challenge is to prove that, even for extreme cases like the above, the OSB condition in Lemma 1 cannot be met except with vanishingly small probability. At the top level, we wish to show that, in any two subintervals $I, J \subseteq [0, D]$ separated by a non-negligible distance, the maximum values $A_I = \max\{j \cdot b_j \,| b_j \in I\}$ and $A_J = \max\{j \cdot b_j \,| b_j \in J\}$ cannot achieve perfect correlation to allow $|A_I - A_J| = O(1)$ except with negligible probability for large n. This claim takes a non-trivial proof since intervals A_I and A_J, being taken from the sorted version b of v, are correlated to a certain degree.

Before proving Theorem 1, we first recast Lemma 1 in an new form which does not reference the quantity w. The main advantage of Lemma 1A is that, the condition now refers only to the quantities $b_1, \cdots, b_n$, thus making it easier to analyze how likely the condition can be satisfied stochastically.

▶ **Lemma 1A** (Optimal Strategic Bid (OSB) Condition). *Let $k^* = k^*(b)$ and $0 < \eta < 1$. If $\delta_{max}(b) \geq \eta$, then there exists $b_j \in [0,\, b_{k^*}(1-\frac{1}{2}\eta)]$ such that $j \cdot b_j \geq k^* \cdot b_{k^*} - \frac{2D^2}{\eta \cdot b_{k^*}}$.*

Proof. Take the w as specified in Lemma 1. The following constraints are satisfied: write $i = i_w$ and $B = k^* \cdot b_{k^*}$, then

$$b_{k^*} - \eta \cdot b_{k^*} \geq w, \tag{1}$$

$$i \cdot w \geq B - D. \tag{2}$$

Let $\Delta_j = b_{k^*} - b_j$ for $j \in \{i, i+1\}$. Let $0 \leq \lambda < 1$ such that $w = \lambda b_i + \lambda' b_{i+1}$ where $\lambda' = 1 - \lambda$. It is easy to verify from Eq. (1), (2) that

$$\lambda \Delta_i + \lambda' \Delta_{i+1} \geq \eta \cdot b_{k^*}, \tag{3}$$

$$\lambda(i \cdot b_i) + \lambda'\left((i+1)b_{i+1}\right) \geq B - D. \tag{4}$$

Note that Eq. (4) implies

$$\max\{i \cdot b_i,\, (i+1)b_{i+1}\} \geq B - D. \tag{5}$$

We now prove Lemma 1A.

Case 1. If $\Delta_i > \frac{1}{2}\eta \cdot b_{k^*}$, then choose $j \in \{i,\, i+1\}$ depending on which gives the larger $j \cdot b_j$. This j satisfies Lemma 1A, as a consequence of Eq. (1) and (5).

Case 2. $\Delta_i \leq \frac{1}{2}\eta \cdot b_{k^*}$. Eq. (3) implies $\lambda' D \geq \frac{1}{2}\eta \cdot b_{k^*}$, and thus

$$\lambda' \geq \frac{\eta \cdot b_{k^*}}{2D}. \tag{6}$$

From Eq. (4) and the fact $i \cdot b_i \leq B$, we have

$$\lambda'\left((i+1)b_{i+1}\right) \geq \lambda' B - D.$$

Using Eq. (6), we obtin

$$(i+1)b_{i+1} \geq B - \frac{2D^2}{\eta \cdot b_{k^*}}.$$

Taking $j = i+1$ satisfies Lemma 1A. ◀

5 Proof of Theorem 1

For simplicity of presentation, we assume that F has support $[1, D]$. Without loss of generality, we can assume that $F(x) < F(D) = 1$ for any $x < D$. Some additional arguments are needed for the general case $[0, D]$; we omit them here.

We will demonstrate that for a random b, the condition stated in Lemma 1A occurs with probability at most $O(\frac{(\log n)^2}{\sqrt{n}})$. That is, stochastically, the pair (b_j, b_{k^*}) with the stated property rarely exists. First some notation. Let $H_F(x) = Pr_{z\sim F}\{z \geq x\} = 1 - F(x-)$. To start the proof, we pick a point $D_1 \in [1, D]$ with the following properties:

P1: $(D_1, D]$ is a forbidden zone for b_{k^*}. Precisely, for a random b, the probability of $b_{k^*} \in (D_1, D]$ is $e^{-\Omega(n)}$.

P2: $Pr_{x\sim F}\{x \in [D_1, D]\} > 0$.

▶ **Fact 1.** *D_1 exists.*

Proof. Let α_{max} be the maximal α achieving $\sup_\alpha\{\alpha \cdot H_F(\alpha)\}$.

Case 1. If $\alpha_{max} = D$, then $(D_1, D]$ is empty and $Pr_{x\sim F}\{x \in [D_1, D]\} = D \cdot H_F(D) \; > 0$.

Case 2. If $\alpha_{max} < D$, then choose any $D_1 \in (\alpha_{max}, D)$. Choose $\Delta = \frac{1}{2}(D_1 - \alpha_{max})$. Then for large n, the probability of $b_{k^*} \in [0, \alpha_{max} + \Delta]$ is $1 - e^{-\Omega(n)}$, satisfying **P1**. We also have $Pr_{x\sim F}\{x \in [D_1, D]\} \geq 1 - F(D_1) > 0$, thus satisfying **P2**. ◀

Divide $[1, D_1]$ into disjoint intervals of length ϵ, that is, write $[1, D_1] = \cup_{\ell=1}^{m} I_\ell$ where $I_\ell = [1 + (\ell - 1)\epsilon, \, 1 + \ell\epsilon)$ for $1 \leq \ell < m$, and $I_m = [1 + (m - 1)\epsilon, \, 1 + m\epsilon]$. Take a random b, which is the sorted list of iid $v_1, \cdots, v_n \sim F$. Let A_ℓ^{max} denote the random variable $\max\{i \cdot b_i | \, b_i \in I_\ell\}$. Let $A_{>\ell}^{max}$ be the random variable $\max\{i \cdot b_i \,|\, b_i \in I_{\ell+1} \cup \cdots \cup I_m\}$. Let W_ℓ denote the event that

$$A_{>(\ell+1)}^{max} - \frac{D^2}{\epsilon} \leq A_\ell^{max} \leq A_{>(\ell+1)}^{max}.$$

Now note that, for $\delta_{max}(b) > 2\epsilon$, the OSB condition for (B_j, b_{k^*}) in Lemma 1A can hold only if either 1) $b_j \in I_\ell$ and $b_{k^*} \in I_{\ell+2} \cup \cdots \cup I_m$ for some ℓ, or 2) $b_{k^*} \in (D_1, D]$.

▶ **Lemma 2.** *$Pr\{\delta_{max}(b) > 2\epsilon\} \leq \sum_{\ell=1}^{m-2} Pr\{W_\ell\} + e^{-\Omega(n)}$.*

Proof. Immediate from property **P1** and Lemma 1A. ◀

The rest of this section is devoted to the proof of the following lemma, which indicates that A_ℓ^{max} and $A_{>(\ell+1)}^{max}$ are not correlated to be nearly identical.

▶ **Lemma 3** (Weak Correlation Lemma). *For each $1 \leq \ell \leq m - 2$, $Pr\{W_\ell\} = O(\frac{(\log n)^2}{\sqrt{n}})$.*

There are two cases to consider.

Case 1. $Pr_{x\sim F}\{x \in I_{\ell+1}\} > 0$;

Case 2. $Pr_{x\sim F}\{x \in I_{\ell+1}\} = 0$.

We give the proof of Case 1. The proof for Case 2 uses the same general idea, and will only be sketched with details omitted here. Assume we have Case 1. Let G be the distribution (normalized) when F is restricted to the interval $L \equiv I_{\ell+1} \cup I_{\ell+2} \cup \cdots \cup I_m \cup [D_1, D]$. Let

$$\rho_0 = Pr_{x\sim G}\{x \in I_{\ell+1}\} > 0, \text{ and}$$
$$\rho_1 = Pr_{x\sim G}\{x \in [D_1, D]\} > 0.$$

Consider the generation of a random b in the following alternative (but equivalent) way.

Phase 1. Generate a random integer N so that

$$Pr\{N = s\} = \binom{n}{s} q^s (1-q)^{n-s}$$

for $0 \le s \le n$, where $q = Pr_{x \sim F}\{x \in L\}$.

Phase 2. Generate $n - N$ iid random numbers $v_1, \cdots, v_{N-n} \sim F|_{I_1 \cup \cdots \cup I_\ell}$, and sort them into decreasing order $b_{N+1}, b_{N+2}, \cdots, b_n$. (Note that A_ℓ^{max} is already determined after the completion of Phase 2.)

Phase 3. Generate $v_1, v_2, \cdots, v_N \sim G$ one number at a time. After step t, we sort the numbers $v_1, v_2, \cdots, v_t$ into decreasing order $b_1^{(t)} \ge b_2^{(t)} \ge \cdots \ge b_t^{(t)}$. Let

$$A^{(t)} = \max\{i \cdot b_i^{(t)} \mid b_i^{(t)} \in I_{\ell+2} \cup I_{\ell+3} \cup \cdots \cup I_m\}.$$

At time $t = N$, $A^{(N)}$ is exactly the same random variable as $A_{>(\ell+1)}^{max}$.

We prove two facts below, from which Lemma 3 follows immediately.

▶ **Fact 2.** *In Phase 1, $N \ge \frac{1}{2} q n$ with probability $1 - e^{-\Omega(n)}$.*

Proof. Chernoff's bound. ◀

After Phase 1 and 2, we have N and $K = A_\ell^{max}$ decided. To prove Lemma 3 we only need to show that, in Phase 3, there is enough randomness so that $A^{(N)}$ is unlikely to have a value within an additive constant $\frac{D^2}{\epsilon}$ to K.

Let us examine the evolution of $A^{(t)}$ as a random process of infinite length. The random sequence $A^{(t)}$ satisfies $A^{(0)} = 0$, and

$$A^{(t)} \begin{cases} = A^{(t-1)} & \text{with probability } \rho_0, \\ \ge A^{(t-1)} + 1 & \text{with probability } \rho_1, \\ \ge A^{(t-1)} & \text{otherwise,} \end{cases}$$

for $t \ge 1$.

▶ **Fact 3.** *At $t = N$, we have*

$$Pr\{|A^{(N)} - K| < \frac{D^2}{\epsilon}\} = O(\frac{(\log N)^2}{\sqrt{N}}) = O(\frac{(\log n)^2}{\sqrt{n}}).$$

Proof Sketch. Let s be the total number of times in the above process when either the second or third selection is made by $A^{(t)}$, for $1 \le t \le N$. Let $A^{(t_1)}, A^{(t_2)}, \cdots, A^{(t_s)}$ be the projected sequence. Note $E(s) = (1 - \rho_0)N$, $Var(s) = \Theta(N)$, and in fact $Pr\{s = u\} = O(\frac{1}{\sqrt{N}})$ for any u. Relabel $A^{(t_i)}$ as $B^{(i)}$, and consider the random sequence $B^{(1)}, B^{(2)}, \cdots$. Let $B^{(i)}, B^{(i+1)}, \cdots, B^{(i')}$ be the portion of the sequence in the range $[K - \frac{D^2}{\epsilon}, K + \frac{D^2}{\epsilon}]$. It is easy to verify that, for the random sequence $B^{(1)}, B^{(2)}, \cdots$,

$$Pr\{i' - i + 1 \ge \frac{D^2}{\epsilon} (\log N)^2\} = O(N^{-\log N}). \tag{7}$$

To see this, note that there is at least a constant $\phi = \frac{\rho_1}{1-\rho_0}$ probability to increase the next $B^{(j)}$ value by 1 (or more). Thus, to increase the value by $\frac{D^2}{\epsilon}$, it takes only $\frac{D^2}{\epsilon}\frac{1}{\phi}$ steps on average, rarely requiring a $(\log N)^2$ factor more steps.

As $Pr\{s = u\} = O(\frac{1}{\sqrt{N}})$ for any u, we conclude that $Pr\{B^{(s)} \in [K - \frac{D^2}{\epsilon}, K + \frac{D^2}{\epsilon}]\} \le O(\frac{(\log n)^2}{\sqrt{n}})$. This completes the proof of Case 1.

In Case 2, $Pr_{x\sim F}\{x \in I_{\ell+1}\} = 0$. We have lost the source of randomness critical for the above argument since $\rho_0 = 0$. Yet the source of randomness can be obtained by splitting I_ℓ into $I'_\ell \cup I''_\ell$ suitably, so that I''_ℓ does not contain any b_j with $j \cdot b_j$ anywhere close to the level of $A^{max}_{>(\ell+1)}$. (This will rely critically on the fact $Pr_{x\sim F}\{x \in I_{\ell+1}\} = 0$.) We omit here the details of implementing this plan, as well as the necessary handling of discontinuities in distribution F. This finishes the proof of Lemma 3 and thus Theorem 1. ◀

6 Proof of Theorem 2

Proof. For any $1 > \epsilon > 0$, we show that

$$\Delta_n^{average}(F) \le \epsilon \tag{8}$$

for all sufficiently large n. First pick $D > 0$ such that $F(D) > 1 - \epsilon/3$. Take n iid $v_i \sim F$ and let $q_{n,m}$ be the probability of exactly m of the $v_i' s$ falling into $[0, D]$. Let $\epsilon' = \epsilon/2$. By the law of large numbers, there exists N_1 such that for all $n \ge N_1$,

$$\sum_{m \le (1-\epsilon')n} q_{n,m} < \epsilon/6. \tag{9}$$

Consider the distribution G, obtained from restricting F to $[0, D]$. By Theorem 1, there exists $N_2 > 0$ such that

$$\Delta_m^{average}(G) < \epsilon/3 \tag{10}$$

for all $m > N_2$. We are now set for proving Theorem 2. By definition of $\Delta_n^{average}$, we have

$$\Delta_n^{average}(F) \le \sum_{m>(1-\epsilon')n} q_{n,m} \left[\frac{m}{n}\Delta_m^{average}(G) + \frac{n-m}{n}\right] + \sum_{m \le (1-\epsilon')n} q_{n,m}.$$

Using Eq. (9)-(10), we obtain for all $n > \max\{N_1, N_2\}$,

$$\Delta_n^{average}(F) \le (\frac{\epsilon}{3} + \epsilon') + \frac{\epsilon}{6} = \epsilon.$$

This proves Eq. (8) and Theorem 2. ◀

7 Proof of Theorem 3

Consider n iid random variables $v_1, \cdots, v_n$ distributed according to the Inverse distribution Inv, and let $b_1 \ge \cdots \ge b_n$ be their sorted sequence. Let $\lambda = 40$, $\lambda' = 1$, and let T_n be the event $(b_1 > \lambda n) \wedge (b_2 < \lambda' n)$. Let V_n be the event $(\max_{2 \le i \le n} i \cdot b_i \le \lambda n)$. Theorem 3 is an immediate consequence of the following two lemmas.

▶ **Lemma 4.** *If v satisfies event $T_n \wedge V_n$, then $\delta_{max}(v) \ge 1 - \frac{\lambda'}{\lambda}$.*

▶ **Lemma 5.**

$$Pr\{T_n \wedge V_n\} \ge \frac{1}{\lambda} e^{-\frac{2}{\lambda'}} - 4\,e^{-\frac{\lambda}{2}}.$$

Lemma 4 and 55 imply

$$\Delta_n^{max}(Inv) \ge c, \quad \textit{where}$$

$$c = (1 - \frac{\lambda'}{\lambda}) \cdot \left(\frac{1}{\lambda} e^{-\frac{2}{\lambda'}} - 4\,e^{-\frac{\lambda}{2}}\right) > 0.$$

This proves Theorem 3.

To prove Lemma 4, note that when $T_n \wedge V_n$ occurs, the highest bidder has a monopolistic price b_1 and a strategic price less than λn. Thus for the highest bidder i, we have $\delta_i(v) \geq 1 - \frac{\lambda n}{b_1} \geq 1 - \frac{\lambda'}{\lambda}$. This proves Lemma 4.

Lemma 5 follows from the following facts:

▶ **Fact 4.** $Pr\{T_n\} \geq \frac{1}{\lambda} e^{-\frac{2}{\lambda'}}$.

Proof.

$$\begin{aligned} Pr\{T_n\} &= \binom{n}{1} \frac{1}{\lambda n} \left(1 - \frac{1}{\lambda' n}\right)^{n-1} \\ &\geq \frac{1}{\lambda} e^{-\frac{2}{\lambda'}}. \end{aligned}$$ ◀

▶ **Fact 5.** $Pr\{V_n | T_n\} \leq 2\, e^{-\frac{\lambda}{2}}$.

Proof. Let $y_1, \cdots, y_n$ be iid distributed according to G, where for $t \in [1, n]$,

$$Pr_{z \sim G}\{z > t\} = \frac{1}{1 - \frac{1}{n}} \left(\frac{1}{t} - \frac{1}{n}\right).$$

Define $Y_n^{max} = \max_{1 \leq i \leq m}\{i \cdot b_i\}$, where $b_1 \geq \cdots \geq b_n$ is the sorted sequence of $y_1, \cdots, y_n$. Clearly,

$$Pr\{V_n | T_n\} \leq Pr\{Y_n^{max} \geq \lambda n\}.$$

To prove Fact 5, it suffices to prove:

$$Pr\{Y_n^{max} \geq \lambda n\} \leq 2\, e^{-\frac{\lambda}{2}}. \tag{11}$$

For any $t \geq 1$, let M_t be the number of y_i's satisfying $y_i \geq t$, and B_t be the event that $t \cdot M_t \geq \frac{\lambda}{2} n$. Let $t_k = \frac{1}{2^k} n$ for $1 \leq k \leq \lfloor \log_2 n \rfloor$. As the event $Y_n^{max} \geq \lambda n$ implies $\vee_{1 \leq k \leq \lfloor \log_2 n \rfloor} B_{t_k}$, we have

$$Pr\{Y_n^{max} \geq \lambda n\} \leq \sum_{i=1}^{\lfloor \log_2 n \rfloor} Pr\{B_{t_k}\}. \tag{12}$$

Observe that $E(M_t) = (1 - \frac{1}{n}) \frac{n}{t}$. Using Chernoff's bound, we have

$$Pr\{B_t\} \leq e^{-\frac{\lambda}{2} \frac{n}{t}}. \tag{13}$$

Equation (11) follows from (12) and (13) immediately. This completes the proof of Fact 5 and Theorem 3. ◀

8 Proof of Theorem 5

Without loss of generality, we assume that A, B are non-empty and that $p_A^{mono} \leq p_B^{mono}$. Let A consist of $y_1 \geq \cdots \geq y_m$ and B consist of $z_1 \geq \cdots \geq z_\ell$, with $y_s = p_A^{mono}$ and $z_t = p_B^{mono}$. Let $A' = \{y_1, y_2, \cdots, y_{s'}\}$ and $B' = \{z_1, z_2, \cdots, z_{t'}\}$ be the winners from A and B respectively. By definition of RSOP,

$$\begin{aligned} y_s &\leq z_{t'}, \\ t' z_{t'} &\leq t\, z_t. \end{aligned}$$

It follows that

$$\begin{aligned} RSOP(v) &= t' y_s + s' z_t \\ &\leq t' z_{t'} + s' z_t \\ &\leq t\, z_t + s' z_t. \end{aligned}$$

Now by definition $R(v) \geq (t + s')z_t$. Theorem 5 follows.

References

1 Moshe Babaioff, Shahar Dobzinski, Sigal Oren, and Aviv Zohar. On bitcoin and red balloons. In *Proceedings of the 13th ACM Conference on Electronic Commerce*, pages 56–73, New York, NY, USA, 2012. Association for Computing Machinery.

2 Joseph Bonneau, Edward W Felten, Steven Goldfeder, Joshua A Kroll, and Arvind Narayanan. Why buy when you can rent? bribery attacks on bitcoin consensus, 2016.

3 Miles Carlsten, Harry Kalodner, S Matthew Weinberg, and Arvind Narayanan. On the instability of bitcoin without the block reward. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 154–167, 2016.

4 Andrew V Goldberg and Jason D Hartline. Competitive auctions for multiple digital goods. In *European Symposium on Algorithms*, pages 416–427. Springer, 2001.

5 Andrew V Goldberg, Jason D Hartline, Anna R Karlin, Michael Saks, and Andrew Wright. Competitive auctions. *Games and Economic Behavior*, 55(2):242–269, 2006.

6 G Huberman, J Leshno, and CC Moallemi. Monopoly without a monopolist: An economic analysis of the bitcoin payment system. ssrn scholarly paper id 3025604. *Social Science Research Network, Rochester, NY*, 3054887, 2017.

7 Joshua A Kroll, Ian C Davey, and Edward W Felten. The economics of bitcoin mining, or bitcoin in the presence of adversaries. In *Proceedings of WEIS*, volume 2013, page 11, 2013.

8 Ron Lavi, Or Sattath, and Aviv Zohar. Redesigning bitcoin's fee market. In *The World Wide Web Conference*, pages 2950–2956, 2019.

Sketching Graphs and Combinatorial Optimization

Robert Krauthgamer
Weizmann Institute of Science, Rehovot, Israel
robert.krauthgamer@weizmann.ac.il

Abstract

Graph-sketching algorithms summarize an input graph G in a manner that suffices to later answer (perhaps approximately) one or more optimization problems on G, like distances, cuts, and matchings. Two famous examples are the Gomory-Hu tree, which represents all the minimum *st*-cuts in a graph G using a tree on the same vertex set $V(G)$; and the cut-sparsifier of Benczúr and Karger, which is a sparse graph (often a reweighted subgraph) that approximates every cut in G within factor $1 \pm \varepsilon$. Another genre of these problems limits the queries to designated terminal vertices, denoted $T \subseteq V(G)$, and the sketch size depends on $|T|$ instead of $|V(G)|$.

The talk will survey this topic, particularly cut and flow problems such as the three examples above. Currently, most known sketches are based on a graph representation, often called edge and vertex sparsification, which leaves room for potential improvements like smaller storage by using another representation, and faster running time to answer a query. These algorithms employ a host of techniques, ranging from combinatorial methods, like graph partitioning and edge or vertex sampling, to standard tools in data-stream algorithms and in sparse recovery. There are also several lower bounds known, either combinatorial (for the graph representation) or based on communication complexity and information theory.

Many of the recent efforts focus on characterizing the tradeoff between accuracy and sketch size, yet many intriguing and very accessible problems are still open, and I will describe them in the talk.

2012 ACM Subject Classification Theory of computation → Sketching and sampling; Mathematics of computing → Combinatorial optimization; Theory of computation → Graph algorithms analysis

Keywords and phrases Sketching, edge sparsification, vertex sparsification, Gomory-Hu tree, mimicking networks, graph sampling, succinct data structures

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.2

Category Invited Talk

Funding Work partially supported by ONR Award N00014-18-1-2364, by the Israel Science Foundation grant #1086/18, and by the Minerva Foundation with funding from the Federal German Ministry for Education and Research.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 2; pp. 2:1–2:1
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

How to Play in Infinite MDPs

Stefan Kiefer
Department of Computer Science, University of Oxford, United Kingdom

Richard Mayr
School of Informatics, University of Edinburgh, United Kingdom

Mahsa Shirmohammadi
CNRS & IRIF, Université de Paris, France

Patrick Totzke
Department of Computer Science, University of Liverpool, United Kingdom

Dominik Wojtczak
Department of Computer Science, University of Liverpool, United Kingdom

Abstract

Markov decision processes (MDPs) are a standard model for dynamic systems that exhibit both stochastic and nondeterministic behavior. For MDPs with finite state space it is known that for a wide range of objectives there exist optimal strategies that are memoryless and deterministic. In contrast, if the state space is infinite, optimal strategies may not exist, and optimal or ε-optimal strategies may require (possibly infinite) memory. In this paper we consider qualitative objectives: reachability, safety, (co-)Büchi, and other parity objectives. We aim at giving an introduction to a collection of techniques that allow for the construction of strategies with little or no memory in countably infinite MDPs.

2012 ACM Subject Classification Theory of computation → Random walks and Markov chains; Mathematics of computing → Probability and statistics

Keywords and phrases Markov decision processes

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.3

Category Invited Talk

Funding *Stefan Kiefer*: Supported by a Royal Society Research Fellowship.

1 Introduction

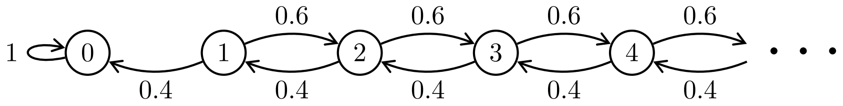
Markov decision processes (MDPs) are a standard model for dynamic systems that exhibit both stochastic and controlled behavior [13]. MDPs play a prominent role in numerous domains, including artificial intelligence and machine learning [16, 15], control theory [3, 1], operations research and finance [4, 14], and formal verification [8, 2]. In an MDP, the system starts in the initial state and makes a sequence of transitions between states. Depending on the type of the current state, either the controller gets to choose an enabled transition (or a distribution over transitions), or the next transition is chosen randomly according to a defined distribution. By fixing a strategy for the controller, one obtains a probability space of runs of the MDP. The goal of the controller is to maximize the probability of a given objective (some set of desired runs), or, more generally, to optimize the expected value of a random variable (some real-valued function on runs).

The type of strategy needed to satisfy an objective optimally (or ε-optimally) is called the *strategy complexity* of the objective. There are different types of strategies, depending on whether one can take the whole history of the run into account (history-dependent; (H)), or whether one is limited to a finite amount of memory (finite memory; (F)) or whether decisions are based only on the current state (memoryless; (M)). Moreover, the strategy

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 3; pp. 3:1–3:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

type depends on whether the controller can randomize (R) or is limited to deterministic choices (D). The simplest type MD refers to memoryless deterministic strategies. *Markov strategies* are strategies that base their decisions only on the current state and the number of steps in the history or the run. Thus they use infinite memory, but only in a very restricted form by maintaining an unbounded step counter.

For finite MDPs, there exist optimal MD-strategies for many (but not all) objectives [5, 6, 7, 13], but the picture is more complex for countably infinite MDPs [11, 12, 13]. For example, given some objective, consider the set of all states for which there exists a strategy that achieves the objective with positive probability. If the MDP is finite then this set is finite and thus there exists some minimal nonzero value, which can often be exploited for the construction of an optimal strategy. These methods do not carry over to infinite MDPs. Here it is possible, even for reachability objectives, that every state has a strategy that achieves the objective with positive probability, but no state, except the target itself, can achieve it almost surely. Such phenomena appear already in infinite-state Markov chains like the classic gambler's ruin problem with unfair coin tosses in the player's favor (0.6 win, 0.4 lose):



The probability of ruin is always positive, but less than 1 in every state except the ruin state itself; cf. [9, Chapter 14]. Another difference to finite MDPs is that optimal strategies need not exist, even for qualitative objectives like reachability or parity. Even if there is a sequence of strategies whose success probabilities converge to 1, there may not exist a strategy with success probability equal to 1. This motivates the investigation of ε-*optimal* strategies, which are those strategies such that no other strategy has a success probability that is more than ε higher.

In this paper we restrict ourselves to MDPs with *countable* state space. Certain theorems such as Theorem 3 are known to be false for MDPs with uncountably many states, see [12]. Uncountable MDPs in general have been studied less, and the underlying measure theory is more complicated.

We aim at providing an introduction to a toolkit for the construction of memoryless or "low-memory" optimal and ε-optimal strategies for certain qualitative objectives like reachability and safety. We will illustrate that these techniques can be combined to construct strategies for more general objectives.

2 Preliminaries

A *probability distribution* over a countable set S is a function $f : S \to [0,1]$ with $\sum_{s \in S} f(s) = 1$. We write $\mathcal{D}(S)$ for the set of all probability distributions over S.

Markov decision processes. In this paper we study Markov decision processes (MDPs) over *countably infinite* state spaces. Formally, an MDP $\mathcal{M} = (S, S_\Box, S_\bigcirc, {\longrightarrow}, P)$ consists of a countable set S of *states*, which is partitioned into a set $S_\Box$ of *controlled states* and a set $S_\bigcirc$ of *random states*, a *transition relation* ${\longrightarrow} \subseteq S \times S$, and a *probability function* $P : S_\bigcirc \to \mathcal{D}(S)$. If $(s, s') \in {\longrightarrow}$, we call s' a *successor* of s. We assume that every state has at least one successor. The probability function P assigns to each random state $s \in S_\bigcirc$ a probability distribution $P(s)$ over its set of successor states. A *sink* is a subset $T \subseteq S$ closed under the $\longrightarrow$ relation. An MDP is *acyclic* if the underlying directed graph $(S, \longrightarrow)$ is acyclic. It is *finitely branching* if every state has finitely many successors and *infinitely branching* otherwise. An MDP without controlled states ($S_\Box = \emptyset$) is a *Markov chain.*

Strategies and Probability Measures. A *run* ρ is an infinite sequence $s_0 s_1 \cdots$ of states such that $(s_i, s_{i+1}) \in \longrightarrow$ for all $i \in \mathbb{N}$. A *partial run* is a finite prefix of a run. A *strategy* is a function $\sigma : S^* S_\Box \to \mathcal{D}(S)$ that assigns to partial runs $\rho s \in S^* S_\Box$ a distribution over the successors of s.

A strategy σ and an initial state $s_0 \in S$ induce a standard probability measure on sets of infinite runs, see, e.g., [10]. Such measurable sets of infinite runs are called *events* or *objectives*. We write $\mathbb{P}^{\sigma}_{\mathcal{M},s_0}(E)$ for the probability of an event $E \subseteq s_0 S^\omega$ of runs starting from s_0. We may drop the subscript $\mathcal{M}$ when it is understood.

Objectives. Given a set $T \subseteq S$ of states, the *reachability* objective $\mathtt{Reach}(T)$ is the set of runs that visit T at least once; and the *safety objective* $\mathtt{Safety}(T)$ is the set of runs that never visit T. *Parity* objectives are defined via a *color function* $\mathit{Col} : S \to \mathbb{N}$ with finite range. The corresponding parity objective is the set of runs such that the largest color that occurs infinitely often along the run is even. We call a parity objective a $\mathcal{C}$-parity objective if $\mathit{Col}(S) \subseteq \mathcal{C}$, i.e., the set of colors is restricted to $\mathcal{C}$. *Büchi* and *co-Büchi* objectives are common names for $\{1,2\}$- and $\{0,1\}$-parity objectives, respectively.

Optimal and ε-optimal Strategies. Given an objective E, the *value* of state s in an MDP $\mathcal{M}$, denoted by $\mathtt{val}_{\mathcal{M},s}(E)$, is the supremum probability of achieving E, i.e., $\mathtt{val}_{\mathcal{M},s}(E) \overset{\text{def}}{=} \sup_{\sigma \in \Sigma} \mathbb{P}^{\sigma}_{\mathcal{M},s}(E)$ where Σ is the set of all strategies. We may drop the subscript $\mathcal{M}$ when it is understood. For $\varepsilon > 0$ and a state $s \in S$, we say that a strategy is *ε-optimal* if $\mathbb{P}^{\sigma}_{\mathcal{M},s}(E) \geq \mathtt{val}_{\mathcal{M},s}(E) - \varepsilon$. A 0-optimal strategy is called *optimal*. An optimal strategy is *almost surely winning* if $\mathtt{val}_{\mathcal{M},s}(E) = 1$.

Strategy Classes. Strategies $\sigma : S^* S_\Box \to \mathcal{D}(S)$ are in general *randomized* (R) in the sense that they take values in $\mathcal{D}(S)$. A strategy σ is *deterministic* (D) if $\sigma(\rho)$ is a Dirac distribution for all partial runs $\rho \in S^* S_\Box$. A strategy is called *memoryless* (M) or *positional* if it only depends on the current state; i.e., a memoryless strategy can be given by a function $\sigma : S_\Box \to \mathcal{D}(S)$. Thus, the simplest strategies are MD strategies, which are both memoryless and deterministic (and thus can be given by a function $\sigma : S_\Box \to S$).

We also consider strategies with memory, but we do not formalize this here. After each transition such a strategy updates its memory mode depending on the taken transition and the previous memory mode. To choose a successor of a controlled state, the strategy can use its memory and the current state but not the partial run that led to the current state. Every strategy can be viewed as a strategy with memory (by using partial runs as memory modes).

For example, k-bit strategies use (at most) k bits of memory; they have (at most) 2^k memory modes. *Markov* strategies use infinite memory but only as a step counter; such strategies depend only on the current state and the number of steps taken so far. A *k-bit Markov* strategy can use both k bits and an (unbounded) step counter.

3 Constructing MD Strategies

In this section we illustrate some techniques to construct MD strategies. We mostly focus on reachability objectives, which we use as a running example. But many ideas apply also to other objectives.

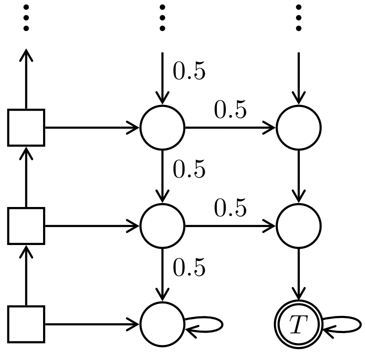


Figure 1 Optimal strategy for reachability may not exist. Here, as in all pictures, we depict controlled states as squares, and random states as circles.

3.1 Reduction to a Finite Case

Suppose we have a (countable) MDP $\mathcal{M} = (S, S_\Box, S_\bigcirc, \longrightarrow, P)$, and we are interested in the reachability objective $\texttt{Reach}(T)$, i.e., we would like to reach a target set $T \subseteq S$.

If S is finite, the situation is as good as it could be:

▶ **Lemma 1** (optimal MD strategies in finite MDPs). *Let $\mathcal{M} = (S, S_\Box, S_\bigcirc, \longrightarrow, P)$ be an MDP, and consider the reachability objective $\texttt{Reach}(T)$ for some $T \subseteq S$. If S is finite then there exists a single MD strategy σ that is optimal for every start state s at the same time; formally, $\mathbb{P}_s^\sigma(\texttt{Reach}(T)) = \texttt{val}_s(\texttt{Reach}(T))$ for all $s \in S$.*

Spelled out, Lemma 1 says that, if the state space is finite, we can fix for each controlled state an outgoing transition in a way that maximizes the probability of reaching the target for every state. The assumption of a finite state space is so powerful that Lemma 1 generalizes from reachability to parity objectives. In fact, even in finite parity *games*, where some controlled states are controlled by a player who wants to maximize the probability of achieving the parity objective, and the remaining controlled states are controlled by a player who wants to *minimize* the probability of achieving the parity objective, both players have optimal MD strategies for all states [17, Theorem 1].

Lemma 1 does not hold without the assumption of S being finite, as optimal strategies may not exist, let alone optimal MD strategies. Indeed, consider the MDP in Figure 1. The controlled states are those in the leftmost column. Suppose you start in the bottom-left state and you would like to reach the target T consisting only of the bottom-right state. The higher you climb the ladder of states in the left column, the bigger you can make the probability to reach T, but eventually you have to turn right and hope for the best. We have $\texttt{val}_s(\texttt{Reach}(T)) = 1$ for all controlled states s, i.e., we can get the probability of reaching T arbitrarily close to 1. But we cannot make that probability equal to 1: there is no strategy that reaches T with probability 1.

Recall that a strategy σ is called ε-optimal for s if $\mathbb{P}_s^\sigma(\texttt{Reach}(T)) \geq \texttt{val}_s(\texttt{Reach}(T)) - \varepsilon$. The following lemma says that every state has ε-optimal *MD* strategies:

▶ **Lemma 2** (non-uniform ε-optimal MD strategies). *Let $\mathcal{M} = (S, S_\Box, S_\bigcirc, \longrightarrow, P)$ be an MDP, and consider the reachability objective $\texttt{Reach}(T)$ for some $T \subseteq S$. For every $\varepsilon > 0$ and every $s \in S$ there exists an MD strategy σ that is ε-optimal for s; formally, $\mathbb{P}_s^\sigma(\texttt{Reach}(T)) \geq \texttt{val}_s(\texttt{Reach}(T)) - \varepsilon$.*

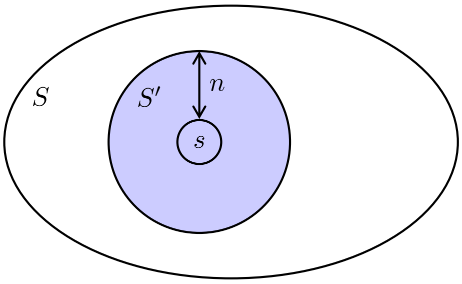


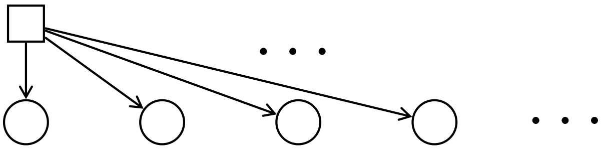
Figure 2 The set S' consists of those states that are reachable from s within at most n steps. Due to finite branching, S' is finite. For n large enough, S' probably suffices to reach the target.

Proof. First assume that $\mathcal{M}$ is finitely branching. Fix $s \in S$ and $\varepsilon > 0$, and let τ be an arbitrary (i.e., not necessarily MD) strategy with $\mathbb{P}^{\tau}_{s}(\mathtt{Reach}(T)) \geq \mathtt{val}_s(\mathtt{Reach}(T)) - \frac{\varepsilon}{2}$. Define a modified reachability objective $\mathtt{Reach}_i(T)$ which means reaching T in exactly i steps. Then we have:

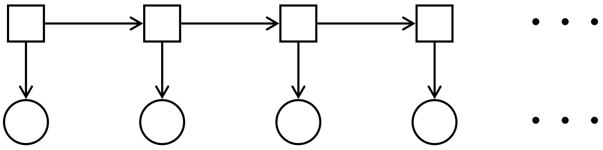
$$\sum_{i=0}^{\infty} \mathbb{P}^{\tau}_{s}(\mathtt{Reach}_i(T)) \;=\; \mathbb{P}^{\tau}_{s}(\mathtt{Reach}(T)) \;\geq\; \mathtt{val}_s(\mathtt{Reach}(T)) - \frac{\varepsilon}{2}$$

It follows that we can pick a number n large enough so that $\sum_{i=0}^{n} \mathbb{P}^{\tau}_{s}(\mathtt{Reach}_i(T)) \geq \mathtt{val}_s(\mathtt{Reach}(T)) - \varepsilon$. From state s, in at most n steps, strategy τ can only use a finite subset $S' \subseteq S$, as $\mathcal{M}$ is finitely branching; see Figure 2. That means, τ manages to reach T with probability at least $\mathtt{val}_s(\mathtt{Reach}(T)) - \varepsilon$ even when it is restricted to the sub-MDP with state space S' (think of leaving S' as losing). But by Lemma 1 this finite sub-MDP has an optimal MD strategy σ. We may extend the definition of σ outside of S' in an arbitrary way. Then, in $\mathcal{M}$, we have $\mathbb{P}^{\sigma}_{s}(\mathtt{Reach}(T)) \geq \mathtt{val}_s(\mathtt{Reach}(T)) - \varepsilon$, as desired.

The assumption that $\mathcal{M}$ is finitely branching can be satisfied using a simple construction: replace every infinitely branching controlled state



by



A similar construction works for random states. Then, construct an MD strategy, as above, from an arbitrary ε-optimal strategy in the new finitely branching MDP. Any MD strategy can be transferred back to the original infinitely branching MDP. ◀

3.2 Ornstein's Plastering Technique

Ornstein [12] proved in 1969 a uniform version of Lemma 2. That is, for every $\varepsilon > 0$ there exists a single MD strategy that is ε-optimal for all states:

▶ **Theorem 3** ([12], uniform ε-optimal MD strategies)**.** *Let $\mathcal{M} = (S, S_\Box, S_\bigcirc, \longrightarrow, P)$ be an MDP, and consider the reachability objective $\mathtt{Reach}(T)$ for some $T \subseteq S$. For every $\varepsilon > 0$ there exists a single MD strategy σ that is ε-optimal for every start state s at the same time; formally, $\mathcal{P}^\sigma_s(\mathtt{Reach}(T)) \geq \mathtt{val}_s(\mathtt{Reach}(T)) - \varepsilon$ for all $s \in S$.*

Ornstein actually proved a stronger statement with multiplicative instead of additive error; i.e., Theorem 3 also holds with $\mathtt{val}_s(\mathtt{Reach}(T)) - \varepsilon$ replaced by $\mathtt{val}_s(\mathtt{Reach}(T)) \cdot (1 - \varepsilon)$.

Proof of Theorem 3. We follow Ornstein's proof [12]. Without loss of generality, we assume that T is a sink. Recall that an MD strategy σ can be viewed as a function $\sigma : S_\Box \to S$ such that for all $s \in S_\Box$, the state $\sigma(s)$ is a successor state of s. Starting from the original MDP $\mathcal{M}$ we successively *fix* more and more controlled states, by which we mean select an outgoing transition and remove all others. While this is in general an infinite (but countable) process, it defines an MD strategy in the limit. Visually, we "plaster" the whole state space by the fixings.

Put the states in some order, i.e., $s_1, s_2, \ldots$ with $S = \{s_1, s_2, \ldots\}$. The plastering proceeds in *rounds*, one round for every state. Let $\mathcal{M}_i$ be the MDP obtained from $\mathcal{M}$ after the fixings of the first $i - 1$ rounds (with $\mathcal{M}_1 = \mathcal{M}$). In round i we fix controlled states in such a way that

(A) the probability, starting from s_i, of reaching the target T using only random and *fixed* controlled states is not much less than the value $\mathtt{val}_{\mathcal{M}_i,s_i}(\mathtt{Reach}(T))$; and

(B) for all states s, the value $\mathtt{val}_{\mathcal{M}_{i+1},s}(\mathtt{Reach}(T))$ is almost as high as $\mathtt{val}_{\mathcal{M}_i,s}(\mathtt{Reach}(T))$.

The purpose of goal (A) is to guarantee good progress towards the target when starting from s_i. The purpose of goal (B) is to avoid fixings that would cause damage to the values of other states.

Now we describe round i. Consider the MDP $\mathcal{M}_i$ after the fixings from the first $i-1$ rounds, and let $\varepsilon_i > 0$. Recall that we wish to fix a part of the state space so that s_i has a high probability of reaching T using only random and fixed controlled states. By Lemma 2 there is an MD strategy σ such that $\mathcal{P}^\sigma_{\mathcal{M}_i,s_i}(\mathtt{Reach}(T)) \geq \mathtt{val}_{\mathcal{M}_i,s_i}(\mathtt{Reach}(T)) - \varepsilon_i^2$. Fixing σ everywhere would accomplish goal (A), but potentially compromise goal (B). So instead we are going to fix σ only for states where σ does well: define

$$G \overset{\text{def}}{=} \{s \in S \mid \mathcal{P}^\sigma_{\mathcal{M}_i,s}(\mathtt{Reach}(T)) \geq \mathtt{val}_{\mathcal{M}_i,s}(\mathtt{Reach}(T)) - \varepsilon_i\}$$

and obtain $\mathcal{M}_{i+1}$ from $\mathcal{M}_i$ by fixing σ on G. (Note that σ does not "contradict" earlier fixings, because in the MDP $\mathcal{M}_i$ the previously fixed states have only one outgoing transition left.) See Figure 3 for an illustration.

We have to check that with this fixing we accomplish the two goals above. Indeed, we accomplish goal (A): by its definition strategy σ is ε_i^2-optimal from s_i, so the probability of ever entering $S \setminus G$ (where σ is less than ε_i-optimal) cannot be large:

$$\mathcal{P}^\sigma_{\mathcal{M}_i,s_i}(\mathtt{Reach}(S \setminus G)) \;\leq\; \varepsilon_i \tag{1}$$

In slightly more detail, this inequality holds because the probability that the ε_i^2-optimal strategy σ enters a state whose value is underachieved by σ by at least ε_i can be at most ε_i. We give a detailed proof of (1) in Section 5.1. It follows from the ε_i^2-optimality of σ and from (1) that we have $\mathcal{P}^\sigma_{\mathcal{M}_i,s_i}(\mathtt{Reach}(T) \wedge \neg\mathtt{Reach}(S \setminus G)) \geq \mathtt{val}_{\mathcal{M}_i,s_i}(\mathtt{Reach}(T)) - \varepsilon_i - \varepsilon_i^2$. So in $\mathcal{M}_{i+1}$ we obtain for *all* strategies σ':

$$\mathcal{P}^{\sigma'}_{\mathcal{M}_{i+1},s_i}(\mathtt{Reach}(T)) \;\geq\; \mathtt{val}_{\mathcal{M}_i,s_i}(\mathtt{Reach}(T)) - \varepsilon_i - \varepsilon_i^2 \tag{2}$$

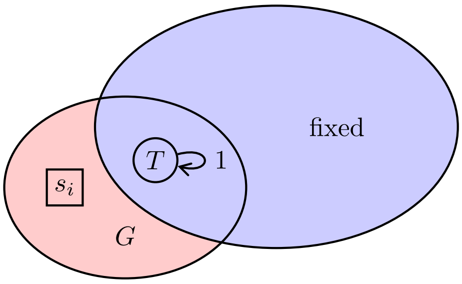


Figure 3 The blue area has been fixed in the first $i-1$ iterations. The smaller ellipse is the set G, where strategy σ does well. The red area will be fixed using σ. Under σ, a run that starts from s_i is likely to lead to the target T without ever leaving G.

We also accomplish goal (B): the difference between $\mathcal{M}_i$ and $\mathcal{M}_{i+1}$ is that σ is fixed on G, but σ performs well from G on. So we obtain for *all* states s:

$$\mathtt{val}_{\mathcal{M}_{i+1},s}(\mathtt{Reach}(T)) \;\geq\; \mathtt{val}_{\mathcal{M}_i,s}(\mathtt{Reach}(T)) - \varepsilon_i \tag{3}$$

In slightly more detail, this inequality holds because any strategy in $\mathcal{M}_i$ can be transformed into a strategy in $\mathcal{M}_{i+1}$, with the difference that once the newly fixed part G is entered, the strategy switches to the strategy σ, which (by the definition of $\mathcal{M}_{i+1}$) is consistent with the fixing and (by the definition of G) is ε_i-optimal from there. We give a detailed proof of (3) in Section 5.2. This completes the description of round i.

Let $\varepsilon \in (0,1)$, and for all $i \geq 1$, choose $\varepsilon_i \overset{\text{def}}{=} \frac{\varepsilon}{2} \cdot 2^{-i}$. Let σ be an arbitrary MD strategy that is compatible with all fixings. (This strategy σ is actually unique.) It follows that σ is playable in all $\mathcal{M}_i$. Consider an arbitrary state s_i. Then it follows from (3) that the value $\mathtt{val}_{\mathcal{M}_i,s_i}(\mathtt{Reach}(T))$ is not much lower than $\mathtt{val}_{\mathcal{M},s_i}(\mathtt{Reach}(T))$, and from (2) that σ realizes most of this value, implying for all $i \geq 1$:

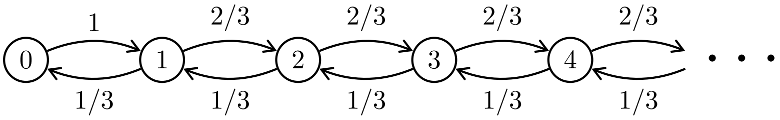
$$\mathbb{P}^{\sigma}_{\mathcal{M},s_i}(\mathtt{Reach}(T)) \;\geq\; \mathtt{val}_{\mathcal{M},s_i}(\mathtt{Reach}(T)) - \varepsilon \tag{4}$$

We give a detailed proof of (4) in Section 5.3. Thus, the MD strategy σ is ε-optimal for all states. ◀

▶ Remark 4. Instead of $\mathtt{Reach}(T)$ the proof above also works for many other objectives, including so-called *tail* events. See Section 3.3 for more discussion about tail events.

3.3 Lévy's Zero-One Law

Consider a Markov chain $\mathcal{M} = (S, \emptyset, S_{\bigcirc}, \longrightarrow, P)$, i.e., an MDP without controlled states. Recall that an event is a set of runs $s_0 s_1 \cdots$. For example, in the Markov chain



we may define an event E as the set of all runs that start in state 0 and revisit state 0 exactly once. Starting in state 1, the probability of ever visiting state 0 can be calculated to be $\frac{1}{2}$. It follows that, starting in state 0, the probability of E is $\frac{1}{2} \cdot (1 - \frac{1}{2}) = \frac{1}{4}$; formally, $\mathbb{P}_0(E) = \frac{1}{4}$. There are two ways of failing to satisfy E: one is to never revisit state 0, the other is to revisit

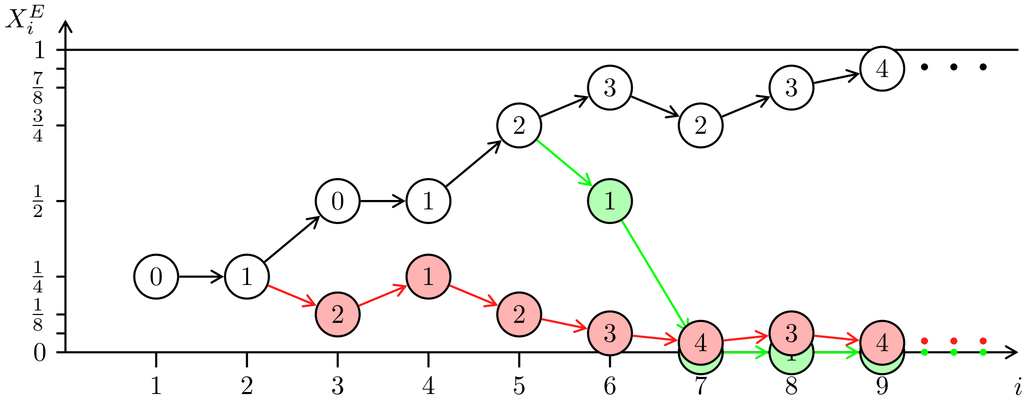


Figure 4 For the event E (exactly one revisit to state 0), three sample runs starting from state 0 are depicted, along with their values $X_1^E, \ldots, X_9^E$. The partially covered "green" run revisits state 0 for a second time, causing $0 = X_7^E = X_8^E = \ldots$ Lévy's zero-one law asserts that X_i^E almost surely converges to 0 or 1.

it more than once. It is because of the latter that for any given finite prefix of a run, we can never be sure that E will hold. For example, if the run starts with 010123, then we have already revisited state 0 and it is unlikely that we will ever do so again: the probability is only $\frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{8}$. So, given this prefix 010123, the probability of satisfying E is $\frac{7}{8}$. However, in this example, no matter what prefix is given, the probability of satisfying E cannot be 1.

Let us define a sequence of random variables, $X_1^E, X_2^E, \ldots$, so that each X_i^E maps a run ρ to the probability that event E will be satisfied, given the prefix of ρ of length i. For example, if $\rho = 010123234\cdots$, then we previously discussed that $X_1^E(\rho) = \frac{1}{4}$ and that $X_6^E(\rho) = \frac{7}{8}$. It is (a consequence of) Lévy's zero-one law that the sequence $X_1^E, X_2^E, \ldots$[1] converges to 0 or 1 almost surely (see also Figure 4):

▶ **Theorem 5** (Lévy's zero-one law for Markov chains)**.** *Let $\mathcal{M} = (S, \emptyset, S_\bigcirc, \longrightarrow, P)$ be a Markov chain, $s_0 \in S$, and E an event of runs starting in s_0. We have $\lim_{i\to\infty} X_i^E \in \{0,1\}$ (and hence this limit exists) almost surely; more formally:*

$$\mathbb{P}_{s_0}\left(\left\{\rho \;\middle|\; \lim_{i\to\infty} X_i^E(\rho) \in \{0,1\}\right\}\right) \;=\; 1\,.$$

Moreover, up to a null set of runs, the limit is 1 *for those runs satisfying E, and* 0 *for those runs not satisfying E.*

As a consequence, the probability of E is equal to the probability that the limit is 1.

For *tail* objectives E, Lévy's zero-one law becomes simpler and clearer. A tail objective is an objective whose occurrence is independent of any finite prefix. The objective E from the example above is not a tail objective because the number of revisits to state 0 may change if we cut off or add a finite prefix. For an example of a tail objective, suppose that the states of an arbitrary Markov chain are classified as accepting and non-accepting states. The *Büchi* objective consists of those runs that visit accepting states infinitely often. Büchi is a tail objective: for any run we can cut off or add any finite prefix without changing whether the run satisfies Büchi. We can also view reachability objectives as tail objectives, provided that the target is a sink, which is often a harmless assumption.

[1] It is conventional to write X_i^E for $X_i^E(\rho)$ if the run (such as a random run produced by $\mathcal{M}$) is understood.

For tail objectives E, the random variable X_i^E depends only on the ith visited state (and not also on the first $i-1$ states as in the general case), and for any run $s_1 s_2 s_3 \cdots$ we have $X_i^E = \mathbb{P}_{s_i}(E)$. For a tail objective E and any state s, a picture analogous to Figure 4 would show each occurrence of state s on the same height, independently of the partial run leading up to the occurrence. As a consequence of Lévy's zero-one law, for any tail objective E, the events

$$E \quad \text{and} \quad \left\{ s_1 s_2 \cdots \;\middle|\; \lim_{i\to\infty} \mathbb{P}_{s_i}(E) = 1 \right\} \quad \text{are equal up to a null set.} \tag{5}$$

This can be used, in conjunction with Ornstein's Theorem 3 about uniform ε-optimal MD strategies, to construct almost surely winning MD strategies:

▶ **Theorem 6** ([12], uniform almost surely winning MD strategies)**.** *Let $\mathcal{M} = (S, S_\Box, S_\bigcirc, \longrightarrow, P)$ be an MDP, and consider the reachability objective $\texttt{Reach}(T)$ for some $T \subseteq S$. Let $S_0 \subseteq S$ be the set of states from which there exists an almost surely winning strategy. Then there exists a single MD strategy σ that is almost surely winning for all $s \in S_0$ at the same time; formally, $\mathbb{P}_s^\sigma(\texttt{Reach}(T)) = 1$ for all $s \in S_0$.*

Proof. Obtain from $\mathcal{M}$ an MDP $\mathcal{M}_0$ by restricting the state space to S_0 and eliminating all transitions that leave S_0. In $\mathcal{M}_0$ all states have an almost surely winning strategy, as an almost surely winning strategy may never enter a state that does not have an almost surely winning strategy. By Theorem 3 there exists, for $\mathcal{M}_0$, a uniform $\frac{1}{2}$-optimal MD strategy σ. Then $\mathbb{P}^\sigma_{\mathcal{M}_0,s}(\texttt{Reach}(T)) \geq \frac{1}{2}$ holds for all states. For any run $s_0 s_1 \cdots$ in $\mathcal{M}_0$ we have $\mathbb{P}^\sigma_{\mathcal{M}_0,s_i}(\neg\texttt{Reach}(T)) \leq \frac{1}{2}$ for all i; in particular, the sequence $\left(\mathbb{P}^\sigma_{\mathcal{M}_0,s_i}(\neg\texttt{Reach}(T))\right)_i$ does not converge to 1. Using (5) for $E = \neg\texttt{Reach}(T)$, we obtain for all $s \in S_0$ that $\mathbb{P}_s^\sigma(E) = 0$, hence, $\mathbb{P}_s^\sigma(\texttt{Reach}(T)) = 1$. ◀

3.4 The Flag Construction

We now move from reachability to co-Büchi objectives: here, a subset of states are marked as "bad", and the goal is to visit bad states only finitely often. Co-Büchi is more general than both reachability and safety objectives: for reachability, make the target a sink and mark all other states as bad; for safety, make the states to be avoided a sink and mark them as bad.

In this section we focus on almost surely winning strategies, and we will argue that for co-Büchi objectives almost surely winning strategies can be chosen MD. However, this does not always hold for infinitely branching MDPs, as the example in Figure 5 shows. Therefore, we assume in the rest of the section that the MDP is finitely branching. We will show:

▶ **Theorem 7** ([11])**.** *Let $\mathcal{M} = (S, S_\Box, S_\bigcirc, \longrightarrow, P)$ be a finitely branching MDP, and consider a co-Büchi objective $\texttt{co-Büchi}(B)$ for some set $B \subseteq S$ of bad states. Let $S_0 \subseteq S$ be the set of states from which there exists an almost surely winning strategy. Then there exists a single MD strategy σ that is almost surely winning for all $s \in S_0$ at the same time; formally, $\mathbb{P}_s^\sigma(\texttt{co-Büchi}(B)) = 1$ for all $s \in S_0$.*

In order to prove Theorem 7, a *safety strategy* may appear promising: in each state minimize the probability of ever visiting a bad state again. The appeal of a safety strategy is twofold:

- If a safety strategy succeeds in never visiting a bad state again, then clearly it visits bad states only finitely often.
- A safety strategy can be chosen uniformly MD (in finitely branching MDPs): in every controlled state pick the successor with the best value.[2]

[2] Such an approach for constructing an optimal strategy does not work for reachability or more general objectives: intuitively, this approach cannot guarantee "progress" towards the goal.

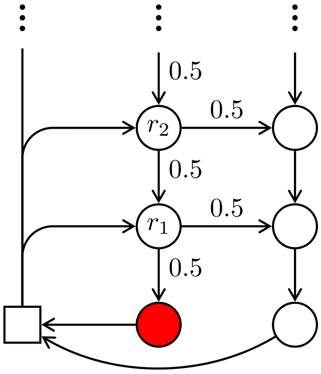


Figure 5 In infinitely branching MDPs with co-Büchi objectives, almost surely winning strategies cannot always be chosen MD. In the MDP above, the bad state is marked red. There exists an almost surely winning strategy, but it requires memory in order to choose r_i for ever higher i.

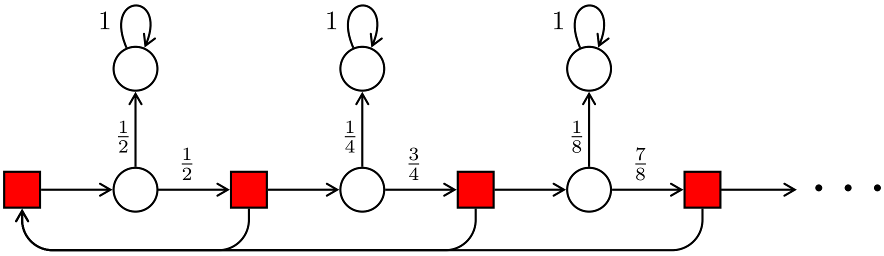


Figure 6 In this MDP (with bad states marked red), a safety strategy always chooses the right outgoing transition, chasing after the ever smaller chance of entering a safe sink state without re-entering any bad state. Starting from the leftmost state, the probability that this strategy achieves the co-Büchi objective is less than 0.72. The “opposite” strategy, which always returns to the leftmost state, succeeds almost surely.

But a safety strategy alone does not suffice for co-Büchi, as the example in Figure 6 shows. In the following proof we construct an almost surely winning MD strategy for co-Büchi by combining an MD strategy for safety with an MD strategy for reachability.

Proof sketch of Theorem 7. Similarly as in the proof of Theorem 6, we can assume without loss of generality that for all states there exists an almost surely winning strategy. We will show that there exists a single MD strategy that is almost surely winning for all states.

We have mentioned previously that there exists an MD uniformly optimal safety strategy σ_{safe}, i.e., for each state, σ_{safe} minimizes the probability of ever revisiting a bad state. For $x \in [0,1]$ define $\mathit{Safe}(x) \subseteq S$ as the set of states with *safety level* at least x. By safety level we mean the probability of never visiting another bad state, assuming σ_{safe} is played. See Figure 7 for an abstract visualization. We *fix* σ_{safe} in $\mathit{Safe}(\frac{1}{3})$, i.e., in the following we will only consider strategies that are compatible with σ_{safe} in $\mathit{Safe}(\frac{1}{3})$, see the right side of Figure 7.

After this fixing, every state still has an almost surely winning strategy. Indeed, consider any state and its almost surely winning strategy before the fixing. We modify the strategy as follows. First we play it as before, but if and when we reach $\mathit{Safe}(\frac{1}{3})$, we switch to σ_{safe}. Now the probability is at least $\frac{1}{3}$ that we never visit a bad state again and thus also achieve the co-Büchi objective. If we do visit a bad state again, we revert to a strategy that is almost surely winning from that bad state in the original MDP. We follow that strategy until we

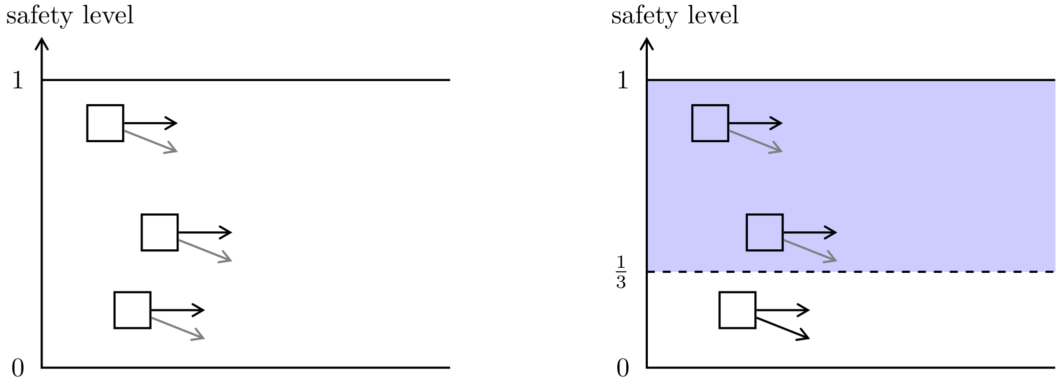


Figure 7 Left: In this diagram the controlled states are arranged according to their safety level. A safety strategy entails not to pick successor states with smaller safety level, so grey transitions are not used. Right: The optimal safety strategy σ_{safe} is fixed for the states with safety level at least $\frac{1}{3}$.

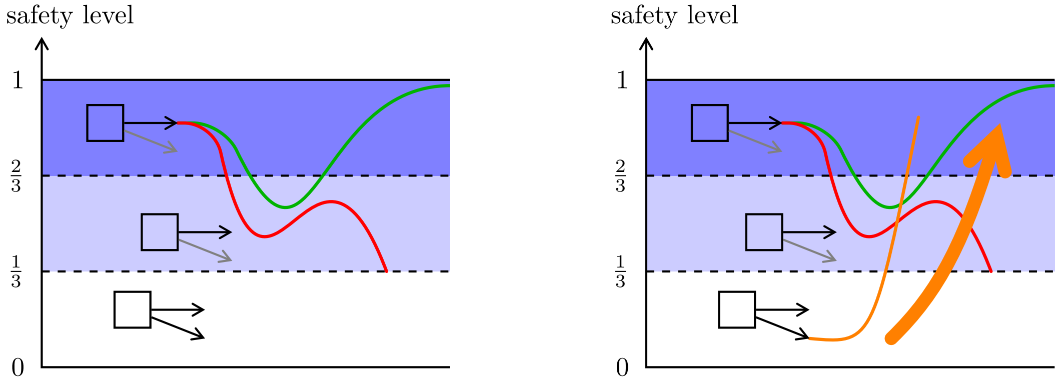


Figure 8 Left: Starting from a state with safety level at least $\frac{2}{3}$ the probability of keeping a safety level of at least $\frac{1}{3}$ forever is at least $\frac{1}{2}$. Right: For the states with safety level less than $\frac{1}{3}$ we aim at reaching safety level at least $\frac{2}{3}$.

possibly reach $Safe(\frac{1}{3})$ again. At this point we switch again to σ_{safe}, thus forever avoiding the bad states with a fresh chance of at least $\frac{1}{3}$. Continuing in this way, we win almost surely. Note that this strategy is not MD, as we have to remember in which phase we are: at any point we either follow a winning strategy of the original MDP, or follow σ_{safe}. We have merely argued here that having fixed σ_{safe} in $Safe(\frac{1}{3})$ has not done any harm.

Before we define an MD strategy for the rest of the state space, consider a state $s \in Safe(\frac{2}{3})$, i.e., s has an even higher safety level of at least $\frac{2}{3}$. Since $Safe(\frac{2}{3}) \subseteq Safe(\frac{1}{3})$, the MD strategy σ_{safe} has been fixed there. Two things might happen starting at s (see the left side of Figure 8): either the run remains in $Safe(\frac{1}{3})$ forever and never visits a bad state; or it eventually leaves $Safe(\frac{1}{3})$ (or even visits a bad state). The second case (leaving $Safe(\frac{1}{3})$ or visiting a bad state) cannot have a very large probability: after all, we start with safety level at least $\frac{2}{3}$, so if the safety level is very likely to drop below $\frac{1}{3}$, we were not very safe to start with. Doing the maths shows that the probability of the second case is at most $\frac{1}{2}$. So starting from $Safe(\frac{2}{3})$, the probability of avoiding bad states forever is at least $\frac{1}{2}$, no matter what strategy is played outside of $Safe(\frac{1}{3})$.

We still need to define an MD strategy for the part of the state space with safety level less than $\frac{1}{3}$. Our ambition is to define it so that from every state we reach almost surely $\mathit{Safe}(\frac{2}{3})$, see the right side of Figure 8. If we succeed in this goal, then we achieve the co-Büchi objective almost surely, because every time we reach $\mathit{Safe}(\frac{2}{3})$ we receive a fresh chance of $\frac{1}{2}$ to avoid bad states forever, as just argued.

First we argue that for each state there is *some* strategy to reach $\mathit{Safe}(\frac{2}{3})$ almost surely. Recall that we have shown above that after the fixing in $\mathit{Safe}(\frac{1}{3})$ every state s still has a strategy σ_s to achieve the co-Büchi objective almost surely. We argue that σ_s reaches $\mathit{Safe}(\frac{2}{3})$ almost surely. Indeed, whenever a run is outside of $\mathit{Safe}(\frac{2}{3})$ there is a risk of more than $\frac{1}{3}$ to visit a bad state. It follows that there is a risk of at least $\frac{1}{3}$ to visit a bad state within a finite time horizon. Since σ_s almost surely achieves the co-Büchi objective, it must avoid that this risk materializes infinitely often. Hence σ_s almost surely reaches $\mathit{Safe}(\frac{2}{3})$.

Using Theorem 6 it follows that, in the MDP after having fixed σ_{safe} in $\mathit{Safe}(\frac{1}{3})$, there is uniform almost surely winning MD strategy σ_{reach} for $\mathtt{Reach}(\mathit{Safe}(\frac{2}{3}))$. In summary, here is our almost surely winning MD strategy for co-Büchi: in $\mathit{Safe}(\frac{1}{3})$ play σ_{safe}, and elsewhere play σ_{reach}. The key point is that these two MD strategies are not conflicting. ◀

▶ Remark 8. Generalizations of Theorem 7 are also considered in [11]:

1. A similar proof shows a version of Theorem 7 for ε-optimal strategies.
2. Theorem 7 generalizes to $\{0, 1, 2\}$-parity objectives, which also encompass Büchi objectives. The theorem further generalizes from almost surely winning to *optimal* strategies as follows: The set $S_0 \subseteq S$ can be taken as the set of states from which there exists an optimal strategy (note that an almost surely winning strategy is optimal). Then there exists a single MD strategy that is optimal for all states in S_0.
3. Theorem 7 does not hold for $\{1, 2, 3\}$-parity objectives.

4 Markov Strategies and Generalizations

We describe a technique to prove the existence of ε-optimal Markov strategies (resp. Markov strategies with one extra bit of memory) for certain types of objectives, based on the work on Büchi objectives in [10].

Obtaining Markov strategies via acyclic MDPs. *Markov strategies* are strategies that base their decisions only on the current state and the number of steps in the history of the run from some initial state s_0. Thus they do use infinite memory, but only in a very restricted form by maintaining an unbounded step counter. Slightly more general are *1-bit Markov strategies* that use one extra bit of extra memory in addition to a step counter.

The existence of ε-optimal (1-bit) Markov strategies for some objective φ on countable MDPs can be proven by first studying the strategy complexity of φ on *acyclic* MDPs, i.e., MDPs where the underlying transition graph is a directed acyclic graph (DAG). Note that a DAG is more general than a tree. If the transition graph is a tree with root s_0 then there always exist ε-optimal positional strategies for any objective, since the entire history is implicit in the current state. This does not hold for a DAG, since the same state s could be reached via (possibly infinitely many) different paths from s_0.

However, for every countable MDP $\mathcal{M}$ with initial state s_0, there is a corresponding acyclic MDP $\mathcal{M}'$ that encodes the step counter into the states, i.e., the states of $\mathcal{M}'$ are of the form (s, i) where s is a state of $\mathcal{M}$ and $i \in \mathbb{N}$ counts the number of steps. Then for every ε-optimal positional strategy for φ in $\mathcal{M}'$ there is a corresponding ε-optimal Markov strategy for φ in $\mathcal{M}$, and vice-versa [10]. Thus, if ε-optimal positional strategies for φ exist in *all* acyclic MDPs then ε-optimal Markov strategies for φ exist in general countable MDPs. The reverse implication does not hold, however, since not all acyclic MDPs encode a step counter, e.g., if some state has infinite in-degree and can be reached from the initial state via paths of arbitrary length.

Acyclic and finitely branching MDPs have nice properties that make it easier to infer the existence of simpler ε-optimal strategies. In the following, we first observe the behavior of a general ε-optimal strategy, and then show how a 1-bit strategy can closely match it (for certain types of objectives).

Observing the behavior of an ε-optimal strategy. Consider an acyclic and finitely branching MDP with initial state s_0. (One could also have a finite set of initial states as in [10], but we use a single state to simplify the presentation.) Let σ be an arbitrary ε-optimal strategy from the initial state s_0. We now observe its behavior, i.e., the induced runs. Let $\mathsf{bubble}_k(\{s_0\})$ be the set of states that can be reached from the initial state s_0 within at most k steps. This set is finite for every finite k, since our MDP is finitely branching. Note that, by acyclicity, any given run can visit a given finite set of states X only finitely often (at most $|X|$ times). However, this does not imply that the probability of re-visiting X must eventually become zero. E.g., it is possible that in the i-th state of some run the probability of re-visiting X (in continuations of this run) is 2^{-i} (i.e., re-visiting X remains always possible, but does not happen almost surely).

Still, a weaker property does hold in acyclic and finitely branching MDPs. It follows from acyclicity [10, Lemma 10] that, after a sufficiently large number of steps, runs are arbitrarily unlikely to visit any given finite set of states again. In particular this holds for the finite set $\mathsf{bubble}_k(\{s_0\})$.

Formally, for every k and $\delta > 0$ there exists some l such that the probability of visiting $\mathsf{bubble}_k(\{s_0\})$ after step l is $\leq \delta$. By definition, states *outside* the set $\mathsf{bubble}_l(\{s_0\})$ are reachable only after a number of steps that is strictly larger than l. Therefore, it is unlikely (probability $\leq \delta$) to visit $\mathsf{bubble}_k(\{s_0\})$ again after some state $s \notin \mathsf{bubble}_l(\{s_0\})$ has been visited for the first time.

These observations allow to define a decreasing sequence $\delta_i \overset{\text{def}}{=} \varepsilon \cdot 2^{-i}$ of small errors and sufficiently large and increasing numbers k_i and l_i with $k_i < l_i < k_{i+1}$ for $i \geq 1$ such that for the finite sets $K_i \overset{\text{def}}{=} \mathsf{bubble}_{k_i}(\{s_0\})$ and $L_i \overset{\text{def}}{=} \mathsf{bubble}_{l_i}(\{s_0\})$ it is unlikely (probability $\leq \delta_i$) to visit K_i after leaving L_i (for the first time). I.e., runs like π_2 in Figure 9 are unlikely. However, the probability of leaving K_i and later returning to K_i (even multiple times) before leaving L_i may be large.

Note that we still have a lot of freedom to choose the numbers k_i and l_i. For the numbers k_i we just need $l_i < k_{i+1}$. The minimal required size of l_i depends on k_i, σ and δ_i, but l_i can be chosen arbitrarily larger than this minimal size.

Let now SEQ be the objective of never visiting a set K_i after leaving L_i (for the first time) for any $i \geq 1$. (I.e., SEQ depends on the chosen numbers k_i, l_i.)

The strategy σ is not only ε-optimal for the objective φ from state s_0, but also 2ε-optimal for the stronger objective $\varphi \wedge \mathsf{SEQ}$, since $\sum_{i\geq 1} \delta_i \leq \varepsilon$.

Constructing simpler strategies. For certain types of objectives φ (e.g., Büchi objectives), one can exploit this pattern of SEQ to construct simpler (1-bit) ε-optimal strategies for φ in acyclic and finitely branching MDPs. This then yields ε-optimal 1-bit Markov strategies in general finitely branching MDPs.

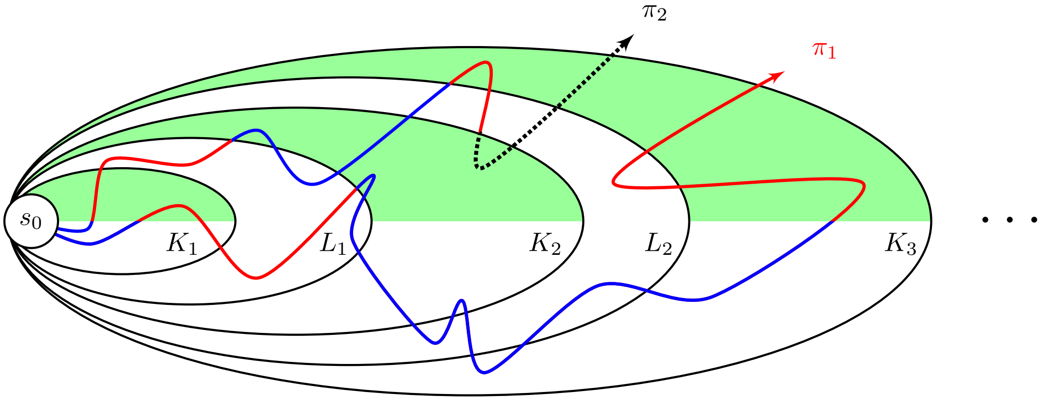


Figure 9 Updates of the extra mode bit along runs π_1, π_2, drawn in blue while the memory-bit is one and in red while the bit is zero. The run π_2 violates SEQ and is drawn as a dotted line once it does. Upon entering the green zone of $K_i \setminus K_{i-1}$, the runs attain the local objective φ_i and flip the mode bit.

Suppose that $\varphi \wedge \mathsf{SEQ}$ can be decomposed into an infinite sequence of local sub-objectives $\varphi_i \wedge \mathsf{SEQ}$ such that, with arbitrarily high probability, satisfying $\varphi_i \wedge \mathsf{SEQ}$ in each finite set $K_i \setminus K_{i-1}$ implies $\varphi \wedge \mathsf{SEQ}$ overall, and vice-versa. (E.g., if φ is a Büchi objective to visit a given set of states F infinitely often then φ_i is the objective to visit the subset of F inside $K_i \setminus K_{i-1}$ (i.e., to visit $F \cap (K_i \setminus K_{i-1})$); cf. [10].) Note that the "vice-versa" part often depends on the fact that the numbers k_i can be chosen sufficiently large to get a sufficiently high probability of satisfying φ_i inside $K_i \setminus K_{i-1}$.

Of course, not every objective φ can be decomposed in this way, e.g., in parity objectives, different runs can win by different colors and local conditions φ_i are insufficient.

Now suppose that in the MDP induced by the finite subspace K_i there exist ε-optimal positional strategies σ_i that attain a high probability of φ_i in $K_i \setminus K_{i-1}$, and additionally maintain a high value w.r.t. future objectives φ_j in $K_j \setminus K_{j-1}$ for all $j > i$. I.e., σ_i has a high attainment for the local sub-objective without compromising future sub-objectives.

One extra bit. The above suggests a scheme to construct an ε-optimal positional strategy σ' for $\varphi \wedge \mathsf{SEQ}$ by playing each local positional strategy σ_i inside $K_i \setminus K_{i-1}$.

However, this is not always sufficient. The problem is that, when playing in $K_i \setminus K_{i-1}$, a run might temporarily go back into the set K_{i-1} (though not into the states that this particular run has previously visited, due to acyclicity). If this run has never yet left L_{i-1}, then going back to K_{i-1} is allowed by SEQ and can be necessary (or even unavoidable). (In contrast, once one has left L_{i-1}, it is possible to henceforth avoid K_{i-1} and still attain φ with high probability, as witnessed by the strategy σ.) But back in K_{i-1} the strategy σ' would play σ_{i-1} towards objective φ_{i-1} (that had already been attained previously) instead of focusing on the current objective φ_i. Although the run will inevitably (by acyclicity) exit K_{i-1} again, it might re-visit K_{i-1} many times, and thus switch the focus back to φ_{i-1} many times. I.e., the strategy σ' might attempt to re-attain the previous objective φ_{i-1} many times over, instead of permanently switching the focus to φ_i once φ_{i-1} has been attained once. Switching the focus back to the previous objective φ_{i-1} too often is wasteful and might damage the ability to attain future objectives φ_j for the $j \geq i$ with high probability, e.g., due to a trade-off between current and future objectives. So this strategy σ' might not always succeed for objective φ (e.g., it cannot work for Büchi objectives [10]).

Instead we want the strategy to always focus on the next objective φ_i after it has completed the previous objective φ_{i-1}. To this end, we need one extra bit of memory, called the *mode bit*, to distinguish two modes of playing: current-mode and next-mode. The mode bit is used to remember whether one has already attained φ_i in $K_i \setminus K_{i-1}$. Upon attaining φ_i in $K_i \setminus K_{i-1}$, one switches from current-mode to next-mode.

One interprets the content of the mode bit (0 or 1) differently for even and odd numbers i, so that the next-mode for $K_i \setminus K_{i-1}$ is the current-mode for $K_{i+1} \setminus K_i$ (and vice-versa). This means that if the strategy plays in next-mode in $K_i \setminus K_{i-1}$ then upon entering $K_{i+1} \setminus K_i$ it automatically switches to current-mode. Dually, if it plays in current-mode in $K_{i+1} \setminus K_i$ and temporarily goes back into $K_i \setminus K_{i-1}$ then it automatically switches to next-mode; cf. Figure 9. The strategy pursues different local goals, depending on the mode bit.

- In current-mode, it continues to focus on attaining φ_i in K_i. Temporarily going back to K_{i-1} does not change the focus on φ_i, because current-mode for K_i is next-mode for K_{i-1}. By suitably choosing φ_i and k_i, one can ensure with high probability that φ_i is attained only in $K_i \setminus L_{i-1}$, i.e., *after* leaving L_{i-1}. So, since one has already left L_{i-1} before this success, it is possible with high probability to avoid K_{i-1} afterwards. In particular, after attaining φ_i in $K_i \setminus L_{i-1}$ the strategy switches the mode-bit to next-mode. The value of the mode bit is the same for next-mode in K_i and for current-mode in K_{i-1}. However, there is only a small danger of ever confusing these, since the probability of visiting K_{i-1} after leaving L_{i-1} is small.
- In next-mode, the focus is on leaving K_i to reach $K_{i+1} \setminus K_i$ and attaining the next objective φ_{i+1}. In particular, upon entering $K_{i+1} \setminus K_i$ the mode bit is interpreted as current mode for $K_{i+1} \setminus K_i$. Moreover, it is then not a problem if the run temporarily goes back from $L_i \setminus K_i$ into K_i, because the focus remains on φ_{i+1} (since current-mode in K_{i+1} is next-mode in K_i).

By combining the positional strategies for the local objectives φ_i with the extra mode bit, one obtains ε-optimal 1-bit strategies on all acyclic finitely branching MDPs. This yields 1-bit Markov strategies on general finitely branching MDPs by the argument above.

For Büchi objectives, one can encode infinite branching into finite branching by a gadget similar to the one used in the proof of Lemma 2. Moreover, the local strategies σ_i can be chosen MD. Thus one obtains deterministic 1-bit Markov ε-optimal strategies. There is also a matching lower bound.

▶ **Theorem 9** ([10], ε-optimal deterministic 1-bit Markov strategies for Büchi objectives). *Given a countable MDP and a Büchi objective, for every $\varepsilon > 0$ and initial state s_0, there exists an ε-optimal deterministic 1-bit Markov strategy. Moreover, neither randomized Markov strategies nor randomized finite-memory strategies are sufficient.*

▶ Remark 10. The whole argument can, of course, be generalized. If the strategies for the local objectives φ_i in acyclic MDPs are not positional but use finite memory, say m bits, then one obtains $(m + 1)$-bit Markov strategies in general MDPs.

5 Missing Proof Details

5.1 Proof of Equation (1)

Proof. For a state $s \in S \setminus G$, define the event L_s as the set of runs that leave G such that s is the first visited state in $S \setminus G$. Then we have:

$$\mathbb{P}^{\sigma}_{\mathcal{M}_i,s_i}(\texttt{Reach}(S \setminus G)) \;=\; \sum_{s \in S \setminus G} \mathbb{P}^{\sigma}_{\mathcal{M}_i,s_i}(L_s) \tag{6}$$

Since T is a sink and using the Markov property:

$$\begin{aligned}\mathbb{P}^{\sigma}_{\mathcal{M}_i,s_i}(\texttt{Reach}(T)) \;&=\; \mathbb{P}^{\sigma}_{\mathcal{M}_i,s_i}(\neg\texttt{Reach}(S\setminus G)\wedge\texttt{Reach}(T)) + \\ &\quad\sum_{s\in S\setminus G}\mathbb{P}^{\sigma}_{\mathcal{M}_i,s_i}(L_s)\cdot\mathbb{P}^{\sigma}_{\mathcal{M}_i,s}(\texttt{Reach}(T))\end{aligned} \tag{7}$$

By the definition of G it follows:

$$\begin{aligned}\mathbb{P}^{\sigma}_{\mathcal{M}_i,s_i}(\texttt{Reach}(T)) \;&\le\; \mathbb{P}^{\sigma}_{\mathcal{M}_i,s_i}(\neg\texttt{Reach}(S\setminus G)\wedge\texttt{Reach}(T)) + \\ &\quad\sum_{s\in S\setminus G}\mathbb{P}^{\sigma}_{\mathcal{M}_i,s_i}(L_s)\cdot(\texttt{val}_{\mathcal{M}_i,s}(\texttt{Reach}(T))-\varepsilon_i)\end{aligned} \tag{8}$$

On the other hand, σ is ε_i^2-optimal for s_i, hence:

$$\begin{aligned}\mathbb{P}^{\sigma}_{\mathcal{M}_i,s_i}(\texttt{Reach}(T)) \;&\ge\; -\varepsilon_i^2+\texttt{val}_{\mathcal{M}_i,s}(\texttt{Reach}(T)) \\ &\ge\; -\varepsilon_i^2+\mathbb{P}^{\sigma}_{\mathcal{M}_i,s_i}(\texttt{Reach}(T)\wedge\neg\texttt{Reach}(S\setminus G)) + \\ &\quad\sum_{s\in S\setminus G}\mathbb{P}^{\sigma}_{\mathcal{M}_i,s_i}(L_s)\cdot\texttt{val}_{\mathcal{M}_i,s}(\texttt{Reach}(T))\end{aligned} \tag{9}$$

By combining (8) and (9) we obtain:

$$\varepsilon_i^2 \;\ge\; \varepsilon_i\cdot\sum_{s\in S\setminus G}\mathbb{P}^{\sigma}_{\mathcal{M}_i,s_i}(L_s) \;=\; \varepsilon_i\cdot\mathbb{P}^{\sigma}_{\mathcal{M}_i,s_i}(\texttt{Reach}(S\setminus G)) \qquad ◀$$

5.2 Proof of Equation (3)

Proof. For a state $s'\in G$, define the event $E_{s'}$ as the set of runs that enter G such that s' is the first visited state in G. Fix any state $s\in S$ and any strategy σ_i in $\mathcal{M}_i$. We transform σ_i into a strategy σ_{i+1} in $\mathcal{M}_{i+1}$ such that σ_{i+1} behaves like σ_i until G is entered, at which point σ_{i+1} switches to the MD strategy σ, which we recall is compatible with $\mathcal{M}_{i+1}$ and is ε_i-optimal from G in $\mathcal{M}_i$. To show (3) it suffices to show that $\mathbb{P}^{\sigma_{i+1}}_{\mathcal{M}_{i+1},s}(\texttt{Reach}(T)) \ge \mathbb{P}^{\sigma_i}_{\mathcal{M}_i,s}(\texttt{Reach}(T))-\varepsilon_i$. We have:

$$\begin{aligned}\mathbb{P}^{\sigma_{i+1}}_{\mathcal{M}_{i+1},s}(\texttt{Reach}(T)) \;&=\; \mathbb{P}^{\sigma_{i+1}}_{\mathcal{M}_{i+1},s}(\neg\texttt{Reach}(G)\wedge\texttt{Reach}(T)) + && \text{as } T \text{ is a sink}\\ &\quad\sum_{s'\in G}\mathbb{P}^{\sigma_{i+1}}_{\mathcal{M}_{i+1},s}(E_{s'})\cdot\mathbb{P}^{\sigma_{i+1}}_{\mathcal{M}_{i+1},s'}(\texttt{Reach}(T)) && \text{Markov property}\\ &=\; \mathbb{P}^{\sigma_i}_{\mathcal{M}_i,s}(\neg\texttt{Reach}(G)\wedge\texttt{Reach}(T)) + && \text{using def. of } \sigma_{i+1}\\ &\quad\sum_{s'\in G}\mathbb{P}^{\sigma_i}_{\mathcal{M}_i,s}(E_{s'})\cdot\mathbb{P}^{\sigma}_{\mathcal{M}_i,s'}(\texttt{Reach}(T))\end{aligned}$$

Further we have for all $s'\in G$:

$$\begin{aligned}\mathbb{P}^{\sigma}_{\mathcal{M}_i,s'}(\texttt{Reach}(T)) \;&\ge\; \texttt{val}_{\mathcal{M}_i,s'}(\texttt{Reach}(T))-\varepsilon_i && \text{as } s'\in G\\ &\ge\; \mathbb{P}^{\sigma_i}_{\mathcal{M}_i,s'}(\texttt{Reach}(T))-\varepsilon_i\end{aligned}$$

Plugging this in above, we obtain:

$$\begin{aligned}
\mathbb{P}^{\sigma_{i+1}}_{\mathcal{M}_{i+1},s}(\mathtt{Reach}(T)) &\geq \mathbb{P}^{\sigma_i}_{\mathcal{M}_i,s}(\neg\mathtt{Reach}(G) \wedge \mathtt{Reach}(T)) + \\
&\qquad \sum_{s' \in G} \mathbb{P}^{\sigma_i}_{\mathcal{M}_i,s}(E_{s'}) \cdot (\mathbb{P}^{\sigma_i}_{\mathcal{M}_i,s'}(\mathtt{Reach}(T)) - \varepsilon_i) \\
&\geq \mathbb{P}^{\sigma_i}_{\mathcal{M}_i,s}(\neg\mathtt{Reach}(G) \wedge \mathtt{Reach}(T)) + \\
&\qquad \Big(\sum_{s' \in G} \mathbb{P}^{\sigma_i}_{\mathcal{M}_i,s}(E_{s'}) \cdot \mathbb{P}^{\sigma_i}_{\mathcal{M}_i,s'}(\mathtt{Reach}(T))\Big) - \varepsilon_i \\
&= \mathbb{P}^{\sigma_i}_{\mathcal{M}_i,s}(\mathtt{Reach}(T)) - \varepsilon_i
\end{aligned}$$

◀

5.3 Proof of Equation (4)

Proof. For all $i \geq 1$ we have:

$$\begin{aligned}
\mathbb{P}^{\sigma}_{\mathcal{M},s_i}(\mathtt{Reach}(T)) &\geq \mathtt{val}_{\mathcal{M}_i,s_i}(\mathtt{Reach}(T)) - \varepsilon_i - \varepsilon_i^2 && \text{by (2)} \\
&\geq \mathtt{val}_{\mathcal{M}_i,s_i}(\mathtt{Reach}(T)) - 2\varepsilon_i && \text{as } \varepsilon_i < 1 \\
&\geq \mathtt{val}_{\mathcal{M}_i,s_i}(\mathtt{Reach}(T)) - \frac{\varepsilon}{2} && \text{choice of } \varepsilon_i \\
&\geq \mathtt{val}_{\mathcal{M},s_i}(\mathtt{Reach}(T)) - \sum_{j=1}^{i-1} \varepsilon_j - \frac{\varepsilon}{2} && \text{by (3)} \\
&\geq \mathtt{val}_{\mathcal{M},s_i}(\mathtt{Reach}(T)) - \varepsilon && \text{choice of } \varepsilon_j
\end{aligned}$$

◀

References

1 P. Abbeel and A. Y. Ng. Learning first-order Markov models for control. In *Advances in Neural Information Processing Systems 17*, pages 1–8. MIT Press, 2004. URL: `http://papers.nips.cc/paper/2569-learning-first-order-markov-models-for-control`.

2 C. Baier and J.-P. Katoen. *Principles of Model Checking*. MIT Press, 2008.

3 V. D. Blondel and J. N. Tsitsiklis. A survey of computational complexity results in systems and control. *Automatica*, 36(9):1249–1274, 2000.

4 N. Bäuerle and U. Rieder. *Markov Decision Processes with Applications to Finance*. Springer-Verlag Berlin Heidelberg, 2011.

5 K. Chatterjee, L. de Alfaro, and T. Henzinger. Trading memory for randomness. In *Annual Conference on Quantitative Evaluation of Systems*, pages 206–217. IEEE Computer Society Press, 2004.

6 K. Chatterjee and T. Henzinger. A survey of stochastic ω-regular games. *Journal of Computer and System Sciences*, 78(2):394–413, 2012.

7 K. Chatterjee, M. Jurdziński, and T. Henzinger. Quantitative stochastic parity games. In *Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 121–130, Philadelphia, PA, USA, 2004. Society for Industrial and Applied Mathematics. URL: `http://dl.acm.org/citation.cfm?id=982792.982808`.

8 E. M. Clarke, T. A. Henzinger, H. Veith, and R. Bloem, editors. *Handbook of Model Checking*. Springer, 2018. `doi:10.1007/978-3-319-10575-8`.

9 W. Feller. *An Introduction to Probability Theory and Its Applications*, volume 1. Wiley & Sons, second edition, 1966.

10 S. Kiefer, R. Mayr, M. Shirmohammadi, and P. Totzke. Büchi objectives in countable MDPs. In *ICALP 2019*, volume 132. LIPIcs, 2019. `doi:10.4230/LIPIcs.ICALP.2019.119`.

11 S. Kiefer, R. Mayr, M. Shirmohammadi, and D. Wojtczak. Parity Objectives in Countable MDPs. In *Annual IEEE Symposium on Logic in Computer Science*, 2017.

12 D. Ornstein. On the existence of stationary optimal strategies. *Proceedings of the American Mathematical Society*, 20:563–569, 1969.

13 M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994.

14 M. Schäl. Markov decision processes in finance and dynamic options. In *Handbook of Markov Decision Processes*, pages 461–487. Springer, 2002.

15 O. Sigaud and O. Buffet. *Markov Decision Processes in Artificial Intelligence.* John Wiley & Sons, 2013.

16 R.S. Sutton and A.G Barto. *Reinforcement Learning: An Introduction.* Adaptive Computation and Machine Learning. MIT Press, 2018.

17 W. Zielonka. Perfect-information stochastic parity games. In *Foundations of Software Science and Computation Structures*, pages 499–513. Springer, 2004.

Scheduling Lower Bounds via AND Subset Sum

Amir Abboud
IBM Almaden Research Center, San Jose, CA, USA
amir.abboud@gmail.com

Karl Bringmann
Saarland University, Saarland Informatics Campus (SIC), Saarbrücken, Germany
Max Planck Institute for Informatics, Saarland Informatics Campus (SIC), Saarbrücken, Germany
bringmann@cs.uni-saarland.de

Danny Hermelin
Department of Industrial Engineering and Management,
Ben-Gurion University of the Negev, Beersheba, Israel
hermelin@bgu.ac.il

Dvir Shabtay
Department of Industrial Engineering and Management,
Ben-Gurion University of the Negev, Beersheba, Israel
dvirs@bgu.ac.il

Abstract

Given N instances $(X_1, t_1), \ldots, (X_N, t_N)$ of Subset Sum, the AND Subset Sum problem asks to determine whether all of these instances are yes-instances; that is, whether each set of integers X_i has a subset that sums up to the target integer t_i. We prove that this problem cannot be solved in time $\widetilde{O}((N \cdot t_{max})^{1-\varepsilon})$, for $t_{max} = \max_i t_i$ and any $\varepsilon > 0$, assuming the $\forall\exists$ Strong Exponential Time Hypothesis ($\forall\exists$-SETH). We then use this result to exclude $\widetilde{O}(n + P_{max} \cdot n^{1-\varepsilon})$-time algorithms for several scheduling problems on n jobs with maximum processing time P_{max}, assuming $\forall\exists$-SETH. These include classical problems such as $1||\sum w_j U_j$, the problem of minimizing the total weight of tardy jobs on a single machine, and $P_2||\sum U_j$, the problem of minimizing the number of tardy jobs on two identical parallel machines.

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

Keywords and phrases SETH, fine grained complexity, Subset Sum, scheduling

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.4

Category Track A: Algorithms, Complexity and Games

Related Version A full version of the paper is available at `http://arxiv.org/abs/2003.07113`.

Funding *Karl Bringmann*: This work is part of the project TIPEA that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 850979).

1 Introduction

The Subset Sum problem is one of the most fundamental problems in computer science and mathematics: Given n integers $X = \{x_1, \ldots, x_n\} \subset \mathbb{N}$, and a target value $t \in \mathbb{N}$, determine whether there is a subset of X that sums[1] to t. This problem appeared in Karp's initial list of 21 NP-complete problems [24], and entire books have been devoted to it and to its closely related variants [25, 30]. Most relevant to this paper is the particular role Subset Sum plays in showing hardness for various problems on integers, essentially being

[1] Note that we can ignore any numbers $x_i > t$, so we will assume throughout the paper that $\max(X) \leq t$.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 4; pp. 4:1–4:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

the most basic such problem where hardness arises exclusively from the additive nature of the problem. In particular, in areas such as operations research, Subset Sum plays a similar role to that of 3-SAT, serving as the core problem used in the vast majority of reductions (see e.g. [9, 11, 15, 24, 28, 32]). Many important problems can be shown to be generalizations of Subset Sum (by easy reductions) including scheduling problems, Knapsack, and Bicriteria Shortest Path. The broad goal of this paper is to understand the fine-grained complexity of such important problems, and more specifically whether the complexity of such generalizations is the same as that of Subset Sum or higher.

While Subset Sum (and its generalizations) is NP-hard, it is well-known that it can be solved in pseudo-polynomial time $O(t \cdot n)$ with the classical dynamic programming algorithm of Bellman [6]. Much more recently, this upper bound was improved to $\widetilde{O}(t + n)$ [7, 23, 27]; this is a significant improvement in the dense regime of the problem, e.g. if $t = O(n^2)$ the new algorithms achieve quadratic as opposed to cubic time. Most recently, in the dense regime the fine-grained complexity of Subset Sum was essentially resolved under the Strong Exponential Time Hypothesis (SETH) by the authors of this paper [1] (the same lower bound was previously known under the incomparable Set Cover Conjecture [12]). SETH [21, 22] postulates that there is no $O(2^{(1-\varepsilon)n})$-time algorithm for deciding the satisfiability of a k-CNF formula, for some $\varepsilon > 0$ independent of k.

▶ **Theorem 1.1** (Hardness of Subset Sum [1]). *Assuming SETH, there is no $\varepsilon > 0$ and $\delta < 1$ such that Subset Sum on n numbers and target t can be solved in time $O(t^{1-\varepsilon} \cdot 2^{\delta n})$.*

The lower bound given by Theorem 1.1 translates directly to several generalizations of Subset Sum, but does this yield tight lower bounds for the generalizations? Or can we prove higher lower bound for them? To answer this kind of question, the OR Subset Sum problem was introduced in [1]: Given N instances $(X_1, t_1), \dots, (X_N, t_N)$ of Subset Sum, determine whether at least one of these instances is a yes-instance; that is, whether there exists an $i \in \{1, \dots, N\}$ such that X_i contains a subset that sums up to t_i. While it seems natural to assume that no algorithm can solve this problem faster than solving each of the N Subset Sum instances independently, it is not clear how to prove this. In fact, an $O(N^{1/10} \cdot \max_i t_i)$ time algorithm for this problem does not directly break the lower bound for Subset Sum. Nevertheless, one can still show a tight lower bound by taking a somewhat indirect route: SAT does have a reduction to its OR variant, and then Theorem 1.1 allows us to reduce OR SAT to OR Subset Sum.

▶ **Theorem 1.2** (Hardness of OR Subset Sum [1]). *Assuming SETH, there are no $\varepsilon, \delta > 0$ such that there is an $O(N^{1+\delta-\varepsilon})$ time algorithm for the following problem: Given N Subset Sum instances, each with $O_{\delta,\varepsilon}(\lg N)$ integers and target $O(N^{\delta})$, determine whether one of these instances is a yes-instances.*

Thus, while Subset Sum admits[2] $\widetilde{O}(n + t)$-time algorithms [7, 23, 27], SETH rules time $\widetilde{O}(N + t)$ for OR Subset Sum. For example, when $N = O(n)$ and $t = O(n^2)$, Subset Sum can be solved in time $O(n^2)$, but OR Subset Sum has a cubic lower bound according to the above theorem. This distinction was used in [1] to show a higher lower bound for a generalization of Subset Sum that is a particularly prominent problem in the operations research community, the Bicriteria Shortest Path problem [19, 41]: Given a graph G with edge lengths and edge costs, two vertices s and t, and a budget B, determine whether there is an s, t-path of total length at most B and total cost at most B. While Theorem 1.1 immediately rules out time

[2] The term $\widetilde{O}()$ is used here and throughout the paper to suppress logarithmic factors.

$B^{1-\varepsilon} \cdot 2^{o(n)}$, it leaves open the possibility of an $\widetilde{O}(B+n)$ algorithm (as is known to exist for Subset Sum). As it turns out, Bicriteria Shortest Path can not only encode a *single* Subset Sum instance, but even *several* instances, and thus Theorem 1.2 yields an $\Omega(n + Bn^{1-\varepsilon})$ lower bound under SETH.

1.1 An Analogue of Theorem 1.2 for AND Subset Sum

While the OR variant in Theorem 1.2 is perfectly suited for showing lower bounds for Bicriteria Shortest Path and other problems of a similar type, there are others, such as the scheduling problems discussed below, whose type can only capture an AND variant: Given N instances of Subset Sum, determine whether *all* are yes-instances. It is natural to wonder whether there is a fine-grained reduction from SAT to AND Subset Sum (either directly or indirectly, by first reducing to AND SAT). Intuitively, the issue is that SAT, Subset Sum, and their OR variants have an $\exists$ quantifier type, while AND SAT and AND Subset Sum have a $\forall\exists$ quantifier type. Reducing one type to another seems very challenging, but fortunately, a morally similar challenge had been encountered before in fine-grained complexity and resolved to some extent as follows.

First, we can observe that the reduction we are looking for is impossible under the Nondeterministic Strong Exponential Time Hypothesis (NSETH) [10] which states that no non-deterministic $O(2^{(1-\varepsilon)n})$-time algorithm can decide whether a given k-CNF is unsatisfiable, for an $\varepsilon > 0$ independent of k. This hypothesis was introduced to show non-reducibility results. Intuitively, NSETH says that even though SAT is easy for nondeterministic algorithms its complement is not. Therefore, if for a certain problem both it and its complement are easy for nondeterministic algorithms then a reduction from SAT is impossible. Note that AND SAT, AND Subset Sum, and their complements admit efficient nondeterministic algorithms: to prove that the AND is "yes" we can guess a solution in each instance, and (for the complement) to prove that the AND is "no" we can guess the index of the instances that is "no". (Notice that the latter is not possible for the OR variants.)

There are already conjectures in fine-grained complexity that can capture problems with a $\forall\exists$ type. In the "n^2 regime", where SAT is faithfully represented by the Orthogonal Vectors (OV) problem[3] which has an $\exists$ type, Abboud, Vassilevska Williams and Wang [2] introduced a hardness hypothesis about the Hitting Set (HS) problem[4] which is the natural $\forall\exists$ type variant of OV. This hypothesis was used to derive lower bounds that cannot (under NSETH) be based on OV or SETH, e.g. for graph median and radius [2, 3, 13] and for Earth Mover Distance [35], and was also studied in the context of model checking problems [18]. Going back to the "2^n regime", the analogous hypothesis, which implies the HS hypothesis, is the following.

▶ **Hypothesis** ($\forall\exists$-SETH). *There is no $0 < \alpha < 1$ and $\varepsilon > 0$ such that for all $k \geq 3$ we can decide in time $O(2^{(1-\varepsilon)n})$, given a k-CNF formula ϕ on n variables $x_1, \dots, x_n$, whether for all assignments to $x_1, \dots, x_{\lceil \alpha \cdot n \rceil}$ there exists an assignment to the rest of the variables that satisfies ϕ, that is, whether:*

$$\forall x_1, \dots, x_{\lceil \alpha \cdot n \rceil} \exists x_{\lceil \alpha \cdot n \rceil + 1}, \dots, x_n : \phi(x_1, \dots, x_n) = \mathit{true}.$$

[3] Given two sets of n binary vectors of dimension $O(\log n)$, decide whether there is a vector in the first set and a vector of the second set that are orthogonal. SETH implies that this problem cannot be solved in time $O(n^{2-\varepsilon})$ [40], and essentially all SETH-based n^2 lower bounds go through this problem.

[4] Given two sets of n binary vectors of dimension $O(\log n)$, decide whether for all vectors in the first set there is an orthogonal vector in the second set. The Hitting Set Hypothesis states that this problem cannot be solved in time $O(n^{2-\varepsilon})$ for any $\varepsilon > 0$.

Note that this hypothesis may also be thought of as the Π_2-SETH, where Π_2 is the second level of the polynomial hierarchy, and one can also think of higher levels of the polynomial hierarchy. Indeed, Bringmann and Chaudhury [8] recently proposed such a version, called Quantified-SETH, in which we can have any constant number $q \geq 1$ of alternating quantifier blocks, with a constant fraction of the variables in each block[5]. Non-trivial algorithms for Quantified-SAT exist [37], but none of them can refute even the stronger of these hypotheses.

It is important to note that while ∀∃-SAT is a strictly harder problem than SAT (as adding more quantifiers can only make the problem harder), in the restricted setting of ∀∃-SETH, where there is a constant fraction of the variables in each quantifier block, the situation is the opposite! A faster algorithm for SAT does imply a faster algorithm for ∀∃-SAT: exhaustively search over all assignments to the universally quantified αn variables and for each assignment solve SAT on $(1-\alpha)n$ variables. A reduction in the other direction is impossible under NSETH[6]. Therefore, ∀∃-SETH is a stronger assumption than SETH, which explains why it is helpful for proving more lower bounds, yet it seems equally plausible (to us). In particular, it gives us a tight lower bound for AND Subset Sum which we will use to show higher lower bounds for scheduling problems.

▶ **Theorem 1.3** (Hardness of AND Subset Sum). *Assuming ∀∃-SETH, there are no $\varepsilon, \delta > 0$ such that the following problem can be solved in time $O(N^{1+\delta-\varepsilon})$: Given N Subset Sum instances, each with $O(N^\varepsilon)$ integers and target $O(N^\delta)$, determine whether all of these instances are yes-instances.*

Note that in comparison with the OR Subset Sum case (Theorem 1.2), the size of our instances is polynomial $O(N^\varepsilon)$ instead of logarithmic $O_{\delta,\varepsilon}(\log N)$. We leave it as an open problem whether this is inherent or Theorem 1.3 can be improved.

It follows from Theorem 1.3 that AND Subset Sum on N instances, each on at most s integers and with target at most t, cannot be solved in time $\widetilde{O}(Ns + t(Ns)^{1-\varepsilon})$. We show that the same holds for the Partition problem, which is the special case of Subset Sum where the target is half of the total input sum. This is the starting point for our reductions in the next section.

▶ **Corollary 1.4.** *Assuming ∀∃-SETH, there is no $\varepsilon > 0$ such that the following problem can be solved in time $\widetilde{O}(Ns + t(Ns)^{1-\varepsilon})$: Given N Partition instances, each with at most s integers and target at most t, determine whether all of these instances are yes-instances.*

1.2 Scheduling lower bounds

To exemplify the power of Theorem 1.3, we use it to show strong lower bounds for several non-preemptive scheduling problems that generalize Subset Sum. These problems include some of the most basic ones such as minimizing the total weight of tardy jobs on a single machine, or minimizing the number of tardy jobs on two parallel machines. Theorem 1.5 below lists all of these problems; they are formally defined in Section 3 and each requires a different reduction. To describe the significance of our new lower bounds more clearly, let us focus on only one of these problems, $P_2||\sum U_j$, for the rest of this section. The input to this problem is a set of n jobs, where each job J_j has a processing time p_j and a due

[5] However, we remark that for the purposes of their paper as well as ours ∀∃-SETH is sufficient; Quantified-SETH is merely mentioned for inspiration. They were motivated by understanding the complexity of the polyline simplification problem from geometry (which turns out to have a ∀∀∃ type).

[6] This is analogous to the "n^2 regime" where HS implies OV but not the other way, assuming NSETH.

date d_j, and the goal is to schedule all jobs on two parallel machines so that the number of jobs exceeding their due dates is minimal. Let $P = \sum_j p_j$ and $P_{max} = \max_j p_j$ denote the sum of processing times and maximum processing time of the input jobs. Observe that $P \leq P_{max} \cdot n$.

The standard dynamic programming algorithm for this problem runs in $O(P \cdot n) = O(P_{max} \cdot n^2)$ time [29], and it is not known whether this running time is the best possible. Nevertheless, there is a well-known easy reduction from Subset Sum on numbers $x_1, \ldots, x_n$ to $P_2||\sum U_j$ that generates an instance with total processing time $P = \sum x_i = O(n \cdot t)$ and $P_{max} = \max x_i = O(t)$. Thus, using Theorem 1.1, we can rule out $P^{1-\varepsilon} \cdot 2^{o(n)}$-time and $P_{max}^{1-\varepsilon} \cdot 2^{o(n)}$-time algorithms for $P_2||\sum U_j$. However, this leaves open the possibility of $\widetilde{O}(P_{max} + n)$-time algorithms, which would be near-linear as opposed to the currently known cubic algorithm in a setting where $P_{max} = \Theta(n)$ and $P = \Theta(n^2)$. One approach for excluding such an upper bound is to first prove the impossibility of an algorithm for Subset Sum with running time $\widetilde{O}(\max_{x \in X} x + n)$. However, such a result has been elusive and is perhaps the most interesting open question in this context [4, 16, 17, 27, 33]. Instead, taking an indirect route, we are able to exclude such algorithms with an $\Omega(n + P_{max} n^{1-\varepsilon})$ lower bound under ∀∃-SETH by showing that $P_2||\sum U_j$ can actually encode the AND of several Subset Sum instances. In particular, in the above regime we improve the lower bound from linear to quadratic.

▶ **Theorem 1.5.** *Assuming ∀∃-SETH, for all $\varepsilon > 0$, none of the following problems have $\widetilde{O}(n + P_{max} \cdot n^{1-\varepsilon})$-time algorithms:*

- $1||\sum w_j U_j$, $1|Rej \leq R|\sum U_j$, $1|Rej \leq R|T_{max}$, *and* $1|r_j \geq 0, Rej \leq R|C_{max}$.
- $P_2||T_{max}$, $P_2||\sum U_j$, $P_2|r_j|C_{max}$, *and* $P_2|\textit{level-order}|C_{max}$.

All problems listed in this theorem are direct generalizations of Subset Sum, and each one admits a $O(P \cdot n) = O(P_{max} \cdot n^2)$-time algorithm via dynamic programming [29, 36, 38].

We note that the distinction between running times depending on P versus P_{max} and n relates to instances with low or high variance in their job processing times. In several experimental studies, it has been reported by researchers that the ability of scheduling algorithms to solve NP-hard problems deteriorates when the variance in job processing time increases (see e.g. [26, 31, 34]). Our results provide theoretical evidence for this claim by showing tighter lower bounds on the time complexity of several scheduling problems based on the maximum processing time P_{max}.

2 Quantified SETH Hardness of AND Subset Sum

In the following we provide a proof for Theorem 1.3, the main technical result of the paper. For this, we present a reduction from Quantified k-SAT to AND Subset Sum which consists of two main steps. The first step uses a tool presented in [1] which takes a (non-quantified) k-SAT instance and reduces it to subexponentially many Subset Sum instances that have relatively small targets. The second step is a new tool, which we develop in Section 2.2, that takes many Subset Sum instances and reduces them to a single instance with only a relatively small increase of the output target.

2.1 Main construction

The following two theorems formally state the two main tools that are used in our construction. Note that for our purpose, the important property here is the manageable increase of the output target in both theorems. The proof of Theorem 2.1 can be found in [1], while the proof of Theorem 2.2 is given in Section 2.2.

▶ **Theorem 2.1** ([1]). *For any $\varepsilon > 0$ and $k \geq 3$, given a k-SAT formula ϕ on n variables, we can in time $2^{\varepsilon n} \cdot n^{O(1)}$ construct $2^{\varepsilon n}$ Subset Sum instances, each with $O(n)$ integers and target at most $2^{(1+\varepsilon)n}$, such that ϕ is satisfiable if and only if at least one of the Subset Sum instances is a yes-instance.*

▶ **Theorem 2.2.** *Given Subset Sum instances $(X_1, t_1), \ldots, (X_N, t_N)$, denoting $n = \max_i |X_i|$ and $t = \max_i t_i$, we can construct in time $(nN \log t)^{O(1)}$ a single Subset Sum instance (X_0, t_0), with $|X_0| = O(nN)$ and $t_0 = t \cdot (nN)^{O(1)}$, such that (X_0, t_0) is a yes-instance if and only if (X_i, t_i) is a yes-instance for some $i \in \{1, \ldots, N\}$.*

Using the two results above, the proof of Theorem 1.3 follows by combining both constructions given by the theorems:

Proof of Theorem 1.3. Let ϕ be a k-SAT formula on n variables and let $0 < \alpha < 1$. We write $n_1 = \lfloor \alpha \cdot n \rfloor$ and $n_2 = n - n_1$, so that $n_1 \leq \alpha n$ and $n_2 \leq (1-\alpha)n + 1$. Our goal is to determine whether $\forall x_1, \ldots, x_{n_1} \exists x_{n_1+1}, \ldots, x_n \colon \phi(x_1, \ldots, x_n)$ is true.

We enumerate all assignments ∂ of the variables $x_1, \ldots, x_{n_1}$, and let ϕ_∂ be the resulting k-SAT formula on n_2 variables after applying ∂. Note that there are 2^{n_1} formulas ϕ_∂.

For each formula ϕ_∂, we run the reduction from Theorem 2.1 with parameter ε_0, resulting in a set $\mathcal{I}_\partial$ of at most $2^{\varepsilon_0 n_2}$ Subset Sum instances such that ϕ_∂ is satisfiable if and only if at least one of the instances in $\mathcal{I}_\partial$ is a yes-instance. Note that each Subset Sum instance in $\mathcal{I}_\partial$ consists of $O(n_2) = O(n)$ integers and has target at most $t = 2^{(1+\varepsilon_0)n_2}$. Moreover, running this reduction for all formulas ϕ_∂ takes time $2^{n_1 + \varepsilon_0 n_2} n^{O(1)}$.

Next, using Theorem 2.2, we reduce $\mathcal{I}_\partial$ to a single Subset Sum instance (X_∂, t_∂) such that (X_∂, t_∂) is yes-instance if and only if ϕ_∂ is a yes-instance, and so ϕ is a yes-instance if and only if *all* (X_∂, t_∂) are yes-instances. Note that we have $|X_\partial| = O(n \cdot 2^{\varepsilon_0 n_2})$ and $t_\partial = O(2^{(1+\varepsilon_0)n_2} \cdot (n \cdot 2^{\varepsilon_0 n_2})^\gamma)$ for some constant $\gamma > 0$ that replaces a hidden constant in Theorem 2.2. Moreover, running this step for all formulas ϕ_∂ takes time $O(2^{n_1} \cdot (n 2^{\varepsilon_0 n_2})^\gamma)$, where again $\gamma > 0$ replaces a hidden constant in Theorem 2.2.

Finally, we assume that for some $\varepsilon', \delta > 0$ we can solve AND Subset Sum on N instances, each with $O(N^{\varepsilon'})$ integers and target $O(N^\delta)$, in time $O(N^{1+\delta-\varepsilon'})$. Set $\varepsilon := \varepsilon'/(1+\delta)$ and note that then we can in particular solve AND Subset Sum on N instances, each with $O(N^\varepsilon)$ integers and target $O(N^\delta)$, in time $O(N^{(1+\delta)(1-\varepsilon)})$; for convenience we use this formulation in the following. Clearly, we can assume $\varepsilon < 1$. Set

$$N := 2^{(1+\varepsilon)n_1} = \Theta(2^{(1+\varepsilon)\alpha n}),$$

and note that the number of instances (X_∂, t_∂) is $2^{n_1} \leq N$. In order to apply the assumed algorithm to the instances (X_∂, t_∂), we need to verify that $|X_\partial| = O(N^\varepsilon)$ and $t_\partial = O(N^\delta)$. To this end, we set $\alpha := 1/(1+\delta)$ and $\varepsilon_0 := \min\{\varepsilon\alpha, \varepsilon/(1+2\gamma)\}$, and check that

$$\begin{aligned} |X_\partial| &= O(n \cdot 2^{\varepsilon_0 n_2}) = O(2^{\varepsilon_0 n}) = O(N^\varepsilon), \\ t_\partial &= O(2^{(1+\varepsilon_0)n_2} \cdot (n \cdot 2^{\varepsilon_0 n_2})^\gamma) = O(2^{(1+\varepsilon_0(1+2\gamma))n_2}). \end{aligned}$$

Using $\varepsilon_0 \leq \varepsilon/(1+2\gamma)$ and $n_2 \leq (1-\alpha)n + 1$, we further simplify this to $t = O(2^{(1+\varepsilon)(1-\alpha)n})$. From our setting of $\alpha = 1/(1+\delta)$ it now follows that $t = O(2^{(1+\varepsilon)\delta\alpha n}) = O(N^\delta)$. Hence, we showed that the assumed algorithm for AND Subset Sum is applicable to the at most N instances (X_∂, t_∂). This algorithm runs in time

$$O(N^{(1+\delta)(1-\varepsilon)}) = O(2^{(1+\delta)(1-\varepsilon)(1+\varepsilon)\alpha n}) = O(2^{(1-\varepsilon^2)n}).$$

Additionally, as analyzed above, the running time incurred by the reduction is bounded by

$$O\big(2^{n_1+\varepsilon_0 n_2} n^{O(1)} + 2^{n_1} \cdot (n 2^{\varepsilon_0 n_2})^{\gamma}\big) = O(2^{n_1+\varepsilon_0(1+2\gamma)n_2}) = O(2^{\alpha n+\varepsilon(1-\alpha)n}) \\ = O(2^{(1-(1-\varepsilon)(1-\alpha))n}).$$

Hence, we can solve the quantified k-CNF formula ϕ in time $O(2^{(1-\min\{\varepsilon^2,(1-\varepsilon)(1-\alpha)\})n})$, which for sufficiently large k violates ∀∃-SETH. ◀

Next, we infer Corollary 1.4 from Theorem 1.3.

Proof of Corollary 1.4. Fix any $\delta > 0$, and let $\varepsilon > 0$ to be chosen later. For given Subset Sum instances $(X_1, t_1), \ldots, (X_N, t_N)$, each with $O(N^{\varepsilon})$ integers and target $O(N^{\delta})$, our goal is to determine whether all of these instances are yes-instances.

For each i, we construct a Partition instance (X_i^*, t_i^*) by setting

$$X_i^* := X_i \cup \Big\{\Big(\sum_{x\in X_i} x\Big) + t_i, 2\cdot\Big(\sum_{x\in X_i} x\Big) - t_i\Big\} \qquad \text{and} \qquad t_i^* := \frac{1}{2}\sum_{x\in X_i^*} x.$$

It is easy to see that the Partition instance (X_i^*, t_i^*) is equivalent to the Subset Sum instance (X_i, t_i). Indeed, the two additional items cannot be put on the same side of the partition, as their sum is too large. Putting them on different sides of the partition, it remains to split X_i into a subset $Y_i \subseteq X_i$ summing to t_i and the remainder $X_i \setminus Y_i$ summing to $(\sum_{x\in X_i} x) - t_i$, to obtain a balanced partition.

Observe that $|X_i^*| = O(|X_i|) = O(N^{\varepsilon})$ and $t_i^* = O(|X_i| \cdot t_i) = O(N^{\delta+\varepsilon})$.

Now assume that we can solve AND Partition on N instances, each with at most s integers and target at most t, in time $O(Ns + t(Ns)^{1-\varepsilon_0})$ for some $\varepsilon_0 > 0$. On the instances $(X_1^*, t_1^*), \ldots, (X_n^*, t_N^*)$, this algorithm would run in time

$$\widetilde{O}(Ns + t(Ns)^{1-\varepsilon_0}) = \widetilde{O}(N^{1+\varepsilon} + N^{\delta+\varepsilon+(1+\varepsilon)(1-\varepsilon_0)}) = \widetilde{O}(N^{1+\varepsilon} + N^{1+\delta+2\varepsilon-\varepsilon_0}).$$

Finally, we pick $\varepsilon := \min\{\delta/2, \varepsilon_0/3\}$ to bound this running time by $O(N^{1+\delta-\varepsilon})$. This violates Theorem 1.3. ◀

2.2 From OR Subset Sum to Subset Sum

We next provide a proof of Theorem 2.2, the second tool used in our reduction from Quantified k-SAT to Subset Sum. We will use the notion of average-free sets.

▶ **Definition 2.3** (m-average-free set). *A set of integers S is called m-average-free if for all (not necessarily distinct) integers $s_1, \ldots, s_{m+1} \in S$ we have:*

$$s_1 + \cdots + s_m = m \cdot s_{m+1} \quad \textit{implies that} \quad s_1 = \cdots = s_{m+1}.$$

▶ **Lemma 2.4** ([5]). *Given $m \geq 2$, $M \geq 1$, and $0 < \varepsilon < 1$, an m-average-free set S of size M with $S \subseteq [0, m^{O(1/\varepsilon)} M^{1+\varepsilon}]$ can be constructed in $M^{O(1)}$ time.*

Proof of Theorem 2.2. Let $(X_1, t_1), \ldots, (X_N, t_N)$ be N given Subset Sum instances, and write $t = \max_i t_i$ and $n = \max_i |X_i|$. We begin by slightly modifying these instances. First, let $t^* = (n+1)t$, and add to each X_i the integer $t^* - t_i$. Clearly, there is a subset of X_i which sums up to t_i if and only if there is a subset of $X_i \cup \{t^* - t_i\}$ that sums up to t^*. Next, we add at most $2(n+1)$ copies of 0 to each instance, ensuring that all instances have the same number of integers $2(n+1)$, and that any instance which has a solution also has one which includes exactly $n+1$ integers. Note that these modifications only change n by a constant factor, and t by a factor $O(n)$, which are negligible for the theorem statement.

Therefore, with slight abuse of notation, henceforth we assume that we are given N Subset Sum instances $(X_1, t_1), \ldots, (X_N, t_N)$ with $t_1 = \ldots = t_N = t$ and $|X_1| = \ldots = |X_N| = 2n$. Moreover, for any i if there exists a subset $Y_i \subseteq X_i$ that sums up to t then we can assume without loss of generality that $|Y_i| = n$.

We construct an n-average-free set $S = \{s_1, \ldots, s_N\}$, with $S \subseteq [0, N^2 \cdot n^{O(1)}]$, using Lemma 2.4. Let $S_{\max} = \max_{s \in S} s$.

We are now ready to describe our construction of (X_0, t_0). It will be convenient to view the integers in (X_0, t_0) as binary encoded numbers, or binary strings, and to describe how they are constructed in terms of *blocks* of consecutive bits. Each integer will consist of seven blocks of fixed sizes. Starting with the least significant bit, the first block has $\lceil \lg t \rceil$ bits and is referred to as the *encoding block*, the third block has $\lceil \lg n \rceil$ bits and is referred to as the *counting block*, the fifth block has $\lceil \log(n \cdot S_{\max}) \rceil = O(\log(nN))$ bits and is referred to as the *verification block*, and the last block consists of a single bit. In between these blocks are blocks containing $\lceil \log(2nN) \rceil$ bits of value 0, whose sole purpose is to avoid overflows.

For each integer $x_{i,j} \in X_i$, we construct a corresponding integer $x^0_{i,j} \in X_0$ as follows (here the "|"-characters are used only to differentiate between blocks, and have no other meaning):

$$x^0_{i,j} \;=\; 0 \;|\; 0\cdots0 \;|\; s_i \;|\; 0\cdots0 \;|\; 0\cdots01 \;|\; 0\cdots0 \;|\; x_{i,j},$$

Additionally, for each $i \in \{1, \ldots, N\}$ we construct an integer $x^0_i \in X_0$ associated with the instance (X_i, t_i) as

$$x^0_i \;=\; 1 \;|\; 0\cdots0 \;|\; n \cdot (S_{\max} - s_i) \;|\; 0\cdots0 \;|\; 0\cdots0 \;|\; 0\cdots0 \;|\; 0.$$

The two sets of integers described above constitute X_0. To complete the construction of the output instance, we construct the target integer t_0 as

$$t_0 \;=\; 1 \;|\; 0\cdots0 \;|\; n \cdot S_{\max} \;|\; 0\cdots0 \;|\; n \;|\; 0\cdots0 \;|\; t^*.$$

Note that $|X_0| = O(\sum_i |X_i|) = O(nN)$ and $t_0 = t \cdot (nN)^{O(1)}$, as required by the theorem statement. Furthermore, the time required to construct (X_0, t_0) is $(nN \log t)^{O(1)}$.

We next argue that (X_0, t_0) is a yes-instance if and only if (X_i, t_i) is a yes-instance for some $i \in \{1, \ldots, N\}$. Suppose that there exists some $i \in \{1, \ldots, N\}$ and some $Y_i \subseteq X_i$ for which $\sum_{x_{i,j} \in Y_i} x_{i,j} = t$. By the discussion at the beginning of this proof, we can assume that $|Y_i| = n$. It is not difficult to verify that all integers in $Y_0 := \{x^0_{i,j} : x_{i,j} \in Y_i\} \cup \{x^0_i\}$ sum up to t_0. Indeed, by construction, the bits in the encoding block of these integers sum up to $\sum_{x_{i,j} \in Y_i} x_{i,j} = t$, the bits in the counting block sum up to n, the bits in the verification block sum up to $n \cdot S_{\max}$, and the last bit sums up to 1.

Conversely, assume that there is some subset $Y_0 \subseteq X_0$ with $\Sigma(Y_0) = \sum_{x \in Y_0} x = t_0$. Let $y_1, \ldots, y_m \in Y_0$ denote all integers of the form $x^0_{i,j}$ in Y_0, and let $x_{i_1,j_1}, \ldots, x_{i_m,j_m} \in X_1 \cup \cdots \cup X_M$ denote the integers that appear in the encoding blocks of $y_1, \ldots, y_m$. Observe that as $m \leq 2nM$, by our construction the highest bit in each overflow block of $\Sigma(Y_0)$ must be 0. It follows that we can argue in each of the encoding block, counting block, verification block, and last block separately. This yields:

- $\sum_\ell x_{i_\ell,j_\ell} = t$, since if this sum is greater than t then the second block of $\Sigma(Y_0)$ would not be all zeros, and if $\sum_\ell x_{i_\ell,j_\ell} < t$ then the encoding block of $\Sigma(Y_0)$ would not be t.
- $m = n$, by a similar argument in the counting block.

- There is exactly one integer of the form $x^0_{i^*}$ in Y_0, for some $i^* \in \{1, \ldots, N\}$, as otherwise the most significant bit of $\Sigma(Y_0)$ would not be 1.
- $i^* = i_1 = \cdots = i_n$: Note that $x^0_{i^*}$ contributes $n \cdot (S_{\max} - s_{i^*})$ to the verification block of $\Sigma(Y_0)$, and so the remaining n integers in Y_0 need to contribute together exactly $n \cdot s_{i^*}$ to this block, since the value of this block is $n \cdot S_{\max}$ in t_0. Since S is an n-average-free set, the only way for this to occur is if all of these integers have s_{i^*} encoded in their verification blocks, implying that $i^* = i_1 = \cdots = i_n$.

Let $i = i^*$ be the index in the last point above. Then $\sum_\ell x_{i,j_\ell} = t$ by the first point above, and so the subset $\{x_{i,j_1}, \ldots, x_{i,j_n}\}$ is a solution for the instance (X_i, t_i). ◀

3 Scheduling Lower Bounds

We next show how to apply Corollary 1.4 to obtain $\Omega(n + P_{max} \cdot n^{1-\varepsilon})$ lower bounds for several scheduling problems. In particular, we provide a complete proof of Theorem 1.5 in a sequence of lemmas below, each exhibiting a reduction from AND Subset Sum (or rather AND Partition) to the scheduling problem at hand.

In each reduction, we start with N Partition instances $(X_1, t_1), \ldots, (X_N, t_N)$; these are Subset Sum instances with $t_i = \frac{1}{2} \sum_{x \in X_i} x$. We write $s = \max_i |X_i|$ and $t = \max_i t_i$. We present reductions that transform these given instances into an instance $\mathcal{I}$ of a certain scheduling problem, such that $\mathcal{I}$ is a yes-instance if and only if (X_i, t_i) is a yes-instance for all i. The constructed instance $\mathcal{I}$ will consist of $n = O(Ns)$ jobs with maximum processing time $P_{max} = O(t)$. Since Corollary 1.4 rules out time $\widetilde{O}(Ns + t(Ns)^{1-\varepsilon})$ for AND Partition, it follows that the scheduling problem is not in time $\widetilde{O}(n + P_{max} \cdot n^{1-\varepsilon})$, for any $\varepsilon > 0$ assuming ∀∃-SETH.

For an instance (X_i, t_i), we let $x_{i,j}$ denote the j-th integer in X_i.

3.1 Scheduling Notation and Terminology

In all scheduling problems considered in this paper, we are given a set of jobs $J_1, \ldots, J_n$ to be scheduled non-preemptively on one or two identical parallel machines. Each job J_j has a *processing time* p_j, and according to the specific problem at hand, it may also have a *due date* d_j, a *release date* r_j, and a *weight* w_j. We always use the same subscript for the job and its parameters. A *schedule* consists of assigning each job J_j a machine $M(J_j)$ and a *starting time* $S_j \in \mathbb{N}^{\geq 0}$. The *completion time* of job j in a given schedule is $C_j = S_j + p_j$, and the *makespan* of the schedule is its maximum completion time $C_{max} = \max_j C_j$. A schedule is *feasible* if no two distinct jobs *overlap* on the same machine; that is, for any pair of distinct jobs J_j and J_k with $M(J_j) = M(J_k)$ and $S_j \leq S_k$ we have $S_k \notin [S_j, C_j)$. Furthermore, when release dates are present, we require that $S_j \geq r_j$ for each job J_j.

A job J_j is said to be tardy in a given schedule if $C_j > d_j$, and otherwise it is said to be early. For each job J_j, we let $U_j \in \{0, 1\}$ denote a Boolean variable with $U_j = 1$ if J_j is tardy and otherwise $U_j = 0$. In this way, $\sum U_j$ denotes the number of tardy jobs in a given schedule, and $\sum w_j U_j$ denote their total weight. We let T_j denote the *tardiness* of a job J_j defined by $T_j = \max\{0, C_j - d_j\}$, and we let $T_{max} = \max_j T_j$ denote the *maximum tardiness* of the schedule. Below we use the standard three field notation $\alpha|\beta|\gamma$ introduced by Graham et al. [20] to denote the various problems, where α denotes the machine model, β denotes the constrains on the problem, and γ is the objective function. Readers unfamiliar with the area of scheduling are also referred to [32] for additional background.

3.2 Problems on Two Machines

We begin by considering scheduling problems on two parallel identical machines, as here our reductions are simpler to describe. Recall that in this setting, a schedule consists of assigning a starting-time S_j and a machine $M(J_j)$ to each input job J_j.

3.2.1 $P_2|$*level-order*$|C_{max}$

Perhaps the easiest application of Theorem 1.3 is makespan minimization on two parallel machines when level-order precedence constraints are present [14, 39]. In this problem, jobs only have processing-times, and they are partitioned into classes $\mathcal{J}_1, \ldots, \mathcal{J}_k$ such that all jobs in any class $\mathcal{J}_i$ must be scheduled after all jobs in $\mathcal{J}_{i-1}$ are completed. The goal is to find a feasible schedule with minimum makespan $C_{max} = \max_j C_j$.

▶ **Lemma 3.1.** *$P_2|$level-order$|C_{max}$ has no $\widetilde{O}(n + P_{max} \cdot n^{1-\varepsilon})$-time algorithm, for any $\varepsilon > 0$, unless $\forall\exists$-SETH is false.*

Proof. First recall that a single Partition instance (X, t) easily reduces to an instance of $P_2||C_{max}$ (*i.e.* without precedence constraints on the jobs) by creating a job with processing time x for each $x \in X$, and then setting the required makespan C to be $C = t$. For reducing multiple Partition instances we can use the precedence constraints: For each instance (X_i, t_i) of Partition, we create a class of jobs $\mathcal{J}_i$ which includes a job $J_{i,j}$ for each $x_{i,j} \in X_i$ with processing time $p_{i,j} = x_{i,j}$. Then since all jobs in class $\mathcal{J}_i$ must be processed after all jobs in $\mathcal{J}_1, \ldots, \mathcal{J}_{i-1}$ are completed, it is easy to see that the $P_2|$*level-order*$|C_{max}$ instance has a feasible schedule with makespan at most $C = \sum_i t_i$ if and only if each Partition instance is a yes-instance.

Indeed, if each X_i has a subset $Y_i \subset X_i$ which sums up to $t_i = \frac{1}{2} \cdot \sum_j x_{i,j}$, then we can schedule all jobs $J_{i,j}$ associated with elements $x_{i,j} \in Y_i$ on the first machine (following all jobs associated with elements in $Y_1, \ldots, Y_{i-1}$), and all jobs $J_{i,j}$ associated with elements $x_{i,j} \notin Y_i$ on the second machine. This gives a feasible schedule with makespan at most C. Conversely, a schedule with makespan at most C must have the last job in $\mathcal{J}_i$ complete no later than $\sum_{i_0 \leq i} t_{i_0}$, for each $i \in \{1, \ldots, N\}$. This in turn can only be done if each X_i can be partitioned into two sets that sum up to t_i, which implies that each (X_i, t_i) is a yes-instance.

Starting from N Partition instances $(X_1, t_1), \ldots, (X_N, t_N)$, each with at most s integers and target at most t, our reduction constructs $n \leq Ns$ jobs with maximum processing time $P_{max} \leq t$. Therefore, any $\widetilde{O}(n + P_{max} \cdot n^{1-\varepsilon})$-time algorithm for $P_2|$*level-order*$|C_{max}$ would yield an $\widetilde{O}(Ns + t(Ns)^{1-\varepsilon})$-time algorithm for AND Partition, which contradicts Corollary 1.4, assuming $\forall\exists$-SETH. ◀

3.2.2 $P_2||T_{max}$ and $P_2||\sum U_j$

We next consider the $P_2||T_{max}$ and $P_2||\sum U_j$ problems, where jobs also have due dates, and the goal is to minimize the maximum tardiness and the total number of tardy jobs, respectively. The reduction here is very similar to the previous reduction. We create for each $i \in \{1, \ldots, N\}$, and each $x_{i,j} \in X_i$, a job $J_{i,j}$ with processing time $p_{i,j} = x_{i,j}$ and due date

$$d_{i,j} = d_i = \sum_{\ell=1}^{i} t_\ell = \frac{1}{2} \cdot \sum_{\ell=0}^{i} \sum_{j} x_{\ell,j}.$$

Observe that for each $i \in \{1, \ldots, N\}$, all jobs $J_{i,j}$ can be scheduled early if and only if X_i can be partitioned into two sets summing up to t_i. Thus, all jobs can be scheduled early if and only if all Partition instances are yes-instances. Note that this corresponds to both objective functions T_{max} and $\sum U_j$ at value 0. Thus, using Corollary 1.4 we obtain:

▶ **Lemma 3.2.** *Both $P_2||T_{max}$ and $P_2||\sum U_j$ have no $\widetilde{O}(n + P_{max} \cdot n^{1-\varepsilon})$-time algorithms, for any $\varepsilon > 0$, assuming ∀∃-SETH.*

3.2.3 $P_2|r_j \geq 0|C_{max}$

Our final dual machine example is the problem of minimizing makespan when release dates are present, the classical $P_2|r_j \geq 0|C_{max}$ problem.

▶ **Lemma 3.3.** *$P_2|r_j \geq 0|C_{max}$ has no $\widetilde{O}(n + P_{max} \cdot n^{1-\varepsilon})$-time algorithm, for any $\varepsilon > 0$, unless ∀∃-SETH is false.*

Proof. Let $(X_1, t_1), \dots, (X_N, t_N)$ be N instances of Partition. For each element $x_{i,j} \in X_i$ we create a job $J_{i,j}$ with processing time $p_{i,j} = x_{i,j}$ and release date $r_{i,j} = \sum_{\ell<i} t_\ell$. Note that there is a schedule for this instance with makespan $\sum_{i=1}^{N} t_i$, where each job is scheduled no earlier than its release date, if and only if each Partition instance is a yes-instance. Also note that the resulting instance has maximum processing time $P_{max} = \max_i t_i$ and total number of jobs $n \leq N \cdot \max_i |X_i|$. As before, using Corollary 1.4 we can now rule out time $\widetilde{O}(n + P_{max} \cdot n^{1-\varepsilon})$, assuming ∀∃-SETH. ◀

3.3 Problems on One Machine

We next consider single machine problems. Obviously, a schedule in this case only needs to specify a starting time S_j for each job J_j, and in case there are no release dates, a schedule can be simply thought of as a permutation of the jobs.

3.3.1 $1||\sum w_j U_j$

One of the most classical single-machine scheduling problems which already appeared in Karp's initial list of 21 NP-complete problems [24] is the problem of minimizing the total weight of tardy jobs. Here each job J_j has a due date d_j and weight w_j, and the goal is to minimize $\sum w_j U_j$.

▶ **Lemma 3.4.** *Assuming ∀∃-SETH, there is no $\widetilde{O}(n + P_{max} \cdot n^{1-\varepsilon})$-time algorithm for $1||\sum w_j U_j$, for any $\varepsilon > 0$.*

Proof. Let $(X_1, t_1), \dots, (X_N, t_N)$ be N instances of Partition. For each $i \in \{1, \dots, N\}$, and for each $x_{i,j} \in X_i$, we create a job $J_{i,j}$ with the following parameters:

- processing time $p_{i,j} = x_{i,j}$,
- weight $w_{i,j} = (N - i + 1) \cdot x_{i,j}$,
- and due date $d_{i,j} = d_i = \sum_{\ell=1}^{i} t_\ell$.

We argue that there is a schedule for all jobs $J_{i,j}$ with total weight of tardy jobs at most $W = \sum_{i=1}^{N} (N - i + 1) \cdot t_i$ if and only if each Partition instance (X_i, t_i) is a yes-instance.

Suppose that each X_i has a subset $Y_i \subseteq X_i$ which sums up to t_i. Let $\mathcal{E}_i = \{J_{i,j} : x_{i,j} \in Y_i\}$ and $\mathcal{T}_i = \{J_{i,j} : x_{i,j} \notin Y_i\}$ for $i \in \{1, \dots, N\}$, and let $\mathcal{E} = \bigcup_i \mathcal{E}_i$ and $\mathcal{T} = \bigcup_i \mathcal{T}_i$. Then any schedule of the form $\mathcal{E}_1, \dots, \mathcal{E}_N, \mathcal{T}$, where the order inside each subset of jobs is arbitrary, schedules all jobs in $\mathcal{E}$ early, and so the total weight of tardy jobs of such a schedule is at most the total weight of $\mathcal{T}$ which is $w(\mathcal{T}) = \sum_i w(\mathcal{T}_i) = \sum_{i=1}^{N} (N - i + 1) \cdot t_i = W$.

Conversely, suppose there is a schedule for the jobs $J_{i,j}$ where the total weight of tardy jobs is at most W. Let $\mathcal{E}_i$ denote the set of early jobs in the schedule with due date d_i, for $i = \{1, \dots, N\}$, and let $\mathcal{E} = \bigcup \mathcal{E}_i$. Then as the total weight of all jobs is $2W$, we have $w(\mathcal{E}) \geq W = \sum_i (N - i + 1) \cdot t_i$. By our construction, this can only happen if we have

$w(\mathcal{E}_i) \geq (N-i+1) \cdot t_i$ for each $i \in \{1,\dots,N\}$, which in turn can only happen if $p(\mathcal{E}_i) \geq t_i$. Since all jobs in each $\mathcal{E}_i$ are early, we have $p(\mathcal{E}_i) \leq t_i$, and so $p(\mathcal{E}_i) = t_i$. It follows that for each $i \in \{1,\dots,N\}$, the set $Y_i = \{x_{i,j} : J_{i,j} \in \mathcal{E}_i\} = \{p_{i,j} : J_{i,j} \in \mathcal{E}_i\}$ sums up to t_i. Thus we have found a solution for each Subset Sum instance (X_i, t_i), and so the lemma follows. ◀

3.3.2 $1|Rej \leq R|\sum U_j$ and $1|Rej \leq R|T_{max}$

In scheduling with rejection problems [38], jobs J_j are allowed not to be scheduled (*i.e.* rejected) at the cost of w_j. Here we consider the case where the total cost of rejected jobs cannot exceed some prespecified bound R. Under this constraint, the $1|Rej \leq R|\sum U_j$ and $1|Rej \leq R|T_{max}$ problems focus on minimizing the number of tardy jobs $\sum U_j$ and the maximum tardiness of any job T_{max}, respectively.

Note that there is a direct reduction from the $1||\sum w_j U_j$ problem to the $1|Rej \leq R|\sum U_j$ and $1|Rej \leq R|T_{max}$ problems: An instance of $1||\sum w_j U_j$ has a schedule with total weight at most W if and only if there are jobs of total weight $R = W$ that can be rejected so that all remaining jobs can be scheduled early. Thus, the lemma below immediately follows from Lemma 3.4 above.

▶ **Lemma 3.5.** *Assuming ∀∃-SETH, both $1|Rej \leq R|\sum U_j$ and $1|Rej \leq R|T_{max}$ have no $\widetilde{O}(n + P_{max} \cdot n^{1-\varepsilon})$-time algorithms, for any $\varepsilon > 0$.*

3.3.3 $1|r_j \geq 0, Rej \leq R|C_{max}$

In this problem, each job J_j has a processing time p_j, a release date r_j, and a weight w_j, and the goal is to find a schedule that rejects jobs with total weight at most R and minimizes the makespan of the remaining non-rejected jobs.

▶ **Lemma 3.6.** *There is no $\widetilde{O}(n + P_{max} \cdot n^{1-\varepsilon})$-time algorithm for $1|r_j \geq 0, Rej \leq R|C_{max}$, for any $\varepsilon > 0$, unless ∀∃-SETH is false.*

Proof. Let $(X_1, t_1), \dots, (X_N, t_N)$ be N instances of Partition. For each $i \in \{1,\dots,N\}$, and for each $x_{i,j} \in X_i$, we create a job $J_{i,j}$ with:

- processing time $p_{i,j} = x_{i,j}$,
- weight $w_{i,j} = i \cdot x_{i,j}$,
- and release date $r_{i,j} = r_i = \sum_{\ell=1}^{i-1} t_\ell$.

We argue that there is a schedule for all jobs $J_{i,j}$ with makespan at most $C = \sum_i t_i$ that rejects jobs with cost at most $R = \sum_i i \cdot t_i$ if and only if each Partition instance (X_i, t_i) is a yes-instance.

Suppose that each X_i has a subset $Y_i \subseteq X_i$ which sums up to t_i. Let $\mathcal{E}_i = \{J_{i,j} : x_{i,j} \in Y_i\}$ and $\mathcal{T}_i = \{J_{i,j} : x_{i,j} \notin Y_i\}$ for $i \in \{1,\dots,N\}$, and let $\mathcal{E} = \bigcup_i \mathcal{E}_i$ and $\mathcal{T} = \bigcup_i \mathcal{T}_i$. Then any schedule of the form $\mathcal{E}_1, \dots, \mathcal{E}_N$, where the jobs in $\mathcal{T}$ are rejected, respects all release dates of jobs in $\mathcal{E}$, and has makespan $C_{max} = \sum_i t_i = C$. Moreover, the total cost of the rejected jobs is $w(\mathcal{T}) = \sum_i w(\mathcal{T}_i) = \sum_{i=1}^N i \cdot t_i = R$.

Conversely, suppose there is schedule for the jobs $J_{i,j}$ that respects all release dates, rejects jobs with weight at most R, and has makespan at most C. Let $\mathcal{E}_i$ denote the set of non-rejected jobs with release date r_i, for $i = \{1,\dots,N\}$, and let $\mathcal{E} = \bigcup \mathcal{E}_i$. Then as the total weight of all jobs is $2R$, we have $w(\mathcal{E}) \leq R = \sum_i i \cdot t_i$. By our construction, this can only happen if we have $w(\mathcal{E}_i) \geq i \cdot t_i$ for each $i \in \{1,\dots,N\}$, which in turn can only happen if $p(\mathcal{E}_i) \geq t_i$. On the other hand, the release date r_{i+1} of jobs in $\mathcal{E}_{i+1}$ can be respected only if $p(\mathcal{E}_i) \leq r_{i+1} = \sum_{\ell=1}^{i} t_\ell$, and so $p(\mathcal{E}_i) = t_i$. It follows that for each $i \in \{1,\dots,N\}$, the set $Y_i = \{x_{i,j} : J_{i,j} \in \mathcal{E}_i\} = \{p_{i,j} : J_{i,j} \in \mathcal{E}_i\}$ sums up to t_i. Thus, we have found a solution for each Subset Sum instance (X_i, t_i), and so the lemma follows. ◀

4 Conclusion

In this paper we considered the AND Subset Sum problem: Given N instances of Subset Sum, determine whether all instances are yes-instances. We showed that the problem is essentially as hard as solving N Subset Sum instances independently, and then used this result to strengthen existing lower bounds for several scheduling problems. Our research is closely related to the question of whether Subset Sum on input (X, t) can be solved in time $\widetilde{O}(\max_{x \in X} x + |X|)$, which is currently a central open problem in the area [4, 16, 17, 27, 33]. Our results answer this question in the negative for several generalizations of Subset Sum. We believe that the line of thought in this paper can provide other results in a similar vein.

Observe that almost all scheduling problems considered in this paper do not have a matching upper-bound of $\widetilde{O}(P_{max} \cdot n)$ to the lower bound constructed in Section 3. The exception is $P_2|\text{level-order}|C_{max}$ which can be solved in time $O(P_{max} \cdot n)$ by using the known $O(P_{max} \cdot n)$-time Subset Sum algorithm [33] (or the faster algorithms given in [7, 27]) on each class of jobs $\mathcal{J}_i$ separately. It would be very interesting to close the gap for other problems listed in Theorem 1.5. This could be done by either devising an $\widetilde{O}(P_{max} \cdot n)$-time algorithm for the problem, or by strengthening our lower bound mechanism.

References

1 Amir Abboud, Karl Bringmann, Danny Hermelin, and Dvir Shabtay. SETH-based lower bounds for subset sum and bicriteria path. *Proc. of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 41–57, 2019.

2 Amir Abboud, Virginia Vassilevska Williams, and Joshua Wang. Approximation and fixed parameter subquadratic algorithms for radius and diameter in sparse graphs. In *Proc. of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 377–391. SIAM, 2016.

3 Bertie Ancona, Monika Henzinger, Liam Roditty, Virginia Vassilevska Williams, and Nicole Wein. Algorithms and hardness for diameter in dynamic graphs. In *Proc. of the 46th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 132 of *LIPIcs*, pages 13:1–13:14, 2019.

4 Kyriakos Axiotis, Arturs Backurs, Ce Jin, Christos Tzamos, and Hongxun Wu. Fast modular subset sum using linear sketching. In *Proc. of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 58–69. SIAM, 2019.

5 Felix A. Behrend. On sets of integers which contain no three terms in arithmetical progression. *Proc. of the National Academy of Sciences*, 32(12):331–332, 1946.

6 Richard E. Bellman. *Dynamic programming.* Princeton University Press, 1957.

7 Karl Bringmann. A near-linear pseudopolynomial time algorithm for subset sum. In *Proc. of of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1073–1084, 2017.

8 Karl Bringmann and Bhaskar Ray Chaudhury. Polyline simplification has cubic complexity. In *Proc. of the 35th International Symposium on Computational Geometry (SoCG)*, pages 18:1–18:16, 2019.

9 Peter Brucker. *Scheduling Algorithms.* Springer, 2006.

10 Marco L. Carmosino, Jiawei Gao, Russell Impagliazzo, Ivan Mihajlin, Ramamohan Paturi, and Stefan Schneider. Nondeterministic extensions of the Strong Exponential Time Hypothesis and consequences for non-reducibility. In *Proc. of the 7th ACM Conference on Innovations in Theoretical Computer Science (ITCS)*, pages 261–270, 2016.

11 T.C. Edwin Cheng, Yakov M. Shafransky, and Chi To Ng. An alternative approach for proving the NP-hardness of optimization problems. *European Journal of Operational Research*, 248(1):52–58, 2016.

12 Marek Cygan, Holger Dell, Daniel Lokshtanov, Dániel Marx, Jesper Nederlof, Yoshio Okamoto, Ramamohan Paturi, Saket Saurabh, and Magnus Wahlström. On problems as hard as CNF-SAT. *ACM Transactions on Algorithms*, 12(3):41, 2016.

13 Erik D. Demaine, Andrea Lincoln, Quanquan C. Liu, Jayson Lynch, and Virginia Vassilevska Williams. Fine-grained I/O complexity via reductions: New lower bounds, faster algorithms, and a time hierarchy. In *Proc. of the 9th Innovations in Theoretical Computer Science Conference (ITCS)*, volume 94 of *LIPIcs*, pages 34:1–34:23, 2018.

14 Danny Dolev and Manfred K. Warmuth. Profile scheduling of opposing forests and level orders. *SIAM Journal on Algebraic and Discrete Methods*, 6(4):665–687, 1985.

15 Jianzhong Du and Joseph Y.-T. Leung. Minimizing total tardiness on one machine is NP-hard. *Mathematics of Operations Research*, 15(3):483–495, 1990.

16 Friedrich Eisenbrand and Robert Weismantel. Proximity results and faster algorithms for integer programming using the Steinitz lemma. In *Proc. of the 29th Annual ACM-SIAM Symposium On Discrete Algorithms (SODA)*, pages 808–816, 2018.

17 Zvi Galil and Oded Margalit. An almost linear-time algorithm for the dense subset-sum problem. *SIAM Journal on Computing*, 20(6):1157–1189, 1991.

18 Jiawei Gao, Russell Impagliazzo, Antonina Kolokolova, and Ryan Williams. Completeness for first-order properties on sparse structures with algorithmic applications. *ACM Transactions on Algorithms*, 15(2):1–35, 2018.

19 Rosario Giuseppe Garroppo, Stefano Giordano, and Luca Tavanti. A survey on multi-constrained optimal path computation: Exact and approximate algorithms. *Computer Networks*, 54(17):3081–3107, 2010.

20 Ronald Graham, Eugene Lawler, Jan K. Lenstra, and Alexander R. Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

21 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001.

22 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001.

23 Ce Jin and Hongxun Wun. A simple near-linear pseudopolynomial time randomized algorithm for subset sum. In *Proc. of the 2nd Symposium On Simplicity in Algorithms (SOSA)*, pages 17:1–17:6, 2018.

24 Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Springer, 1972.

25 Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack problems.* Springer, 2004.

26 Ketan Khowala, Ahmet B. Keha, and John Fowler. A comparison of different formulations for the non-preemptive single machine total weighted tardiness scheduling problem. In *Proc. of the 2nd Multidisciplinary International Conference on Scheduling: Theory and Application (MISTA)*, pages 643–651, 2008.

27 Konstantinos Koiliaris and Chao Xu. A faster pseudopolynomial time algorithm for subset sum. *ACM Transaction on Algorithms*, 15(3):40:1–40:20, 2019.

28 Mikhail Y. Kovalyov and Erwin Pesch. A generic approach to proving NP-hardness of partition type problems. *Discrete Applied Mathematics*, 158(17):1908–1912, 2010.

29 Eugene L. Lawler and James M. Moore. A functional equation and its application to resource allocation and sequencing problems. *Management Science*, 16(1):77–84, 1969.

30 Silvano Martello and Paolo Toth. *Knapsack problems: algorithms and computer implementations.* John Wiley & Sons, Inc., 1990.

31 Yunpeng Pan and Leyuan Shi. On the equivalence of the max-min transportation lower bound and the time-indexed lower bound for single-machine scheduling problems. *Mathematical Programming*, 110(3):543–559, 2007.

32 Michael Pinedo. *Scheduling: Theory, Algorithms and Systems.* Prentice-Hall, 2008.

33 David Pisinger. Linear time algorithms for knapsack problems with bounded weights. *Journal of Algorithms*, 32:1–14, 1999.

34 Gary L. Ragatz. A branch-and-bound method for minimum tardiness sequencing on a single processor with sequence dependent setup times. In *Proc. of the 24th Annual Meeting of the Decision Sciences Institute (DSI)*, page 1375–1377, 1993.

35 Dhruv Rohatgi. Conditional hardness of earth mover distance. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, volume 145 of *LIPIcs*, pages 12:1–12:17, 2019.

36 Michael H. Rothkopf. Scheduling independent tasks on parallel processors. *Management Science*, 12(5):437–447, 1966.

37 Rahul Santhanam and Ryan Williams. Beating exhaustive search for quantified boolean formulas and connections to circuit complexity. In *Proc. of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 231–241. SIAM, 2014.

38 Dvir Shabtay, Nufar Gaspar, and Moshe Kaspi. A survey on offline scheduling with rejection. *Journal of Scheduling*, 16(1):3–28, 2013.

39 Tianyu Wang and Odile Bellenguez. The complexity of parallel machine scheduling of unit-processing-time jobs under level-order precedence constraints. *Journal of Scheduling*, pages 263–269, January 2019.

40 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science*, 348(2-3):357–365, 2005.

41 Ossama Younis and Sonia Fahmy. Constraint-based routing in the internet: Basic principles and recent research. *IEEE Communications Surveys and Tutorials*, 5(1):2–13, 2003.

On the Fine-Grained Complexity of Parity Problems

Amir Abboud
IBM Almaden Research Center, San Jose, CA, USA
amir.abboud@ibm.com

Shon Feller
University of Haifa, Israel
shonfeller1@gmail.com

Oren Weimann
University of Haifa, Israel
oren@cs.haifa.ac.il

Abstract

We consider the *parity* variants of basic problems studied in fine-grained complexity. We show that finding the exact solution is just as hard as finding its parity (i.e. if the solution is even or odd) for a large number of classical problems, including All-Pairs Shortest Paths (APSP), Diameter, Radius, Median, Second Shortest Path, Maximum Consecutive Subsums, Min-Plus Convolution, and 0/1-Knapsack.

A direct reduction from a problem to its parity version is often difficult to design. Instead, we revisit the existing hardness reductions and tailor them in a problem-specific way to the parity version. Nearly all reductions from APSP in the literature proceed via the (subcubic-equivalent but simpler) Negative Weight Triangle (NWT) problem. Our new modified reductions also start from NWT or a non-standard parity variant of it. We are not able to establish a subcubic-equivalence with the more natural *parity counting* variant of NWT, where we ask if the number of negative triangles is even or odd. Perhaps surprisingly, we justify this by designing a reduction from the seemingly-harder Zero Weight Triangle problem, showing that parity is (conditionally) strictly harder than decision for NWT.

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

Keywords and phrases All-pairs shortest paths, Fine-grained complexity, Diameter, Distance product, Min-plus convolution, Parity problems

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.5

Category Track A: Algorithms, Complexity and Games

Related Version A full version of the paper is available at `https://arxiv.org/abs/2002.07415`.

Funding *Shon Feller*: Supported in part by Israel Science Foundation grant 592/17.
Oren Weimann: Supported in part by Israel Science Foundation grant 592/17.

1 Introduction

The blossoming field of fine-grained complexity is concerned with understanding the time complexity of basic computational problems in a precise way. The main approach is to hypothesize the hardness of a few core problems and then reduce them to a large number of other problems, establishing tight conditional lower bounds for them. A cornerstone finding in this field is that there is a class of more than ten problems that are all *subcubic equivalent* to the All-Pairs Shortest Paths (APSP) problem, in the sense that if any of them can be solved in truly subcubic $O(n^{3-\varepsilon})$ time (for some $\varepsilon > 0$) then all of them can. Most of the problems in this "APSP-class" are related to distance computations in graphs such as computing the radius of the graph or deciding if the graph contains a negative weight triangle (NWT). In this work, we investigate the fine-grained complexity of the natural

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 5; pp. 5:1–5:19
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

parity versions of such problems: are they easier, harder, or do they have the same time complexity? Depending on the problem, the natural parity version could have a different type; let us consider the two main types that will appear in this paper and illustrate them with examples.

- **Parity Computation:** The Radius-Parity problem asks whether the radius of the graph is even or odd. Similarly, the APSP-Parity problem asks to compute, for each pair of nodes, whether the distance between them is even or odd. This type is often natural for optimization problems.
- **Parity Counting:** The NWT-Parity problem asks if the number of negative triangles in the graph is even or odd, or equivalently, it asks to count the number of negative triangles modulo 2. Similarly, SAT-Parity asks if the number of satisfying assignments to a given formula is even or odd. This type is often natural for decision problems where we are looking for a solution satisfying a certain property[1].

The parity computation versions are clearly no harder than the original problem: If we know the radius exactly we also know its parity (if it is even or odd). In fact, sometimes knowing the parity can be much easier than computing the entire answer. For instance, while computing the maximum number of nodes in a matching requires super-linear time, knowing the parity is trivial (it is always 0). On the other hand, parity counting versions can make the problem much harder. A famous example is 2-SAT: the decision version takes linear time but the parity version is probably not in P [42]. Thus, in general, the natural parity version could be easier or harder than the original problem.

Various questions related to parity arose naturally in different contexts in computer science throughout the years. For instance, the SAT-Parity problem played a key role in the proof of Toda's theorem [40], which is one of the earliest and most fundamental results in the large body of works on counting complexity [23]. There, parity counting problems are extensively studied, being of intermediate complexity between the decision problems and the counting problems (see e.g. [5, 8, 34, 41–44]). The first type of problems are less studied in terms of worst-case complexity but are morally related to hard-core predicates in cryptography [45] where it is desirable that the parity (least significant bit) of a function is hard to guess. The motivation for our work is twofold: (1) many of the parity versions are interesting on their own and we would like to know their complexity, and (2) this investigation could lead to a deeper understanding of the structure among the original (non-parity) versions.

1.1 Our Results

1.1.1 The APSP Class

Our first set of results concern the APSP equivalence class. We have gone through the problems in this class from the works of [3, 20, 51] and tried to classify the complexity of their parity versions. Our first theorem shows that, with the notable exception of Negative Weight Triangle (NWT), all the parity versions are subcubic-equivalent to APSP and therefore also to the original (non-parity) problems. These problems and their parity versions are listed and defined in Table 1 together with our results for each of them and where they appear in the paper.

[1] The parity counting version could be viewed simply as the parity computation version of the counting version of the problem, so the first type could be considered the "real" parity version. However, parity counting is widely referred to as the parity version in the literature.

Table 1 The APSP class: problem definitions, known results, and our results (in bold). We denote subcubic equivalent as SE and as SER when it is under a randomized reduction. The first seven problems output a single value and their parity version computes the parity of this value. The next three problems output multiple values and their parity version computes the parity of every value. The last two problems are the only parity counting problems, in which we distinguish between the parity version (asking for the parity of the number of such triangles) and the vertex parity version (asking for the parity of the number of vertices that belong to such triangles).

Problem	Definition	Complexity
Median	$\min_u \sum_v d(u,v)$	SE to APSP [3], **Parity is SE to APSP (Sec. 2)**
Wiener Index	$\sum_u \sum_v d(u,v)$	SE to APSP [20], **Parity is SER to APSP (Sec. 3.2, 3.3)**
Radius	$\min_u \max_v d(u,v)$	SE to APSP [3], **Parity is SE to APSP (Sec. 8)**
Sum of Eccentricities	$\sum_u \max_v d(u,v)$	**SE to APSP, (Sec. 7),** **Parity is SER to APSP (Sec. 3.4)**
Integer Betweenness Centrality	find the number of vertices pairs with a shortest path passing through a given vertex x	SE to APSP [3], $(1+\varepsilon)$-approx. is SER to Diameter [3] **Parity is SER to APSP (Sec. 3.5, 3.6)**
Second Shortest Path	given vertices s,t, find the length of the second shortest s-to-t path	SE to APSP [51], **Parity is SE to APSP (Full version)**
Maximum Subarray	given a matrix, find the maximum total value in a submatrix	SE to APSP [6, 39], **Parity is SE to APSP (Full version)**
APSP	Compute all distances $d(u,v)$	**Parity is SE to APSP (Sec. 4)**
Min-Plus Matrix Multiplication	given $n \times n$ matrices A and B, compute the matrix C where $C[i,j] = \min_k\{A[i,k] + B[k,j]\}$	SE to APSP (folklore), **Parity is SE to APSP (Sec. 4)**
Replacement Paths	for every edge e on a shortest s-to-t path, find the length of the shortest s-to-t path that avoids e	SE to APSP [51], **Parity is SE to APSP (Full version)**
Negative Weight Triangle	determine if there is a triangle of total negative weight	SE to APSP [51], $(1+\epsilon)$-approx. Counting is SER to APSP [19]. **Randomized reductions from APSP and 3SUM to Parity and Counting (Sec. 5.1, 5.2), Vertex Parity is SER to APSP (Sec. 3.1)**
Zero Weight Triangle	determine if there is a triangle of total zero weight	Reduction from APSP and 3SUM [50], **Randomized reduction to Parity and Vertex Parity (Sec. 3.1, 5.2)**

▶ **Theorem 1.** *The following problems are subcubic-equivalent:*

- *All-Pairs Shortest Paths and its parity computation,*
- *Min-Plus Matrix Multiplication and its parity computation,*
- *Radius and its parity computation,*
- *Median and its parity computation,*
- *Wiener Index and its parity computation,*
- *Replacement Paths and its parity computation,*
- *Second Shortest Path and its parity computation,*
- *Vertex in Negative Weight Triangle and its parity computation,*

- *Integer Betweenness Centrality and its parity computation,*
- *Maximum Subarray and its parity computation,*
- *Sum of Eccentricities*[2] *and its parity computation.*

This adds more than ten natural problems to the APSP-equivalence class. For all problems in Theorem 1, the reduction from the parity version to the original problem is straightforward (since they are parity computation rather than parity counting problems), while the reduction in the other direction is not. For instance, it is not at all clear how to establish the hardness of Median-Parity by reducing from Median to it. Instead, we find it much more convenient to start from NWT which is the canonical APSP-complete problem and the starting point for nearly all APSP-hardness reductions.

Some of our results take the known reductions from NWT and modify them to establish the hardness of the parity versions, e.g. for Median-Parity. Notably, reductions of this kind are deterministic. For other parity problems such as Wiener-Index it is more convenient to reduce from a parity version of NWT. However, (the most natural) NWT-Parity is a parity counting problem which makes it seem harder than NWT and therefore inappropriate as a starting point for reductions. Instead, we identify a different variant that we call NWT-Vertex-Parity (asking if the number of vertices that belong to a negative triangle is even or odd) which turns out to be subcubic-equivalent to NWT and a very useful intermediate problem. Reductions of this kind seem to require randomization.

Finally, we investigate the intriguing NWT-Parity problem. This is the modulo 2 version of the NWT-counting problem (asking for the number of negative triangles) that was recently studied by Dell and Lapinskas [19] in their work on the fine-grained complexity of approximate counting. With standard subsampling techniques, one can show that NWT reduces to NWT-Parity. But are they subcubic-equivalent? We show that such an equivalence would imply breakthroughs in fine-grained complexity, therefore suggesting that the parity version is strictly harder. Our next theorem shows that NWT-Parity can solve a problem that is considered strictly harder than APSP: the problem of deciding whether a graph has a Zero Weight Triangle (ZWT). As discussed below, if the same reduction can be shown between the original (non-parity) problems it would be a major breakthrough.

▶ **Theorem 2.** *There is a deterministic subcubic-reduction from the Zero Weight Triangle Parity problem to the Negative Weight Triangle Parity problem.*

The ZWT problem is considered one of the "hardest" n^3-problems since a subcubic algorithm for it would refute *two* of the main conjectures in fine-grained complexity: it would give a subcubic algorithm for APSP and a subquadratic algorithm for 3SUM [35, 50, 51]. The 3SUM Conjecture states that we cannot decide in truly subquadratic $O(n^{2-\varepsilon})$ time if among a set of n numbers there are three that sum to zero. The class of problems that are 3SUM-hard contains dozens of problems mostly from computational geometry (see [24, 28] for a partial list), but also in other domains, e.g. [4, 29, 35]. One of the central open questions in the field is whether the APSP class and the 3SUM class can be unified; in particular, whether APSP is 3SUM-hard. One way to prove this is to reduce ZWT to APSP, and our result shows that NWT-Parity to NWT suffices:

▶ **Corollary 3.** *If the Negative Weight Triangle Parity problem is subcubic-equivalent to the Negative Weight Triangle problem, then APSP is 3SUM-hard.*

[2] This natural problem was not considered before to our knowledge, but it is closely related to Median, Radius, and the other distance computation problems.

A more quantitative reason for supposing that ZWT is harder than NWT is that their current upper bounds, while all mildly-subcubic, are significantly far apart. All the problems in the APSP equivalence class can be solved in $n^3/2^{\Omega(\sqrt{\log n})}$ time [47], which is faster than $O(n^3/\log^c n)$ for all $c > 0$. For ZWT, on the other hand, nothing better than $O(n^3/\log^c n)$ is known for a small $c \leq 2$, and even small improvements would lead to a faster mildly subquadratic algorithm for 3SUM beating the current fastest $O(n^2(\log\log n)^2/\log n)$ [26]. Dell and Lapinskas [19] achieve an $n^3/2^{\Omega(\sqrt{\log n})}$ upper bound for the approximate counting version of NWT but not for exact counting. We show that even for the parity version such a result has breakthrough consequences for 3SUM. For NWT, this (conditionally) separates the counting and parity versions from the decision and approximate counting versions.

Table 2 The other (non-APSP) problems. We denote subcubic (subquadratic) equivalent as SE (SQE) and as SER (SQER) when it is under a randomized reduction. As before, the parity version of problems that output a single (multiple) value(s) computes the parity of this value (all these values). The last problems is the only parity counting problem, in which the parity version asks for the parity of the number of vertices that do not belong to any negative triangle.

Problem	**Definition**	**Complexity**
Diameter	$\max_u \max_v d(u,v)$	**Parity is SE to Diameter (Sec. 8)**
Maximum Row Sum	$\max_u \sum_v d(u,v)$	**Reduction from Co-Negative Triangle to Parity (Full version)**
Reach Centrality	compute the maximum distance between a given vertex x and the closest endpoint of any shortest path passing through x	SE to Diameter [3], **Parity is SE to Diameter (Full version)**
0/1-Knapsack	given items (w_i, v_i) and a weight t, find a subset I that maximizes $\sum_{i\in I} v_i$ subject to $\sum_{i\in I} w_i \leq t$	SQER to Min-Plus Convolution, variants are SQE to Min-Plus Convolution [17,30], **Parity is SQER to Min-Plus Convolution, variants are SQE to Min-Plus Convolution (Sec. 6)**
Tree Sparsity	given a node-weighted tree, find the maximum weight of a subtree of size k	SQE to Min-Plus Convolution [7,17], **Parity is SQE to Min-Plus Convolution (Full version)**
Min-Plus Convolution	given n-length vectors A and B, compute the vector C where $C[k] = \min_{i+j=k}\{A[i]+B[j]\}$	Reduction to APSP and 3SUM [13,17], **SQE to Parity (Sec. 4.2)**
Maximum Consecutive Subsums	given an n-length vector A, compute the vector B where $B[k] = \max_i\{\sum_{j=0}^{k-1} A[i+j]\}$	SQE to Min-Plus Convolution [17,31], **Parity is SQE to Min-Plus Convolution (Sec. 4.3)**
Co-Negative Triangle	find a vertex that does not belong to any negative triangle	Reduction to Diameter [10], **Reduction to Maximum Row Sum Parity (Full version)**

1.1.2 Other Classes

In our second set of results we ask whether parity computation problems are as hard also for problems that are outside the APSP class. We have gone through other fine-grained complexity results from the works of [7,17,30,31] and tried to establish the same results for the parity versions. All problems we consider are defined in Table 2 together with our results and where they appear in the paper. The general message is that, in all cases we considered, the same hardness reductions (if modified carefully) can establish the hardness of the (seemingly easier) parity version as well. We mention a few concrete examples.

In the context of distance computations in graphs, the central open question is whether the Diameter problem is subcubic equivalent to APSP. Meanwhile, Diameter has its own (smaller) equivalence class which includes problems such as Reach Centrality [3]. We prove that Diameter-Parity and Reach Centrality-Parity are subcubic equivalent to Diameter.

Another interesting problem in fine-grained complexity whose importance is rapidly increasing is the Min-Plus Convolution problem [17]. The naïve algorithm for this problem runs in $O(n^2)$ time, and a truly subquadratic $O(n^{2-\varepsilon})$ algorithm is conjectured to be impossible. This problem is one of the easiest n^2 problems since it can be reduced to both APSP (i.e. a subcubic algorithm for APSP yields a subquadratic algorithm for Min-Plus Convolution) and to 3SUM (an opposing situation to that of ZWT). This means that all of the APSP and 3SUM lower bounds can be based on this conjecture, but also that Min-Plus Convolution is unlikely to be equivalent to either of them (as it would imply a unification of the classes). Recently, a few other problems have been shown to be harder, e.g. [2], or subquadratic-equivalent to it, e.g. Maximum consecutive subsums [17, 31], 0/1-Knapsack [17, 30], and a $(1+\varepsilon)$-approximation for Subset Sum [14]. We prove that these equivalences hold for the parity versions as well (except the latter problem for which we did not find a natural parity version). Our reduction from the Maximum consecutive subsums problem to its parity version in Section 4.3 is quite involved and it uses specific properties of the addition operator. One can obtain such a reduction indirectly and more easily via Min-Plus Convolution, however, we believe that our reduction gives more insight into the problem and into the usage of the addition operator.

▶ **Theorem 4.** *The following problems are subquadratic-equivalent:*
- *Min-Plus Convolution and its parity computation,*
- *Maximum Consecutive Subsums and its parity computation,*
- 0/1-*Knapsack and its parity computation,*
- *Tree Sparsity and its parity computation.*

1.2 Related Work

While parity counting problems are extensively studied in classical complexity theory, the parity computation problems seem to have received less attention. In many cases, the standard NP-hardness reduction from SAT gives instances in which the solution is always either k or $k-1$, which directly implies the NP-hardness of the parity version as well. Some of the results in fine-grained complexity also have this property. For example, the quadratic hardness result for Diameter in sparse graphs [36] shows that it is hard to distinguish diameter 2 from 3 and immediately gives the same lower bound for parity. However, for many other problems, such as the ones we consider, this is not the case and a careful problem-specific treatment is required.

Theorem 2 and its corollaries conditionally separate NWT from its parity and counting versions. Such separations are famously known in classical complexity, e.g. for 2-SAT [42]. In fine-grained complexity, a (conditional) separation for a variant of the Orthogonal Vectors problem between near-linear time decision [48] and quadratic time exact counting [46] was recently achieved. Notably, the approximate counting version is also in near-linear time [19] and the parity version is open.

The parity counting version of the Strong Exponential Time Hypothesis was studied in a seminal paper on the fine-grained complexity of NP-hard problems [16]. The central question left open in that paper (and is still wide open, see [1]) is whether SAT can be reduced to Set-Cover in a fine-grained way; interestingly, the authors have shown such a reduction for the parity counting versions.

Exact and approximate counting problems have received a lot of attention in parameterized [15, 22] and fine-grained complexity [18]. In a recent development, the k-Clique counting problem was shown to have worst-case to average-case reductions [9, 25]. It is likely that our result for NWT-Parity can be extended to Negative Weight k Clique Parity showing that it is as hard as Zero Weight k Clique. The decision version of the latter problem was used as the basis for public-key cryptography schemes [32].

Due to the large amount of works on APSP-hardness and equivalences we did not manage to exhaustively enumerate all of them and investigate the complexity of the parity versions, e.g. for problems on stochastic context-free grammars [38] or dynamic graphs [4, 37]. Still, we expect that the ideas in this work can be extended to show the hardness of those parity computation problems as well.

Besides parity computation and parity counting, there is a third natural type of parity problems where we take a problem and replace one of the operations (e.g. summation) with a parity. For example, the 3XOR problem is a variant of 3SUM where we are given a set of n binary vectors of size $O(\log n)$ and are asked if there is a triple whose bit-wise XOR is all zero. 3XOR is the subject of study of several papers [11, 12, 21] and it seems just as hard as 3SUM but a reduction in either direction has been elusive [27].

1.3 Preliminaries

In all graph problems we assume that the graphs have n nodes and $O(n^2)$ edges. In all the weighted problems we consider, we assume the weights are integers in $[-M, M]$ (and generally it is assumed that $M = poly(n)$).

Intuitively, a fine-grained reduction [49, 51] from problem A with current upper bound $O(n^a)$ to problem B with current upper bound $O(n^b)$ is a Turing-reduction proving that if B is solvable in time $O(n^{b-\varepsilon})$, for some $\varepsilon > 0$, then A is solvable in time $O(n^{a-\varepsilon'})$, for some $\varepsilon' > 0$. More formally, an (a, b)-fine-grained reduction from A to B is a (possibly randomized) algorithm solving A on instances of size n using t calls to an oracle for B on instances of sizes $n_1, \dots, n_t$, such that for all $\varepsilon > 0$: $\sum_{i=1}^{t} (n_i)^{b-\varepsilon} \leq n^{a-\varepsilon'}$ for some $\varepsilon' > 0$. In this paper, unless otherwise stated, we assume that the reduction is randomized. A $(3, 3)$-fine-grained reduction is called a subcubic-reduction and two problems are called subcubic-equivalent if there are subcubic-reductions in both ways. Similarly, two problems are subquadratic-equivalent if there are $(2, 2)$-fine-grained reductions between them in both ways.

2 APSP to Median Parity

In this section, we show a subcubic reduction from the Negative Weight Triangle problem (hence also from APSP [51]) on a directed graph G with integral edge weights in $[-M, M]$ to Median Parity. We first describe the reduction of [3] from Negative Weight Triangle to Median and then modify it to become a reduction to Median Parity.

2.1 Negative Weight Triangle to Median [3]

The instance G' to the Median problem (illustrated in Figure 1) is an undirected graph constructed as follows. First, for any two (not necessarily different) vertices u, v if there is no edge (u, v) in G then we add an edge (u, v) of weight $w(u, v) = 4M$ to G (this will not form a new negative triangle). Each vertex u of G has five copies in G' denoted $u_A, u_B, u_{B'}, u_C, u_{C'}$. Let H be a sufficiently large number (say $H = 100M$). For any two (not necessarily different) vertices u, v of G we add the following edges to G': (u_A, v_B) of weight $3H + w(u,v)$, $(u_A, v_{B'})$ of weight $3H - w(u,v)$, (u_A, v_C) of weight $6H - w(v,u)$[3], $(u_A, v_{C'})$ of weight $3H + w(v,u)$[3], (u_A, v_A) of weight H, and (u_B, v_C) of weight $3H + w(u,v)$.

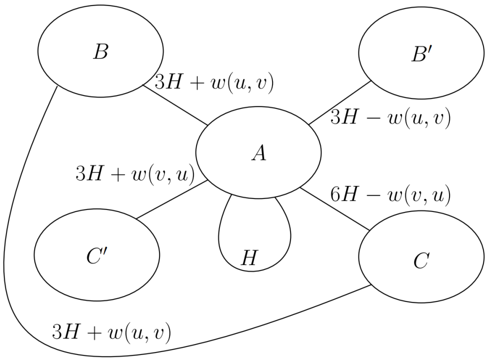


Figure 1 The graph G' in the reduction from Negative Weight Triangle to Median.

▶ **Lemma 5** ([3]). *G does not contain a negative triangle iff the median of G' is $(16n-1)H$.*

Proof. Consider first a vertex u_X with $X \neq A$. We claim that the sum of distances $\sum_{v \in V(G')} d_{G'}(u_X, v)$ is at least $(19n - 5)H$. To see this, first observe that the sum is minimized when $X = B$. This is because shortest paths from vertices in B' and C' go through A, and because every C-to-A distance is larger than any B-to-A distance by at least H. We therefore focus on $X = B$: The distance from u_B to u_B is zero and the distance from u_B to v_B (for $v \neq u$) is at least $5H$ (since H is large enough, $3H + w(u,t) + 3H + w(t,v) > 5H$), so the sum of distances from u_B to all vertices of B is $5H(n-1)$. Similarly, for every vertex v of G, the distances from u_B to $v_A, v_{B'}, v_C, v_{C'}$ are at least $2H, 5H, 2H, 5H$ respectively. Overall, the sum of distances from u_B is at least $(5n-5)H + 2nH + 5nH + 2nH + 5nH = (19n-5)H$.

Next consider a vertex u_A. Let $F(u,v) = \min\{0, \min_{t \in V(G)}\{w(v,u) + w(u,t) + w(t,v)\}\}$. Observe that $F(u,v) = 0$ if the edge (v,u) is not part of any negative triangle in G, and $F(u,v) < 0$ otherwise. We claim that the sum of distances $\sum_{v \in V(G')} d_{G'}(u_A, v)$ is exactly $(16n-1)H + \sum_{v \in V(G)} F(u,v)$. To see this, consider the distances from u_A. Distances to v_A, v_B, and $v_{B'}$ are H (for $v \neq u$), $3H + w(u,v)$, and $3H - w(u,v)$ respectively. Over all such vertices the sum of the distances is therefore $(n-1)H + 6nH = (7n-1)H$. The distance to $v_{C'}$ is $3H + w(v,u)$ and the distance to v_C is the minimum between $6H - w(v,u)$ (using a single edge) and $3H + w(u,t) + 3H + w(t,v)$ for some vertex t (using two edges, through some t_B). Summing those two distances together, we get $9H + w(v,u) + \min_{t \in V(G)}\{-w(v,u), w(u,t) + w(t,v)\} = 9H + F(u,v)$. Overall, we get that $\sum_{v \in V(G')} d_{G'}(u_A, v) = (16n-1)H + \sum_{v \in V(G)} F(u,v)$ as claimed. This implies that the median vertex must come from A and that the median value is $(16n-1)H$ iff every $F(u,v) = 0$ (i.e. G does not contain a negative triangle). ◀

2.2 Negative Weight Triangle to Median Parity

We now modify the above reduction so that it reduces to MEDIAN PARITY instead of MEDIAN. We assume n is odd (otherwise add an isolated vertex to G). Let *Med* be the value of the median of G'. We multiply all the edge weights of G' by $4n$ (notice that this multiplies the

[3] Notice the different order of the vertices.

median value Med by $4n$). We do this in order to make sure that small changes in edge weights would not change any shortest path, and also to make sure that subtracting n from distance sums would not change the median vertex.

We show how to find the median of G' using $O(\log n)$ executions of Median Parity: Given a set of vertices $T \subseteq A$ (initialized to be A), pick an arbitrary subset S of T of half of its size. Temporarily (i.e restore weights at the end of the iteration) subtract 1 from all the S-to-B and S-to-C edges and add 1 to all the S-to-B' edges. Now solve Median Parity on G'. If the median value is odd, set $T \leftarrow S$. If the median value is even, set $T \leftarrow T/S$. We continue recursively for $O(\log n)$ steps until T contains a single vertex. We then check if this vertex participates in a negative triangle in G.

For the correctness of the above procedure, inductively assume that T contains the median vertex of G'. Notice that the temporary changes to the edge weights do not change the identity of shortest paths in G', only their value. In particular, the sum of distances from every vertex $u_A \in S$ decreases exactly by n, and for any vertex $u_A \in A \backslash S$ the sum remains the same. To see this, consider first a vertex $u_A \in S$. The sum of its distances to any v_B and $v_{B'}$ remains the same (one is larger by 1 and one is smaller by 1) and its distance to v_C is decreased by 1 (recall that the shortest path is either the direct edge (u_A, v_C) or two edges $(u_A, t_B), (t_B, v_C)$). Therefore, the sum of distances from $u_A \in S$ to all vertices of G' decreases by exactly n. As for vertices in $u_A \in A \backslash S$, we do not change weights of edges in their shortest paths so their sum of distances is unchanged.

If the median is from S, then its sum of distances in G' was originally Med. Since we multiplied the edge weights by $4n$ and subtracted n from its sum, the median value is now $4n \cdot Med - n$. This value is odd and indeed we set $T \leftarrow S$. If on the other hand the median is not from S, then the sum of distances from any vertex of S was originally at least $Med + 1$, and is therefore now at least $4n \cdot (Med + 1) - n$. This value is strictly bigger than the value $4n \cdot Med$ of the median. The value $4n \cdot Med$ is even and indeed we set $T \leftarrow T/S$.

3 Negative Triangle Vertex Parity

In this section, we show that Negative Triangle Vertex Parity (finding if the number of vertices that belong to a negative triangle is odd or even) is subcubic equivalent to APSP under randomized reductions. We then use Negative Triangle Vertex Parity in order to establish a subcubic equivalence with the Parity versions of Wiener Index, Sum of Eccentricities, and Integer Betweenness Centrality.

3.1 APSP to Negative Triangle Vertex Parity

We now show a probabilistic (one side error) reduction from Negative Weight Triangle (NWT) to Negative Triangle Vertex Parity (NTVP). We assume without loss of generality that the NWT instance G is undirected (otherwise, we turn G into an undirected tripartite graph (by adding vertices v_1, v_2, v_3 for every v in $V(G)$ and edges $(u_1, v_2), (u_2, v_3), (u_3, v_1)$ for every (u, v) in $E(G)$) with the property that there is a negative triangle in G iff there is a negative triangle in the tripartite graph). The NTVP instance G' is also undirected and is created as follows: Choose $V_1 \subseteq V(G)$ uniformly, and let $\overline{V} = V(G) \backslash V_1$. For every $u_1 \in V_1$ we add a vertex u_2, and let the union of all u_2 vertices be V_2. For every edge (u_1, v_1) in $V_1 \times V_1$ we add the edge (u_2, v_2) and for every edge (u_1, v) in $V_1 \times \overline{V}$ we add the edge (u_2, v). Notice that the graph induced by $\overline{V} \cup V_2$ is G, and the same for $\overline{V} \cup V_1$.

Since there are no edges between V_1 and V_2, every triangle is either in $\overline{V} \cup V_1$ or in $\overline{V} \cup V_2$. Furthermore, for every vertex $u_1 \in V_1$, if u_1 belongs to a negative triangle in G then both u_1 and u_2 belong to negative triangles in G', thus contributing 2 (even) to the parity NTVP(G')

of the number of vertices that belong to a negative triangle in G'. Therefore, vertices in V_1 do not affect the parity NTVP(G'). In other words, NTVP(G') is the parity of vertices in $\overline{V}$ with a negative triangle. If G contains a negative triangle, then the probability of odd NTVP(G') is exactly $1/2$ (since each vertex with a negative triangle is chosen to be in $\overline{V}$ with probability $1/2$). If G does not contain a negative triangle, then the probability of even NTVP(G') is exactly 1. By repeating this process $O(\log n)$ times we can amplify the probability of success to $1 - 1/n^c$ for any constant c.

We remark that the above reduction can also be used to reduce Zero Weight Triangle to its vertex parity version.

3.2 Negative Triangle Vertex Parity to Wiener Index Parity (Directed)

We handle the directed case here and the undirected case in Section 3.3. Assume n is even by adding a vertex with no negative triangles, if needed. The reduction graph G' is constructed as in [3, 51] (see Figure 2): Each vertex u of G has five copies in G' denoted u_S, u_A, u_B, u_C, u_D. Let H be a sufficiently large *even* number (say $H = 100M$). For every $(X, Y) \in \{(A, B), (B, C), (C, D)\}$ and $u, v \in V(G)$, add the edge (u_X, v_Y) with weight $2H + 2w(u, v)$. For every $u \neq v \in V(G)$, add an edge (u_A, v_D) with weight $5H$. For every $u \in V(G)$, we add the edge (u_S, u_A) with weight $H + 1$ and the edge (u_S, u_D) with weight $7H$. Turn G' into a clique by replacing any missing edge with an edge of weight $16H$.

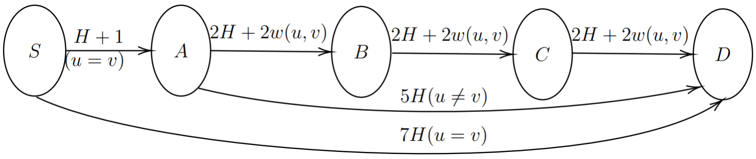


Figure 2 The graph G' in the reduction from Negative Triangle Vertex Parity to Wiener Index Parity. Edges of weight $16H$ are absent.

We now show that the Negative Triangle Vertex Parity of G (NTVP(G)) is equal to the Wiener Index Parity of G' (WIP(G')). Since H is an even number, the only edges in G' that have an odd length are the (u_S, u_A) edges of length $H + 1$. Therefore, the parity WIP(G') is determined by the S to $A \cup B \cup C \cup D$ distances.

First observe that the sum of distances from S to $A \cup B \cup C$ is even. This is because for any vertex u_S in S the following shortest paths consist of a single edge of weight $H + 1$: the u_S-to-v_A (for $v = u$) path, the u_S-to-v_B (for $v = u$ or $v \neq u$) paths, and the u_S-to-v_C (for $v = u$ or $v \neq u$) paths. Thus, the total number of odd edges in the sum of distances from S to $A \cup B \cup C$ is $n(2n + 1)$, which is even since n is even.

It remains to consider the distances from S to D. For $u \neq v$, $d_{G'}(u_S, v_D) = 6H + 1$, and the sum of such distances is $n(n - 1)(6H + 1)$ (even). We are left with the sum of distances $d_{G'}(u_S, u_D)$. If u belongs to a negative triangle in G and k is the minimal weight of such cycle then $d_{G'}(u_S, u_D) = 7H + 2k + 1$ (odd). If u does not belong to any negative triangle then $d_{G'}(u_S, u_D) = 7H$ (even). Therefore the sum of distances is odd iff there is an odd number of vertices belonging to a negative triangle.

3.3 Negative Triangle Vertex Parity to Wiener Index Parity (Undirected)

In undirected graphs, to avoid a trivial Wiener Index Parity of 0, the Wiener Index is defined as the sum of $d(u, v)$ over every unordered (rather than ordered) pair $\{u, v\}$.

We assume that every triangle has odd length by multiplying every edge-weight by 4 and adding 1 (this preserves the sign of negative and non-negative triangles). We construct a graph G' similarly to [3, 51] and to Section 3.2 but the approach differs in the analysis of correctness: Each vertex u of G has four copies in G' denoted u_A, u_B, u_C, u_D. Let $H = 100M$ (sufficiently large *even* number). For every $(X, Y) \in \{(A, B), (B, C), (C, D)\}$ and $u, v \in V(G)$, add the edge (u_X, v_Y) of weight $2H + w(u, v)$. For every $u \neq v \in V(G)$, add an edge (u_A, v_D) of weight $5H$. For every $u = v \in V(G)$, add an edge (u_A, u_D) of weight $6H$.

Let m be the number of edges in G and let W be the sum of edge weights in G. We claim that $\mathrm{WIP}(G') - W$ is odd iff $\mathrm{NTVP}(G)$ is odd: The sum of A-to-B distances is $W + 2H \cdot m$. This sum has the same parity as W, which we cancel out by subtracting W from $\mathrm{WIP}(G')$. Notice that the sum of B-to-C (A-to-C) distances and the sum of C-to-D (B-to-D) distances are equal thus by adding both sums the parity of $\mathrm{WIP}(G')$ does not change. Similarly, the sum of the X-to-X distances for every $X \in \{A, B, C, D\}$ is the same and $\mathrm{WIP}(G')$ is not changed. We are left with the A-to-D distances. The sum of u_A-to-v_D distances for $u \neq v$ is $5H \cdot n(n-1)$ (even). If u is not in a negative triangle, then $d(u_A, u_D) = 6H$ (even) by using the direct edge (u_A, u_D). If u is in a negative triangle, the u_A-to-u_D distance is $6H$ plus the weight of the minimum weight triangle of u (odd). Therefore $\mathrm{WIP}(G') - W$ is odd iff there is an odd number of vertices with a negative triangle.

3.4 Negative Triangle Vertex Parity to Sum of Eccentricities Parity

The reduction is obtained by tweaking the reduction of Section 3.3. We add to G' an additional vertex y. For every $u \in V(G)$, we add the edge (y, u_D) of weight $7H$ and the edges $(y, u_A), (y, u_B)$ and (y, u_C) each of weight $5H$.

Notice that these changes to G' do not affect the distances between vertices of $V(G')\backslash\{y\}$ since every path that goes through y has weight of at least $10H$. Recall that H is an even number. The *eccentricity* of a vertex u is defined as $\max_v d(u, v)$. The eccentricity of vertices in $B \cup C$ is $5H$ (even), since their distance to y is $5H$ and their distance to any other vertex is bounded by $4H + 2M$ (i.e. smaller than $5H$). The eccentricity of vertices in D is $7H$ (even), since their distance to y is $7H$ and their distance to any other vertex is bounded by $6H$ (maximized by a vertex in A). The eccentricity of y is $7H$ (even). Finally, the eccentricity of a vertex u_A in A is $d(u_A, u_D)$, since the u_A-to-u_D distance is at least $6H - 3M$ and any other distance is bounded by $5H$ (maximized by y and some v_D). This means that, as shown in Section 3.3, the parity of $\sum_u d(u_A, u_D)$ equals $\mathrm{NTVP}(G)$.

3.5 Negative Triangle Vertex Parity to Integer Betweenness Centrality Parity

The reduction is deterministic and uses a similar graph G' to the one used in the reduction of [3] from Negative Weight Triangle to Betweenness Centrality: Each vertex u of G has four copies in G' denoted u_A, u_B, u_C, u_D. Let $H = 100M$ (sufficiently large number). For every $(X, Y) \in \{(A, B), (B, C), (C, D)\}$ and $u, v \in V(G)$, add the edge (u_X, v_Y) with weight $2H + w(u, v)$. Add a single vertex x and for every vertex $v \in V(G)$, add the edges $(u_A, x), (x, v_D)$ with weight $3H$. Add two sets of vertices Z, O each of size $\lceil \log n \rceil$. Let $z_i \in Z, o_i \in O$ be the i'th vertex of the sets. If the i'th bit in u's binary representation is 0, add an edge (u_A, z_i) with weight $2H$ and an edge (o_i, u_D) with weight $3H$. Otherwise, add an edge (u_A, o_i) with weight $2H$ and an edge (z_i, u_D) with weight $3H$. This dependency on the binary representation assures that every u_A and v_D are connected with a path (of weight $5H$) through O or through Z except for the case where $u = v$. See Figure 3.

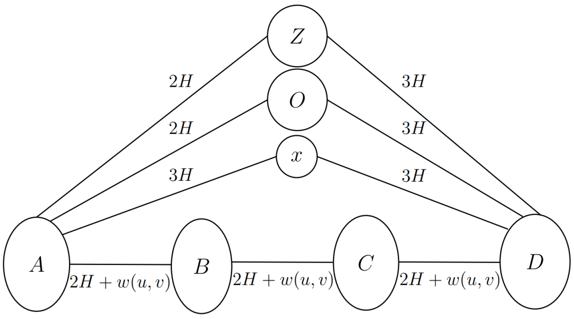


Figure 3 A representation of G' in the reduction from Negative Weight Triangle to Integer Betweenness Centrality Parity.

Consider the Integer Betweenness Centrality Parity of the vertex x in G'. Assume n is even (otherwise add a vertex to G with no negative triangle). Notice that the only pairs with a shortest path through x can be of the form (u_A, u_D) (pairs (u_A, v_D) with $v \neq u$ have shorter paths of weight $5H$ through Z or O). Furthermore, there is a shortest u_A-to-u_D path through x iff u is not in a negative triangle. This is because the distance between u_A and u_D is the minimum between $6H$ (going through x) and $6H + w(u,v) + w(v,t) + w(t,u)$ for some $v, t \in V(G)$. Therefore, the number of pairs (u_A, u_D) with shortest paths through x is n minus the number of vertices in a negative triangle, hence the parity of the number of paths going through x in G' is the same as the parity of the number of vertices in G with a negative triangle.

3.6 APSP to Integer Betweenness Centrality Parity

We provide a probabilistic (one sided error) reduction from Negative Weight Triangle that does not go through Negative Triangle Vertex Parity. We continue from where we stopped in Section 3.5. Recall that the number of pairs that have a shortest path through x is n minus the number of vertices in a negative triangle. If there is an odd number of pairs then we return that a negative triangle exists. Otherwise, there is an even number of vertices with a negative triangle. We then choose a set $S \subseteq V(G)$ uniformly, and limit A, D to the vertices in S (B, C remain the same). If the number of paths going through x is odd we report that there is a negative triangle, otherwise we report that there is none. If a negative triangle exists, S has an odd number of vertices with a negative triangle with probability $1/2$, and we detect an odd number of pairs. Otherwise, S always has 0 (even) vertices with a negative triangle, and we succeed with probability 1. We can repeat the process $O(\log n)$ times and amplify the probability of success to $1 - 1/n^c$ for any constant c.

4 APSP to Min-Plus Matrix Multiplication Parity

4.1 Min-Plus Multiplication to Min-Plus Multiplication Parity

Given two $n \times n$ matrices A and B we wish to compute $C = A \otimes B$ where $C[i,j] = \min_k\{A[i,k] + B[k,j]\}$. First assume that for every i, j the value $C[i,j]$ is obtained by a unique index k. Let K be the $n \times n$ matrix such that $K[i,j]$ is the unique index k of $C[i,j]$. We show how to compute K by using Min-Plus Matrix Multiplication Parity.

Define $\hat{A} = 2A$, and for any $t \in [\log n]$ define k_t as the t'th bit of k and $\hat{B}_t$ to be the matrix such that $\hat{B}_t[k,j] = 2B[k,j] + k_t$. We compute the parity of $\hat{C}_t = \hat{A} \otimes \hat{B}_t$ for every $t \in [\log n]$. We claim that the parity of $\hat{C}_t[i,j]$ is the t'th bit of $K[i,j]$. This is because

$\hat{C}_t[i,j] = \min_k\{2A[i,k] + 2B[k,j] + k_t\}$. The parity of this value is 0 if the unique index k that minimizes $A[i,k] + B[k,j]$ has $k_t = 0$ and is 1 otherwise. Therefore, from this parity we can recover the t'th bit of $K[i,j]$.

To remove the assumption on the uniqueness of k, we define matrices A' and B' as $A'[i,k] = (n+1) \cdot A[i,k] + k$ and $B'[k,j] = (n+1) \cdot B[k,j]$. Observe that A' and B' have the uniqueness of k property. This is because if $A'[i,k_1] + B'[k_1,j] = A'[i,k_2] + B'[k_2,j]$ for some k_1, k_2 then $(n+1) \cdot (A[i,k_1] + B[k_1,j]) + k_1 = (n+1) \cdot (A[i,k_2] + B[k_2,j]) + k_2$ and, since k_1 and k_2 are smaller than $n+1$, it follows that $k_1 = k_2$. Furthermore, in order to compute $A \otimes B$ it suffices to compute $C' = A' \otimes B'$. Because if $A'[i,k_1] + B'[k_1,j] \leq A'[i,k_2] + B'[k_2,j]$ then (since k_1 and k_2 are smaller than $n+1$) $A[i,k_1] + B[k_1,j] \leq A[i,k_2] + B[k_2,j]$.

As a corollary, we get that APSP is subcubic equivalent to APSP Parity (i.e. the problem of deciding the parity of every pairwise distance in the graph): Let M be a bound on the absolute values in A and B. Create a graph consisting of vertices a_i, b_i, c_i for every $i \in [n]$ and the edges $(a_i, b_j), (b_i, c_j)$ with weights $A[i,j] + 3M$ and $B[i,j] + 3M$ respectively for every i, j. The distance $d(a_i, c_j) = 6M + \min_k\{A[i,k] + B[k,j]\}$ and therefore has the same parity as $(A \otimes B)[i,j]$. Notice that the reduction can be modified (with a folklore trick) to show that even computing $A \otimes A$ Parity is hard. Let D be the $n \times n$ matrix with every element equals to $3M$. Let E be the $2n \times 2n$ matrix $\begin{bmatrix} A & B \\ D & D \end{bmatrix}$. Then $E \otimes E$ equals $\begin{bmatrix} X & Y \\ Z & W \end{bmatrix}$ where $Y = A \otimes B$ since $Y[i,j] = \min\{(A \otimes B)[i,j], (B \otimes D)[i,j]\} = (A \otimes B)[i,j]$.

4.2 Min-Plus Convolution to Min-Plus Convolution Parity

Given vectors A and B each of length n, we wish to compute their convolution C where $C[i] = \min_{i=j+k}\{A[j] + B[k]\}$. The approach is the same as in Section 4.1. We assume each value $C[i]$ is obtained by a unique index k, otherwise we multiply A and B by $n+1$ and add to each $B[k]$ the value k (as in Section 4.1). Let K be the vector such that $K[i]$ is the unique index k of $C[i]$. Define $\hat{A} = 2A$, and for any $t \in [\log n]$ define k_t is the t'th bit of k and $\hat{B}_t$ to be the vector such that $\hat{B}_t[k] = 2B[k] + k_t$. Let $\hat{C}_t[i]$ be the convolution of $\hat{A}$ and $\hat{B}_t$. Then the t'th bit of $K[i]$ is the same as the parity of $\hat{C}_t[i]$. This is because $\hat{C}_t[i] = \min_{i=j+k}\{2A[j] + 2B[k] + k_t\}$.

4.3 Maximum Consecutive Subsums to Maximum Consecutive Subsums Parity

Given a vector X of length n, the maximum consecutive subsums problem asks to compute $\max_i \sum_{j=1}^{k} X[i+j]$ for every $k \in [n]$. To achieve this, we first compute (in linear time) the vector A where $A[k] = \sum_{j=1}^{k} X[j]$. The problem then reduces to computing Diff(A) where Diff$(A)[k] = \max_i\{A[k+i] - A[i]\}$. In fact, since $X[k] = A[k] - A[k-1]$, there is also a reduction in the opposite direction and so the two problems are equivalent (and their parity versions are equivalent). In this section, we show that given the parity of Diff(A) (i.e. the parity of every element in Diff(A)) we can compute Diff(A) itself.

Given a vector A, we wish to compute Diff(A). We assume that for every k, the value Diff$(A)[k] = \max_i\{A[k+i] - A[i]\}$ is obtained by a unique index i. Otherwise, we multiply every $A[k]$ by (n^2+1) and add k^2 (similarly to Section 4.1). Let I be the vector of such unique indices, and let J be the vector where $J[k] = I[k] + k$. By definition, Diff$(A)[k] = A[J[k]] - A[I[k]]$. We define A_t to be the vector such that $A_t[k] = 4 \cdot A[k] + k_t$ (where k_t is the t'th bit of k). Notice that $A_t[j] - A_t[i] = 4 \cdot (A[j] - A[i]) + (j_t - i_t)$ where $(j_t - i_t) \in \{-1, 0, 1\}$. Thus, for every k, Diff$(A)[k]$ is maximized when $j = J[k]$ and $i = I[k]$

(regardless of the values of j_t and i_t). This is because for every $i \neq I[k]$ and $j = k + i$ it holds that $4\cdot(A[j]-A[i])+(j_t-i_t) \leq 4\cdot(A[J[k]]-A[I[k]]-1)+1 < 4\cdot(A[J[k]]-A[I[k]])+(J[k]_t-I[k]_t)$. Observe that that the parity of $A_t[j] - A_t[i]$ is $j_t \oplus i_t$, where $\oplus$ is the bitwise XOR operation.

For every $t \in [\log n]$ we compute the parity of $\mathsf{Diff}(A_t)$. The computed parity of $\mathsf{Diff}(A_t)[k]$ is $J[k]_t \oplus I[k]_t$. Given $J[k]_1 \oplus I[k]_1, \ldots, J[k]_{\log n} \oplus I[k]_{\log n}$, we want to compute $J[k]$ and $I[k]$. Recall that $J[k] = k + I[k]$. Let $c_1, \ldots, c_{\log n}$ be the carry bits in the binary addition of $I[k]$ and k. We know that $I[k]_t \oplus k_t \oplus c_t = J[k]_t$ so by substituting $J[k]_t \oplus I[k]_t$ we compute every $c_t = J[k]_t \oplus I[k]_t \oplus k_t$. Given c_t, c_{t+1}, k_t, and $J[k]_t \oplus I[k]_t$ we wish to compute $J[k]_t$ and $I[k]_t$. However, this can only be done when $J[k]_t \oplus I[k]_t = 1$. In this case $I[k]_t = \neg J[k]_t = c_{t+1}$. This is because $k_t \oplus c_t = J[k]_t \oplus I[k]_t = 1$ so $k_t + c_t = 1$ and therefore $c_{t+1} = 1$ iff $I[k]_t + k_t + c_t \geq 2$ iff $I[k]_t = 1$. We are left with the bits where $J[k]_t = I[k]_t$. Let $A_{t,p}$ be the vector such that $A_{t,p}[k] = 4 \cdot A[k] + (k_t \to k_p)$ (compared to A_t, we replace k_t with k_t implies k_p). For every $(t,p) \in [\log n]^2$ we compute the parity of $\mathsf{Diff}(A_{t,p})$. The parity of $\mathsf{Diff}(A_{t,p})[k]$ is $(J[k]_t \to J[k]_p) \oplus (I[k]_t \to I[k]_p)$ and is denoted as $b_{t,p}$. Given that $J[k]_t$ and $I[k]_t$ have not been computed yet, we know that $J[k]_t = I[k]_t$, hence for every p it holds that $b_{t,p} = (I[k]_t \to J[k]_p) \oplus (I[k]_t \to I[k]_p)$. Notice that $J[k] > I[k]$ therefore there must be an index p' where $I[k]_{p'} \neq J[k]_{p'}$ (which we previously found) thus $b_{t,p'} = (I[k]_t \to \neg I[k]_{p'}) \oplus (I[k]_t \to I[k]_{p'})$. Observe that $I[k]_t = 0$ iff $b_{t,p'} = 0$. Overall, we find $I[k]$ for every k using $O(\log^2 n)$ Diff parity computations and $\tilde{O}(n)$ reduction time.

5 Zero Weight Triangle Counting to Negative Triangle Counting

In this section we show a simple but surprising reduction from counting zero weight triangles to counting negative triangles. We show a deterministic reduction from Zero Weight Triangle to Negative Triangle Counting and a randomized reduction from Zero Weight Triangle to Negative Triangle Parity.

5.1 Zero Weight Triangle Counting (Parity) to Negative Triangle Counting (Parity)

We want to count the number of triangles with weight zero in G. Let Δ be the number of triangles in G. We can compute Δ in matrix-multiplication $O(n^\omega)$ time[4]. Let $\Delta^0, \Delta^+, \Delta^-$ be the number of zero, positive, and negative weight triangles in G' respectively. Given a subcubic algorithm for Negative Triangle Counting, we can compute Δ^-. By negating weights in G we can compute Δ^+ as well. We then compute $\Delta^0 = \Delta - \Delta^+ - \Delta^-$. This simple reduction also reduces Zero Weight Triangle Parity to Negative Triangle Parity.

5.2 Zero Weight Triangle to Zero Weight Triangle Parity

Given a graph G, we want to find whether there is a zero weight triangle. We create a graph G' as follows: For every vertex $u \in V(G)$, we create three copies u_A, u_B, u_C in G', and for every edge $(u,v) \in E(G)$ we add the edges (u_A, v_B),(u_B, v_C), (u_C, v_A) to G' (with the same weight as (u,v)). Notice that there is a zero weight triangle in G iff there is a zero weight triangle in G'. We now create a graph G'' by removing from G' each edge (u_B, v_C) with probability $\frac{1}{2}$, and removing from G' each vertex u_A with probability $\frac{1}{2}$. We report that a zero weight triangle exists in G iff there is an odd number of zero weight triangles in G''. We now show that this reduction works with probability at least $\frac{1}{4}$.

[4] We can also compute Δ with Negative Triangle Counting by changing every weight in G to -1.

If there is no zero weight triangle in G', we succeed with probability 1. If there is a zero weight triangle in G', then let u_A be a vertex of G' that participates in some zero weight triangle. Since we removed each edge (v_B, t_C) with probability $\frac{1}{2}$ then, with probability $\frac{1}{2}$, the vertex u_A participates in an odd number of zero weight triangles in G''. Let $\Delta^0_{v_A}$ be the number of zero weight triangles in G'' that v_A participates in. The total number of zero weight triangles in G'' is $\sum_{v\in V(G)} \Delta^0_{v_A} = \sum_{v\in V(G)\setminus\{u\}} \Delta^0_{v_A} + \Delta^0_{u_A}$. If $\Delta^0_{u_A}$ is odd then, with probability $\frac{1}{2}$, our decision whether to remove u_A leads to an odd number of zero weight triangles in G''. Overall, the probability of success is therefore at least $\frac{1}{4}$.

6 Min-Plus Convolution to Knapsack Parity

In the KNAPSACK problem, given a set of n items (w_i, v_i) and a target weight t, we wish to pick a multiset of items I that maximizes $\sum_{i\in I} v_i$ subject to $\sum_{i\in I} w_i \leq t$. When I is required to be a set (and not a multiset) the problem is called 0/1-KNAPSACK. In the INDEXED KNAPSACK problem, we have $w_i = i$ and $t = n$. Finally, the COIN CHANGE problem [30, 33] is the same as INDEXED KNAPSACK but with the additional restriction $\sum_{i\in I} i = n$.

The KNAPSACK and the 0/1-KNAPSACK problems are equivalent to Min-plus Convolution under randomized reductions [17]. The INDEXED KNAPSACK and the COIN CHANGE problem are equivalent to Min-plus Convolution under deterministic reductions [30]. In this section, we show that the parity versions of all the above problems are equivalent to Min-plus Convolution.

6.1 Super-Additivity Testing to Knapsack [17]

Given a vector $A[0], \ldots, A[n-1]$, the SUPER-ADDITIVITY TESTING problem asks whether $A[i] + A[j] \leq A[i+j]$ for every i, j. The problem is subquadratic equivalent to Min-plus Convolution under deterministic reductions [17]. We now give a brief description of the reduction in [17] from SUPER-ADDITIVITY TESTING to KNAPSACK.

First, it is shown in [17] that we can assume without loss of generality that $0 = A[0] < A[1] < \cdots < A[n-1] = M$. Let $D = Mn + 1$, the instance of KNAPSACK consists of two types of items: Type-A items are $(i, A[i])$ and Type-B items are $(2n-1-i, D-A[i])$. It remains to show that, when setting $t = 2n-1$, the optimal sum of values $\sum_{i\in I} v_i$ equals D iff A is super-additive. Since $D > \sum_i A[i]$, the optimal solution must take at least one Type-B item, and it cannot take more than one because the weight would exceeds t. If A is not super-additive, then for some i, j it holds that $A[i] + A[j] > A[i+j]$ and therefore the three items $\{(i, A[i]), (j, A[j]), (2n-1-i-j, D-A[i+j])\}$ constitute a valid solution whose value is larger than D. If A is super-additive, then every two Type-A items $(i, A[i]), (j, A[j])$ can be replaced by $(i+j, A[i+j])$ without changing the total weight. Thus, for any i, the solution $\{(i, A[i]), (2n-1-i, D-A[i])\}$ is optimal and its value is exactly D.

6.2 Super-Additivity Testing to Knapsack Parity

We now modify the above the reduction to obtain a reduction to KNAPSACK PARITY. We first remove the item $(0, A[0])$ (since $A[0] = 0$ it does not contribute any value). We then replace every Type-A item $(i, A[i])$ by $(i, 2A[i])$, every Type-B item $(2n-1-i, D-A[i])$ (with $i \neq 1$) by $(2n-1-i, 2(D-A[i]))$, and the Type-B item $(2n-1-1, D-A[1])$ by $(2n-1-1, 2(D-A[1])+1)$. We show that A is super-additive iff the value of the optimal solution is odd.

Once again, every optimal solution must consist of exactly one Type-B item since if there are no Type-B items then the value does not exceed $2D$ as $2D > \sum_i 2A[i]$, and with more than one Type-B items the weight exceeds $t = 2n-1$. If A is not super-additive, then there are i, j such that $k = i + j \geq 2$ and $A[i] + A[j] > A[k]$ therefore the items $\{(i, 2A[i]), (j, 2A[j]), (2n - 1 - i - j, 2D - 2A[k])\}$ constitute a valid solution whose value is larger than $2D + 1$. Notice that $k \geq 2$ since if $k = 0$ then the total value is $2D$ (thus not optimal), and if $k = 1$ then we include the item $(2n - 1 - 1, 2(D - A[1]) + 1)$ and since $t = 2n - 1$ we can only add the item $(1, 2A[1])$ leading to a non-optimal solution with value $2D + 1$. Therefore, the optimal solution does not use the item $(2n - 1 - 1, 2(D - A[1]) + 1)$ and hence it has an even value. On the other hand, if A is super-additive, then the solution $\{(1, 2A[1]), (2n - 1 - 1, 2(D - A[1]) + 1)\}$ is optimal and has value exactly $2D + 1$ (odd). This is because among the solutions that include a Type-B item $(2n - 1 - i - j, 2D - 2A[k])$ with $k \neq 1$, once again by super-additivity, $\{(k, 2A[k]), (2n - 1 - k, 2D - 2A[k])\}$ has the maximal value of $2D$ (i.e. smaller than $2D + 1$).

6.3 Super-Additivity Testing to 0/1-knapsack Parity

We now show how to modify the above reductions to be reductions to 0/1-KNAPSACK and 0/1-KNAPSACK PARITY. In the above reductions, when A is super-additive the optimal solution does not use any item more than once, and its total value V is either D or $2D + 1$. When A is not super-additive, there is a solution with a higher value than V. There is only one case where this solution may use the same item more than once. This happens when A is not super-additive in the following way: $A[i] + A[j] \leq A[i + j]$ for every $i \neq j$ but $A[i] + A[j] > A[i + j]$ for some $i = j$. Therefore, in $O(n)$ time we can check for every i whether $2A[i] \leq A[2i]$ and only if the answer is yes we apply the reduction.

Note that the above reductions also apply to the INDEXED KNAPSACK PARITY problem. This is because the target weight t equals the total number of items $2n-1$, and each item has a unique weight in $[2n - 1]$. The reductions also apply to COIN CHANGE PARITY: When A is super-additive, the optimal solution for COIN CHANGE (which is also an optimal solution for KNAPSACK) has weight $2n - 1$ (equal to the number of items) and an odd value $(2D + 1)$. When A is not super-additive, the optimal solution for COIN CHANGE (which is possibly not an optimal solution for KNAPSACK) has weight $2n - 1$ (equal to the number of items) and an even value (larger than $2D + 1$).

7 APSP to Sum of Eccentricities

In this section, we show a subcubic reduction from RADIUS (hence also from APSP) on a graph G to SUM OF ECCENTRICITIES on a graph G'. Let R be the radius of G. In order to compute R it suffices to find whether $R \geq k$ for any given $k \in [Mn]$ (since then we can binary search for R). The constructed graph G' is similar to the one in the reduction of [3] from DIAMETER to POSITIVE BETWEENESS CENTRALITY: We create G' by multiplying the edge weights of G by 2 and then adding a vertex x and the edges (x, u) and (u, x) each of weight k for every $u \in V(G)$.

▶ **Lemma 6.** *$R \geq k$ iff the sum of eccentricities of G' is $\sum_u \max_v d_{G'}(u, v) = 2kn + k$.*

Proof. This is the same as claiming that $R \geq k$ iff $\sum_{u \neq x} \max_v d_{G'}(u, v) = 2kn$. If $R \geq k$, then every vertex $u \neq x$ can use x to get to its furthest vertex with a path of length $2k \leq 2R$. Observe that any other path would be of length at least $2R$ (because we have multiplied all edge weights by 2). Therefore, $\sum_{u \neq x} \max_v d_{G'}(u, v) = 2kn$. If on the other hand $R < k$, then the distance in G' from the radius vertex of G to any other vertex is at most $\max\{2R, k\} < 2k$ so this vertex adds less than $2k$ to the sum. All the other vertices add at most $2k$ to the sum, and thus $\sum_{u \neq x} \max_v d_{G'}(u, v) \neq 2kn$. ◀

8 Radius to Radius Parity and Diameter to Diameter Parity

In this section, we show that computing the Radius R (resp. Diameter D) of a graph G subcubicaly reduces to computing the parity of R (resp. D). As usual, to compute R, D it suffices to find whether $R, D \geq k'$ for $k' \in [Mn]$. Let $k' = (k+1)/2$ for some odd $k \geq 1$. We create a reduction graph G' similarly to [3] and to Section 7: We multiply the edge weights of G by 2 and add a vertex x with (v,x),(x,v) edges of weight k for every $v \in V(G)$.

Consider first the radius of G'. If the radius vertex of G' is x then its value is k. Otherwise, its value is either $\max\{2R, k\}$ (by using the edge to x and the same path as in G to all other vertices) or $2k$ (by using a path through x). Therefore, the radius of G' has value $\min\{k, 2R\}$. If $2R \geq k+1$ (i.e $R \geq k'$), the radius is k (odd). Otherwise $2R < k+1$ (i.e $R < k'$) and the radius is $2R$ (even). Next consider the diameter of G'. If x is an endpoint of the diameter of G' then the diameter value is k. Otherwise, the diameter value is either $2D$ (by taking the same path as in G) or $2k$ (by using a path through x). Therefore the diameter of G' has value $\max\{k, \min\{2D, 2k\}\}$. If $2D \leq k$ (i.e. $D < k'$), then the diameter is k (odd). Otherwise $2D \geq k+1$ (i.e. $D \geq k'$), and the diameter is either $2D$ or $2k$ (even in both cases).

References

1 Amir Abboud. Fine-grained reductions and quantum speedups for dynamic programming. In *46th ICALP*, pages 8:1–8:13, 2019.

2 Amir Abboud, Vincent Cohen-Addad, and Philip N Klein. New hardness results for planar graph problems in p and an algorithm for sparsest cut. In *52nd STOC*, 2020. To appear.

3 Amir Abboud, Fabrizio Grandoni, and Virginia Vassilevska Williams. Subcubic equivalences between graph centrality problems, APSP and diameter. In *26th SODA*, pages 1681–1697, 2015.

4 Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *55th FOCS*, pages 434–443. IEEE, 2014.

5 Vikraman Arvind and Piyush P Kurur. Graph isomorphism is in spp. In *43rd FOCS*, pages 743–750, 2002.

6 Arturs Backurs, Nishanth Dikkala, and Christos Tzamos. Tight hardness results for maximum weight rectangles. In *43rd ICALP*, pages 81:1–81:13, 2016.

7 Arturs Backurs, Piotr Indyk, and Ludwig Schmidt. Better approximations for tree sparsity in nearly-linear time. In *28th SODA*, pages 2215–2229, 2017.

8 Richard Beigel, Harry Buhrman, and Lance Fortnow. Np might not be as easy as detecting unique solutions. In *30th STOC*, pages 203–208, 1998.

9 Enric Boix-Adserà, Matthew Brennan, and Guy Bresler. The average-case complexity of counting cliques in erdős-rényi hypergraphs. In *60th FOCS*, pages 1256–1280, 2019.

10 Mahdi Boroujeni, Sina Dehghani, Soheil Ehsani, Mohammad Taghi Hajiaghayi, and Saeed Seddighin. Subcubic equivalences between graph centrality measures and complementary problems. *CoRR*, abs/1905.08127, 2019. arXiv:1905.08127.

11 Charles Bouillaguet and Claire Delaplace. Faster algorithms for the sparse random 3XOR problem. Preprint, 2019. URL: https://hal.archives-ouvertes.fr/hal-02306917.

12 Charles Bouillaguet, Claire Delaplace, and Pierre-Alain Fouque. Revisiting and improving algorithms for the 3xor problem. *IACR Transactions on Symmetric Cryptology*, pages 254–276, 2018.

13 David Bremner, Timothy M. Chan, Erik D. Demaine, Jeff Erickson, Ferran Hurtado, John Iacono, Stefan Langerman, Mihai Patrascu, and Perouz Taslakian. Necklaces, convolutions, and X+Y. *Algorithmica*, 69(2):294–314, 2014.

14 Karl Bringmann. Approximating subset sum is equivalent to min-plus-convolution. *CoRR*, abs/1912.12529, 2019. arXiv:1912.12529.

15 Radu Curticapean. Counting problems in parameterized complexity. In *13th IPEC 2018*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

16 Marek Cygan, Holger Dell, Daniel Lokshtanov, Dániel Marx, Jesper Nederlof, Yoshio Okamoto, Ramamohan Paturi, Saket Saurabh, and Magnus Wahlström. On problems as hard as CNF-SAT. *ACM Trans. Algorithms*, 12(3):41:1–41:24, 2016.

17 Marek Cygan, Marcin Mucha, Karol Wegrzycki, and Michal Wlodarczyk. On problems equivalent to (min,+)-convolution. *ACM Trans. Algorithms*, 15(1):14:1–14:25, 2019.

18 Holger Dell. Fine-grained complexity classification of counting problems. https://simons.berkeley.edu/talks/holger-dell-2016-03-28, 2016.

19 Holger Dell and John Lapinskas. Fine-grained reductions from approximate counting to decision. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 281–288, 2018.

20 Erik D. Demaine, Andrea Lincoln, Quanquan C. Liu, Jayson Lynch, and Virginia Vassilevska Williams. Fine-grained I/O complexity via reductions: New lower bounds, faster algorithms, and a time hierarchy. In *9th ITCS*, pages 34:1–34:23, 2018.

21 Martin Dietzfelbinger, Philipp Schlag, and Stefan Walzer. A subquadratic algorithm for 3XOR. *CoRR*, abs/1804.11086, 2018. arXiv:1804.11086.

22 Jörg Flum and Martin Grohe. The parameterized complexity of counting problems. *SIAM Journal on Computing*, 33(4):892–922, 2004.

23 Lance Fortnow. Counting complexity. *Complexity theory retrospective II*, pages 81–107, 1997.

24 Anka Gajentaan and Mark H. Overmars. On a class of $o(n^2)$ problems in computational geometry. *Comput. Geom.*, 5:165–185, 1995.

25 Oded Goldreich and Guy N. Rothblum. Counting t-cliques: Worst-case to average-case reductions and direct interactive proof systems. In *59th FOCS*, pages 77–88, 2018.

26 Allan Grønlund and Seth Pettie. Threesomes, degenerates, and love triangles. *J. ACM*, 65(4):22:1–22:25, 2018.

27 Zahra Jafargholi and Emanuele Viola. 3xor,3sum, triangles. *Algorithmica*, 74(1):326–343, 2016.

28 James King. A survey of 3SUM-hard problems. Preprint, 2019. URL: http://www.ccs.neu.edu/home/viola/classes/papers/King04Survey3sum.pdf.

29 Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Higher lower bounds from the 3sum conjecture. In *27th SODA*, pages 1272–1287. SIAM, 2016.

30 Marvin Künnemann, Ramamohan Paturi, and Stefan Schneider. On the fine-grained complexity of one-dimensional dynamic programming. In *44th ICALP*, pages 21:1–21:15, 2017.

31 Eduardo Sany Laber, Wilfredo Bardales Roncalla, and Ferdinando Cicalese. On lower bounds for the maximum consecutive subsums problem and the (min,+)-convolution. In *11th ISIT*, pages 1807–1811, 2014.

32 Rio LaVigne, Andrea Lincoln, and Virginia Vassilevska Williams. Public-key cryptography in the fine-grained setting. In *39th CRYPTO*, pages 605–635, 2019.

33 Andrea Lincoln, Adam Polak, and Virginia Vassilevska Williams. Monochromatic triangles, intermediate matrix products, and convolutions. In *11th ITCS*, pages 53:1–53:18, 2020.

34 Christos H Papadimitriou and Stathis K Zachos. Two remarks on the power of counting. In *Theoretical Computer Science*, volume 145, pages 269–276, 1983.

35 Mihai Patrascu. Towards polynomial lower bounds for dynamic problems. In *42nd STOC*, pages 603–610, 2010.

36 L. Roditty and V. Vassilevska Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In *45th STOC*, pages 515–524, 2013.

37 Liam Roditty and Uri Zwick. On dynamic shortest paths problems. In *12th ESA*, pages 580–591, 2004.

38 Barna Saha. Language edit distance and maximum likelihood parsing of stochastic grammars: Faster algorithms and connection to fundamental graph problems. In *6th FOCS*, pages 118–135, 2015.

39 Tadao Takaoka. Efficient algorithms for the maximum subarray problem by distance matrix multiplication. *Electr. Notes Theor. Comput. Sci.*, 61:191–200, 2002.

40 Seinosuke Toda. PP is as hard as the polynomial-time hierarchy. *SIAM J. Comput.*, 20(5):865–877, 1991.

41 Leslie G Valiant. The complexity of computing the permanent. *Theoretical computer science*, 8(2):189–201, 1979.

42 Leslie G Valiant. Accidental algorthims. In *47th FOCS*, pages 509–517, 2006.

43 Leslie G Valiant. Some observations on holographic algorithms. *computational complexity*, 27(3):351–374, 2018.

44 Leslie G Valiant and Vijay V Vazirani. Np is as easy as detecting unique solutions. In *17th STOC*, pages 458–463, 1985.

45 Maria Isabel González Vasco and Mats Näslund. A survey of hard core functions. *Cryptography and Computational Number Theory*, 20:227–255, 2001.

46 Ryan Williams. Counting solutions to polynomial systems via reductions. In *1st SOSA*, pages 6:1–6:15, 2018.

47 Ryan Williams. Faster all-pairs shortest paths via circuit complexity. *SIAM J. Comput.*, 47(5):1965–1985, 2018.

48 Ryan Williams and Huacheng Yu. Finding orthogonal vectors in discrete structures. In *25th SODA*, pages 1867–1877, 2014.

49 Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *ICM*, 2018. Invited talk.

50 Virginia Vassilevska Williams and Ryan Williams. Finding, minimizing, and counting weighted subgraphs. In *41st STOC*, pages 455–464, 2009.

51 Virginia Vassilevska Williams and Ryan Williams. Subcubic equivalences between path, matrix, and triangle problems. *J. ACM*, 65(5):27:1–27:38, 2018.

Optimal Streaming Algorithms for Submodular Maximization with Cardinality Constraints

Naor Alaluf
Department of Mathematics and Computer Science, Open University of Israel, Ra'anana, Israel
naoralaluf@gmail.com

Alina Ene
Department of Computer Science, Boston University, MA, USA
aene@bu.edu

Moran Feldman
Department of Computer Science, University of Haifa, Israel
moranfe@cs.haifa.ac.il

Huy L. Nguyen
Khoury College of Computer and Information Science, Northeastern University, Boston, MA, USA
hlnguyen@cs.princeton.edu

Andrew Suh
Department of Computer Science, Boston University, MA, USA
asuh9@bu.edu

Abstract

We study the problem of maximizing a *non-monotone* submodular function subject to a cardinality constraint in the streaming model. Our main contributions are two single-pass (semi-)streaming algorithms that use $\tilde{O}(k) \cdot \text{poly}(1/\varepsilon)$ memory, where k is the size constraint. At the end of the stream, both our algorithms post-process their data structures using *any offline algorithm* for submodular maximization, and obtain a solution whose approximation guarantee is $\frac{\alpha}{1+\alpha} - \varepsilon$, where α is the approximation of the offline algorithm. If we use an exact (exponential time) post-processing algorithm, this leads to $\frac{1}{2} - \varepsilon$ approximation (which is nearly optimal). If we post-process with the algorithm of [5], that achieves the state-of-the-art offline approximation guarantee of $\alpha = 0.385$, we obtain 0.2779-approximation in polynomial time, improving over the previously best polynomial-time approximation of 0.1715 due to [17]. One of our algorithms is combinatorial and enjoys fast update and overall running times. Our other algorithm is based on the multilinear extension, enjoys an improved space complexity, and can be made deterministic in some settings of interest.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms; Mathematics of computing → Combinatorial optimization; Mathematics of computing → Probabilistic algorithms

Keywords and phrases Submodular maximization, streaming algorithms, cardinality constraint

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.6

Category Track A: Algorithms, Complexity and Games

Related Version `https://arxiv.org/abs/1911.12959`

Funding *Naor Alaluf*: The work of Naor Alaluf was supported in part by ISF grant number 1357/16.
Alina Ene: The work of Alina Ene was supported in part by NSF CAREER grant CCF-1750333, NSF grant CCF-1718342, and NSF grant III-1908510.
Moran Feldman: The work of Moran Feldman was supported in part by ISF grant number 1357/16.
Huy L. Nguyen: The work of Huy L. Nguyen was supported in part by NSF CAREER grant CCF-1750716 and NSF grant CCF-1909314.
Andrew Suh: The work of Andrew Suh was supported in part by NSF CAREER grant CCF-1750333, NSF grant CCF-1718342, and NSF grant III-1908510.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 6; pp. 6:1–6:19
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

In this paper, we study the problem of maximizing a *non-monotone* submodular function subject to a cardinality (size) constraint in the streaming model. This problem captures problems of interest in a wide-range of domains, such as machine learning, data mining, combinatorial optimization, algorithmic game theory, social networks, and many others. A representative application is data summarization, where the goal is to select a small subset of the data that captures the salient features of the overall dataset [2]. One can model these problems as submodular maximization with a cardinality constraint: the submodular objective captures how informative the summary is, as well as other considerations such as how diverse the summary is, and the cardinality constraint ensures that the summary is small. Obtaining such a summary is very beneficial when working with massive data sets, that may not even fit into memory, since it makes it possible to analyze the data using algorithms that would be prohibitive to run on the entire dataset.

There have been two main approaches to deal with the large size of modern data sets: the *distributed* computation approach partitions the data across many machines and uses local computation on the machines and communication across the machines in order to perform the analysis, and the *streaming* computation approach processes the data in a stream using only a small amount of memory and (ideally) only a single pass over the data. Classical algorithms for submodular maximization, such as the Greedy algorithm, are not suitable in these settings since they are centralized and require many passes over the data. Motivated by the applications as well as theoretical considerations, there has been a significant interest in studying submodular maximization problems both in the distributed and the streaming setting, leading to many new results and insights [22, 29, 2, 9, 11, 26, 4, 28, 3, 14, 27, 17, 31, 1].

Despite this significant progress, several fundamental questions remain open both in the streaming and distributed setting. In the streaming setting, which is the main focus of this paper, submodular maximization is fairly well understood when the objective function is additionally *monotone* – i.e., we have $f(A) \leq f(B)$ whenever $A \subseteq B$. For example, the Greedy approach, which obtains an optimal $(1 - 1/e)$-approximation in the centralized setting when the function is monotone [30], can be adapted to the streaming model [22, 2]. This yields the single-threshold Greedy algorithm: make a single pass over the data and select an item if its marginal gain exceeds a suitably chosen threshold. If the threshold is chosen to be $\frac{1}{2}\frac{f(\mathrm{OPT})}{k}$, where $f(\mathrm{OPT})$ is the value of the optimal solution and k is the cardinality constraint, then the single-threshold Greedy algorithm is guaranteed to achieve $\frac{1}{2}$-approximation. Although the value of the optimal solution is unknown, it can be estimated based on the largest singleton value even in the streaming setting [2]. Remarkably, this approximation guarantee is optimal in the streaming model even if we allow unbounded computational power: Feldman et al. [19] showed that any algorithm for monotone submodular maximization that achieves an approximation better than $\frac{1}{2}$ requires $\Omega\left(\frac{n}{k^3}\right)$ memory, where n is the length of the stream. Additionally, the single-threshold Greedy algorithm enjoys a fast update time of $O(\varepsilon^{-1} \log k)$ marginal value computations per item, and it uses $O(\varepsilon^{-1} k \log k)$ space.

In contrast, the general problem with a *non-monotone* objective has proved to be considerably more challenging. Even in the centralized setting, the Greedy algorithm fails to achieve any approximation guarantee when the objective is non-monotone. Thus, several approaches have been developed for handling non-monotone objectives in this setting, including local search [15, 24, 23], continuous optimization [18, 13, 5] and sampling [6, 16]. The currently best approximation guarantee is 0.385 [5], and the strongest inapproximability is 0.491 [20], and it remains a long-standing open problem to settle the approximability of submodular maximization subject to a cardinality constraint.

Adapting the above techniques to the streaming setting is challenging, and the approximation guarantees are weaker. The main approach for non-monotone maximization in the streaming setting has been to extend the local search algorithm of Chakrabarti and Kale [9] from monotone to non-monotone objectives. This approach was employed in a sequence of works [11, 17, 27], leading to the currently best approximation of $\frac{1}{3+2\sqrt{2}} \approx 0.1715$.[1] This naturally leads to the following questions.

- *What is the optimal approximation ratio achievable for submodular maximization in the streaming model? Is it possible to achieve $\frac{1}{2} - \varepsilon$ approximation using an algorithm that uses only* $\mathrm{poly}(k, 1/\varepsilon)$ *space?*
- *Is there a good streaming algorithm for non-monotone functions based on the single-threshold Greedy algorithm that works so well for monotone functions?*
- *Can we exploit existing heuristics for the offline problem in the streaming setting?*

Our contributions. In this work, we give an affirmative answer to all of the above questions. Specifically, we give streaming algorithms[2] that perform a single pass over the stream and output a set of size $k \cdot \mathrm{poly}(1/\varepsilon)$ that can be post-processed using *any offline algorithm* for submodular maximization. The post-processing is itself quite straightforward: we simply run the offline algorithm on the output set to obtain a solution of size at most k. We show that, if the offline algorithm achieves α-approximation, then we obtain $\left(\frac{\alpha}{1+\alpha} - \varepsilon\right)$-approximation.

Our main result implies that if we post-process using an exact (exponential time) algorithm, we obtain $(\frac{1}{2} - \varepsilon)$-approximation. This matches the inapproximability result proven by [19] for the special case of a monotone function. Furthermore, we show that in the non-monotone case any streaming algorithm guaranteeing $(\frac{1}{2} + \varepsilon)$-approximation for some positive constant ε must use in fact $\Omega(n)$ space.[3] Thus, we essentially settle the approximability of the problem if exponential-time computation is allowed.

The best (polynomial-time) approximation guarantee that is currently known in the offline setting is $\alpha = 0.385$ [5]. If we post-process using this algorithm, we obtain 0.2779-approximation in polynomial time, improving over the previously best polynomial-time approximation of 0.1715 due to [17]. The offline algorithm of [5] is based on the multilinear extension, and thus is quite slow. One can obtain a more efficient overall algorithm by using the combinatorial random Greedy algorithm of [6] that achieves $\frac{1}{e}$-approximation. Furthermore, any existing heuristic for the offline problem can be used for post-processing, exploiting their effectiveness beyond the worst case.

Our techniques. The two streaming algorithms that we present enjoy the same approximation guarantee, but differ in other properties. Our first algorithm (StreamProcess) is a combinatorial algorithm that achieves very fast update time and overall running time. StreamProcess takes inspiration both from the single-threshold Greedy algorithm for *monotone* maximization and distributed algorithms that randomly partition the data [26, 4, 3]: it randomly partitions the elements into $1/\varepsilon$ parts as they arrive in the stream and runs

[1] Chekuri et al. [11] claimed an improved approximation ratio of $\frac{1}{2+e} - \varepsilon$ for a cardinality constraint, but an error was later found in the proof of this improved ratio [10]. We defer the details to the full version.

[2] Formally, our algorithms are semi-streaming algorithms, i.e., their space complexity is nearly linear in k. Since this is unavoidable for algorithms designed to output an approximate solution (as opposed to just estimating the value of the optimal solution), we ignore the difference between streaming and semi-streaming algorithms in this paper and use the two terms interchangably.

[3] This result is a simple adaptation of a result due to Buchbinder et al. [7]. For completeness, we include the proof in the full version of the paper.

the single-threshold Greedy algorithm on each part; this process is repeated independently and in parallel $O(\ln(1/\varepsilon)/\varepsilon)$ times. Since the main engine behind our algorithm is the very efficient and practical single-threshold Greedy algorithm, our StreamProcess algorithm inherits its very efficient update time and practical potential. Compared to the optimal streaming algorithm for monotone maximization discussed above, our algorithm is quite similar: the monotone algorithm runs $O(\log k/\varepsilon)$ instances of single-threshold Greedy, each of which processes all n items; StreamProcess runs $O(\ln(1/\varepsilon)/\varepsilon^2) \cdot O(\log k/\varepsilon)$ instances of single-threshold Greedy, each of which processes $O(\varepsilon \cdot n)$ items with high probability.

Our second algorithm (StreamProcessExtension) is based on the multilinear extension of the submodular function. This algorithm is similar to the single-threshold Greedy algorithm, but adds fractions of elements rather than whole elements to the solution it maintains. The extension based approach of this algorithm allows us to save on the space usage. Furthermore, when the multilinear extension can be evaluated deterministically, this approach leads to a deterministic algorithm. However, the time complexity of this approach depends on the complexity of evaluating the multilinear extension, which is quite high if we are only given value oracle access to f. Thus, given such restricted access, this approach leads to higher update and overall running time.

We note that combining the single-threshold Greedy with randomization is difficult because it requires delicate care of the event that the single-threshold Greedy algorithm fills up the budget. In particular, this was the source of the subtle error mentioned above in one of the results of [11]. Our approach here for handling this issue is simple in retrospect. In our combinatorial algorithm, we consider two cases depending on the probability that the budget is filled up in a run (this is a good event since the resulting solution has good value). If this probability is sufficiently large (at least ε), we repeat the basic algorithm $O(\ln(1/\varepsilon)/\varepsilon)$ times to boost the probability of this good event to $1 - \varepsilon$. Otherwise, the probability that the budget is not filled up in a run is at least $1 - \varepsilon$, and conditioning on this event changes the probabilities by only a $1 - \varepsilon$ factor.

In our extension based algorithm, the decisions of the algorithm are based on the values taken by derivatives of the extension, which are values of expectations over appropriately chosen distributions. On the one hand, this allows our algorithm to include a random component, which is a component that appears (at least implicitly) in all of the known algorithms for non-monotone submodular maximization. On the other hand, since expectations have deterministic values, the algorithm we get is deterministic enough that it suffices for us to consider at each time only one of two possible cases: the case in which the budget fills up, and the case in which it does not.

Paper structure. In Section 2, defines the notation that we use and presents some known lemmata. Section 3 presents and analyzes our combinatorial algorithm (StreamProcess). Finally, in Section 4, we present and analyze our extension based algorithm.

2 Preliminaries

Basic notation. Let V denote a finite ground set of size $n := |V|$. We occasionally assume without loss of generality that $V = \{1, 2, \dots, n\}$, and use, e.g., $x = (x_1, x_2, \dots, x_n)$ to denote a vector in $\mathbb{R}^V$. For two vectors $x, y \in \mathbb{R}^V$, we let $x \vee y$ and $x \wedge y$ be the vectors such that $(x \vee y)_e = \max\{x_e, y_e\}$ and $(x \wedge y)_e = \min\{x_e, y_e\}$ for all $e \in V$. For a set $S \subseteq V$, we let $\mathbf{1}_S$ denote the indicator vector of S, i.e., the vector that has 1 in every coordinate $e \in S$ and 0 in every coordinate $e \in V \setminus S$. Given an element $e \in V$, we use $\mathbf{1}_e$ as a shorthand for $\mathbf{1}_{\{e\}}$. Furthermore, if S is a random subset of V, we use $\mathbb{E}[\mathbf{1}_S]$ to denote the vector p such that $p_e = \Pr[e \in S]$ for all $e \in V$ (i.e., the expectation is applied coordinate-wise).

Submodular functions. In this paper, we consider the problem of maximizing a non-negative submodular function subject to a cardinality constraint. A set function $f\colon 2^V \to \mathbb{R}$ is submodular if $f(A) + f(B) \geq f(A \cap B) + f(A \cup B)$ for all subsets $A, B \subseteq V$.

Continuous extensions. We make use of two standard continuous extensions of submodular functions. The first of these extensions is known as the *multilinear extension*. To define this extension, we first need to define the random set $\mathtt{R}(x)$. For every vector $x \in [0,1]^V$, $\mathtt{R}(x)$ is defined as a random subset of V that includes every element $e \in V$ with probability x_e, independently. The multilinear extension F of f is now defined for every $x \in [0,1]^V$ by

$$F(x) = \mathbb{E}\big[f\big(\mathtt{R}(x)\big)\big] = \sum_{A \subseteq V} f(A) \cdot \Pr[\mathtt{R}(x) = A] = \sum_{A \subseteq V} \left(f(A) \cdot \prod_{e \in A} x_e \cdot \prod_{e \notin A} (1 - x_e) \right) .$$

One can observe from the definition that F is indeed a multilinear function of the coordinates of x, as suggested by its name. Thus, if we use the shorthand $\partial_e F(x)$ for the first partial derivative $\frac{\partial F(x)}{\partial x_e}$ of the multilinear extension F, then $\partial_e F(x) = F(x \vee \mathbf{1}_e) - F\big(x \wedge \mathbf{1}_{V \setminus \{e\}}\big)$.

In the analysis of our extension based algorithm, we need an upper bound on the possible increase in the value of $F(x)$ when some of the indices of x are zeroed. Corollary 2 provides such an upper bound. It readily follows from the following known lemma by Buchbinder et al. [6].

▶ **Lemma 1** (Lemma 2.2 from [6]). *Let $f\colon 2^V \to \mathbb{R}_{\geq 0}$ be a non-negative submodular function. Denote by $A(p)$ a random subset of $A \subseteq V$ where each element appears with probability at most p (not necessarily independently). Then, $\mathbb{E}[f(A(p))] \geq (1-p) \cdot f(\varnothing)$.*

In the statement of Corollary 2, and in the rest of the paper, we denote by $\operatorname{supp}(x)$ the support of vector x, i.e., the set $\{e \in V \mid x_e > 0\}$.

▶ **Corollary 2.** *Let $f\colon 2^V \to \mathbb{R}_{\geq 0}$ be a non-negative submodular function, let p be a number in the range $[0,1]$ and let $x, y \in [0,1]^V$ be two vectors such that $\operatorname{supp}(x) \cap \operatorname{supp}(y) = \varnothing$ and $y_e \leq p$ for every $e \in V$. Then, the multilinear extension F of f obeys $F(x+y) \geq (1-p) \cdot F(x)$.*

The analyses of both our algorithms make use of the *Lovász extension* $\hat{f}$ of f. The Lovász extension $\hat{f}\colon [0,1]^V \to \mathbb{R}$ is defined as follows. For every $x \in [0,1]^V$, $\hat{f}(x) = \mathbb{E}_{\theta \sim [0,1]}[f(\{e \in V\colon x_e \geq \theta\})]$, where we use the notation $\theta \sim [0,1]$ to denote a value chosen uniformly at random from the interval $[0,1]$. The Lovász extension $\hat{f}$ of a non-negative submodular function has the following properties: (1) convexity: $c\hat{f}(x) + (1-c)\hat{f}(y) \geq \hat{f}(cx + (1-c)y)$ for all $x, y \in [0,1]^V$ and all $c \in [0,1]$ [25]; (2) restricted scale invariance: $\hat{f}(cx) \geq c\hat{f}(x)$ for all $x \in [0,1]^V$ and all $c \in [0,1]$; (3) it lower bounds the multilinear extension, i.e., $F(x) \geq \hat{f}(x)$ for every $x \in [0,1]^V$ [32, Lemma A.4].

3 Combinatorial Algorithm

Our combinatorial streaming algorithm is shown in Algorithm 1. For simplicity, we describe the algorithm assuming the knowledge of an estimate of the value of the optimal solution, $f(\mathrm{OPT})$. To remove this assumption, we use the standard technique introduced by [2]. The basic idea is to use the maximum singleton value $v = \max_e f(\{e\})$ as a k-approximation of $f(\mathrm{OPT})$. Given this approximation, one can guess a $1+\varepsilon$ approximation of $f(\mathrm{OPT})$ from a set of $O(\log(k/\alpha)/\varepsilon)$ values ranging from v to kv/α (α is the approximation guarantee of the offline algorithm OfflineAlg that we use in the post-processing step). The final streaming algorithm is simply $O(\log(k/\alpha)/\varepsilon)$ copies of the basic algorithm running in parallel with different guesses. As new elements appear in the stream, the value $v = \max_e f(\{e\})$ also increases over time and thus, existing copies of the basic algorithm with small guesses are dropped and new copies with higher guesses are added. An important observation is that when we introduce a new copy with a large guess, starting it from mid-stream has exactly the same outcome as if we started it from the beginning of the stream: all previous elements have marginal gain much smaller than the guess and smaller than the threshold so they would have been rejected anyway. We refer to [2] for the full details.

▶ **Theorem 3.** *There is a streaming algorithm* StreamProcess *for non-negative, non-monotone submodular maximization with the following properties ($\varepsilon > 0$ is any desired accuracy and it is given as input to the algorithm):*

- *The algorithm makes a single pass over the stream.*
- *The algorithm uses $O\left(\frac{k \log(k/\alpha)\log(1/\varepsilon)}{\varepsilon^3}\right)$ space.*
- *The update time per item is $O\left(\frac{\log(k/\alpha)\log(1/\varepsilon)}{\varepsilon^2}\right)$ marginal gain computations.*

At the end of the stream, we post-process the output of StreamProcess *using any offline algorithm* OfflineAlg *for submodular maximization. The resulting solution is a $\frac{\alpha}{1+\alpha} - \varepsilon$ approximation, where α is the approximation of* OfflineAlg*.*

Algorithm 1 Streaming algorithm for $\max_{|S| \le k} f(S)$. PostProcess uses any offline algorithm OfflineAlg with approximation α. Lines shown in blue are comments. The algorithm does **not** store the sets $V_{i,j}$, they are defined for analysis purposes only.

```
StreamProcess($f, k, \varepsilon, \kappa$)
$r \leftarrow \Theta(\ln(1/\varepsilon)/\varepsilon)$
$m \leftarrow 1/\varepsilon$
$S_{i,j} \leftarrow \varnothing$ for all $i \in [r], j \in [m]$
$V_{i,j} \leftarrow \varnothing$ for all $i \in [r], j \in [m]$ // not stored, defined for analysis purposes only
for each arriving element $e$ do
    for $i = 1$ to $r$ do
        choose an index $j \in [m]$ uniformly and independently at random
        $V_{i,j} \leftarrow V_{i,j} \cup \{e\}$ // not stored, defined for analysis purposes only
        if $f(S_{i,j} \cup \{e\}) - f(S_{i,j}) \ge \kappa$ and $|S_{i,j}| < k$ then
            $S_{i,j} \leftarrow S_{i,j} \cup \{e\}$
return $\{S_{i,j} : i \in [r], j \in [m]\}$

PostProcess($f, k, \varepsilon$)
$\kappa \leftarrow \frac{\alpha}{1+\alpha} \cdot \frac{1}{k} \cdot f(\mathrm{OPT})$ // threshold
$\{S_{i,j}\} \leftarrow$ StreamProcess($f, k, \varepsilon, \kappa$)
if $|S_{i,j}| = k$ for some $i$ and $j$ then
    return $S_{i,j}$
else
    $U \leftarrow \bigcup_{i,j} S_{i,j}$
    $T \leftarrow$ OfflineAlg($f, k, U$)
    return $\arg\max\{f(S_{1,1}), f(T)\}$
```

Algorithm 2 Single threshold Greedy algorithm. The algorithm processes the elements in the order in which they arrive in the stream, and it uses the same threshold κ as StreamProcess.

```
STGreedy($f, N, k, \kappa$):
$S \leftarrow \varnothing$
for each $e \in N$ in the stream order do
    if $f(S \cup \{e\}) - f(S) \geq \kappa$ and $|S| < k$ then
        $S \leftarrow S \cup \{e\}$
return $S$
```

In the remainder of this section, we analyze Algorithm 1 and show that it achieves a $\frac{\alpha}{1+\alpha} - \varepsilon$ approximation, where α is the approximation guarantee of the offline algorithm OfflineAlg.

We divide the analysis into two cases, depending on the probability of the event that a set $S_{i,1}$ (for some $i \in [r]$) constructed by StreamProcess has size k. For every $i \in [r]$, let $\mathcal{F}_i$ be the event that $|S_{i,1}| = k$. Since each of the r repetitions (iterations of the for loop of StreamProcess) use independent randomness to partition V, the events $\mathcal{F}_1, \dots, \mathcal{F}_r$ are independent. Additionally, the events $\mathcal{F}_1, \dots, \mathcal{F}_r$ have the same probability. We divide the analysis into two cases, depending on whether $\Pr[\mathcal{F}_1] \geq \varepsilon$ or $\Pr[\mathcal{F}_1] < \varepsilon$. In the first case, since we are repeating $r = \Theta(\ln(1/\varepsilon)/\varepsilon)$ times, the probability that there is a set $S_{i,j}$ of size k is at least $1 - \varepsilon$, and we obtain the desired approximation since $f(S_{i,j}) \geq \kappa\,|S_{i,j}| = \kappa k = \frac{\alpha}{1+\alpha} f(\mathrm{OPT})$. In the second case, we have $\Pr\left[\,\overline{\mathcal{F}_1}\,\right] \geq 1 - \varepsilon$ and we argue that $\bigcup_{i,j} S_{i,j}$ contains a good solution. We now give the formal argument for each of the cases.

The case $\Pr[\mathcal{F}_1] \geq \varepsilon$

As noted earlier, the events $\mathcal{F}_1, \dots, \mathcal{F}_r$ are independent and have the same probability. Thus,

$$\Pr\left[\,\overline{\mathcal{F}_1 \cup \cdots \cup \mathcal{F}_r}\,\right] \leq (1-\varepsilon)^r \leq \exp(-\varepsilon r) \leq \varepsilon$$

since $r = \Theta(\ln(1/\varepsilon)/\varepsilon)$. Thus $\Pr[\mathcal{F}_1 \cup \cdots \cup \mathcal{F}_r] \geq 1 - \varepsilon$.

Conditioned on the event $\mathcal{F}_1 \cup \cdots \cup \mathcal{F}_r$, we obtain the desired approximation due to the following lemma. The lemma follows from the fact that the marginal gain of each selected element is at least κ.

▶ **Lemma 4.** *We have $f(S_{i,j}) \geq \kappa\,|S_{i,j}|$ for all $i \in [r]$, $j \in [m]$.*

We can combine the two facts and obtain the desired approximation as follows. Let $\mathcal{S}$ be the random variable equal to the solution returned by PostProcess. We have

$$\mathbb{E}[f(\mathcal{S})] \geq \mathbb{E}[f(\mathcal{S})|\mathcal{F}_1 \cup \cdots \cup \mathcal{F}_r]\Pr[\mathcal{F}_1 \cup \cdots \cup \mathcal{F}_r] \geq (1-\varepsilon)\kappa k = (1-\varepsilon)\frac{\alpha}{1+\alpha} f(\mathrm{OPT})$$

The case $\Pr[\mathcal{F}_1] < \varepsilon$

In this case, we show that the solution $\arg\max\{f(T), f(S_{1,1})\}$ returned on the last line of PostProcess has good value in expectation. Our analysis borrows ideas and techniques from the work of Barbosa et al. [3]: the probabilities p_e defined below are analogous to the probabilities used in that work; the division of OPT into two sets based on these probabilities is analogous to the division employed in Section 7.3 in that work; Lemma 6 shows a consistency property for the single threshold greedy algorithm that is analogous to the consistency property shown for the standard greedy algorithm and other algorithms by Barbosa *et al.* Barbosa *et al.* use these concepts in a different context (specifically, *monotone* maximization in the *distributed* setting). When applied to our context – *non-monotone* maximization in the *streaming* setting – the framework of Barbosa *et al.* requires $\Omega(\sqrt{nk})$ memory if used with a single pass (alternatively, they use $\Omega(\min\{k, 1/\varepsilon\})$ passes) and achieves worse approximation guarantees.

Notation and definitions. For analysis purposes only, we make use of the Lovasz extension $\hat{f}$. We fix an optimal solution $\mathrm{OPT} \in \arg\max\{f(A) \colon A \subseteq V, |A| \leq k\}$. Let $\mathcal{V}(1/m)$ be the distribution of $1/m$-samples of V, where a $1/m$-sample of V includes each element of V independently at random with probability $1/m$. Note that $V_{i,j} \sim \mathcal{V}(1/m)$ for every $i \in [r]$, $j \in [m]$ (see StreamProcess). Additionally, for each $i \in [r]$, $V_{i,1}, \ldots, V_{i,m}$ is a partition of V into $1/m$-samples.

For a subset $N \subseteq V$, we let STGreedy(N) be the output of the single threshold greedy algorithm when run as follows (see also Algorithm 2 for a formal description of the algorithm): the algorithm processes the elements of N *in the order in which they arrive in the stream* and it uses the same threshold κ as StreamProcess; starting with the empty solution and continuing until the size constraint of k is reached, the algorithm adds an element to the current solution if its marginal gain is above the threshold. Note that $S_{i,j} = \mathrm{STGreedy}(V_{i,j})$ for all $i \in [r], j \in [m]$. For analysis purposes only, we also consider STGreedy(N) for sets N that do not correspond to any set $V_{i,j}$.

For each $e \in V$, we define

$$p_e = \begin{cases} \Pr_{X \sim \mathcal{V}(1/m)}\left[e \in \mathrm{STGreedy}(X \cup \{e\})\right] & \text{if } e \in \mathrm{OPT} \\ 0 & \text{otherwise} \end{cases}$$

We partition OPT into two sets:

$$O_1 = \{e \in \mathrm{OPT} \colon p_e \geq \varepsilon\} \quad O_2 = \mathrm{OPT} \setminus O_1$$

We also define the following subset of O_2:

$$O_2' = \{e \in O_2 \colon e \notin \mathrm{STGreedy}\left(V_{1,1} \cup \{e\}\right)\}.$$

Note that (O_1, O_2) is a deterministic partition of OPT, whereas O_2' is a random subset of O_2. The role of the sets O_1, O_2, O_2' will become clearer in the analysis. The intuition is that, using the repetition, we can ensure that each element of O_1 ends up in the collected set $U = \bigcup_{i,j} S_{i,j}$ with good probability: each iteration $i \in [r]$ ensures that an element $e \in O_1$ is in $S_{i,1} \cup \cdots \cup S_{i,m}$ with probability $p_e \geq \varepsilon$ and, since we repeat $r = \Theta(\ln(1/\varepsilon)/\varepsilon)$ times, we will ensure that $\mathbb{E}[\mathbf{1}_{O_1 \cap U}] \geq (1-\varepsilon)\mathbf{1}_{O_1}$. We also have that $\mathbb{E}\left[\mathbf{1}_{O_2'}\right] \geq (1-\varepsilon)\mathbf{1}_{O_2}$: an element $e \in O_2 \setminus O_2'$ ends up being picked by STGreedy when run on input $V_{1,1} \cup \{e\}$, which is a low probability event for the elements in O_2; more precisely, the probability of this event is equal to p_e (since $V_{1,1} \sim \mathcal{V}(1/m)$) and $p_e \leq \varepsilon$ (since $e \in O_2$). Thus $\mathbb{E}\left[\mathbf{1}_{(O_1 \cap U) \cup O_2'}\right] \geq (1-\varepsilon)\mathbf{1}_{\mathrm{OPT}}$, which implies that the expected value of $(O_1 \cap U) \cup O_2'$ is at least $(1-\varepsilon)f(\mathrm{OPT})$. However, whereas $O_1 \cap U$ is available in the post-processing phase, elements of O_2' may not be available and they may account for most of the value of O_2. The key insight is to show that $S_{1,1}$ makes up for the lost value from these elements.

We start the analysis with two helper lemmas, which follow from standard arguments that have been used in previous works. The first of these lemmas follows from an argument based on the Lovasz extension and its properties.

▶ **Lemma 5.** *Let* $0 \leq u \leq v \leq 1$*. Let* $S \subseteq V \setminus \mathrm{OPT}$ *and* $O \subseteq \mathrm{OPT}$ *be random sets such that* $\mathbb{E}[\mathbf{1}_S] \leq u\mathbf{1}_{V\setminus\mathrm{OPT}}$ *and* $\mathbb{E}[\mathbf{1}_O] \geq v\mathbf{1}_{\mathrm{OPT}}$*. Then* $\mathbb{E}[f(S \cup O)] \geq (v-u)f(\mathrm{OPT})$*.*

The following lemma establishes a consistency property for the STGreedy algorithm, analogous to the consistency property shown and used by Barbosa et al. for algorithms such as the standard Greedy algorithm. The proof is also very similar to the proof shown by Barbosa et al.

▶ **Lemma 6.** *Conditioned on the event* $|S_{1,1}| < k$*, we have* $\textit{STGreedy}(V_{1,1} \cup O_2') = \textit{STGreedy}(V_{1,1}) = S_{1,1}$*.*

We now proceed with the main analysis. Recall that PostProcess runs the algorithm OfflineAlg on U to obtain a solution T, and returns the better of the two solutions $S_{1,1}$ and T. In the following lemma, we show that the value of this solution is proportional to $f(S_{1,1} \cup (O_1 \cap U))$. Note that $S_{1,1} \cup (O_1 \cap U)$ may not be feasible, since we could have $|S_{1,1}| > |O_2|$, and hence the scaling based on $\frac{|O_2|}{k}$.

▶ **Lemma 7.** *We have* $\max\{f(S_{1,1}), f(T)\} \geq \frac{\alpha}{1+\alpha\left(1-\frac{|O_2|}{k}\right)} f(S_{1,1} \cup (O_1 \cap U))$*.*

Proof. To simplify notation, we let $S_1 = S_{1,1}$. Let $b = |O_2|$. First, we analyze $f(T)$. Let $X \subseteq S_1$ be a random subset of S_1 such that $|X| \leq b$ and $\mathbb{E}[\mathbf{1}_X] = \frac{b}{k}\mathbf{1}_{S_1}$. We can select such a subset as follows: we first choose a permutation of S_1 uniformly at random, and let $\tilde{X}$ be the first $s := \min\{b, |S_1|\}$ elements in the permutation. For each element of $\tilde{X}$, we add it to X with probability $p := |S_1|b/(sk)$. For each $e \in S_1$, we have

$$\Pr[e \in X] = \Pr\left[e \in X | e \in \tilde{X}\right] \Pr\left[e \in \tilde{X}\right] = p\frac{s}{|S_1|} = \frac{b}{k}$$

For each $e \notin S_1$, we have $\Pr[e \in X] = 0$. Thus $\mathbb{E}[\mathbf{1}_X] = \frac{b}{k}\mathbf{1}_{S_1}$.

Since $X \cup ((O_1 \cap U) \setminus S_1)$ is a feasible solution contained in U and OfflineAlg achieves an α-approximation, we have

$$f(T) \geq \alpha f(X \cup ((O_1 \cap U) \setminus S_1))$$

By taking expectation over X only (more precisely, the random sampling that we used to select X) and using that $\hat{f}$ is a convex extension, we obtain:

$$\begin{aligned} f(T) &\geq \alpha\mathbb{E}_X\left[f(X \cup ((O_1 \cap U) \setminus S_1))\right] = \alpha\mathbb{E}_X\left[\hat{f}\left(\mathbf{1}_{X\cup((O_1\cap U)\setminus S_1)}\right)\right] \\ &\geq \alpha\hat{f}\left(\mathbb{E}_X\left[\mathbf{1}_{X\cup((O_1\cap U)\setminus S_1)}\right]\right) = \alpha\hat{f}\left(\frac{b}{k}\mathbf{1}_{S_1} + \mathbf{1}_{(O_1\cap U)\setminus S_1}\right) \end{aligned}$$

Next, we lower bound $\max\{f(S_1), f(T)\}$ using a convex combination $(1-\theta)f(S_1) + \theta f(T)$ with coefficient $\theta = 1/(1+\alpha\left(1-\frac{b}{k}\right))$. Note that $1-\theta = \theta\alpha\left(1-\frac{b}{k}\right)$. By taking this convex combination, using the previous inequality lower bounding $f(T)$, and the convexity and restricted scale invariance of $\hat{f}$, we obtain:

$$\begin{aligned}
&\max\{f(S_1), f(T)\} \geq (1-\theta)f(S_1) + \theta f(T) = \theta\alpha\left(1-\frac{b}{k}\right)f(S_1) + \theta f(T) \\
&\geq \theta\alpha\left(1-\frac{b}{k}\right)\hat{f}\left(\mathbf{1}_{S_1}\right) + \theta\alpha\hat{f}\left(\frac{b}{k}\mathbf{1}_{S_1} + \mathbf{1}_{(O_1\cap U)\setminus S_1}\right) \\
&= \theta\alpha\left(2-\frac{b}{k}\right)\left(\frac{1-\frac{b}{k}}{2-\frac{b}{k}}\hat{f}\left(\mathbf{1}_{S_1}\right) + \frac{1}{2-\frac{b}{k}}\hat{f}\left(\frac{b}{k}\mathbf{1}_{S_1} + \mathbf{1}_{(O_1\cap U)\setminus S_1}\right)\right) \\
&\geq \theta\alpha\left(2-\frac{b}{k}\right)\hat{f}\left(\frac{1-\frac{b}{k}}{2-\frac{b}{k}}\mathbf{1}_{S_1} + \frac{1}{2-\frac{b}{k}}\left(\frac{b}{k}\mathbf{1}_{S_1} + \mathbf{1}_{(O_1\cap U)\setminus S_1}\right)\right) \\
&= \theta\alpha\left(2-\frac{b}{k}\right)\hat{f}\left(\frac{1}{2-\frac{b}{k}}\mathbf{1}_{S_1\cup(O_1\cap U)}\right) \geq \frac{\alpha}{1+\alpha\left(1-\frac{b}{k}\right)}f(S_1\cup(O_1\cap U)). \qquad \blacktriangleleft
\end{aligned}$$

(We note that we chose θ to make the coefficients of $\mathbf{1}_{S_1}$ and $\mathbf{1}_{(O_1\cap U)\setminus S_1}$ equal, and this allowed us to relate the value of the final solution to $f(S_1\cup(O_1\cap U))$.)

Next, we analyze the expected value of $f(S_{1,1}\cup(O_1\cap U))$. We do so in two steps: first we analyze the marginal gain of O_2' on top of $S_{1,1}$ and show that it is suitably small, and then we analyze $f(S_{1,1}\cup(O_1\cap U)\cup O_2')$ and show that its expected value is proportional to $f(\mathrm{OPT})$. We use the notation $f(A|B)$ to denote the marginal gain of A on top of B, i.e., $f(A|B) = f(A\cup B) - f(B)$.

▶ **Lemma 8.** *We have* $\mathbb{E}[f\left(O_2'|S_{1,1}\right)] \leq \kappa b + \varepsilon f(\mathrm{OPT})$.

Proof. As before, to simplify notation, we let $S_1 = S_{1,1}$ and $V_1 = V_{1,1}$. We break down the expectation using the law of total expectation as follows:

$$\begin{aligned}
\mathbb{E}[f\left(O_2'|S_1\right)] &= \mathbb{E}[f\left(O_2'|S_1\right)\,|\,|S_1|<k]\cdot\underbrace{\Pr[|S_1|<k]}_{\leq 1} + \underbrace{\mathbb{E}[f\left(O_2'|S_1\right)\,|\,|S_1|=k]}_{\leq f(\mathrm{OPT})}\cdot\underbrace{\Pr[|S_1|=k]}_{\leq\varepsilon} \\
&\leq \mathbb{E}[f\left(O_2'|S_1\right)\,|\,|S_1|<k] + \varepsilon f(\mathrm{OPT})
\end{aligned}$$

Above, we have used that $f(O_2'|S_1) \leq f(O_2') \leq f(\mathrm{OPT})$, where the first inequality follows by submodularity. We have also used that $\Pr[|S_1| = k] = \Pr[\mathcal{F}_1] \leq \varepsilon$. Thus it only remains to show that $\mathbb{E}[f\left(O_2'|S_1\right)\,|\,|S_1|<k] \leq \kappa b$.

We condition on the event $|S_1| < k$ for the remainder of the proof. By Lemma 6, we have $\textsc{STGreedy}(V_1\cup O_2') = S_1$. Since $|S_1| < k$, each element of $O_2'\setminus S_1$ was rejected because its marginal gain was below the threshold when it arrived in the stream. This, together with submodularity, implies that $f\left(O_2'|S_1\right) \leq \kappa\,|O_2'| \leq \kappa b$. ◀

▶ **Lemma 9.** *We have* $\mathbb{E}[f(S_{1,1}\cup(O_1\cap U)\cup O_2')] \geq (1-2\varepsilon)f(\mathrm{OPT})$.

Proof. We apply Lemma 5 to the following sets:

$$\begin{aligned}
&S = S_{1,1}\setminus\mathrm{OPT} \\
&O = (S_{1,1}\cap\mathrm{OPT})\cup(O_1\cap U)\cup O_2'
\end{aligned}$$

We show below that $\mathbb{E}[\mathbf{1}_O] \leq \varepsilon\mathbf{1}_{V\setminus\mathrm{OPT}}$ and $\mathbb{E}[\mathbf{1}_O] \geq (1-\varepsilon)\mathbf{1}_{\mathrm{OPT}}$. Assuming these bounds, we can take $u = \varepsilon$ and $v = 1-\varepsilon$ in Lemma 5, which gives the desired result.

Since $S \subseteq S_{1,1} \subseteq V_{1,1}$ and $V_{1,1}$ is a $(1/m)$-sample of V, we have $\mathbb{E}[\mathbf{1}_S] \leq \frac{1}{m}\mathbf{1}_{V\setminus\mathrm{OPT}} = \varepsilon\mathbf{1}_{V\setminus\mathrm{OPT}}$. Thus it only remains to show that, for each $e\in\mathrm{OPT}$, we have $\Pr[e\in O] \geq 1-\varepsilon$. Since $(O_1\cap U)\cup O_2' \subseteq O$, it suffices to show that $\Pr[e\in(O_1\cap U)\cup O_2'] \geq 1-\varepsilon$, or equivalently that $\Pr[e\in(O_1\setminus U)\cup(O_2\setminus O_2')] \leq \varepsilon$.

Recall that (O_1, O_2) is a deterministic partition of OPT. Thus e belongs to exactly one of O_1 and O_2 and we consider each of these cases in turn.

Suppose that $e \in O_1$. A single iteration of the for loop of StreamProcess ensures that e is in $S_{i,1} \cup \cdots \cup S_{i,m}$ with probability $p_e \geq \varepsilon$. Since we perform $r = \Theta(\ln(1/\varepsilon)/\varepsilon)$ independent iterations, we have $\Pr[e \notin U] \leq (1-\varepsilon)^r \leq \exp(-\varepsilon r) \leq \varepsilon$.

Suppose that $e \in O_2$. We have

$$\Pr[e \in O_2 \setminus O_2'] = \Pr[e \in \text{STGreedy}\,(V_{1,1} \cup \{e\})] = p_e \leq \varepsilon$$

where the first equality follows from the definition of O_2', the second equality follows from the definition of p_e and the fact that $V_{1,1} \sim \mathcal{V}(1/m)$, and the inequality follows from the definition of O_2. ◀

Lemmas 8 and 9 immediately imply the following:

▶ **Lemma 10.** *We have* $\mathbb{E}[f\,(S_{1,1} \cup (O_1 \cap U))] \geq (1-3\varepsilon)f(\text{OPT}) - \kappa b$.

Finally, Lemmas 7 and 10 give the approximation guarantee:

▶ **Lemma 11.** *We have* $\mathbb{E}[\max\{f(S_{1,1}), f(T)\}] \geq \left(\frac{\alpha}{1+\alpha} - 3\varepsilon\right) f(\text{OPT})$.

4 Extension based algorithm

Using our extension based algorithm, we prove the following theorem.

▶ **Theorem 12.** *Assume there exists an α-approximation offline algorithm OfflineAlg for maximizing a non-negative submodular function subject to cardinality constraint whose space complexity is nearly linear in the size of the ground set. Then, for every constant $\varepsilon \in (0,1]$, there exists an $(\frac{\alpha}{1+\alpha} - \varepsilon)$-approximation semi-streaming algorithm for maximizing a non-negative submodular function subject to a cardinality constraint. The algorithm stores at most $O(k\varepsilon^{-2})$ elements.*[4]

In this section, we introduce a simplified version of the algorithm used to prove Theorem 12. This simplified version (given as Algorithm 3) captures our main new ideas, but makes two simplifying assumptions that can be avoided using standard techniques.

- The first assumption is that Algorithm 3 has access to an estimate τ of $f(\text{OPT})$ obeying $(1-\varepsilon/8) \cdot f(\text{OPT}) \leq \tau \leq f(\text{OPT})$. Such an estimate can be produced using well-known techniques, at the cost of a slight increase in the space complexity of the algorithm. In the full version of this paper we formally show that one such technique due to [21] can be used for that purpose, and that it increases the space complexity of the algorithm only by a factor of $O(\varepsilon^{-1} \log \alpha^{-1})$.
- The second assumption is that Algorithm 3 has value oracle access to the multilinear extension F. If the time complexity of Algorithm 3 is not important, then this assumption is of no consequence since a value oracle query to F can be emulated using an exponential number of value oracle queries to f. However, the assumption becomes problematic when we would like to keep the time complexity of the algorithm polynomial and we only have value oracle access to f. Thus, we explain in the full version of this paper how to drop this

[4] Formally, the number of elements stored by the algorithm also depends on $\log \alpha^{-1}$. Since α is typically a positive constant, or at least lower bounded by a positive constant, we omit this dependence from the statement of the theorem.

assumption via sampling. Interestingly, the rounding step and this sampling technique are the only parts of the extension based algorithm that employ randomness. Since the rounding can be made deterministic given either exponential time or value oracle access to F, we get the following observation.

▶ **Observation 13.** *If* OfflineAlg *is deterministic, then the algorithm whose existence is guaranteed by Theorem 12 is also* deterministic *when it is allowed either exponential computation time or value oracle access to F.*

Algorithm 3 has two constant parameters $p \in (0, 1)$ and $c > 0$ and maintains a fractional solution $x \in [0, 1]^V$. This fractional solution starts empty, and the algorithm adds to it fractions of elements as they arrive. Specifically, when an element e arrives, the algorithm considers its marginal contribution with respect to the current fractional solution x. If this marginal contribution exceeds the threshold of $c\tau/k$, then the algorithm tries to add to x a p-fraction of e, but might end up adding a smaller fraction of e if adding a full p-fraction of e to x will make x an infeasible solution, i.e., make $\|x\|_1 > k$ (note that $\|x\|_1$ is the sum of the coordinates of x).

After viewing all of the elements, Algorithm 3 uses the fractional solution x to generate two sets S_1 and S_2 that are feasible (integral) solutions. The set S_1 is generated by rounding the fractional solution x. Two rounding procedures, named Pipage Rounding and Swap Rounding, were suggested for this task in the literature [8, 12]. Both procedures run in polynomial time and guarantee that the output set S_1 of the rounding is always feasible, and that its expected value with respect to f is at least the value $F(x)$ of the fractional solution x. The set S_2 is generated by applying OfflineAlg to the support of the vector x, which produces a feasible solution that (approximately) maximizes f among all subsets of the support whose size is at most k. After computing the two feasible solutions S_1 and S_2, Algorithm 3 simply returns the better one of them.

Algorithm 3 StreamProcessExtension (simplified) (p, c).

```
Let $x \leftarrow \mathbf{1}_\varnothing$.
for each arriving element $e$ do
    if $\partial_e F(x) \geq \frac{c\tau}{k}$ then $x \leftarrow x + \min\{p, k - \|x\|_1\} \cdot \mathbf{1}_e$.
Round the vector $x$ to yield a feasible solution $S_1$ such that $\mathbb{E}[f(S_1)] \geq F(x)$.
Find another feasible solution $S_2 \subseteq \mathrm{supp}(x)$ by running OfflineAlg with $\mathrm{supp}(x)$
  as the ground set.
return the better solution among $S_1$ and $S_2$.
```

Let us denote by $\hat{x}$ the final value of the fractional solution x (i.e., its value when the stream ends). We begin the analysis of Algorithm 3 with the following useful observation.

▶ **Observation 14.** *If $\|\hat{x}\|_1 < k$, then $\hat{x}_e = p$ for every $e \in \mathrm{supp}(\hat{x})$. Otherwise (when $\|\hat{x}\|_1 = k$), this is still true for every element $e \in \mathrm{supp}(\hat{x})$ except for maybe a single element.*

Proof. For every element e added to the support of x by Algorithm 3, the algorithm sets x_e to p unless this will make $\|x\|_1$ exceed k, in which case the algorithm set x_e to be the value that will make $\|x\|_1$ equal to k. Thus, after a single coordinate of x is set to a value other than p (or the initial 0), $\|x\|_1$ becomes k and Algorithm 3 stops changing x. ◀

Using the last observation we can now bound the space complexity of Algorithm 3, and show (in particular) that it is a semi-streaming algorithm for a constant p when the space complexity of OfflineAlg is nearly linear.

▶ **Observation 15.** *Algorithm 3 can be implemented so that it stores at most $O(k/p)$ elements.*

Proof. To calculate the sets S_1 and S_2, Algorithm 3 needs access only to the elements of V that appear in the support of x. Thus, the number of elements it needs to store is $O(|\operatorname{supp}(\hat{x})|) = O(k/p)$, where the equality follows from Observation 14. ◀

We now divert our attention to analyzing the approximation ratio of Algorithm 3. The first step in this analysis is lower bounding the value of $F(\hat{x})$, which we do by considering two cases, one when $\|\hat{x}\|_1 = k$, and the other when $\|\hat{x}\|_1 < k$. The following lemma bounds the value of $F(\hat{x})$ in the first of these cases. Intuitively, this lemma holds since $\operatorname{supp}(\hat{x})$ contains many elements, and each one of these elements must have increased the value of $F(x)$ significantly when added (otherwise, Algorithm 3 would not have added this element to the support of x).

▶ **Lemma 16.** *If $\|\hat{x}\|_1 = k$, then $F(\hat{x}) \geq c\tau$.*

Proof. Denote by $e_1, e_2, \ldots, e_\ell$ the elements in the support of $\hat{x}$, in the order of their arrival. Using this notation, the value of $F(\hat{x})$ can be written as follows.

$$\begin{aligned} F(\hat{x}) &= F(\mathbf{1}_\varnothing) + \sum_{i=1}^{\ell} \left(F\big(\hat{x} \wedge \mathbf{1}_{\{e_1, e_2, \ldots, e_i\}}\big) - F\big(\hat{x} \wedge \mathbf{1}_{\{e_1, e_2, \ldots, e_{i-1}\}}\big) \right) \\ &= F(\mathbf{1}_\varnothing) + \sum_{i=1}^{\ell} \left(\hat{x}_{e_i} \cdot \partial_{e_i} F\big(\hat{x} \wedge \mathbf{1}_{\{e_1, e_2, \ldots, e_{i-1}\}}\big) \right) \\ &\geq F(\mathbf{1}_\varnothing) + \frac{c\tau}{k} \cdot \sum_{i=1}^{\ell} \hat{x}_{e_i} = F(\mathbf{1}_\varnothing) + \frac{c\tau}{k} \cdot \|\hat{x}\|_1 \geq c\tau \enspace , \end{aligned}$$

where the second equality follows from the multilinearity of F, and the first inequality holds since Algorithm 3 selects an element e_i only when $\partial_{e_i} F\big(\hat{x} \wedge \mathbf{1}_{\{e_1, e_2, \ldots, e_{i-1}\}}\big) \geq \frac{c\tau}{k}$. The last inequality holds since f (and thus, also F) is non-negative and $\|\hat{x}\|_1 = k$ by the assumption of the lemma. ◀

Consider now the case in which $\|\hat{x}\|_1 < k$. Recall that our objective is to lower bound $F(\hat{x})$ in this case as well. Towards this goal, we bound the expression $F(\hat{x} + \mathbf{1}_{\mathrm{OPT} \setminus \operatorname{supp}(\hat{x})})$ from below and above in the following two lemmata.

▶ **Lemma 17.** *If $\|\hat{x}\|_1 < k$, then $F\big(\hat{x} + \mathbf{1}_{\mathrm{OPT} \setminus \operatorname{supp}(\hat{x})}\big) \geq (1-p) \cdot \big[p \cdot f(\mathrm{OPT}) + (1-p) \cdot f\big(\mathrm{OPT} \setminus \operatorname{supp}(\hat{x})\big)\big]$.*

Proof. Since $\|\hat{x}\|_1 < k$, Observation 14 guarantees that $\hat{x}_e = p$ for every $e \in \operatorname{supp}(\hat{x})$. Thus $\hat{x} = p \cdot \mathbf{1}_{\mathrm{OPT} \cap \operatorname{supp}(\hat{x})} + p \cdot \mathbf{1}_{\operatorname{supp}(\hat{x}) \setminus \mathrm{OPT}}$, and therefore,

$$\begin{aligned} F\big(\hat{x} + \mathbf{1}_{\mathrm{OPT} \setminus \operatorname{supp}(\hat{x})}\big) &= F\big(p \cdot \mathbf{1}_{\mathrm{OPT} \cap \operatorname{supp}(\hat{x})} + p \cdot \mathbf{1}_{\operatorname{supp}(\hat{x}) \setminus \mathrm{OPT}} + \mathbf{1}_{\mathrm{OPT} \setminus \operatorname{supp}(\hat{x})}\big) \\ &\geq (1-p) \cdot F\big(p \cdot \mathbf{1}_{\mathrm{OPT} \cap \operatorname{supp}(\hat{x})} + \mathbf{1}_{\mathrm{OPT} \setminus \operatorname{supp}(\hat{x})}\big) \\ &\geq (1-p) \cdot \hat{f}\big(p \cdot \mathbf{1}_{\mathrm{OPT} \cap \operatorname{supp}(\hat{x})} + \mathbf{1}_{\mathrm{OPT} \setminus \operatorname{supp}(\hat{x})}\big) \\ &= (1-p) \cdot \Big[p \cdot f(\mathrm{OPT}) + (1-p) \cdot f\big(\mathrm{OPT} \setminus \operatorname{supp}(\hat{x})\big)\Big] \enspace , \end{aligned}$$

where the first inequality follows from Corollary 2, the second inequality holds since the Lovász extension lower bounds the multilinear extension, and the last equality follows from the definition of the Lovász extension. ◀

In the following lemma, and the rest of the section, we use the notation $b = k^{-1} \cdot |\mathrm{OPT} \setminus \mathrm{supp}(\hat{x})|$. Intuitively, the lemma holds since the fact that the elements of $\mathrm{OPT} \setminus \mathrm{supp}(\hat{x})$ where not added to the support of x implies that their marginal contribution is small.

▶ **Lemma 18.** *If $\|\hat{x}\|_1 < k$, then $F\big(\hat{x} + \mathbf{1}_{\mathrm{OPT} \setminus \mathrm{supp}(\hat{x})}\big) \leq F(\hat{x}) + bc\tau$.*

Proof. The elements in $\mathrm{OPT} \setminus \mathrm{supp}(\hat{x})$ were rejected by Algorithm 3, which means that their marginal contribution with respect to the fractional solution x at the time of their arrival was smaller than $c\tau/k$. Since the fractional solution x only increases during the execution of the algorithm, the submodularity of f guarantees that this is true also with respect to $\hat{x}$. More formally, we get

$$\partial_e F(\hat{x}) < \frac{c\tau}{k} \quad \forall\, e \in \mathrm{OPT} \setminus \mathrm{supp}(\hat{x}) \ .$$

Using the submodularity of f again, this implies

$$F\big(\hat{x} + \mathbf{1}_{\mathrm{OPT} \setminus \mathrm{supp}(\hat{x})}\big) \leq F(\hat{x}) + \sum_{e \in \mathrm{OPT} \setminus \mathrm{supp}(\hat{x})} \partial_e F(\hat{x}) \leq F(\hat{x}) + |\mathrm{OPT} \setminus \mathrm{supp}(\hat{x})| \cdot \frac{c\tau}{k} = F(\hat{x}) + bc\tau. \blacktriangleleft$$

Combining the last two lemmata immediately yields the promised lower bound on $F(\hat{x})$. To understand the second inequality in the following corollary, recall that $\tau \leq f(\mathrm{OPT})$.

▶ **Corollary 19.** *If $\|\hat{x}\|_1 < k$, then $F(\hat{x}) \geq (1-p) \cdot \Big[p \cdot f(\mathrm{OPT}) + (1-p) \cdot f\big(\mathrm{OPT} \setminus \mathrm{supp}(\hat{x})\big)\Big] - bc\tau \geq [p(1-p) - bc]\tau + (1-p)^2 \cdot f\big(\mathrm{OPT} \setminus \mathrm{supp}(\hat{x})\big)$.*

Our next step is to get a lower bound on the expected value of $f(S_2)$. One easy way to get such a lower bound is to observe that $\mathrm{OPT} \cap \mathrm{supp}(\hat{x})$ is a subset of the support of $\hat{x}$ of size at most k, and thus, is a candidate to be OPT; which implies $\mathbb{E}[f(S_2)] \geq \alpha \cdot f(\mathrm{OPT} \cap \mathrm{supp}(\hat{x}))$ since the algorithm OfflineAlg used to find S_2 is an α-approximation algorithm. The following lemma proves a more involved lower bound by considering the vector $(b\hat{x}) \vee \mathbf{1}_{\mathrm{OPT} \cap \mathrm{supp}(\hat{x})}$ as a fractional candidate to be OPT (using the rounding methods discussed above it, it can be converted into an integral candidate of at least the same value). The proof of the lemma lower bounds the value of the vector $(b\hat{x}) \vee \mathbf{1}_{\mathrm{OPT} \cap \mathrm{supp}(\hat{x})}$ using the concavity of the function $F((t \cdot \hat{x}) \vee \mathbf{1}_{\mathrm{OPT} \cap \mathrm{supp}(\hat{x})})$ as well as ideas used in the proofs of the previous claims.

▶ **Lemma 20.** *If $\|\hat{x}\|_1 < k$, then $\mathbb{E}[f(S_2)] \geq \alpha b(1 - p - cb)\tau + \alpha(1-b) \cdot f(\mathrm{OPT} \cap \mathrm{supp}(\hat{x}))$.*

Proof. Consider the vector $(b\hat{x}) \vee \mathbf{1}_{\mathrm{OPT} \cap \mathrm{supp}(\hat{x})}$. Clearly,

$$\begin{aligned}\big\|(b\hat{x}) \vee \mathbf{1}_{\mathrm{OPT} \cap \mathrm{supp}(\hat{x})}\big\|_1 &\leq b \cdot \|\hat{x}\|_1 + \big\|\mathbf{1}_{\mathrm{OPT} \cap \mathrm{supp}(\hat{x})}\big\|_1 \\ &\leq |\mathrm{OPT} \setminus \mathrm{supp}(\hat{x})| + |\mathrm{OPT} \cap \mathrm{supp}(\hat{x})| = |\mathrm{OPT}| \leq k \ ,\end{aligned}$$

where the second inequality holds by the definition of b since $\|\hat{x}\|_1 < k$. Thus, due to the existence of the rounding methods discussed in Section 4, there must exist a set S of size at most k obeying $f(S) \geq F((b\hat{x}) \vee \mathbf{1}_{\mathrm{OPT} \cap \mathrm{supp}(\hat{x})})$. Since S_2 is produced by OfflineAlg, whose approximation ratio is α, this implies $\mathbb{E}[f(S_2)] \geq \alpha \cdot F((b\hat{x}) \vee \mathbf{1}_{\mathrm{OPT} \cap \mathrm{supp}(\hat{x})})$. Thus, to prove the lemma it suffices to show that $F((b\hat{x}) \vee \mathbf{1}_{\mathrm{OPT} \cap \mathrm{supp}(\hat{x})})$ is always at least $b(1 - p - cb)\tau + (1-b) \cdot f(\mathrm{OPT} \cap \mathrm{supp}(\hat{x}))$.

The first step towards proving the last inequality is getting a lower bound on $F(\hat{x} \vee \mathbf{1}_{\mathrm{OPT} \cap \mathrm{supp}(\hat{x})})$. Recall that we already showed in the proof of Lemma 18 that

$$\partial_e F(\hat{x}) < \frac{c\tau}{k} \quad \forall\, e \in \mathrm{OPT} \setminus \mathrm{supp}(\hat{x}) \enspace .$$

Thus, the submodularity of f implies

$$\begin{aligned} F(\hat{x} \vee \mathbf{1}_{\mathrm{OPT}}) &\leq F(\hat{x} \vee \mathbf{1}_{\mathrm{OPT} \cap \mathrm{supp}(\hat{x})}) + \sum_{e \in \mathrm{OPT} \setminus \mathrm{supp}(\hat{x})} \partial_e F(\hat{x}) \\ &\leq F(\hat{x} \vee \mathbf{1}_{\mathrm{OPT} \cap \mathrm{supp}(\hat{x})}) + \frac{c\tau \cdot |\mathrm{OPT} \setminus \mathrm{supp}(\hat{x})|}{k} = F(\hat{x} \vee \mathbf{1}_{\mathrm{OPT} \cap \mathrm{supp}(\hat{x})}) + cb\tau \enspace . \end{aligned}$$

Rearranging this inequality yields

$$F(\hat{x} \vee \mathbf{1}_{\mathrm{OPT} \cap \mathrm{supp}(\hat{x})}) \geq F(\hat{x} \vee \mathbf{1}_{\mathrm{OPT}}) - cb\tau \geq (1-p) \cdot f(\mathrm{OPT}) - cb\tau \geq (1-p-cb)\tau \enspace ,$$

where the second inequality holds by Corollary 2 since Observation 14 guarantees that every coordinate of $\hat{x}$ is either 0 or p. This gives us the promised lower bound on $F(\hat{x} \vee \mathbf{1}_{\mathrm{OPT} \cap \mathrm{supp}(\hat{x})})$.

We now note that the submodularity of f implies that $F((t \cdot \hat{x}) \vee \mathbf{1}_{\mathrm{OPT} \cap \mathrm{supp}(\hat{x})})$ is a concave function of t within the range $[0, 1]$. Since b is inside this range,

$$\begin{aligned} F((b\hat{x}) \vee \mathbf{1}_{\mathrm{OPT} \cap \mathrm{supp}(\hat{x})}) &\geq b \cdot F(\hat{x} \vee \mathbf{1}_{\mathrm{OPT} \cap \mathrm{supp}(\hat{x})}) + (1-b) \cdot f(\mathrm{OPT} \cap \mathrm{supp}(\hat{x})) \\ &\geq b(1-p-cb)\tau + (1-b) \cdot f(\mathrm{OPT} \cap \mathrm{supp}(\hat{x})) \enspace , \end{aligned}$$

which completes the proof of the lemma. ◀

Using the last two claims we can now obtain a lower bound on the value of the solution of Algorithm 3 in the case of $\|\hat{x}\|_1 < k$ which is a function of α, τ and p alone. We note that both the guarantees of Corollary 19 and Lemma 20 are lower bounds on the expected value of the output of the algorithm in this case since $\mathbb{E}[f(S_1)] \geq F(\hat{x})$. Thus, any convex combination of these guarantees is also such a lower bound, and the proof of the following corollary basically proves a lower bound for one such convex combination – for the specific value of c stated in the corollary.

▶ **Corollary 21.** *If $\|\hat{x}\|_1 < k$ and c is set to $\frac{\alpha(1-p)}{\alpha+1}$, then $\mathbb{E}[\max\{f(S_1), f(S_2)\}] \geq \frac{(1-p)\alpha\tau}{\alpha+1}$.*

Proof. The corollary follows immediately from the non-negativity of f when $p = 1$. Thus, we may assume $p < 1$ in the rest of the proof.

By the definition of S_1, $\mathbb{E}[f(S_1)] \geq F(\hat{x})$. Thus, by Corollary 19 and Lemma 20,

$$\begin{aligned} \mathbb{E}[\max\{f(S_1), f(S_2)\}] &\geq \max\{\mathbb{E}[f(S_1)], \mathbb{E}[f(S_2)]\} \\ &\geq \max\{[p(1-p) - bc]\tau + (1-p)^2 \cdot f\big(\mathrm{OPT} \setminus \mathrm{supp}(\hat{x})\big), \\ &\qquad \alpha b(1-p-cb)\tau + \alpha(1-b) \cdot f(\mathrm{OPT} \cap \mathrm{supp}(\hat{x}))\} \\ &\geq \frac{\alpha(1-b)}{\alpha(1-b) + (1-p)^2} \cdot \big[[p(1-p) - bc]\tau + (1-p)^2 \cdot f\big(\mathrm{OPT} \setminus \mathrm{supp}(\hat{x})\big)\big] \\ &\quad + \frac{(1-p)^2}{\alpha(1-b) + (1-p)^2} \cdot [\alpha b(1-p-cb)\tau + \alpha(1-b) \cdot f(\mathrm{OPT} \cap \mathrm{supp}(\hat{x}))] \enspace . \end{aligned}$$

To keep the following calculations short, it will be useful to define $q = 1-p$ and $d = 1-b$. Using this notation and the fact that the submodularity and non-negativity of f guarantee together $f\big(\mathrm{OPT} \setminus \mathrm{supp}(\hat{x})\big) + f\big(\mathrm{OPT} \cap \mathrm{supp}(\hat{x})\big) \geq f(\mathrm{OPT}) \geq \tau$, the previous inequality implies

$$\begin{aligned}\frac{\mathbb{E}[\max\{f(S_1),f(S_2)\}]}{\alpha\tau} &\geq \frac{(1-b)[p(1-p)-bc]+b(1-p)^2(1-p-bc)+(1-b)(1-p)^2}{\alpha(1-b)+(1-p)^2}\\ &= \frac{d[q(1-q)-(1-d)c]+q^2(1-d)[q-(1-d)c]+dq^2}{\alpha d+q^2}\\ &= \frac{d[q-(1-d)c]+q^2(1-d)[q-(1-d)c]}{\alpha d+q^2} = \frac{[d+q^2(1-d)][q-(1-d)c]}{\alpha d+q^2}\\ &= \frac{q[d+q^2(1-d)](d\alpha+1)}{(\alpha+1)(d\alpha+q^2)} = \frac{d^2\alpha+d\alpha q^2-d^2\alpha q^2+d+q^2-dq^2}{d\alpha+q^2}\cdot\frac{q}{\alpha+1}\ , \end{aligned} \tag{1}$$

where the fourth equality holds by plugging in the value we assume for c.

The second fraction in the last expression is independent of the value of d, and the derivative of the first fraction in this expression as a function of d is

$$\begin{aligned}&\frac{(2d\alpha+\alpha q^2-2d\alpha q^2+1-q^2)[d\alpha+q^2]-\alpha(d^2\alpha+d\alpha q^2-d^2\alpha q^2+d+q^2-dq^2)}{[d\alpha+q^2]^2}\\ &\qquad\qquad\qquad\qquad = \frac{1-q^2}{[d\alpha+q^2]^2}\cdot[q^2(1-\alpha)+d\alpha(d\alpha+2q^2)]\ ,\end{aligned}$$

which is always non-negative since both q and α are numbers between 0 and 1. Thus, we get that the minimal value of the expression (1) is obtained for $d=0$ for any choice of q and α. Plugging this value into d yields

$$\mathbb{E}[\max\{f(S_1),f(S_2)\}] \geq \frac{q\alpha\tau}{\alpha+1} = \frac{(1-p)\alpha\tau}{\alpha+1}\ . \qquad ◀$$

Note that Lemma 16 and Corollary 21 prove the same lower bound on the expectation $\mathbb{E}[\max\{f(S_1),f(S_2)\}]$ when c is set to the value it is set to in Corollary 21 (because $\mathbb{E}[\max\{f(S_1),f(S_2)\}] \geq \mathbb{E}[f(S_1)] \geq F(\hat{x})$). Thus, we can summarize the results we have proved so far using the following proposition.

▶ **Proposition 22.** *Algorithm 3 is a semi-streaming algorithm storing $O(k/p)$ elements. Moreover, for the value of the parameter c given in Corollary 21, the output set produced by this algorithm has an expected value of at least $\frac{\alpha\tau(1-p)}{\alpha+1}$.*

Using the last proposition, we can now prove the following theorem. As discussed at the beginning of the section, in the full version of this paper we explain how the assumption that τ is known can be dropped at the cost of increasing of a slight increase in the number of of elements stored by the algorithm, which yields Theorem 12.

▶ **Theorem 23.** *For every constant $\varepsilon \in (0,1]$, there exists a semi-streaming algorithm that assumes access to an estimate τ of $f(\mathrm{OPT})$ obeying $(1-\varepsilon/8)\cdot f(\mathrm{OPT}) \leq \tau \leq f(\mathrm{OPT})$ and provides $(\frac{\alpha}{1+\alpha}-\varepsilon)$-approximation for the problem of maximizing a non-negative submodular function subject to cardinality constraint. This algorithm stores at most $O(k\varepsilon^{-1})$ elements.*

Proof. Consider the algorithm obtained from Algorithm 3 by setting $p=\varepsilon/2$ and c as is set in Corollary 21. By Proposition 22, this algorithm stores only $O(k/p)=O(k\varepsilon^{-1})$ elements, and the expected value of its output set is at least

$$\frac{\alpha\tau(1-p)}{\alpha+1} \geq \frac{\alpha(1-\varepsilon/8)(1-\varepsilon/2)}{\alpha+1}\cdot f(\mathrm{OPT}) \geq \frac{\alpha(1-\varepsilon)}{\alpha+1}\cdot f(\mathrm{OPT}) \geq \left(\frac{\alpha}{\alpha+1}-\varepsilon\right)\cdot f(\mathrm{OPT})\ ,$$

where the first inequality holds since τ obeys, by assumption, $\tau \geq (1-\varepsilon/8)\cdot f(\mathrm{OPT})$. ◀

Further discussion. Before concluding, let us discuss in more detail the value that should be assigned to the parameter p of Algorithm 3. In the proof of Theorem 23, we chose p to be very small. This makes sense whenever α is independent of p since the formula given by Proposition 22 for the guaranteed value of the output is non-increasing in p. However, for some choices of OfflineAlg the value of α might depend on p, and thus, it might be beneficial to choose a value for p which is not very small. To see why α might depend on p, note that the input passed to OfflineAlg by Algorithm 3 is of size at most $\lceil k/p \rceil$ due to Observation 14; and therefore, the ratio between the size of the ground set and k in this input is roughly $1/p$. Hence, α depends on p if the approximation ratio of OfflineAlg depends on the above ratio; which is the case, e.g., for one of the algorithms described in [6].

As a corollary of the above discussion, we get that for some offline algorithms a smart choice of p can yield a better approximation guarantee than the one stated in Theorem 23. At the current point this corollary is not very useful since the state-of-the-art offline approximation algorithm has an approximation ratio which is independent of p; however, this might change in the future.

References

1 Naor Alaluf and Moran Feldman. Making a sieve random: Improved semi-streaming algorithm for submodular maximization under a cardinality constraint. *CoRR*, abs/1906.11237, 2019. arXiv:1906.11237.

2 Ashwinkumar Badanidiyuru, Baharan Mirzasoleiman, Amin Karbasi, and Andreas Krause. Streaming submodular maximization: Massive data summarization on the fly. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 671–680. ACM, 2014.

3 Rafael da Ponte Barbosa, Alina Ene, Huy L Nguyen, and Justin Ward. A new framework for distributed submodular maximization. In *IEEE Foundations of Computer Science (FOCS)*, pages 645–654, 2016.

4 Rafael D.P. Barbosa, Alina Ene, Huy L. Nguyen, and Justin Ward. The power of randomization: Distributed submodular maximization on massive datasets. In *International Conference on Machine Learning (ICML)*, 2015.

5 Niv Buchbinder and Moran Feldman. Constrained submodular maximization via a non-symmetric technique. *Mathematics of Operations Research*, 44(3):988–1005, 2019.

6 Niv Buchbinder, Moran Feldman, Joseph Naor, and Roy Schwartz. Submodular maximization with cardinality constraints. In *SODA*, pages 1433–1452, 2014. doi:10.1137/1.9781611973402.106.

7 Niv Buchbinder, Moran Feldman, and Roy Schwartz. Online submodular maximization with preemption. *ACM Trans. Algorithms*, 15(3):30:1–30:31, 2019. doi:10.1145/3309764.

8 Gruia Călinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a monotone submodular function subject to a matroid constraint. *SIAM J. Comput.*, 40(6):1740–1766, 2011. doi:10.1137/080733991.

9 Amit Chakrabarti and Sagar Kale. Submodular maximization meets streaming: Matchings, matroids, and more. *Mathematical Programming*, 154(1-2):225–247, 2015.

10 Chandra Chekuri. Personal communication, 2018.

11 Chandra Chekuri, Shalmoli Gupta, and Kent Quanrud. Streaming algorithms for submodular function maximization. In *International Colloquium on Automata, Languages and Programming (ICALP)*, pages 318–330. Springer, 2015.

12 Chandra Chekuri, Jan Vondrák, and Rico Zenklusen. Dependent randomized rounding via exchange properties of combinatorial structures. In *FOCS*, pages 575–584, 2010. doi:10.1109/FOCS.2010.60.

13 Alina Ene and Huy L Nguyen. Constrained submodular maximization: Beyond 1/e. In *IEEE Foundations of Computer Science (FOCS)*, pages 248–257. IEEE, 2016.

14 Alessandro Epasto, Vahab Mirrokni, and Morteza Zadimoghaddam. Bicriteria distributed submodular maximization in a few rounds. In *PACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 25–33, 2017.

15 Uriel Feige, Vahab S Mirrokni, and Jan Vondrák. Maximizing non-monotone submodular functions. *SIAM Journal on Computing*, 40(4):1133–1153, 2011.

16 Moran Feldman, Christopher Harshaw, and Amin Karbasi. Greed is good: Near-optimal submodular maximization via greedy optimization. In *Proceedings of the 30th Conference on Learning Theory, COLT 2017, Amsterdam, The Netherlands, 7-10 July 2017*, pages 758–784, 2017. URL: `http://proceedings.mlr.press/v65/feldman17b.html`.

17 Moran Feldman, Amin Karbasi, and Ehsan Kazemi. Do less, get more: Streaming submodular maximization with subsampling. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 732–742, 2018.

18 Moran Feldman, Joseph Naor, and Roy Schwartz. A unified continuous greedy algorithm for submodular maximization. In *IEEE Foundations of Computer Science (FOCS)*, 2011. `doi:10.1109/FOCS.2011.46`.

19 Moran Feldman, Ashkan Norouzi-Fard, Ola Svensson, and Rico Zenklusen. The one-way communication complexity of submodular maximization with applications to streaming and robustness, 2020. To appear in STOC 2020.

20 Shayan Oveis Gharan and Jan Vondrák. Submodular maximization by simulated annealing. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2011.

21 Ehsan Kazemi, Marko Mitrovic, Morteza Zadimoghaddam, Silvio Lattanzi, and Amin Karbasi. Submodular streaming in all its glory: Tight approximation, minimum memory and low adaptive complexity. In *ICML*, pages 3311–3320, 2019. URL: `http://proceedings.mlr.press/v97/kazemi19a.html`.

22 Ravi Kumar, Benjamin Moseley, Sergei Vassilvitskii, and Andrea Vattani. Fast greedy algorithms in mapreduce and streaming. In *PACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 1–10, 2013.

23 Jon Lee, Vahab S Mirrokni, Viswanath Nagarajan, and Maxim Sviridenko. Non-monotone submodular maximization under matroid and knapsack constraints. In *ACM Symposium on Theory of Computing (STOC)*, pages 323–332, 2009.

24 Jon Lee, Maxim Sviridenko, and Jan Vondrák. Submodular maximization over multiple matroids via generalized exchange properties. *Mathematics of Operations Research*, 35(4):795–806, 2010.

25 László Lovász. Submodular functions and convexity. In *Mathematical Programming The State of the Art, XIth International Symposium on Mathematical Programming, Bonn, Germany, August 23-27, 1982*, pages 235–257, 1982. `doi:10.1007/978-3-642-68874-4_10`.

26 Vahab Mirrokni and Morteza Zadimoghaddam. Randomized composable core-sets for distributed submodular maximization. In *ACM Symposium on Theory of Computing (STOC)*, 2015.

27 Baharan Mirzasoleiman, Stefanie Jegelka, and Andreas Krause. Streaming non-monotone submodular maximization: Personalized video summarization on the fly. In *Thirty-second AAAI conference on artificial intelligence*, 2018.

28 Baharan Mirzasoleiman, Amin Karbasi, Ashwinkumar Badanidiyuru, and Andreas Krause. Distributed submodular cover: Succinctly summarizing massive data. In *Advances in Neural Information Processing Systems*, pages 2881–2889, 2015.

29 Baharan Mirzasoleiman, Amin Karbasi, Rik Sarkar, and Andreas Krause. Distributed submodular maximization: Identifying representative elements in massive data. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 2049–2057, 2013.

30 G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions – I. *Mathematical Programming*, 14(1):265–294, 1978.

31 Ashkan Norouzi-Fard, Jakub Tarnawski, Slobodan Mitrovic, Amir Zandieh, Aidasadat Mousavifar, and Ola Svensson. Beyond 1/2-approximation for submodular maximization on massive data streams. In *International Conference on Machine Learning*, pages 3826–3835, 2018.

32 Jan Vondrák. Symmetry and approximability of submodular maximization problems. *SIAM J. Comput.*, 42(1):265–304, 2013. doi:10.1137/110832318.

Dynamic Averaging Load Balancing on Cycles

Dan Alistarh
IST Austria, Klosterneuburg, Austria

Giorgi Nadiradze
IST Austria, Klosterneuburg, Austria

Amirmojtaba Sabour
IST Austria, Klosterneuburg, Austria

Abstract

We consider the following dynamic load-balancing process: given an underlying graph G with n nodes, in each step $t \geq 0$, one unit of load is created, and placed at a randomly chosen graph node. In the same step, the chosen node picks a random neighbor, and the two nodes *balance* their loads by averaging them. We are interested in the expected gap between the minimum and maximum loads at nodes as the process progresses, and its dependence on n and on the graph structure.

Variants of the above graphical balanced allocation process have been studied previously by Peres, Talwar, and Wieder [10], and by Sauerwald and Sun [12]. These authors left as open the question of characterizing the gap in the case of *cycle graphs* in the *dynamic* case, where weights are created during the algorithm's execution. For this case, the only known upper bound is of $\mathcal{O}(n \log n)$, following from a majorization argument due to [10], which analyzes a related graphical allocation process.

In this paper, we provide an upper bound of $\mathcal{O}(\sqrt{n} \log n)$ on the expected gap of the above process for cycles of length n. We introduce a new potential analysis technique, which enables us to bound the difference in load between k-hop neighbors on the cycle, for any $k \leq n/2$. We complement this with a "gap covering" argument, which bounds the maximum value of the gap by bounding its value across all possible subsets of a certain structure, and recursively bounding the gaps within each subset. We provide analytical and experimental evidence that our upper bound on the gap is tight up to a logarithmic factor.

2012 ACM Subject Classification Mathematics of computing → Combinatorics

Keywords and phrases Algorithms, Load Balancing

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.7

Category Track A: Algorithms, Complexity and Games

Related Version A full version of the paper is available at `https://arxiv.org/abs/2003.09297`.

Funding *Dan Alistarh*: European Research Council funding award PR1042ERC01
Giorgi Nadiradze: European Research Council funding award PR1042ERC01

Acknowledgements The authors sincerely thank Thomas Sauerwald and George Giakkoupis for insightful discussions, and Mohsen Ghaffari, Yuval Peres, and Udi Wieder for feedback on earlier versions of this draft. We also thank the ICALP anonymous reviewers for their very useful comments.

1 Introduction

We consider balls-into-bins processes where a sequence of m weights are placed into n bins via some randomized procedure, with the goal of minimizing the load imbalance between the most loaded and the least loaded bin. This family of processes has been used to model several practical allocation problems, such as load-balancing [3, 7, 11], hashing [5], or even relaxed data structures [2, 1].

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 7; pp. 7:1–7:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

One way to put our results into context is to view them as a variation of the well-known d-choice process, in which, in each step, a new weight is generated, and is placed in the least loaded of d uniform random choices. If $d = 1$, then we have the uniform random choice scheme, whose properties are well understood. In particular, if we place $m = n$ unit weights into the bins, then it is known that the most loaded bin will have expected $\Theta(\log n/\log\log n)$ load, whereas if $m = \Omega(n \log n)$ we have that the expected maximum load is $m/n + \Theta(\sqrt{m \log n/n})$. Seminal work by Azar, Broder, Karlin, and Upfal [3] showed that, if we place n unit weights into n bins by the d-choice process with $d \geq 2$, then, surprisingly, the maximum load is reduced to $\Theta(\log\log n/\log d)$. A technical tour-de-force by Berenbrink, Czumaj, Steger, and Vöcking [4] extended this result to the "heavily-loaded" case where $m \gg n$, showing that in this case the maximum load is $m/n + \log\log n/\log d + O(1)$ with failure probability at most $1/\text{poly } n$. An elegant alternative proof for a slightly weaker version of this result was later provided by Talwar and Wieder [13].

More recently, Peres, Talwar, and Wieder [10] analyzed the *graphical* version of this process, where the bins are the vertices of a graph, an *edge* is chosen at every step, and the weight is placed at the less loaded endpoint of the edge, breaking ties arbitrarily. (The classic 2-choice process corresponds to the case where the graph is a clique.) The authors focus on the evolution of the gap between the highest and lowest loaded bins, showing that, for graphs of β-edge-expansion [10], this gap is $O(\log n/\beta)$, with probability $1 - 1/\text{poly } n$.

An alternative way to frame our results is to consider *static load-balancing*, where each node starts with an arbitrary initial load, and the endpoints *average* their initial loads whenever the edge is chosen. Then, the balancing process can be mapped to a Markov chain, whose convergence is well-understood in terms of the spectral gap of the underlying graph [12]. Sauerwald and Sun [12] considered this static case in the *discrete* setting, where the fixed initial load can only be divided to *integer* tokens upon each averaging step, for which they gave strong upper bounds for a wide range of graph families.

By contrast to this previous work, in this paper we consider the less complex *continuous averaging case*, where exact averaging of the weights is possible, but in the more challenging *dynamic* scenario, where weights arrive in each step rather than being initially allocated, which is closer to the setting of the d-choice process discussed above.

One question left open by previous work concerns the evolution of the gap in the dynamic case on graphs of low expansion, such as cycles. In particular, for cycles, the only known upper bound on the expected gap in the dynamic case is of $O(n \log n)$, following from [10], whereas the only lower bound is the immediate $\Omega(\log n)$ gap lower bound for the clique. Closing this gap for cycle graphs is known to be a challenging open problem [9]. As suggested in [10], to deal with the cycle case, there is a need for a new approach, which takes the structure of the load balancing graph into account.

Contribution. In this paper, we address this question for the case where averaging is performed on a cycle graph. Let $Gap(t)$ be a difference between highest and lowest loads of the nodes at time step t. We provide the upper bound on the gap in the dynamic, heavily-loaded case, via a new potential argument. More formally, for any $t > 0$, we show that for a cycle graph with n vertices:

$$\mathbb{E}[Gap(t)] = O(\sqrt{n}\log(n)). \tag{1}$$

We complement this result with a lower bound of $\Omega(n)$ on the $\mathbb{E}[Gap(t)^2]$, as well as additional experimental evidence suggesting that our upper bound is tight within a logarithmic factor. Our results extend to *weighted* input. That is, we can allow our input to come from any distribution W, such that $\mathbb{E}[W^2] \leq M^2$, for some $M > 0$.

Technical Argument. Our upper bound result is based on two main ideas. The first introduces a new parametrized hop-potential function, which measures the squared difference in load between any k-hop neighbors on the graph, where $k \geq 1$ is a fixed hop parameter. Let $G = (V, E)$ be our input graph, where $V = \{1, 2, ..., n\}$. Throughout the paper, for any $1 \leq i \leq n$ we assume that the nodes $i + n$ and $i - n$ are the same as the node i. Let $x_i(t)$ be the load of node i node at step t. Then, we define the k-hop potential as:

$$\phi_k(t) = \sum_{i=1}^{n} (x_i(t) - x_{i+k}(t))^2.$$

The first technical step in the proof is to understand the expected ("steady-state") value of the k-hop potential. We show that, in expectation, the k-hop potential has a recursive structure on regular graphs. While the expected values of k-hop potentials cannot be computed precisely, we can isolate upper and lower bounds on their values for cycles. In particular, for the k-hop potential on an n-cycle, we prove the following bound:

$$\mathbb{E}[\phi_k(t)] \leq k(n-k) - 1, \forall k \geq 1. \tag{2}$$

In the second technical step, we shift gears, aiming to bound the *maximum possible value* of the gap between any two nodes, leveraging the fact that we understand the hop potential for any $k \geq 1$. We achieve this via a "gap covering" technique, which characterizes the maximum value of the gap across all possible subsets of a certain type.

More precisely, in the case of a cycle of length $n = 2^m$, for each node i and hop count k, we define the set family A_k^i to be formed of nodes $\{i, i + 2^k, i + 2 \times 2^k, i + 3 \times 2^k, \dots\}$. (Since we are on a cycle, $i = i + 2^{m-k} 2^k$.) Then for any $1 \leq i \leq n$ and $k > 0$, we will have

$$\sum_{i=1}^{n} Gap_{A_{k-1}^i}(t) \leq \sum_{i=1}^{n} Gap_{A_k^i}(t) + \frac{n}{\sqrt{2^{k-1}}} \sqrt{\phi_{2^{k-1}}(t)}, \tag{3}$$

where $Gap_X(t)$ is the maximal gap inside the set X at time t. Intuitively, this result allows us recursively characterize the gap value at various "resolutions" across the graph.

Finally, we notice that we can "cover" the gap across between *any* two nodes by carefully unwinding the recursion in the above inequality, considering all possible subsets of a well-chosen structure, and recursively bounding the gaps within each subset. (This step is particularly delicate in the case where n is not a power of two, see Section 5.) We obtain that

$$\mathbb{E}[Gap(t)] = O(\sqrt{n} \log(n)), \tag{4}$$

as claimed. The logarithmic slack is caused by the second term on the right-hand-side of (2). We note that this technique extends to the case where inserted items are *weighted*, where the weights are coming from some distribution of bounded second moment.

Lower Bound. It is interesting to ask whether this upper bound is tight. To examine this question, we revisit the recursive structure of the k-hop potential, which we used to obtain the lower bound in Equation 2. We can leverage this structure to obtain a *lower bound* on the expected k-hop potential as well. Starting from this lower bound, we can turn the upper bound argument "inside out," to obtain a linear lower bound on the *expected squared gap*:

$$\mathbb{E}[Gap(t)^2] = \Omega(n). \tag{5}$$

This second moment bound strongly suggests that our above analysis is tight within logarithmic factors. We conjecture that the bound is also tight with regards to the expected gap, and examine this claim empirically in Section 6.

Extensions and Overview. The analysis template we described above is general, and could be extended to other graph families, such as regular expanders. In particular, we note that the recursive structure of the k-hop potentials is preserved for such graphs. The main technical steps in analyzing a new graph family are to (1) identify the right upper bound on the k-hop potential (the analogue of (1)); and (2) identify the right set family for the gap covering argument, and its recursive structure (the analogue of (2)). Obtaining tight bounds for these quantities is not straightforward, since they do not seem to be immediately linked to well-studied graph properties. Here, we focus on obtaining tight bounds on the gap for cycles, which is technically non-trivial, and leave the extensions for other graph families as future work. To substantiate our generality claim, we exhibit an application of our analysis technique to Harary graphs [6] in a full version of the paper.

We discuss the relation between our results and bounds for the graphical power-of-two process on a cycle [10] in Section 7.

Related Work. As we have already discussed broad background, we will now mainly focus on the technical differences from previous work. As stated, we are the first to specifically consider the *dynamic* case for *continuous averaging* on cycles. In the *static* case with *discrete averaging*, the problem has been considered by Sauerwald and Sun [12]. However, their techniques would not apply in our case, since we consider that weights would be introduced *dynamically*, during the processes' execution.

To our knowledge, the only non-trivial upper bound on the gap of the process we consider which would follow from previous work is of $\mathcal{O}(n \log n)$, by the potential analysis of [10]: they consider 2-choice load balancing, and one can re-do their potential analysis for (continuous) averaging load balancing, yielding the same bounds. However, as our bounds show, the resulting analysis is quite loose in the case of cycles, yielding an $\Omega(\sqrt{n})$ gap. This is a consequence of the majorization technique used, which links dynamic averaging on the cycle and a very weak form of averaging on the clique.

Our potential analysis is substantially different from that of [10], as they track a sum of exponential potentials across the entire graph. By contrast, our analysis tracks the squared load differences between k-hop neighbors, establishing recurrences between these potentials. We notice that this is also different from the usual square potentials used for analyzing averaging load balancing, e.g. [8], which usually compare against the *global mean*, as opposed to pairwise potential differences. Our approach is also different from the classic analyses of e.g. [3], which perform probabilistic induction on the number of bins at a given load, assuming a clique.

Generally, our technique can be seen as performing the induction needed to bound the gap not on the bin loads, as is common in previous work, e.g. [3], but *over the topology of the graph*. This approach is natural, since we wish to obtain tight, topology-specific bounds, but we believe we are the first to propose and analyze it successfully.

2 Averaging on the Cycle: Upper Bounding the Gap

Preliminaries. We consider a cycle graph $G = (V, E)$ where $V = \{1, 2, ..., n\}$, such that each node i is connected to its left and right neighbors, $i - 1$ and $i + 1$ (recall that for any $1 \leq i \leq n$ the nodes $i + n$ and $i - n$ are the same as the node i).

We consider a stochastic process following real time $t \geq 0$, in which, in each step, a weight $w(t)$ is generated from a same distribution W. We associate a real-valued *load* value $x_i(t)$ with each node i. In step t, an edge $(i, i + 1)$ is chosen uniformly at random, and the two endpoints nodes update their weights as follows:

$$x_i(t+1) = x_{i+1}(t+1) = \frac{x_i(t) + x_{i+1}(t) + w(t)}{2}.$$

We will assume that the second moment of the distribution W is bounded. That is: $E[W^2] \leq M^2$, for some $M > 0$. For simplicity, we will assume that weights are normalized by M. This gives us that $\mathbb{E}[W^2] \leq 1$.

Let $X(t) = (x_1(t), x_2(t), ..., x_n(t))$ be the vector of the bin weights after t balls have been thrown. First, we define the following potential functions:

$$\forall k \in \{1, 2, \dots, n-1\} : \phi_k(t) := \sum_{i=1}^{n} (x_i(t) - x_{i+k}(t))^2.$$

Notice that for every $1 \leq i \leq n$, we have that $\phi_i(t) = \phi_{n-i}(t)$. We want to analyze what is the value of these functions in expectation after an additional ball is thrown, for a given load vector $X(t)$.

We start with $\phi_1(t+1)$:

$$\begin{aligned}\mathbb{E}[\phi_1(t+1)|X(t), w(t)] &= \sum_{i=1}^{n} \frac{1}{n}\Bigg(\Big(\frac{x_i(t) + x_{i+1}(t) + w(t)}{2} - x_{i+2}(t)\Big)^2 \\ &\quad + \Big(\frac{x_i(t) + x_{i+1}(t) + w(t)}{2} - x_{i-1}(t)\Big)^2 \\ &\quad + \sum_{j \neq i-1, i, i+1} (x_j(t) - x_{j+1}(t))^2\Bigg) \\ &= \frac{n-3}{n}\phi_1(t) + \frac{1}{2} + \frac{1}{2n}(\phi_1(t) + 2\phi_2(t)) \\ &= \frac{n-2}{n}\phi_1(t) + \frac{1}{2}(w(t)^2 - \frac{\phi_1(t)}{n}) + \frac{1}{n}\phi_2(t).\end{aligned}$$

Now, we proceed with calculating the expected value of $\phi_k(t+1)$, for $2 \leq k \leq \lfloor n/2 \rfloor$:

$$\begin{aligned}\mathbb{E}[\phi_k(t+1)|X_t, w(t)] &= \sum_{i=1}^{n} \frac{1}{n}\Bigg(\Big(\frac{x_i(t) + x_{i+1}(t) + w(t)}{2} - x_{i-k}(t)\Big)^2 \\ &\quad + \Big(\frac{x_i(t) + x_{i+1}(t) + w(t)}{2} - x_{i+1-k}(t)\Big)^2 \\ &\quad + \Big(\frac{x_i(t) + x_{i+1}(t) + w(t)}{2} - x_{i+k}(t)\Big)^2 \\ &\quad + \Big(\frac{x_i(t) + x_{i+1}(t) + w(t)}{2} - x_{i+1+k}(t)\Big)^2 \\ &\quad + \sum_{j \neq i-k, i+1-k, i+k, i+1+k} (x_j(t) - x_{j+k}(t))^2\Bigg) \\ &= \frac{n-2}{n}\phi_k(t) + (w(t)^2 - \frac{\phi_1(t)}{n}) + \frac{\phi_{k+1}(t)}{n} + \frac{\phi_{k-1}(t)}{n}.\end{aligned}$$

Note that in the above calculations for $\phi_1(t+1)$ and $\phi_k(t+1)$, for $k > 1$ the terms which contain $w(t)$ as linear multiplicative term disappear because we can assume that loads $x_1(t), x_2(t), ..., x_n(t)$ are normalized (this will not change our potentials) and we have:

$$\sum_{i=1}^{n} w(t)x_i(t) = 0. \tag{6}$$

If we remove conditioning on $w(t)$ and express these equations for $k = 1, 2, \dots, n-1$, we get:

$$\begin{cases} \mathbb{E}[\phi_1(t+1)|X(t)] = (\frac{n-2}{n})\phi_1(t) + \frac{1}{2}(\mathbb{E}[W^2] - \frac{\phi_1(t)}{n}) + \frac{\phi_2(t)}{n}. \\ \mathbb{E}[\phi_2(t+1)|X(t)] = (\frac{n-2}{n})\phi_2(t) + (\mathbb{E}[W^2] - \frac{\phi_1(t)}{n}) + \frac{\phi_1(t)}{n} + \frac{\phi_3(t)}{n}. \\ \dots \\ \mathbb{E}[\phi_{\lfloor \frac{n}{2} \rfloor}(t+1)|X(t)] = (\frac{n-2}{n})\phi_{\lfloor \frac{n}{2} \rfloor}(t) + (\mathbb{E}[W^2] - \frac{\phi_1(t)}{n}) \\ \qquad + \frac{\phi_{\lfloor \frac{n}{2} \rfloor - 1}(t)}{n} + \frac{\phi_{\lfloor \frac{n}{2} \rfloor + 1}(t)}{n}. \\ \dots \\ \mathbb{E}[\phi_{n-2}(t+1)|X(t)] = (\frac{n-2}{n})\phi_{n-2}(t) \\ \qquad + (\mathbb{E}[W^2] - \frac{\phi_1(t)}{n}) + \frac{\phi_{n-3}(t)}{n} + \frac{\phi_{n-1}(t)}{n}. \\ \mathbb{E}[\phi_{n-1}(t+1)|X(t)] = (\frac{n-2}{n})\phi_{n-1}(t) + \frac{1}{2}(\mathbb{E}[W^2] - \frac{\phi_1(t)}{n}) + \frac{\phi_{n-2}(t)}{n}. \end{cases}$$

Using the above equations we can prove the following:

▶ **Lemma 1.** *For every $t \geq 0$ and $1 \leq k \leq n-1$, we have that*

$$\mathbb{E}[\phi_k(t)] \leq (k(n-k)-1)\mathbb{E}[W^2] \leq k(n-k)-1. \tag{7}$$

Proof. Let $\Phi(t) = (\phi_1(t), \phi_2(t), ..., \phi_{n-1}(t))$ be the vector of values of our potentials at time step t and let $Y = (y_1, y_2, ..., y_{n-1})$, be the vector containing our desired upper bounds for each potential. That is: for each $1 \leq i \leq n-1$, we have that $y_i = (i(n-i)-1)\mathbb{E}[W^2]$. An interesting and easily checkable thing about the vector Y is that

$$\mathbb{E}[\Phi(t+1)|\Phi(t) = Y] = Y. \tag{8}$$

Next, consider the vector $Z(t) = (z_1(t), z_2(t), ...z_{n-1}(t)) = Y - \Phi(t)$. Our goal is to show that for every step t and coordinate i, $\mathbb{E}[z_i(t)] \geq 0$. we have that

$$\begin{aligned} \mathbb{E}[z_1(t+1)|X(t)] &= y_1 - \mathbb{E}[\phi_1(t+1)|X(t)] \\ &= (\frac{n-2}{n})y_1 + \frac{1}{2}(\mathbb{E}[W^2] - \frac{y_1}{n}) + \frac{y_2}{n} - \left((\frac{n-2}{n})\phi_1(t) + \frac{1}{2}(\mathbb{E}[W^2] - \frac{\phi_1(t)}{n}) + \frac{\phi_2(t)}{n} \right) \\ &= (\frac{n-2}{n})z_1(t) - \frac{z_1(t)}{2n} + \frac{z_2(t)}{n}. \end{aligned}$$

and for $2 \leq i \leq \lfloor \frac{n}{2} \rfloor$, we have that

$$\mathbb{E}[z_i(t+1)|X(t)] = (\frac{n-2}{n})z_i(t) - \frac{z_1(t)}{n} + \frac{z_{i-1}(t)}{n} + \frac{z_{i-1}(t)}{n}.$$

Hence we get the following equations(recall that $z_i(t) = z_{n-i}(t)$):

$$\begin{cases} n \times \mathbb{E}[z_1(t+1)|X(t)] = (n-2-\frac{1}{2})z_1(t) + z_2(t). \\ n \times \mathbb{E}[z_2(t+1)|X(t)] = -z_1(t) + z_1(t) + (n-2)z_2(t) + z_3(t). \\ n \times \mathbb{E}[z_3(t+1)|X(t)] = -z_1(t) + z_2(t) + (n-2)z_3(t) + z_4(t). \\ \dots \\ n \times \mathbb{E}[z_{\lfloor \frac{n}{2} \rfloor}(t+1)|X(t)] = -z_1(t) + z_{\lfloor \frac{n}{2} \rfloor - 1}(t) + (n-2)z_{\lfloor \frac{n}{2} \rfloor}(t) + z_{\lfloor \frac{n}{2} \rfloor + 1}(t). \end{cases} \tag{9}$$

Next, using induction on t, we show that for every $t \geq 0$

$$0 \leq \mathbb{E}[z_1(t)] \leq \mathbb{E}[z_2(t)] \leq ... \leq \mathbb{E}[z_{\lfloor \frac{n}{2} \rfloor}(t)]. \tag{10}$$

The base case holds trivially since $Z(0) = Y$. For the induction step, assume that $0 \leq \mathbb{E}[z_1(t)] \leq \mathbb{E}[z_2(t)] \leq ... \leq \mathbb{E}[z_{\lfloor \frac{n}{2} \rfloor}(t)]$. First, we have that

$$n\mathbb{E}[z_1(t+1)] = n\mathbb{E}_{X(t)}[\mathbb{E}[z_1(t+1)|X(t)]] = (n-2-\frac{1}{2})\mathbb{E}[z_1(t)] + \mathbb{E}[z_2(t)] \geq 0.$$

Additionally, we have that:

$$\begin{aligned} n\mathbb{E}[z_1(t+1)] &= (n-2-\frac{1}{2})\mathbb{E}[z_1(t)] + \mathbb{E}[z_2(t)] \leq (n-2)\mathbb{E}[z_1(t)] + \mathbb{E}[z_2(t)] \\ &\leq (n-2)\mathbb{E}[z_2(t)] + \mathbb{E}[z_3(t)] = n\mathbb{E}[z_2(t+1)]. \end{aligned}$$

For $2 \leq i \leq \lfloor \frac{n}{2} \rfloor - 2$, we have that

$$\begin{aligned} n\mathbb{E}[z_i(t+1)] &= -\mathbb{E}[z_1(t)] + \mathbb{E}[z_{i-1}(t)] + (n-2)\mathbb{E}[z_i(t)] + \mathbb{E}[z_{i+1}(t)] \\ &\leq -\mathbb{E}[z_1(t)] + \mathbb{E}[z_i(t)] + (n-2)\mathbb{E}[z_{i+1}(t)] + \mathbb{E}[z_{i+2}(t)] \\ &= n\mathbb{E}[z_{i+1}(t+1)]. \end{aligned}$$

Next, observe that by our assumption:
$\mathbb{E}[z_{\lfloor \frac{n}{2} \rfloor+1}(t)] = \mathbb{E}[z_{\lceil \frac{n}{2} \rceil-1}(t)] \geq \mathbb{E}[z_{\lfloor \frac{n}{2} \rfloor-2}(t)]$. Finally, by using this observation we get that

$$\begin{aligned} &n\mathbb{E}[z_{\lfloor \frac{n}{2} \rfloor-1}(t+1)] = -\mathbb{E}[z_1(t)] + \mathbb{E}[z_{\lfloor \frac{n}{2} \rfloor-2}(t)] + (n-2)\mathbb{E}[z_{\lfloor \frac{n}{2} \rfloor-1}(t)] + \mathbb{E}[z_{\lfloor \frac{n}{2} \rfloor}(t)] \\ &\quad \leq -\mathbb{E}[z_1(t)] + \mathbb{E}[z_{\lfloor \frac{n}{2} \rfloor+1}(t)] + \mathbb{E}[z_{\lfloor \frac{n}{2} \rfloor-1}(t)] + (n-3)\mathbb{E}[z_{\lfloor \frac{n}{2} \rfloor-1}(t)] + \mathbb{E}[z_{\lfloor \frac{n}{2} \rfloor}(t)] \\ &\quad \leq -\mathbb{E}[z_1(t)] + \mathbb{E}[z_{\lfloor \frac{n}{2} \rfloor+1}(t)] + \mathbb{E}[z_{\lfloor \frac{n}{2} \rfloor-1}(t)] + (n-2)\mathbb{E}[z_{\lfloor \frac{n}{2} \rfloor}(t)] \\ &\quad = n\mathbb{E}[z_{\lfloor \frac{n}{2} \rfloor}(t+1)]. \end{aligned}$$

This completes the proof of the theorem. ◀

3 Upper Bound on the Gap for $n = 2^m$

In this section we upper bound a gap in expectation for $n = 2^m$ case. The proof for the general case is quite technical but not necessarily more interesting, and is provided in the Section 5.

We begin with some definitions. For a set $A \subseteq \{1, 2, \ldots, n\}$, let

$$Gap_A(t) = \max_{i \in A}(x_i(t)) - \min_{i \in A}(x_i(t)).$$

Also, let A_k^i be $\{i, i+2^k, i+2 \times 2^k, i+3 \times 2^k, \ldots\}$ (Notice that $i = i + 2^{m-k}2^k$). Our proof works as follows: for each $1 \leq i \leq n$ and $0 < k \leq m$, we look at the vertices given by the sets A_k^i and $A_k^{i+2^{k-1}}$ and try to characterise the gap after we merge those sets (Note that this will give us the gap for the set $A_{k-1}^i = A_k^i \cup A_k^{i+2^{k-1}}$). Using this result, we are able to show that $\sum_{i=1}^n Gap_{A_{k-1}^i}(t)$ is upper bounded by $\sum_{i=1}^n Gap_{A_k^i}(t)$ plus n times maximum load difference between vertices at hop distance 2^{k-1}. Next, we use 2^{k-1} hop distance potential $\phi_{2^{k-1}}(t)$ to upper bound maximum load between the vertices at hop distance 2^{k-1}. Using induction on k, we are able to upper bound $\sum_{i=1}^n Gap_{A_0^i}(t)$ in terms of $\sum_{i=1}^n Gap_{A_m^i}(t)$ and $\sum_{k=1}^m \phi_{2^{k-1}}(t)$. Notice that by our definitions, for each i, $Gap_{A_m^i}(t) = 0$ (A_i^m contains only vertex i) and $Gap_{A_0^i}(t) = Gap(t)$ (A_0^i contains all vertices). Hence, what is left is to use the upper bounds for the hop distance potentials, which we derived in the previous section.

We start by proving the following useful lemma.

▶ **Lemma 2.** *For any $1 \leq i \leq n$ and $k > 0$, we have that*

$$2Gap_{A^i_{k-1}}(t) \leq 2 \max_{j \in A^i_{k-1}} |x_j(t) - x_{j+2^{k-1}}(t)| + Gap_{A^{i+2^{k-1}}_k}(t) + Gap_{A^i_k}(t). \tag{11}$$

Proof. Fix vertex i. Note that $A^i_{k-1} = A^i_k \cup A^{i+2^{k-1}}_k$. Let $u = \arg\max_{j \in A^i_{k-1}} x_j(t)$ and let $v = \arg\min_{j \in A^i_{k-1}} x_j(t)$. We consider several cases on the membership of nodes u and v, and bound the gap in each one:

Case 1. $u \in A^i_k$ and $v \in A^i_k$. Then $Gap_{A^i_{k-1}}(t) = Gap_{A^i_k}(t)$ and we have that

$$\begin{aligned} Gap_{A^i_k}(t) &= |x_u(t) - x_v(t)| \\ &\leq |x_{u+2^{k-1}}(t) - x_u(t)| + |x_{v+2^{k-1}}(t) - x_v(t)| + |x_{u+2^{k-1}}(t) - x_{v+2^{k-1}}(t)| \\ &\leq |x_{u+2^{k-1}}(t) - x_u(t)| + |x_{v+2^{k-1}}(t) - x_v(t)| + Gap_{A^{i+2^{k-1}}_k}(t) \\ &\leq 2 \max_{j \in A^i_{k-1}} |x_j(t) - x_{j+2^{k-1}}(t)| + Gap_{A^{i+2^{k-1}}_k}(t). \end{aligned}$$

Where we used the fact that both $u + 2^{k-1}$ and $v + 2^{k-1}$ belong to $A^{i+2^{k-1}}_k$. This gives us that

$$2Gap_{A^i_{k-1}}(t) \leq 2 \max_{j \in A^i_{k-1}} |x_j(t) - x_{j+2^{k-1}}(t)| + Gap_{A^{i+2^{k-1}}_k}(t) + Gap_{A^i_k}(t). \tag{12}$$

Case 2. $u \in A^i_k$ and $v \in A^{i+2^{k-1}}_k$. Then we have that:

$$\begin{aligned} Gap_{A^i_{k-1}}(t) &= |x_u(t) - x_v(t)| \leq |x_u(t) - x_{v+2^{k-1}}(t)| + |x_{v+2^{k-1}}(t) - x_v(t)| \\ &\leq Gap_{A^i_k}(t) + \max_{j \in A^i_{k-1}} (|x_j(t) - x_{j+2^{k-1}}(t)|) \end{aligned}$$

and

$$\begin{aligned} Gap_{A^i_{k-1}}(t) &= |x_u(t) - x_v(t)| \leq |x_u(t) - x_{u+2^{k-1}}(t)| + |x_{u+2^{k-1}}(t) - x_v(t)| \\ &\leq Gap_{A^{i+2^{k-1}}_k}(t) + \max_{j \in A^i_{k-1}} (|x_j(t) - x_{j+2^{k-1}}(t)|) \end{aligned}$$

Where we used $v + 2^{k-1} \in A^k_i$ and $u + 2^{k-1} \in A^{i+2^{k-1}}_k$. Hence, we again get that

$$2Gap_{A^i_{k-1}}(t) \leq 2 \max_{j \in A^i_{k-1}} |x_j(t) - x_{j+2^{k-1}}(t)| + Gap_{A^{i+2^{k-1}}_k}(t) + Gap_{A^i_k}(t). \tag{13}$$

Case 3. $u \in A^{i+2^{k-1}}_k$ and $v \in A^{i+2^{k-1}}_k$, is similar to Case 1.

Case 4. $v \in A^i_k$ and $u \in A^{i+2^{k-1}}_k$, is similar to Case 2. ◀

Next, we upper bound the quantity $\sum_{i=1}^n \max_{j \in A^i_k} |x_j(t) - x_{j+2^k}(t)|$.

▶ **Lemma 3.**

$$\sum_{i=1}^{n} \max_{j \in A^i_k} |x_j(t) - x_{j+2^k}(t)| \leq \frac{n}{\sqrt{2^k}} \sqrt{\phi_{2^k}(t)}. \tag{14}$$

Proof. Notice that for any i and $i' \in A_k^i$, we have that $A_k^i = A_k^{i'}$, hence $\max_{j \in A_k^i} |x_j(t) - x_{j+2^k}(t)| = \max_{j \in A_k^i} |x_j(t) - x_{j+2^k}(t)|$ and this means that

$$\sum_{i=1}^{n} \max_{j \in A_k^i} |x_j(t) - x_{j+2^k}(t)| = \frac{n}{2^k} \sum_{i=1}^{2^k} \max_{j \in A_k^i} |x_j(t) - x_{j+2^k}(t)|$$
$$\leq \frac{n}{2^k} \sqrt{2^k} \sqrt{\sum_{i=1}^{2^k} \max_{j \in A_k^i} |x_j(t) - x_{j+2^k}(t)|^2}$$
$$\leq \frac{n}{2^k} \sqrt{2^k} \sqrt{\sum_{j=1}^{n} |x_j(t) - x_{j+2^k}(t)|^2} = \frac{n}{\sqrt{2^k}} \sqrt{\phi_{2^k}(t)}$$

Where we used a fact that sets $A_k^1, A_k^2, ..., A_k^{2^k}$ are disjoint. ◀

Finally, using the two Lemmas above and Theorem 1 we can upper bound the expected gap at step t:

▶ **Theorem 4.** *For every $t \geq 0$, we have that*

$$\mathbb{E}[Gap(t)] = O(\sqrt{n} \log(n)).$$

Proof. From Lemma 2 we have that

$$\sum_{i=1}^{n} 2Gap_{A_{k-1}^i(t)} \leq \sum_{i=1}^{n} Gap_{A_k^i}(t) + \sum_{i=1}^{n} Gap_{A_k^{i+2^{k-1}}}(t)$$
$$+ \sum_{i=1}^{n} 2 \max_{j \in A_{k-1}^i} |x_j(t) - x_{j+2^{k-1}}(t)|$$
$$= 2\sum_{i=1}^{n} Gap_{A_k^i}(t) + 2\sum_{i=1}^{n} \max_{j \in A_{k-1}^i} |x_j(t) - x_{j+2^{k-1}}(t)|.$$

After dividing the above inequality by 2 and applying Lemma 3 we get that:

$$\sum_{i=1}^{n} Gap_{A_{k-1}^i}(t) \leq \sum_{i=1}^{n} Gap_{A_k^i}(t) + \frac{n}{\sqrt{2^{k-1}}} \sqrt{\phi_{2^{k-1}}(t)}.$$

Notice that $\sum_{i=1}^{n} Gap_0^i(t) = nGap(t)$ and we also have that

$$\sum_{i=1}^{n} Gap_{\frac{n}{2}}^i(t) = \sum_{i=1}^{n} |x_i(t) - x_{i+\frac{n}{2}}(t)| \leq \sqrt{n} \sqrt{\sum_{i=1}^{n} |x_i(t) - x_{i+\frac{n}{2}}(t)|^2} = \sqrt{n} \sqrt{\phi_{\frac{n}{2}}(t)}$$

Hence, we get that

$$nGap(t) = \sum_{i=1}^{n} Gap_0^i(t) \leq \sum_{i=1}^{n} Gap_{\frac{n}{2}}^i(t) + \sum_{k=1}^{m-1} \frac{n}{\sqrt{2^{k-1}}} \sqrt{\phi_{2^{k-1}}(t)}$$
$$\leq \sqrt{n} \sqrt{\phi_{\frac{n}{2}}(t)} + \sum_{k=1}^{m-1} \frac{n}{\sqrt{2^{k-1}}} \sqrt{\phi_{2^{k-1}}(t)}.$$

Next, we apply Jensen and Theorem 1:

$$\begin{aligned}
n\mathbb{E}[Gap(t)] &\leq \sqrt{n}\mathbb{E}\sqrt{\phi_{\frac{n}{2}}(t)} + \sum_{k=1}^{m-1} \frac{n}{\sqrt{2^{k-1}}}\mathbb{E}\sqrt{\phi_{2^{k-1}}(t)} \\
&\leq \sqrt{n}\sqrt{\mathbb{E}[\phi_{\frac{n}{2}}(t)]} + \sum_{k=1}^{m-1} \frac{n}{\sqrt{2^{k-1}}}\sqrt{\mathbb{E}[\phi_{2^{k-1}}(t)]} \\
&\leq \sqrt{n}\sqrt{\left(\frac{n}{2}\right)^2} + \sum_{k=1}^{m-1} \frac{n}{\sqrt{2^{k-1}}}\sqrt{2^{k-1}(n-2^{k-1})} \\
&\leq mn\sqrt{n} = n(\log n)\sqrt{n}.
\end{aligned}$$

This gives us the proof of the theorem. ◀

4 Gap Lower Bound

Next we prove the following theorem, which provides strong evidence that our bound on the gap is tight within a logarithmic factor.

▶ **Theorem 5.** *The following limit holds:*

$$\lim_{t\to\infty} \mathbb{E}[Gap(t)^2] = \Omega(n\mathbb{E}[W^2])).$$

Proof. In this case we want to prove that not only does vector $Z(t)$ have positive coordinates in expectation, but also $\mathbb{E}[z_{\lfloor\frac{n}{2}\rfloor}]$ converges to 0 . This will give us that $\phi_{\lfloor\frac{n}{2}\rfloor}$ approaches it's upper bound $(\lfloor\frac{n}{2}\rfloor\lceil\frac{n}{2}\rceil - 1)\mathbb{E}[W^2]$ in expectation. Then, we can show that there exist two nodes(At distance $\lfloor\frac{n}{2}\rfloor$) such that the expected square of difference between their loads is $\Omega(n\mathbb{E}[w^2])$.

Recall from Equations 9 that

$$n\mathbb{E}[z_{\lfloor\frac{n}{2}\rfloor(t+1)}] = -\mathbb{E}[z_1(t)] + \mathbb{E}[z_{\lfloor\frac{n}{2}\rfloor+1}(t)] + \mathbb{E}[z_{\lfloor\frac{n}{2}\rfloor-1}(t)] + (n-2)\mathbb{E}[z_{\lfloor\frac{n}{2}\rfloor}(t)].$$

We also know that Inequalities 10 hold for every t, hence we get that

$$\mathbb{E}[z_{\lfloor\frac{n}{2}\rfloor(t+1)}] \leq \mathbb{E}[z_{\lfloor\frac{n}{2}\rfloor(t)}] - \frac{\mathbb{E}[z_1(t)]}{n}.$$

The above inequality in combination with Inequalities 10 means that

$$\begin{aligned}
\mathbb{E}[z_{\lfloor\frac{n}{2}\rfloor}(t+\lfloor\frac{n}{2}\rfloor+1)] &\leq \mathbb{E}[z_{\lfloor\frac{n}{2}\rfloor}(t+1)] - \sum_{i=t}^{t+\lfloor\frac{n}{2}\rfloor}\frac{\mathbb{E}[z_1(i)]}{n} &\quad (15)\\
&\leq \mathbb{E}[z_{\lfloor\frac{n}{2}\rfloor}(t+1)] - \frac{\mathbb{E}[z_1(t+\lfloor\frac{n}{2}\rfloor)]}{n} &\quad (16)
\end{aligned}$$

Again by using Equations 9 and Inequalities 10, we can show that for every $1 \leq i \leq \lfloor\frac{n}{2}\rfloor - 1$:

$$\mathbb{E}[z_i(t+1)] \geq \frac{\mathbb{E}[z_{i+1}(t)]}{n}.$$

This gives us that:

$$\begin{aligned}
\mathbb{E}[z_1(t+\lfloor\frac{n}{2}\rfloor)] &\geq \left(\frac{1}{n}\right)\mathbb{E}[z_2(t+\lfloor\frac{n}{2}\rfloor-1)] \geq \left(\frac{1}{n}\right)^2\mathbb{E}[z_3(t+\lfloor\frac{n}{2}\rfloor-2)] \\
&\geq \dots \\
&\geq \left(\frac{1}{n}\right)^{\lfloor\frac{n}{2}\rfloor-1}\mathbb{E}[z_{\lfloor\frac{n}{2}\rfloor}(t+\lfloor\frac{n}{2}\rfloor-(\lfloor\frac{n}{2}\rfloor-1))] = \left(\frac{1}{n}\right)^{\lfloor\frac{n}{2}\rfloor-1}\mathbb{E}[z_{\lfloor\frac{n}{2}\rfloor}(t+1)].
\end{aligned}$$

By plugging the above inequality in inequality 15 we get that

$$\mathbb{E}[z_{\lfloor \frac{n}{2} \rfloor}(t+\lfloor \frac{n}{2} \rfloor+1)] \le \mathbb{E}[z_{\lfloor \frac{n}{2} \rfloor}(t+1)] - \frac{\mathbb{E}[z_1(t+\lfloor \frac{n}{2} \rfloor)]}{n}$$

$$\le \mathbb{E}[z_{\lfloor \frac{n}{2} \rfloor}(t+1)] - \left(\frac{1}{n}\right)^{\lfloor \frac{n}{2} \rfloor-1}\mathbb{E}[z_{\lfloor \frac{n}{2} \rfloor}(t+1)] = \left(1-\left(\frac{1}{n}\right)^{\lfloor \frac{n}{2} \rfloor-1}\right)\mathbb{E}[z_{\lfloor \frac{n}{2} \rfloor}(t+1)]$$

Because $\left(1-\left(\frac{1}{n}\right)^{\lfloor \frac{n}{2} \rfloor-1}\right) < 1$ and does not depend on t, we get that $\lim_{t\to\infty}\mathbb{E}[z_{\lfloor \frac{n}{2} \rfloor}(t)] = 0$..
This means that $\lim_{t\to\infty}\mathbb{E}[\phi_{\lfloor \frac{n}{2} \rfloor}(t)] = \Omega(n^2\mathbb{E}[W^2])$.
Let $Gap_{\lfloor \frac{n}{2} \rfloor}(t) = \max_{1\le i\le n}|x_i(t) - x_{i+\lfloor \frac{n}{2} \rfloor}(t)|$. Note that:
$Gap(t)^2 \ge Gap_{\lfloor \frac{n}{2} \rfloor}(t)^2 \ge \frac{\phi_{\lfloor \frac{n}{2} \rfloor}(t)}{n}$. Hence $\lim_{t\to\infty}\mathbb{E}[Gap(t)^2] = \Omega(n\mathbb{E}[W^2])$.

Unfortunately we are not able to obtain the lower bound on the gap, since our approach uses the fact that the upper bounds on k-hop potentials are 'tight'. Since our potentials are quadratic, we are not able to derive any kind of lower on for the gap itself. Intuitively, this will be an issue with any argument which uses convex potential. ◀

5 Upper Bound on the Gap, General Case

To prove the Theorem 4 for the general case, we need to redefine our sets A_i^k. In order to do this, for each k we define 2^k dimensional vector $\Delta_k = (\delta_k^1, \delta_k^2, ..., \delta_k^{2^k})$. For $k = 0$, we have that $\Delta_k = (n)$. For $\lfloor \log n \rfloor \ge k > 0$ we set $\Delta_k = (\alpha_k, \delta_{k-1}^1 - \alpha_k, \alpha_k, \delta_{k-1}^2 - \alpha_k, ..., \alpha_k, \delta_{k-1}^{2^{k-1}} - \alpha_k)$.

Where,

$$\alpha_k = \begin{cases} \lfloor \frac{n}{2^{k-1}} \rfloor/2, & \text{if } \lfloor \frac{n}{2^{k-1}} \rfloor \text{ is even.} \\ \left\lfloor \lceil \frac{n}{2^{k-1}} \rceil/2 \right\rfloor, & \text{otherwise.} \end{cases}$$

First we prove the following lemma:

▶ **Lemma 6.** *For any $\lfloor \log n \rfloor \ge k > 0$, we have that*
(1) $\sum_{i=1}^{2^k} \delta_k^i = n$.
(2) *For any $1 \le i \le 2^k$, $\delta_k^i \in \{\lceil \frac{n}{2^k} \rceil \lfloor \frac{n}{2^k} \rfloor\}$ (Notice that this means $\alpha_k = \lfloor \frac{n}{2^k} \rfloor$ or $\alpha_k = \lceil \frac{n}{2^k} \rceil$).*

Proof. We prove the lemma using induction on k. Base case $k = 0$ holds trivially. For the induction step, assume that Properties 1 and 2 hold for $k-1$, we aim to prove that they hold for k as well. We have that $\sum_{i=1}^{2^k} \delta_k^i = \sum_{i=1}^{2^{k-1}} (\alpha_k + \delta_{k-1}^i - \alpha_k) = \sum_{i=1}^{2^{k-1}} \delta_{k-1}^i = n$. To prove Property 2 we consider several cases:

Case 1. $\frac{n}{2^{k-1}} = 2q$, for some integer q.
We have that $\alpha_k = q$, and hence for any $1 \le i \le 2^{k-1}$, $\delta_{k-1}^i - \alpha_k = q$. Since $\lfloor \frac{n}{2^k} \rfloor = q$, Property 2 holds.

Case 2. $\frac{n}{2^{k-1}} = 2q + 1$, for some integer q.
We have that $\alpha_k = q$, and hence for any $1 \le i \le 2^{k-1}$, $\delta_{k-1}^i - \alpha_k = q + 1$. Since $\lfloor \frac{n}{2^k} \rfloor = q$ and $\lceil \frac{n}{2^k} \rceil = q + 1$, Property 2 holds.

Case 3. $\frac{n}{2^{k-1}} = 2q + \epsilon$, for some integer q and $0 < \epsilon < 1$.
We have that $\lfloor \frac{n}{2^{k-1}} \rfloor = 2q$ and $\lceil \frac{n}{2^{k-1}} \rceil = 2q + 1$. Additionally, $\alpha_k = q$, and hence for any $1 \le i \le 2^{k-1}$, $(\delta_{k-1}^i - \alpha_k) \in \{q, q+1\}$. Since $\lfloor \frac{n}{2^k} \rfloor = q$ and $\lceil \frac{n}{2^k} \rceil = q + 1$, Property 2 holds.

Case 4. $\frac{n}{2^{k-1}} = 2q + 1 + \epsilon$, for some integer q and $0 < \epsilon < 1$.
We have that $\lfloor \frac{n}{2^{k-1}} \rfloor = 2q + 1$ and $\lceil \frac{n}{2^{k-1}} \rceil = 2q + 2$. Additionally, $\alpha_k = q + 1$, and hence for any $1 \le i \le 2^{k-1}$, $(\delta_{k-1}^i - \alpha_k) \in \{q, q+1\}$. Since $\lfloor \frac{n}{2^k} \rfloor = q$ and $\lceil \frac{n}{2^k} \rceil = q + 1$, Property 2 holds. ◀

Next, for $\lfloor \log n \rfloor \geq k > 0$ we set

$$A_i^k = \{i, i + \delta_k^1, i + \delta_k^1 + \delta_k^2, ..., i + \sum_{j=1}^{2^k - 1} \delta_k^j\}.$$

It is easy to see that for any $\lfloor \log n \rfloor \geq k > 0$ and i, we have that $|A_k^i| = 2^k$, $A_k^i = A_{k-1}^i \cup A_{k-1}^{i+\alpha_k}$ and $A_{k-1}^i \cap A_{k-1}^{i+\alpha_k} = \emptyset$. Also notice that for any $u \in A_{k-1}^i$, there exists $v \in A_{k-1}^{i+\alpha_k}$, such that $u + \alpha_k = v$ or $v + \alpha_k = u$ (For any $u \in A_{k-1}^{i+\alpha_k}$ there exists $v \in A_{k-1}^i$ with the same property).

Next we prove the lemma which is similar to the lemma for $n = 2^m$ case:

▶ **Lemma 7.** *For any $1 \leq i \leq n$ and $\lfloor \log n \rfloor \geq k > 0$, we have that*

$$2Gap_{A_k^i}(t) \leq 2 \max_{j \in A_k^i} |x_j(t) - x_{j+\alpha_k}(t)| + Gap_{A_{k-1}^{i+\alpha_k}}(t) + Gap_{A_{k-1}^i}(t). \tag{17}$$

Proof. Let $u = \arg\max_{j \in A_{k-1}^i} x_j(t)$ and let $v = \arg\min_{j \in A_{k-1}^i} x_j(t)$. We consider several cases:

Case 1. $u \in A_{k-1}^i$ and $v \in A_{k-1}^i$. Notice that in this case $Gap_{A_{k-1}^i}(t) = Gap_{A_k^i}(t)$. Let $u' \in A_{k-1}^{i+\alpha_k}$ be the vertex such that $u + \alpha_k = u'$ or $u' + \alpha_k = u$ and let $v' \in A_{k-1}^{i+\alpha_k}$ be the vertex such that $v + \alpha_k = v'$ or $v' + \alpha_k = v$. We have that

$$\begin{aligned} Gap_{A_k^i}(t) &= |x_u(t) - x_v(t)| \\ &\leq |x_{u'}(t) - x_u(t)| + |x_{v'}(t) - x_v(t)| + |x_{u'}(t) - x_{v'}(t)| \\ &\leq |x_{u'}(t) - x_u(t)| + |x_{v'} - x_v(t)| + Gap_{A_{k-1}^{i+\alpha_k}}(t) \\ &\leq 2 \max_{j \in A_k^i} |x_j(t) - x_{j+\alpha_k}(t)| + Gap_{A_{k-1}^{i+2^{k-1}}}(t). \end{aligned}$$

This gives us that

$$2Gap_{A_k^i}(t) \leq 2 \max_{j \in A_k^i} |x_j(t) - x_{j+\alpha_k}(t)| + Gap_{A_{k-1}^{i+\alpha_k}}(t) + Gap_{A_{k-1}^i}(t). \tag{18}$$

Case 2. $u \in A_{k-1}^i$ and $v \in A_{k-1}^{i+\alpha_k}$. Let $u' \in A_{k-1}^{i+\alpha_k}$ be the vertex such that $u + \alpha_k = u'$ or $u' + \alpha_k = u$ and let $v' \in A_{k-1}^i$ be the vertex such that $v + \alpha_k = v'$ or $v' + \alpha_k = v$. We have that:

$$\begin{aligned} Gap_{A_k^i}(t) = |x_u(t) - x_v(t)| &\leq |x_u(t) - x_{v'}(t)| + |x_{v'}(t) - x_v(t)| \\ &\leq Gap_{A_{k-1}^i}(t) + \max_{j \in A_k^i}(|x_j(t) - x_{j+\alpha_k}(t)|) \end{aligned}$$

and

$$\begin{aligned} Gap_{A_k^i}(t) = |x_u(t) - x_v(t)| &\leq |x_u(t) - x_{u'}(t)| + |x_{u'}(t) - x_v(t)| \\ &\leq Gap_{A_{k-1}^{i+\alpha_k}}(t) + \max_{j \in A_k^i}(|x_j(t) - x_{j+\alpha_k}(t)|) \end{aligned}$$

Hence, we again get that

$$2Gap_{A_k^i}(t) \leq 2 \max_{j \in A_k^i} |x_j(t) - x_{j+\alpha_k}(t)| + Gap_{A_{k-1}^{i+\alpha_k}}(t) + Gap_{A_{k-1}^i}(t). \tag{19}$$

Case 3. $u \in A_{k-1}^{i+\alpha_k}$ and $v \in A_{k-1}^{i+\alpha_k}$, is similar to Case 1.

Case 4. $v \in A_{k-1}^i$ and $u \in A_{k-1}^{i+\alpha_k}$, is similar to Case 2. ◀

Next, we upper bound $\sum_{i=1}^{n} \max_{j \in A_k^i} |x_j(t) - x_{j+\alpha_k}(t)|$.

▶ **Lemma 8.**

$$\sum_{i=1}^{n} \max_{j \in A_k^i} |x_j(t) - x_{j+\alpha_k}(t)| \leq \left\lceil \frac{n}{\lfloor \frac{n}{2^k} \rfloor} \right\rceil \sqrt{\lfloor \frac{n}{2^k} \rfloor} \sqrt{\phi_{\alpha_k}(t)} \tag{20}$$

Proof. Notice that for any $1 \leq u \leq n$ and sets $A_k^u, A_k^{u+1}, ..., A_k^{u+\lfloor \frac{n}{2^k} \rfloor - 1}$ are disjoint, because for any $1 \leq j \leq 2^k$, $\delta_k^j \geq \lfloor \frac{n}{2^k} \rfloor$ (This means that for any $1 \leq i \leq n$, distances between consecutive vertices in A_k^i are at least $\lfloor \frac{n}{2^k} \rfloor$). Using this fact and Cauchy-Schwarz inequality we get that

$$\begin{aligned}
&\sum_{i=u}^{u+\lfloor \frac{n}{2^k} \rfloor - 1} \max_{j \in A_k^i} |x_j(t) - x_{j+\alpha_k}(t)| \\
&\leq \sqrt{\lfloor \frac{n}{2^k} \rfloor} \sqrt{\sum_{i=u}^{u+\lfloor \frac{n}{2^k} \rfloor - 1} \max_{j \in A_k^i} |x_j(t) - x_{j+\alpha_k}(t)|^2} \\
&\leq \sqrt{\lfloor \frac{n}{2^k} \rfloor} \sqrt{\sum_{j=1}^{n} |x_j(t) - x_{j+\alpha_k}(t)|^2} = \sqrt{\lfloor \frac{n}{2^k} \rfloor} \sqrt{\phi_{\alpha_k}(t)}
\end{aligned}$$

Since the above inequality holds for any u we can write that:

$$\sum_{i=1}^{n} \max_{j \in A_k^i} |x_j(t) - x_{j+\alpha_k}(t)| \leq \left\lceil \frac{n}{\lfloor \frac{n}{2^k} \rfloor} \right\rceil \sqrt{\lfloor \frac{n}{2^k} \rfloor} \sqrt{\phi_{\alpha_k}(t)} \tag{21}$$

◀

With the above lemmas in place, we are ready to prove Theorem 4 for general n.

From Lemma 7 we have that

$$\begin{aligned}
\sum_{i=1}^{n} 2Gap_{A_k^i(t)} \leq \sum_{i=1}^{n} Gap_{A_{k-1}^i}(t) + \sum_{i=1}^{n} Gap_{A_{k-1}^{i+\alpha_k}}(t) \\
+ \sum_{i=1}^{n} 2 \max_{j \in A_k^i} |x_j(t) - x_{j+\alpha_k}(t)| \\
= 2\sum_{i=1}^{n} Gap_{A_{k-1}^i}(t) + 2\sum_{i=1}^{n} \max_{j \in A_k^i} |x_j(t) - x_{j+\alpha_k}(t)|.
\end{aligned}$$

After dividing the above inequality by 2 and applying Lemma 8: we get that:

$$\sum_{i=1}^{n} Gap_{A_k^i}(t) \leq \sum_{i=1}^{n} Gap_{A_{k-1}^i}(t) + \left\lceil \frac{n}{\lfloor \frac{n}{2^k} \rfloor} \right\rceil \sqrt{\lfloor \frac{n}{2^k} \rfloor} \sqrt{\phi_{\alpha_k}(t)}.$$

Notice that for any i, $Gap_0^i(t) = 0$. Hence, we get that

$$\sum_{i=1}^{n} Gap_{A_{\lfloor \log n \rfloor}^i}(t) \leq \sum_{k=1}^{\lfloor \log n \rfloor} \left\lceil \frac{n}{\lfloor \frac{n}{2^k} \rfloor} \right\rceil \sqrt{\lfloor \frac{n}{2^k} \rfloor} \sqrt{\phi_{\alpha_k}(t)}.$$

Let $i' = \arg\min_i Gap_{A^i_{\lfloor \log n \rfloor}}(t)$. Notice that consecutive vertices in $A^{i'}_{\lfloor \log n \rfloor}$ are 1 or 2 edges apart, hence for any $1 \le i \le n$, either $i \in A^{i'}_{\lfloor \log n \rfloor}$ or $i + 1 \in A^{i'}_{\lfloor \log n \rfloor}$. This gives us that

$$\begin{aligned} Gap(t) &\le Gap_{A^{i'}_{\lfloor \log n \rfloor}}(t) + \max_i |x_i(t) - x_{i+1}(t)| \\ &= Gap_{A^{i'}_{\lfloor \log n \rfloor}}(t) + \sqrt{\max_i |x_i(t) - x_{i+1}(t)|^2} \le Gap_{A^{i'}_{\lfloor \log n \rfloor}}(t) + \sqrt{\phi_1(t)}. \end{aligned}$$

By combining the above two inequalities we get that

$$\begin{aligned} nGap(t) &\le nGap_{A^{i'}_{\lfloor \log n \rfloor}}(t) + n\sqrt{\phi_1(t)} \le \sum_{i=1}^{n} Gap_{A^{i}_{\lfloor \log n \rfloor}}(t) + n\sqrt{\phi_1(t)} \\ &\le \sum_{k=1}^{\lfloor \log n \rfloor} \left\lceil \frac{n}{\lfloor \frac{n}{2^k} \rfloor} \right\rceil \sqrt{\lfloor \frac{n}{2^k} \rfloor} \sqrt{\phi_{\alpha_k}(t)} + n\sqrt{\phi_1(t)}. \end{aligned}$$

Next, we apply Jensen's inequality and Lemma 1 (We are going to use a looser upper bound: $\mathbb{E}[\phi_i(t)] \le i(n-i) - 1 \le in$)

$$\begin{aligned} n\mathbb{E}[Gap(t)] &\le n\mathbb{E}\sqrt{[\phi_1(t)]} + \sum_{k=1}^{\lfloor \log n \rfloor} \left\lceil \frac{n}{\lfloor \frac{n}{2^k} \rfloor} \right\rceil \sqrt{\lfloor \frac{n}{2^k} \rfloor} \mathbb{E}\sqrt{\phi_{\alpha_k}(t)} \\ &\le n\sqrt{\mathbb{E}[\phi_1(t)]} + \sum_{k=1}^{\lfloor \log n \rfloor} \left\lceil \frac{n}{\lfloor \frac{n}{2^k} \rfloor} \right\rceil \sqrt{\lfloor \frac{n}{2^k} \rfloor} \sqrt{\mathbb{E}[\phi_{\alpha_k}(t)]} \\ &\le n\sqrt{n} + \sum_{k=1}^{\lfloor \log n \rfloor} \Bigg(\left\lceil \frac{n}{\lfloor \frac{n}{2^k} \rfloor} \right\rceil \sqrt{\lfloor \frac{n}{2^k} \rfloor} \sqrt{\alpha_k n} = O(n\sqrt{n}\log n). \end{aligned}$$

This completes the proof.

6 Experimental Validation

On the practical side, we implemented our load balancing algorithm with unit weight increments on a cycle. The results confirm our hypothesis that the gap is of order $\Theta(\sqrt{n})$. In Figure 1 we ran our experiment 100 times and calculated average gap over the all runs. x-axis shows number of balls thrown(which is the same as the number of increments) and y-axis is current average gap divided by $\sqrt{n}$. The experiment shows that once the number of thrown balls is large enough, the gap stays between $\sqrt{n}$ and $1.4\sqrt{n}$.

7 Discussion and Future Work

We have shown that in the case of dynamic averaging on a cycle the gap between highest and lowest loaded bins is upper bounded by $O(\sqrt{n}\log n)$ in expectation. Additionally we showed that the expected square of the gap is lower bounded by $\Omega(n)$. It the future, it would be interesting to further tighten our results, matching our experimental analysis. We conjecture that the "correct" bound on the expected gap is of $\Theta(\sqrt{n})$. As already discussed, we also plan to extend our results to more general graph families, in particular grid graphs.

Comparison of two-choice and averaging load balancing. Finally, it is interesting to ask if it possible to extend our gap bounds in the case of the classic two-choice load balancing process. In particular, it is possible to show that the gap in the case of averaging process is

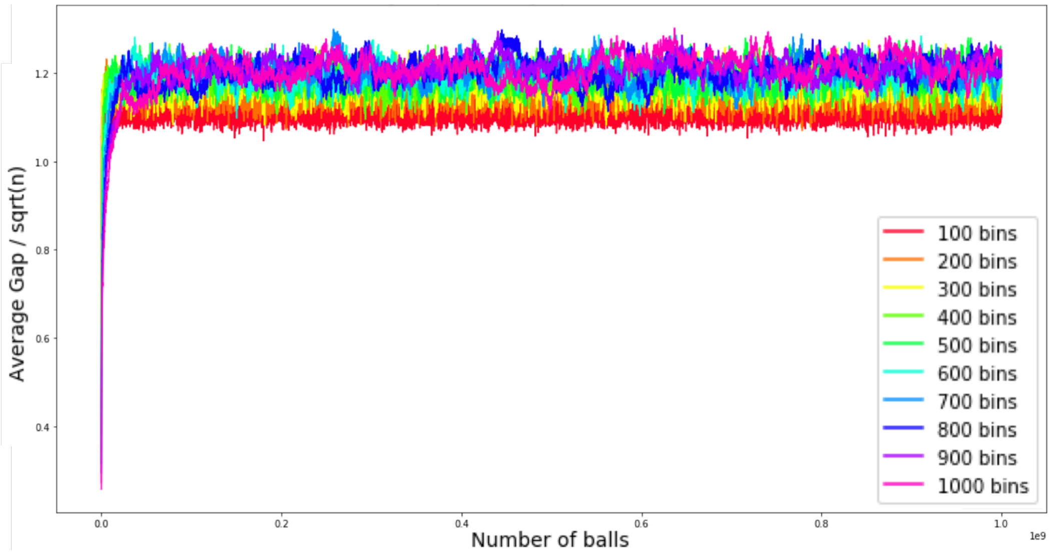


Figure 1 The evolution of average gap divided by square root of n, where n is the number of bins.

always smaller in expectation than the gap in the case of two choice process? Intuitively this should be the case, since the load balancing operation in the case of averaging can be viewed as picking up a random edge, incrementing the load of the less loaded endpoint, and then averaging the values. The extra averaging step should not make the gap larger. Indeed, the exponential potential used to analyse the gap in [10] can be used to upper bound the gap for averaging, since the exponential function is convex and averaging values does not increase it (by Jensen's inequality).

Unfortunately, it is not clear if averaging helps to actually *decrease* the exponential potential. Additionally, this argument shows that averaging does not make the gap worse if applied to the particular technique of upper bounding the gap, and it is not clear if the gap itself is actually smaller, if we use averaging on top of the two-choice process. We conjecture that there exists a majorization argument which is based on *how often* the process performs the averaging step. More precisely, we consider the setting where after the increment step (using two choice), we perform averaging with probability β. The gap should decrease in expectation as we increase β. Note that the only result which lower bounds the gap for the two-choice process on the cycle is the straightforward $\Omega(\log n)$ lower bound which can be shown for the clique [10]; so what makes the existence of the majorization argument interesting is that it would allow us to show that the lower bound we derived on the second moment of the gap while always performing averaging step on the cycle ($\beta = 1$) can be automatically used as the lower bound on the gap for two choice on the cycle ($\beta = 0$). We plan to investigate this connection in future work.

References

1 Dan Alistarh, Trevor Brown, Justin Kopinsky, Jerry Z Li, and Giorgi Nadiradze. Distributionally linearizable data structures. *arXiv preprint*, 2018. arXiv:1804.01018.

2 Dan Alistarh, Justin Kopinsky, Jerry Li, and Giorgi Nadiradze. The power of choice in priority scheduling. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 283–292. ACM, 2017.

3 Yossi Azar, Andrei Z Broder, Anna R Karlin, and Eli Upfal. Balanced allocations. *SIAM journal on computing*, 29(1):180–200, 1999.

4 Petra Berenbrink, Artur Czumaj, Angelika Steger, and Berthold Vöcking. Balanced allocations: The heavily loaded case. In *Proceedings of the Thirty-second Annual ACM Symposium on Theory of Computing*, STOC '00, pages 745–754, New York, NY, USA, 2000. ACM. doi:10.1145/335305.335411.

5 Alan Frieze, Páll Melsted, and Michael Mitzenmacher. An analysis of random-walk cuckoo hashing. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 490–503. Springer, 2009.

6 Frank Harary. The maximum connectivity of a graph. *Proceedings of the National Academy of Sciences of the United States of America*, 48(7):1142, 1962.

7 Michael Mitzenmacher. The power of two choices in randomized load balancing. *IEEE Transactions on Parallel and Distributed Systems*, 12(10):1094–1104, 2001.

8 Shanmugavelayutham Muthukrishnan, Bhaskar Ghosh, and Martin H Schultz. First-and second-order diffusive methods for rapid, coarse, distributed load balancing. *Theory of computing systems*, 31(4):331–354, 1998.

9 Yuval Peres, 2015. Personal Communication.

10 Yuval Peres, Kunal Talwar, and Udi Wieder. Graphical balanced allocations and the (1+β)-choice process. *Random Structures & Algorithms*, 47(4):760–775, 2015.

11 Andrea W Richa, M Mitzenmacher, and R Sitaraman. The power of two random choices: A survey of techniques and results. *Combinatorial Optimization*, 9:255–304, 2001.

12 Thomas Sauerwald and He Sun. Tight bounds for randomized load balancing on arbitrary network topologies. In *2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*, pages 341–350. IEEE, 2012.

13 Kunal Talwar and Udi Wieder. Balanced allocations: A simple proof for the heavily loaded case. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming – 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, volume 8572 of *Lecture Notes in Computer Science*, pages 979–990. Springer, 2014. doi:10.1007/978-3-662-43948-7_81.

Asynchronous Majority Dynamics in Preferential Attachment Trees

Maryam Bahrani
Princeton University, NJ, USA
mbahrani@alumni.princeton.edu

Nicole Immorlica
Microsoft Research, Cambridge, MA, USA
nicimm@microsoft.com

Divyarthi Mohan
Princeton University, NJ, USA
dm23@cs.princeton.edu

S. Matthew Weinberg
Princeton University, NJ, USA
smweinberg@princeton.edu

Abstract

We study information aggregation in networks where agents make binary decisions (labeled incorrect or correct). Agents initially form independent private beliefs about the better decision, which is correct with probability $1/2 + \delta$. The dynamics we consider are asynchronous (each round, a single agent updates their announced decision) and non-Bayesian (agents simply copy the majority announcements among their neighbors, tie-breaking in favor of their private signal).

Our main result proves that when the network is a tree formed according to the preferential attachment model [5], with high probability, the process stabilizes in a correct majority within $O(n \log n / \log \log n)$ rounds. We extend our results to other tree structures, including balanced M-ary trees for any M.

2012 ACM Subject Classification Theory of computation → Algorithmic game theory and mechanism design; Networks → Network dynamics

Keywords and phrases Opinion Dynamics, Information Cascades, Preferential Attachment, Majority Dynamics, non-Bayesian Asynchronous Learning, Stochastic Processes

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.8

Category Track A: Algorithms, Complexity and Games

Related Version A full version of the paper is available at `https://arxiv.org/abs/1907.05823`.

Funding *S. Matthew Weinberg*: Supported by NSF CCF-1717899.

1 Introduction

Individuals form opinions about the world both through private investigation and through discussion with one-another. A citizen, trying to decide which candidate's economic policies will lead to more jobs, might form an initial belief based on her own employment history. However, her stated opinion might be swayed by the opinions of her friends. The dynamics of this process, together with the social network structure of the individuals, can result in a variety of societal outcomes. Even if individuals are well-informed, i.e., are more likely to have correct than incorrect initial beliefs, certain dynamics and/or network structures can cause large portions of the population to form mistaken opinions.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 8; pp. 8:1–8:14
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

A substantial body of work exists modeling these dynamics mathematically, which we overview in Section 1.2. This paper focuses on the model of *asynchronous majority dynamics*. Initially, individuals have private beliefs over a binary state of the world, but no publicly stated opinion. Initial beliefs are independent: Correct with probability $1/2 + \delta$, and Incorrect with probability $1/2 - \delta$. In each time step, a random individual is selected to announce a public opinion. Each time an individual announces a public opinion, they simply copy the majority of their neighbors' announcements, tie-breaking in favor of their private belief. This is clearly naive: a true Bayesian would reason about the redundancy of information among the opinions of her friends, for example. Majority (or other non-Bayesian) dynamics are generally considered a more faithful model of agents with bounded rationality (e.g. voters), whereas Bayesian dynamics are generally considered a more faithful model of fully rational actors (e.g. financial traders). We consider asynchronous announcements[1] which are a more faithful model of human decisions (e.g. citizens deciding which candidate is better).

It's initially tempting to conjecture that these dynamics in a connected network should result in a Correct consensus; after all, the majority is initially Correct (with high probability) by assumption. Nonetheless, it's well-understood that individuals can fail miserably to learn. Suppose for instance that the individuals form a complete graph. Then in asynchronous majority dynamics, whichever individual is selected to announce first will have their opinion copied by the entire network. As this opinion is Incorrect with constant probability, there's a good chance that the entire network makes the wrong decision (this is known as an *information cascade*, and is not unique to asynchonous majority dynamics [4, 7]). So the overarching goal in these works is to understand in *which* graphs the dynamics stabilize in correctness with high probability.

For most previously studied dynamics (discussed in Section 1.2), "correctness" means a Correct consensus. This is because the models terminate in a consensus with probability 1, and the only question is whether this consensus is correct or not. With majority dynamics, it is certainly possible that the process stabilizes without a consensus. To see this, suppose individuals form a line graph. In this case, two adjacent individuals with the same initial belief are likely to form a "road block" (if both announce before their other neighbors), sticking to their initial beliefs throughout the process. In this case, with high probability a constant fraction of individuals terminate with a Correct opinion, but also a constant fraction terminate with an Incorrect opinion. As consensus is no longer guaranteed, we're instead interested in understanding network structures for which the dynamics converge, with high probability, to a *majority* of nodes having the Correct opinion (i.e., if a majority vote were to be taken, would it be correct w.h.p.?).

Prior work shows that, to reach a Correct consensus, it's sufficient for the social network to be sparse (every individual has only a constant number of neighbors) and expansive (every group of individuals have many friends outside the group) [12], and the tools developed indeed make strong use of both assumptions. Many networks of interest, however, like the hierarchy of employees in a corporation, are neither sparse nor expansive. Therefore, the focus of this paper is to push beyond these assumptions and develop tools for more general graphs.

[1] Unlike synchronous models where all agents announce simultaneously.

1.1 Our Results and Techniques

We focus our attention on trees, the simplest graphs outside the reach of prior techniques. In addition to modeling certain types of social networks (including hierarchical ones, or communication networks in which redundancy is expensive), and forming the backbone of many more, trees already present a number of technical challenges whose absence enabled the prior results. We study preferential attachment trees, which are well-studied graphs with rich structure[2]. Our main result is the following:

▶ **Theorem 1.** *Let G be a tree. Then with probability $1 - o(1)$*[3]*, asynchronous majority dynamics in G stabilizes in a* Correct *majority if:*

- *G is formed according to the preferential attachment model.*[4]
- *G is a balanced, M-ary tree of any degree.*[5]

Beyond Prior Tools

In prior work [12], the authors have two key ideas. Without yet getting into full details, one key idea crucially invokes sparsity to claim that most pairs of nodes u, v have distance $d(u, v) = \Omega(\ln n/\ln \ln n)$, which allows them to conclude that after $O(n \ln n/\ln \ln n)$ steps, most nodes are announcing a Correct opinion.

Still, just the fact that the dynamics hit a Correct majority along the way does not imply that the Correct majority will hold thru termination. To wrap up, they crucially invoke expansiveness (building off an argument of [24]) to claim that once there is a Correct majority, it spreads to a Correct consensus with high probability.

Both properties are necessary for prior work, and both properties fail in trees. For instance, the star graph is a tree, but $d(u, v) \leq 2$ for all u, v (precluding their "majority at $O(n \ln n/\ln \ln n)$" argument). Additionally, trees are not expansive. In particular, the line graph discussed earlier is a tree which hits a Correct majority at some point (as this tree happens to be sparse), but does not converge to consensus, so there is no hope for an argument like this. However, we believe that the process stabilizes in a correct majority in all trees.

▶ **Conjecture.** *Let G be any tree. Then the asynchronous majority dynamics stabilizes in a* Correct *majority with probability $1 - o(1)$.*

New Tools

Our main technical innovation is an approach to reason about majority without going through consensus. Specifically, we show in Sections 5 and 6 for preferential attachment trees, or balanced M-ary trees, that with probability $1 - o(1)$, a $1 - o(1)$ fraction of nodes have *finalized* after $O(n \ln n/\ln \ln n)$ steps. That is, after $O(n \ln n/\ln \ln n)$ steps, most (but not all) of the network has stabilized. The main barrier to extending our results to general trees is Section 5, as we require additional structure on the graphs to prove that the process stabilizes quickly. We postpone further details to Section 5, but just wish to highlight this approach as a fairly significant deviation from prior work.

[2] The more general preferential attachment graphs are a popular model of real networks.

[3] As $n \to \infty$ the probability converges to 1, where n is the number of nodes in G.

[4] That is, G is created by adding nodes one at a time. When a node is added, it attaches a single edge to a random previous node, selected proportional to its degree.

[5] That is, G can be rooted at some node v. All non-leaf nodes have M children, and all leaves have the same distance to v.

From here, our task is now reduced to showing that a Correct majority exists w.h.p. after $O(n \ln n / \ln \ln n)$ steps. Our main insight here is that most nodes with $d(u,v) = O(\ln n / \ln \ln n)$ must have some high-degree nodes along the path from u to v. We prove that such nodes act like a "road block," causing announcements on either side to be independent with high probability (and all nodes with $d(u,v) = \Omega(\ln n / \ln \ln n)$ can be handled with similar arguments to prior work).

1.2 Related Work

Information aggregation in social networks is an enormous field, and we will not come close to overviewing it in its entirety. Below, we'll briefly summarize the most related literature, restricting attention to works that consider two states of the world and independent initial beliefs are independently Correct with probability $1/2 + \delta$.

Bayesian Dynamics

In Bayesian models, agents are fully rational and sequentially perform Bayesian updates to their public opinion based on the public opinions of their neighbors. Seminal works of Banerjee [4] and Bikchandani, Hirshleifer, and Welch [7] first identified the potential of information cascades in this model. Subsequent works consider numerous variations, aiming to understand what assumptions on the underlying network or information structure results in Correct consensus [28, 3, 2]. Many other works studied repeated interactions of Bayesian agents in Social Networks [13, 27, 20, 26, 25, 23]. While the high-level goals of these works align with ours, technically they are mostly unrelated as we consider non-Bayesian dynamics.

Voter and DeGroot Dynamics

Prior work also considered other non-Bayesian dynamics. In voter dynamics, individuals update by copying a random neighbor [10, 18]. Similar dynamics (such as 3-majority, or k-majority) are analyzed from a distributed computing perspective with an emphasis on the rate of convergence to consensus [6, 15, 14]. In the DeGroot model, individuals announce an opinion in $[0,1]$ (as opposed to $\{0,1\}$), and update by averaging their neighbors [11, 16]. The biggest difference between these works and ours is that consensus is reached with probability 1 in these models on any connected graph, which doesn't hold for majority dynamics.

Majority Dynamics

The works most related to ours consider majority dynamics. Even synchronous majority dynamics may not result in a consensus (consider again the line graph). These works, like ours, therefore seek to understand what graph structures result in a Correct majority. Mossel, Neeman, and Tamuz study synchronous majority dynamics and prove that a Correct majority arises as long as the underlying graph is sufficiently symmetric, or sufficiently expansive (in the latter case, they prove that the network further reaches consensus) [24]. Feldman et al. study asynchronous majority dynamics and prove that a Correct consensus arises when the underlying graph is sparse and expansive [12]. Work of [29] further studies "retention of information," which asks whether *any* recovery procedure (not necessarily a majority vote) at stabilization can recover the ground truth with high probability. In connection to these, our work simply pushes the boundary beyond what classes of graphs are understood in prior work.

The key difference between the synchronous and asynchronous models is captured by the complete graph. In asynchronous dynamics, a Correct majority occurs only with probability $1/2 + \delta$, whereas in synchronous dynamics a Correct consensus occurs with probability $1 - \exp(-\Omega(n))$. This is because in step one, every node simply announces their private belief, and in step two everyone updates to the majority, which is Correct with probability $1 - \exp(-\Omega(n))$. So while the models bear some similarity, and some tools are indeed transferable (e.g. the expansiveness lemma of [24] used in [12]), much of the anlayses will necessarily diverge.

Preferential Attachment and Balanced M-ary Trees

There is also substantial prior work studying aggregation dynamics in trees. Here, the most related work is [19, 22], which studies synchronous majority dynamics in balanced M-ary trees. Less related are works which study "bottom-up" dynamics in balanced M-ary trees [21, 31, 30], k-majority dynamics in preferential attachment trees [1], or model cascades themselves as a preferential attachment tree [17]. While these works provide ample motivation for restricting attention to preferential attachment trees, or balanced M-ary trees, they bear no technical similarity to ours.

2 Model and Preliminaries

We consider an undirected tree $G = (V, E)$ with $|V(G)| = n$ and $|E(G)| = m$. We denote by $\deg(v)$ the degree of a node $v \in V(G)$, $N(v)$ to be its neighbors $\{u, (u, v) \in E\}$, and $d(u, v)$ to be the length of the unique path between u and v, and let $P(u, v)$ denote the ordered list of vertices on this path (i.e. starting with u and ending with v). We'll also denote by $D(G) = \max_{u,v}\{d(u, v)\}$ the diameter of G.

Individuals initially have one of two private beliefs, which we'll refer to as Correct (or 1) and Incorrect (or 0). That is, each $v \in V(G)$ receives an independent private signal $X(v) \in \{0, 1\}$, and $\Pr[X(v) = 1] = 1/2 + \delta$, for some constant $0 < \delta < 1/2$.

Individuals also have a *publicly announced* opinion (which we will simply refer to as an announcement). We define $C^t(v) \in \{\perp, 0, 1\}$ to be the public announcement of $v \in V(G)$ at time t. Initially, no announcements have been made, *i.e.* $C^0(v) = \perp$ for all v. In each subsequent step, a *single node* v^t is chosen uniformly at random from $V(G)$ and updates her announcement (announcements of all other nodes stay the same)[6]. v^t updates her announcement using *majority dynamics*. That is, if $N_1^t(v)$ denotes the number of v's neighbors with a Correct announcement at time t, and $N_0^t(v)$ denotes the number of v's neighbors with an Incorrect announcement, then:

$$C^t(v) = \begin{cases} 1 & \text{if } N_1^{t-1}(v) > N_0^{t-1}(v), \text{ and } v = v^t, \\ 0 & \text{if } N_1^{t-1}(v) < N_0^{t-1}(v), \text{ and } v = v^t, \\ X(v) & \text{if } N_1^{t-1}(v) = N_0^{t-1}(v), \text{ and } v = v^t, \\ C^{t-1}(v) & \text{if } v \neq v^t. \end{cases}$$

Note that we will treat δ as an absolute constant. Therefore, the only variable taken inside Big-Oh notation is n, the number of nodes (and, for instance, when we write $o(1)$ we mean any function of n that approaches 0 as n approaches ∞).

[6] This makes the process *asynchronous*.

As shown in [12], it is easy to see that in any network this process stabilizes with high probability in $O(n^2)$ steps. That is, the network reaches a state where no node will want to change its announcement and thus the process terminates.

2.1 Concentration Bounds and Tools from Prior Work

Our work indeed makes use of some tools from prior work to get started, which we state below. The concept of a *critical time*, defined below, is implicit in [12].

▶ **Definition 2.** *The* critical time[7] *from u to u, $T(u,u)$, is the first time that node u announces. The critical time from u to v, $T(u,v)$, is recursively defined as the first time that v announces* after *the critical time from u to x, where x is the neighbor of v in $P(u,v)$. We further denote the* critical chain *from u to v as the ordered list of critical times from u to x for all x on $P(u,v)$.*

The following lemma is a formal statement of ideas from prior work (a proof appears in Appendix A of the full version). To parse it, it will be helpful to think of the process as first drawing a countably infinite sequence S of nodes to announce, which then allows each $C^t(v)$ to be written as a deterministic function of the random variables $\{X(u), u \in V\}$. Lemma 3 below states that in fact, for early enough t, initial beliefs for only a proper subset of V suffice.

▶ **Lemma 3** ([12]). *For all t, and all v, $C^t(v)$ can be expressed as a function of the subset of signals $\{X(u), T(u,v) \leq t\}$.*

The final theorem we take from prior work is due to Mossel et al., and is used to claim that at minimum the *expected* number of CORRECT nodes at termination is at least $(1/2+\delta)n$.

▶ **Theorem 4** ([24]). *Let f be an odd, monotone Boolean function. Let $X_1, \ldots, X_n$ be input bits, each sampled i.i.d. from a distribution that is 1 with probability $p \geqslant 1/2$ and 0 otherwise. Then $\mathbb{E}[f(X_1, \ldots, X_n)] \geqslant p$.*

Note that, as long as v has announced at least once by t, $C^t(v)$ is an odd, monotone, Boolean function in variables $\{X(u), u \in V\}$,[8] and therefore $\Pr[C^t(v) = 1] \geq 1/2 + \delta$ for all v and $t \geq T(v,v)$.

Finally, we'll make use of the following concentration bound on $T(u,v)$ repeatedly. Its proof is a simple application of a Chernoff bound and appears in Appendix A of the full version.

▶ **Lemma 5.** *For all $0 < \beta < 1$:*

- $\Pr[T(u,v) > 8 \cdot \max\{\ln(1/\beta), d(u,v)+1\} \cdot n] \leq \beta^2$.
- $\Pr[T(u,v) < (d(u,v)+1) \cdot \beta \cdot n] \leq e^{-\beta d(u,v)(1-\beta)^2/3}$.
- $\Pr[T(u,v) < (d(u,v)+1) \cdot \beta \cdot n] \leq (e\beta)^{d(u,v)} = e^{(1+\ln\beta) \cdot d(u,v)}$.

[7] This definition can be naturally extended to any general graph.

[8] That is, flipping all $X(v)$ simultaneously to $1 - X(v)$ would cause $C^t(v)$ to flip (odd), and changing any subset of initial beliefs from 0 to 1 cannot change $C^t(v)$ from 1 to 0 (monotone).

3 Key Concepts

Before getting into our proofs, we elaborate some key concepts that will be used throughout. In Proposition 7 below, we analyze the connection between critical chains and switches in announcements. Intuitively, Proposition 7 is claiming that every fresh announcement can cause other nodes to switch a previous announcement along critical chains, but that these are the *only* switches that can occur.

▶ **Definition 6.** *Let v change her announcement at t, and her previous announcement be made at $t' > 0$. We say that node u is a* cause *of v changing her announcement at t if $C^t(u) = C^t(v)$, and $C^{t'}(u) \neq C^t(v)$. Observe that every such change in announcement has a cause.*

▶ **Proposition 7.** *If $C^t(v) \neq C^{t-1}(v)$, then there exists a node u such that:*

- *$t = T(u, v)$ (i.e. the influence of u just reaches v at time t).*
- *$C^{T(u,u)}(u) = C^t(v)$ (i.e. v is updating to match u's initial announcement).*
- *Denote $u = x_0, x_1, \ldots, x_{d(u,v)} = v$ the path $P(u, v)$. Then* every *x_i, $i > 0$, has $C^{T(u,x_i)-1}(x_i) = C^{t-1}(v)$ and $C^{T(u,x_i)}(x_i) = C^t(v)$, and x_{i-1} caused this change (i.e.* every *node along the path from u to v* changed *to match u's initial announcement).*

Proof. The proof proceeds by induction on t. Consider $t = 1$ as a base case. If $C^1(v) \neq C^0(v) = \perp$, then it must be because v announced at time 1, meaning that $1 = T(v, v)$ as desired.

Now assume that for all v and all $t' < t$ the claim holds, and consider time t. If v does not announce at time t then the claim vacuously holds. If v announces at time t but does *not* change their announcement, then again the claim vacuously holds. If v announces at time t for the first time, then v itself is the desired u and the claim holds. The remaining case is if v changes their previous announcement that was made at time $t' < t$ (and v did not announce between t' and t).

Let's consider the state of affairs at time t', when v announced some opinion A. This means that, at time t', a majority (tie-breaking for $X(v)$) of v's neighbors were announcing A. Yet, at time t, a majority (tie-breaking for $X(v)$) of v's neighbors were announcing $B = 1 - A$. Therefore, *some* node adjacent to v must have switched its announcement to B at some $t'' \in (t', t)$, and stays B till time t (and caused the change). Call this node x. We now wish to invoke the inductive hypothesis for x at t''.

The inductive hypothesis claims there there is some u (maybe $u = x$) such that u made B as its first announcement, and then every node y along the critical chain from u to x switched from A to B at $T(u, y)$ (caused by its predecessor), and that $t'' = T(u, x)$. Let's first consider the case that v is not on the path from u to x (and therefore x is on the path from u to v, since they are adjacent). Then as $T(u, x) = t'' \in (t', t)$, and v does not announce in (t', t), we see that $T(u, v) = t$ (immediately by definition of critical times). Moreover, as $P(u, v)$ is simply $P(u, x)$ concatenated with v, the inductive hypothesis already guarantees that u announced B at $T(u, u)$, and that every node y on $P(u, v)$ switched from A to B at $T(u, y)$. So the last step is to show that in fact v *must* not be on the path from u to x, and then the inductive step will be complete.

Finally, we show that we cannot have v on the path from u to x, completing the inductive step. Assume for contradiction that v were on the path from u to x. Then as $t'' = T(u, x)$, we would necessarily have $t' \geq T(u, v)$ (immediately from definition of critical times). However, by hypothesis, $C^{t''}(v) = C^{t'}(v) = A$ (the first equality is simply because v does not announce in $(t', t'']$), contradicting the inductive hypothesis that v caused x to change (because of u), which would imply instead that $C^{T(u,v)}(v) = C^{t''}(v) = B$. So v cannot be on the path from u to x. ◀

Below, we make use of Proposition 7 to prove that, in any tree[9], the process terminates quickly (proof in Appendix B of the full version). Note that [12] already proves that the process on trees terminates with probability $1 - o(1)$ after $O(n^2)$ steps, so Corollary 8 is a strict improvement when the diameter $D(G) = o(n)$.

▶ **Corollary 8.** *Let T_{stable} denote the last time that a node changes its announcement. Then with probability $1 - o(1)$, $T_{stable} \leq 8 \cdot \max\{2\ln(n), D(G) + 1\} \cdot n$.*

Finally, we prove one last proposition which will be used in future sections regarding the probability that a single node announces Correct throughout the process (the proof appears in Appendix B of the full version). Beginning with v's first announcement, because the graph is a tree, prior to v's first announcement all of v's neighbors' announcements are independent. Therefore, it initially seems like we should expect v's initial announcement to be Correct except with probability exponentially small in $\deg(v)$ – indeed, this would hold if the dynamics were synchronous. However, since the dynamics are *asynchronous*, there's a good chance that v announces before any of its neighbors and simply announces $X(v)$. That is, the probability that v's initial announcement is Incorrect is at least $\frac{1/2-\delta}{\deg(v)}$, so we cannot hope for such strong guarantees. This observation highlights one (of several) crucial differences between synchronous and asynchronous dynamics. Still, the proposition below shows roughly that the only bad event is v announcing before many of its neighbors. Below for a set S, we'll use $C_S^t(v)$ to denote the following modified dynamics: First, set $C_S^t(v) = $ Incorrect for all $v \in S$, and all t. Then, run the asynchronous majority dynamics as normal. In other words, the modified dynamics hard-code an Incorrect announcement for all nodes in S and otherwise run asynchronous majority dynamics as usual (this extension will be necessary for a later argument).

▶ **Definition 9.** *We say that a node v is* safe thru T *if $C^t(v) \in \{\perp, 1\}$ for all $t \leq T$. We further say that a node v is* safe thru T, even against S *if $C_S^t(v) \in \{\perp, 1\}$ for all $t \leq T$.*

▶ **Proposition 10.** *For all a, there exist constants[10] b, c such that for any S with $|S| = a$, $v \notin S$ and $T \leq n \cdot e^{b\deg(v)}$, v is safe thru T, even against S, with probability at least $1 - c/\deg(v)$.*

4 Forming an Initial Majority

In this section, we prove that in any tree, a Correct majority forms after a near-linear number of steps (but may later fade). The main idea is to show that the announcements of most pairs of nodes are independent with probability $1 - o(1)$, and use Chebyshev's inequality to show that the number of Correct announcements therefore concentrates around its expectation. The independence argument is the crux of the proof. To show it, we consider three cases depending on the length and degree sequence of the path between a pair of nodes. If the path is long, then, similar to prior work, there is simply not enough time for the pair to influence each other. If the path is short, but (some of) the intermediate nodes have high degrees, then these effectively block influence because the announcements of these high-degree nodes is effectively independent of what's happening on the path. Finally, if the path is short and the intermediate nodes have low degrees, then the pair certainly may influence each other. However, a counting argument shows there can only be a vanishingly small fraction of such pairs. The main result of this section is the following:

[9] Proposition 7 and Corollary 8 hold for any general graph. Since we are only interested in trees, we restrict our proofs to just trees for brevity.

[10] does not depend on n, but may depend on δ.

▶ **Theorem 11** (Majority in trees). *For sufficiently large n, any tree on n nodes and any $T \leq \frac{n \ln n}{32 \ln \ln n}$, after T steps, with probability at least $1 - O(e^{-\ln n/(24 \ln \ln n)})$, the announcements of at least $(\frac{1}{2} + \frac{\delta}{2} - e^{-T/n}) \cdot n$ nodes are* Correct *.*

Further, for all constants $\gamma > 0$, there exists a constant $\alpha > 0$ such that for sufficiently large n, when $T \leq \frac{n \ln^{1-\gamma} n}{\ln \ln n}$, after T steps, with probability at least $1 - n^{-\alpha}$, the announcements of at least $(\frac{1}{2} + \frac{\delta}{2} - e^{-T/n}) \cdot n$ nodes are Correct *.*

First, we analyze the expected number of Correct nodes using Theorem 4 (proof in Appendix C of the full version). Note that, the probability that a node v has announced by T is at least $(1 - e^{-T/n})$[11].

▶ **Lemma 12.** *At any time T, the expected number of nodes v with $C^T(v) =$ Correct is at least $(1/2 + \delta - e^{-T/n})n$.*

From here, we now need to show that the number of Correct announcements concentrates around its expectation. To this end, we'll show that most pairs of nodes can be written as functions of disjoint initial beliefs, and are therefore independent. Ideas from [12] formally show that this suffices:

▶ **Definition 13.** *We say that two nodes u, v are ε-disjoint at t if there exist random variables X_u, X_v, written as functions of disjoint sets of initial beliefs (and therefore independent), such that $\Pr[C^t(u) = X_u] \geq 1 - \varepsilon$ and $\Pr[C^t(v) = X_v] \geq 1 - \varepsilon$.*

▶ **Lemma 14** (Inspired by [12]). *Let ε^t_{uv} be such that u, v are ε^t_{uv}-disjoint at t. Then if $\sum_{u,v} \varepsilon^t_{uv} = D$, the number of* Correct *nodes at time t is within $\delta n/2$ of its expectation with probability $1 - \frac{4n+16D}{\delta^2 n^2}$.*

So our remaining task is to upper bound $\sum_{u,v} \varepsilon^t_{uv}$, and this is the point where we diverge from prior work. For a given pair u, v, there are three possible cases. Below, case one is most similar to prior work, and cases two/three are fairly distinct.

Case One: Long Paths

One possibility is that $d(u, v) \geq f(n)$, for some $f(n)$ to be decided later. The following proposition implies that at any time T, the announcements of pairs of nodes at a large enough distance are almost independent. The proof is provided in Appendix C of the full version.

▶ **Proposition 15.** *Let $d(u, v) \geq \max\{kT, f(n)\}$ for $k \geq 4$. Then $\varepsilon^T_{uv} \leq 2e^{-f(n)/24}$, and $\varepsilon^T_{uv} \leq 2e^{(1-\ln k)\cdot f(n)}$.*

Case Two: Short Paths A

Another possibility is that $d(u, v) < f(n)$. Here, there will be two subcases. First, maybe it's the case that $d(u, v)$ is small *and* the product of degrees on the path from u to v is small. In this case, it very well could be that ε^t_{uv} is large, which is bad. However, we prove that there cannot be many such pairs (and so in total they contribute $o(n^2)$ to the sum). The following lemma shows in fact that even if we remove the restriction that $d(u, v) = O(T)$, there simply cannot be many pairs of nodes such that the product of degrees on $P(u, v)$ is small (proof in Appendix C of the full version).

[11] The probability that v wasn't chosen in all T rounds is $\left(1 - \left(1 - \frac{1}{n}\right)^T\right)$.

▶ **Lemma 16.** *Let K be the set of pairs of nodes (u, v) such that $\prod_{w \in P(u,v)} \deg(w) \leq X$. Then $|K| \leq Xn/2$.*

Case Three: Short Paths B

The final possibility is that $d(u, v) < f(n)$, and also that $\prod_{w \in P(u,v)} \deg(w)$ is large. In this case, we will prove that with probability $1 - o(1)$ there is some block in $P(u, v)$ causing u's and v's announcemnts to be independent (proof in Appendix C of the full version).

▶ **Definition 17.** *We say that a node $x \in P(u, v)$* cuts *u from v thru T if some node y in $P(u, x)$ is safe thru T even against S_y, where S_y are y's (at most) two neighbors in $P(u, v)$.*

▶ **Lemma 18.** *Let T be any time and p_x be the probability that x cuts u from v thru T* and also *cuts v from u thru T. Then u and v are p_x-disjoint at T.*

Next, we wish to show that with good probability there is indeed a node on $P(u, v)$ that cuts u from v and also v from u (proofs in Appendix C of the full version).

▶ **Lemma 19.** *There exist absolute constants b, d such that for any pairs of nodes $u, v \in V$ with $\prod_{w \in P(u,v)} \deg(w) = X$, $d(u, v) \leq \frac{\ln(X)}{d}$, and $T \leq ne^{bX^{1/(4d(u,v))}}$, there exists an x such that that with probability $1 - X^{-1/8}$, x cuts u from v thru T and also v from u thru T.*

▶ **Corollary 20.** *There exist absolute constants b, d such that for pairs of nodes any $u, v \in V$ with $\prod_{w \in P(u,v)} \deg(w) = X$, $d(u, v) \leq \frac{\ln(X)}{d}$, and $T \leq ne^{bX^{1/(4d(u,v))}}$), u and v are $X^{-1/8}$-disjoint at T.*

Now, we'll put together case one, Lemma 18 and Corollary 20 together to prove Theorem 11, which is mostly a matter of setting parameters straight (and appears in Appendix C of the full version).

To conclude, at this point we have proven that a majority takes hold after $n\frac{\ln n}{32 \ln \ln n}$ steps for any tree. The remaining work is to prove that it does not disappear.

5 Stabilizing Quickly

In this section, we identify properties of a tree which cause it to stabilize quickly. Our main theorem will then follow by proving that both balanced M-ary trees and preferential attachment trees have this property. The main idea is to consider nodes that are "close" to leaves in the following formal sense:

▶ **Definition 21.** *We say that a node v is an (X, Y)-leaf in G if there exists a rooting of G such that v has $\leq X$ descendants, and the longest path from v to one of its descendants is at most Y. Note that leaves are $(0, 0)$-leaves. When we refer to a node's parent, children, or descendants, it will be with respect to this rooting.*

▶ **Definition 22.** *We say that a node v is:*

- finalized *at T, if $C^t(v) = C^T(v)$ for all $t \geq T$.*
- nearly-finalized *at T with respect to u if there exists a $t' \geq T$ such that v is finalized at t' and for all $t \in (T, t')$ when v announces, it either updates $C^t(v) = C^t(u)$, if $C^t(u) \neq \perp$, or $C^t(v) = C^{t-1}(v)$, if $C^t(u) = \perp$.*

Intuitively, a node is finalized if it is done changing its announcement. A node v is nearly-finalized with respect to u if v is not quite finalized, but changes in u are the only reason why v would change its announcement (and moreover, v will copy u every announcement until v finalizes).

The main result of this section is as follows:

▶ **Theorem 23.** *Let v be an (X, Y)-leaf. Then with probability $1 - Xe^{-T/nY}$, v is nearly-finalized at T with respect to its parent.*

The main insight for the proof of Theorem 23 will be the following lemmas. Below, Lemma 24 asserts that once all of v's children are nearly-finalized with respect to v, any changes in v's opinion are to copy its parent, and Lemma 25 builds off this to claim that we can relate the time until v nearly-finalizes to its critical times. Importantly, Lemma 25 does *not* require *all* critical paths to hit v, but only those from its descendents.

▶ **Lemma 24.** *Let all of v's children be nearly-finalized with respect to v at T, and let u be v's parent. Let also $t > t' \geq T$ be two timesteps during which v announced. Then if $C^t(v) \neq C^{t-1}(v)$, we must have $C^t(v) = C^t(u)$.*

▶ **Lemma 25.** *Let $T_v := \max\{T(x, v), x$ is a descendant of $v\}$. Then v is nearly-finalized at T_v with respect to its parent.*

These above lemmas suffice to prove Theorem 23. The proofs of these lemmas and Theorem 23 appear in Appendix D of the full version.

We will also need the following implications of Theorem 23. Below, Lemma 26 will be helpful in proving Corollary 27. Corollary 27 lets us claim that while nearly-finalized nodes are not themselves finalized, their existence implies the existence of other finalized nodes. This will be helpful in wrapping up in the following section, since the process only terminates once nodes are finalized. The proofs of Lemma 26 and Corollary 27 appear in Appendix D of the full version.

▶ **Lemma 26.** *For any $t > T_v$, if a child of v changes their announcement at t, v becomes finalized at t.*

▶ **Corollary 27.** *For any T, with probability $1 - e^{-T/n}$, v has $\lfloor(\deg(v) - 1)/2\rfloor$ children who are finalized at $T_v + T$.*

Moreover, if v is an (X, Y)-leaf and finalized at $t \geq T_v$, then with probability $1 - Xe^{-T/nY}$, all of v's descendants are finalized at $t + T$.

6 Wrapping Up: Preferential Attachment and Balanced *M*-ary Trees

In this section, we show how to make use of Theorem 23 to conclude that a $1 - o(1)$ fraction of nodes are finalized by $\frac{n \ln n}{32 \ln \ln n}$. Proofs for the two cases follow different paths, but both get most of their mileage from the developments in Section 5.

6.1 Preferential Attachment Trees Stabilize Quickly

Let's first be clear what we mean by a preferential attachment tree.[12]

[12] Note that this is the standard definition of preferential attachment used for heuristic arguments, e.g. [5]. Most prior rigorous work uses a slightly modified definition that produces a forest instead of a tree in order to rigorously analyze (say) the degree distribution [9, 8]. As we are only interested in (fairly loose) bounds on the degrees, our results are rigorous in the standard model.

▶ **Definition 28** (Preferential Attachment Tree). *n nodes arrive sequentially, attaching a single edge to a pre-existing node at random proportional to its degree. Specifically:*

- *Let v_i denote the i^{th} node to arrive.*
- *Let $\deg_t(v_i)$ denote the degree of node v_i after a total of t nodes have arrived.*
- *There is a special node v_0, which only v_1 connects to upon arrival, and no future nodes.*
- *When v_{i+1} arrives, v_{i+1} attaches a single edge to a previous node, choosing node v_j, $j \in [1, i]$, with probability $\frac{\deg_i(v_j)}{2i-1}$.*

Our main argument for preferential attachment trees is that most nodes are in a "good" subtree, defined below. All subsequent proofs are in Appendix E of the full version. At a high level the plan is as follows: first, we prove that because most nodes are (X, Y)-leaves for small X, Y, these nodes quickly become nearly-finalized. Next, we prove that most such nodes are part of a small subtree whose parent is likely to be safe thru the entire process. Therefore, the parent of this subtree is finalized early, and once the subtree becomes nearly-finalized, it finalizes quickly as well.

▶ **Definition 29.** *Say that a subtree rooted at v is* good *if:*

- *v is a (X, Y)-leaf, for $X = \ln^{O(1)} n$ and $Y = O(\ln \ln n)$.*
- *v's parent has degree at least $\ln^{\Omega(1)} n$.*

▶ **Proposition 30.** *Let the subtree rooted at v be good, and let the diameter of the entire graph be $O(\ln n)$. Then with probability $1 - o(1)$, the entire subtree rooted at v is finalized by $n \frac{\ln n}{32 \ln \ln n}$.*

▶ **Proposition 31.** *For a tree built according to the preferential attachment model, the following simultaneously hold with probability $1 - o(1)$.*

- *$n - o(n)$ nodes are in good subtrees.*
- *The diameter of the entire graph is $O(\ln n)$.*

▶ **Theorem 32.** *A tree built according to the preferential attachment model stabilizes in a* Correct *majority with probability $1 - o(1)$.*

The proofs for Proposition 30, Proposition 31, and Theorem 32 appear in Appendix E of the full version.

6.2 Balanced *M*-ary Trees Stablize Quickly

Let's first be clear what we mean by a balanced M-ary tree.

▶ **Definition 33.** *We say a tree is a balanced M-ary tree if there is a root v such that all non-leaf nodes have exactly M children, and all root-leaf paths have the same length.*

Our plan of attack is as follows (all proofs are in Appendix E of the full version). First, the case for large M (say, $M > \ln n$) is actually fairly straight-forward as a result of Proposition 10. This is because every pair of nodes has a high-degree block on their path, meaning that the "Case Three" argument used in Section 4 actually applies all the way until the process terminates. The $M \leq \ln n$ case is more interesting, and requires the tools developed in Section 5.

Here, the plan is as follows. Corollary 27 roughly lets us claim that all nearly-finalized nodes must have a decent number of finalized children, and moreover that all these finalized children have finalized descendents. Iterating this counting inductively through children, we see that actually most descendents of nearly-finalized nodes of sufficient height must themselves be finalized.

Formally, the approach is to first get a bound on the height for which we can claim that nodes are indeed nearly-finalized with high probability (Corollary 34, immediately from Theorem 23). $\ln\ln n$ turns out to be a good choice.

▶ **Corollary 34.** *Let v be distance h from a leaf. Then v is an $(2M^h, h)$-leaf, and therefore v is nearly-finalized with respect to its parent at $\frac{n\ln n}{64\ln\ln n}$ with probability $1 - 2M^h \cdot e^{-\frac{\ln n}{64h\ln\ln n}} = 1 - e^{-\frac{\ln n}{64h\ln\ln n} + h\ln(2M)}$.*

In particular, if $h = o(\sqrt{\frac{\ln n}{\ln\ln n \cdot \ln M}})$, then v is nearly-finalized with respect to its parent at $\frac{n\ln n}{64\ln\ln n}$ with probability $1 - o(1)$.

▶ **Proposition 35.** *Let v have height $h = \ln\ln n$ in a balanced M-ary tree for $M \leq \ln n$. Then with probability $1 - o(1)$, at most $2M^h \cdot (2/3)^h = o(M^h)$ descendents of v are not finalized by $\frac{n\ln n}{32\ln\ln n}$.*

▶ **Theorem 36.** *Any M-ary tree stablizes in a* Correct *majority with probability $1 - o(1)$.*

The proofs of Proposition 35 and Theorem 36 appear in Appendix E of the full version.

References

1 Mohammed Amin Abdullah, Michel Bode, and Nikolaos Fountoulakis. Local majority dynamics on preferential attachment graphs. In *Proceedings of the 12th International Workshop on Algorithms and Models for the Web Graph - Volume 9479*, WAW 2015, pages 95–106, Berlin, Heidelberg, 2015. Springer-Verlag. `doi:10.1007/978-3-319-26784-5_8`.

2 Daron Acemoglu, Munther A. Dahleh, Ilan Lobel, and Asuman Ozdaglar. Bayesian learning in social networks. *Review of Economic Studies*, 78(4):1201–1236, 2011.

3 Abhijit Banerjee and Drew Fudenberg. Word-of-mouth learning. *Games and Economic Behavior*, 46(1):1–22, January 2004. URL: `http://ideas.repec.org/a/eee/gamebe/v46y2004i1p1-22.html`.

4 Abhijit V. Banerjee. A simple model of herd behavior. *The Quarterly Journal of Economics*, 107(3):797–817, 1992.

5 Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999. `doi:10.1126/science.286.5439.509`.

6 Luca Bechetti, Andrea E.F. Clementi, Emanuele Natale, Francesco Pasquale, and Luca Trevisan. Stabilizing consensus with many opinions. In *ACM Symposium on Discrete Algorithms (SODA)*, 2016.

7 Sushil Bikhchandani, David Hirshleifer, and Ivo Welch. A theory of fads, fashion, custom, and cultural change in informational cascades. *Journal of Political Economy*, 100(5):992–1026, October 1992.

8 Béla Bollobás and Oliver Riordan. The diameter of a scale-free random graph. *Combinatorica*, 24(1):5–34, 2004. `doi:10.1007/s00493-004-0002-2`.

9 Béla Bollobás, Oliver Riordan, Joel Spencer, and Gábor E. Tusnády. The degree sequence of a scale-free random graph process. *Random Struct. Algorithms*, 18(3):279–290, 2001. `doi:10.1002/rsa.1009`.

10 Peter Clifford and Aidan Sudbury. A model for spatial conflict. *Biometrika*, 60(3):581–588, 1973. URL: `http://www.jstor.org/stable/2335008`.

11 Morris H. DeGroot. Reaching a consensus. *Review of Economic Studies*, 69(345):118–121, 1974.

12 Michal Feldman, Nicole Immorlica, Brendan Lucier, and S. Matthew Weinberg. Reaching consensus via non-bayesian asynchronous learning in social networks. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2014, September 4-6, 2014, Barcelona, Spain*, 2014. `doi:10.4230/LIPIcs.APPROX-RANDOM.2014.192`.

13 Douglas Gale and Shachar Kariv. Bayesian learning in social networks. *Games and Economic Behavior*, 45(2):329–346, 2003. Special Issue in Honor of Robert W. Rosenthal. `doi:10.1016/S0899-8256(03)00144-1`.

14 Mohsen Ghaffari and Johannes Lengler. Nearly-tight analysis for 2-choice and 3-majority consensus dynamics. In *ACM Symposium on Principles of Distributed Computing (PODC)*, 2018.

15 Mohsen Ghaffari and Merav Parter. A polylogarithmic gossip algorithm for plurality consensus. In *ACM Symposium on Principles of Distributed Computing (PODC)*, 2016.

16 Benjamin Golub and Matthew O. Jackson. Naïve learning in social networks and the wisdom of crowds. *American Economic Journal: Microeconomics*, 2(1):112–149, 2010.

17 Vicenç Gómez, Hilbert J Kappen, and Andreas Kaltenbrunner. Modeling the structure and evolution of discussion cascades. In *Proceedings of the 22nd ACM conference on Hypertext and hypermedia*, pages 181–190. ACM, 2011.

18 Richard A. Holley and Thomas M. Liggett. Ergodic theorems for weakly interacting infinite systems and the voter model. *The Annals of Probability*, 3(4):643–663, 1975. URL: `http://www.jstor.org/stable/2959329`.

19 C Douglas Howard. Zero-temperature ising spin dynamics on the homogeneous tree of degree three. *Journal of applied probability*, 37(3):736–747, 2000.

20 Y. Kanoria and O. Tamuz. Tractable bayesian social learning on trees. *IEEE Journal on Selected Areas in Communications*, 31(4):756–765, 2013.

21 Yashodhan Kanoria and Andrea Montanari. Subexponential convergence for information aggregation on regular trees. In *Decision and Control and European Control Conference (CDC-ECC), 2011 50th IEEE Conference on*, pages 5317–5322. IEEE, 2011.

22 Yashodhan Kanoria, Andrea Montanari, et al. Majority dynamics on trees and the dynamic cavity method. *The Annals of Applied Probability*, 21(5):1694–1748, 2011.

23 E. Mossel, N. Olsman, and O. Tamuz. Efficient bayesian learning in social networks with gaussian estimators. In *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, 2016.

24 Elchanan Mossel, Joe Neeman, and Omer Tamuz. Majority dynamics and aggregation of information in social networks. In *Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2013.

25 Elchanan Mossel, Allan Sly, and Omer Tamuz. Asymptotic learning on bayesian social networks. *Probability Theory and Related Fields*, 2014.

26 Manuel Mueller-Frank. A general framework for rational learning in social networks. *Theoretical Economics*, 8(1):1–40, 2013.

27 Dinah Rosenberg, Eilon Solan, and Nicolas Vieille. Informational externalities and emergence of consensus. *Games and Economic Behavior*, 66(2):979–994, 2009.

28 Lones Smith and Peter Sorensen. Pathological outcomes of observational learning. *Econometrica*, 68(2):371–398, March 2000. URL: `http://ideas.repec.org/a/ecm/emetrp/v68y2000i2p371-398.html`.

29 Omer Tamuz and Ran Tessler. Majority dynamics and the retention of information. In *Working paper*, 2013.

30 Y. P. Tian, X. J. Sun, and O. Tian. Detection performance of the majority dominance rule in m-ary relay trees with node and link failures. *IEEE Transactions on Signal Processing*, 66(6):1469–1482, March 2018. `doi:10.1109/TSP.2017.2788418`.

31 Zhenliang Zhang, Edwin KP Chong, Ali Pezeshki, William Moran, and Stephen D Howard. Learning in hierarchical social networks. *IEEE Journal of Selected Topics in Signal Processing*, 7(2):305–317, 2013.

The Power of Many Samples in Query Complexity

Andrew Bassilakis
Stanford University, CA, USA
abass20@stanford.edu

Andrew Drucker
University of Chicago, IL, USA
andy.drucker@gmail.com

Mika Göös
Stanford University, CA, USA
goos@stanford.edu

Lunjia Hu
Stanford University, CA, USA
lunjia@stanford.edu

Weiyun Ma
Stanford University, CA, USA
wyma@cs.stanford.edu

Li-Yang Tan
Stanford University, CA, USA
liyang@cs.stanford.edu

Abstract

The randomized query complexity $\mathsf{R}(f)$ of a boolean function $f\colon \{0,1\}^n \to \{0,1\}$ is famously characterized (via Yao's minimax) by the least number of queries needed to distinguish a distribution $\mathcal{D}_0$ over 0-inputs from a distribution $\mathcal{D}_1$ over 1-inputs, maximized over all pairs $(\mathcal{D}_0, \mathcal{D}_1)$. We ask: Does this task become easier if we allow query access to infinitely many samples from either $\mathcal{D}_0$ or $\mathcal{D}_1$? We show the answer is *no*: There exists a hard pair $(\mathcal{D}_0, \mathcal{D}_1)$ such that distinguishing $\mathcal{D}_0^\infty$ from $\mathcal{D}_1^\infty$ requires $\Theta(\mathsf{R}(f))$ many queries. As an application, we show that for any composed function $f \circ g$ we have $\mathsf{R}(f \circ g) \geq \Omega(\mathsf{fbs}(f)\mathsf{R}(g))$ where fbs denotes fractional block sensitivity.

2012 ACM Subject Classification Theory of computation → Oracles and decision trees; Theory of computation → Probabilistic computation

Keywords and phrases Query complexity, Composition theorems

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.9

Category Track A: Algorithms, Complexity and Games

Funding *Lunjia Hu*: supported by NSF Award IIS-1908774 and a VMware fellowship.
Weiyun Ma: supported by a Stanford Graduate Fellowship.
Li-Yang Tan: supported by NSF grant CCF-1921795 and CAREER Award CCF-1942123.

Acknowledgements We thank Shalev Ben-David for correspondence about their ongoing work [5, 6]. AD thanks Mark Braverman for interesting discussions of related topics.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 9; pp. 9:1–9:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Randomized query complexity (see [8] for a classic survey) is often studied using Yao's minimax principle [20]. The principle states that for every boolean function $f\colon \{0,1\}^n \to \{0,1\}$

$$\textbf{Yao's minimax:} \qquad \mathsf{R}_\epsilon(f) \;=\; \max\nolimits_{\mathcal{D}} \mathsf{D}_\epsilon(f,\mathcal{D}).$$

- Here $\mathsf{R}_\epsilon(f)$ is the randomized ϵ-error query complexity of f. More precisely, $\mathsf{R}_\epsilon(f)$ equals the least number of queries a randomized algorithm (decision tree) must make to the input bits $x_i \in \{0,1\}$ of an unknown input $x \in \{0,1\}^n$ in order to output $f(x)$ with probability at least $1-\epsilon$ (where the probability is over the coin tosses of the algorithm). We often set $\epsilon = 1/3$ and omit ϵ from notation, as it is well known that this choice only affects constant factors in query complexity.
- $\mathcal{D}$ is a distribution over the inputs $\{0,1\}^n$. We may assume wlog that $\mathcal{D}$ is *balanced*: $\mathcal{D} = \frac{1}{2}\mathcal{D}_0 + \frac{1}{2}\mathcal{D}_1$ where $\mathcal{D}_b$ is a distribution over $f^{-1}(b)$.
- $\mathsf{D}_\epsilon(f,\mathcal{D})$ is the *distributional* ϵ-error query complexity of f relative to $\mathcal{D}$. More precisely, $\mathsf{D}_\epsilon(f,\mathcal{D})$ equals the least number of queries a *deterministic* algorithm must make to an input $x \sim \mathcal{D}$ in order to output $f(x)$ with probability at least $1-\epsilon$ (where the probability is over $x \sim \mathcal{D}$).

1.1 Correlated samples problem

One way to think about the distributional complexity of f relative to $\mathcal{D} = \frac{1}{2}\mathcal{D}_0 + \frac{1}{2}\mathcal{D}_1$ is as the following task: A deterministic algorithm is given query access to a sample from either $\mathcal{D}_0$ or $\mathcal{D}_1$ and it needs to decide which is the case. In this work, we ask: Does this task become easier if we allow query access to an unlimited number of independent samples from either $\mathcal{D}_0$ or $\mathcal{D}_1$? In short,

Is it easier to distinguish $\mathcal{D}_0^\infty$ from $\mathcal{D}_1^\infty$ than it is to distinguish $\mathcal{D}_0$ from $\mathcal{D}_1$?

More formally, we define the *correlated samples problem* for f relative to $\mathcal{D} = \frac{1}{2}\mathcal{D}_0 + \frac{1}{2}\mathcal{D}_1$ by

$$\mathsf{Corr}_\epsilon(f,\mathcal{D}) \;\coloneqq\; \min_{k\geq 1} \mathsf{D}_\epsilon(f^k, \tfrac{1}{2}\mathcal{D}_0^k + \tfrac{1}{2}\mathcal{D}_1^k).$$

Here $f^k\colon (\{0,1\}^n)^k \to \{0,1\}^k$ is the function that evaluates k copies of f on disjoint inputs. We also use the notation $\mathcal{D}^k \coloneqq \mathcal{D} \times \cdots \times \mathcal{D}$ (k times) for the k-fold product distribution. In particular, under $\frac{1}{2}\mathcal{D}_0^k + \frac{1}{2}\mathcal{D}_1^k$, the function f^k outputs either 0^k or 1^k; the correlated samples problem is to decide which is the case. We note that the expression to be minimized on the right side is a non-increasing function of k (access to more samples is only going to help). We may also assume wlog that $k \leq n$ (when an algorithm queries a sample for the first time, we may assume it is the first unqueried sample so far).

Shaltiel examples. It is not hard to give examples of input distributions where access to multiple correlated samples *does* help. Such examples were already discussed by Shaltiel [18] in the context of direct product theorems. For instance, consider the n-bit $\textsc{Xor}_n$ function. It is well known that $\mathsf{R}_\epsilon(\textsc{Xor}_n) = n$ for all $\epsilon > 0$. Define a balanced input distribution (here $\mathcal{U}$ is a uniform random bit in $\{0,1\}$)

$$\mathcal{D} \;\coloneqq\; \begin{cases} 0\,\mathcal{U}^{n-1} & \text{with probability } 99\%, \\ 1\,\mathcal{U}\,0^{n-2} & \text{with probability } 1\%. \end{cases}$$

This distribution is hard 99% of the time: if the first bit is 0, an algorithm has to compute $\textsc{Xor}_{n-1}$ relative to $\mathcal{U}^{n-1}$, which requires $n-1$ queries. For the remaining 1%, the distribution is easy: if the first bit is 1, the output can be deduced from the second bit. Here multiple correlated samples help a lot (for $\epsilon = 1/3$):

$$\begin{aligned} \mathsf{D}(\textsc{Xor}_n, \mathcal{D}) &= \Omega(n), \\ \mathsf{Corr}(\textsc{Xor}_n, \mathcal{D}) &= O(1). \end{aligned}$$

Indeed, given a single sample from $\mathcal{D}$, an algorithm is likely to have to solve the hard case of the distribution. By contrast, given multiple correlated samples, we can query the first bit for a large constant number of samples. This will give us a high chance to encounter at least one easy sample.

Error reduction. An important fact (which fails in the single-sample setting!) is that we can amplify the success probability of any algorithm for correlated samples. This is achieved by a variant of the usual trick: repeatedly run the algorithm on fresh samples to gain more confidence about the output.[1]

▶ **Fact 1.1.** $\mathsf{Corr}_\epsilon(f, \mathcal{D}) \leq O(\log(1/\epsilon)/\delta^2) \cdot \mathsf{Corr}_{1/2-\delta}(f, \mathcal{D})$ *for every* $(f, \mathcal{D})$.

The aforementioned Shaltiel example $(\textsc{Xor}_n, \mathcal{D})$ can alternatively be computed as follows: By querying the first two bits of a single sample $x \sim \mathcal{D}$ one can predict $\textsc{Xor}_n(x)$ to within error 49.5%. Now apply Fact 1.1 to reduce the error below 1/3 at the cost of a constant-factor blowup in query cost.

1.2 Main result

We study whether Shaltiel examples can be avoided if we restrict our attention to the hardest possible input distribution. Namely, we define a distribution-free complexity measure by

$$\mathsf{Corr}_\epsilon(f) \;:=\; \max_{\mathcal{D}} \mathsf{Corr}_\epsilon(f, \mathcal{D}).$$

Our main result is that multiple correlated samples do not help for the hardest distribution.

▶ **Theorem 1.2.** $\mathsf{Corr}(f) = \Theta(\mathsf{R}(f))$ *for any (partial) boolean function* f.

The main challenge in proving Theorem 1.2 is precisely the existence of Shaltiel examples: How to construct hard distributions that do not contain any hidden easy parts? We resolve it by building decision trees that can exploit the easy parts not only in its own input distribution, but in various other distributions as well.

1.3 Application 1: Selection problem

Next we describe a consequence of our main result to a natural query task that we dub the *selection problem*. A similar problem, called *choose*, was studied by [4] in communication complexity.

[1] In more detail: An algorithm T with error $1/2 - \delta$ has $|p_0 - p_1| \geq 2\delta$ where $p_i := \Pr[T(x_i) = 1]$ for $x_i \sim \mathcal{D}_i^k$. Reducing error below $\epsilon > 0$ boils down to distinguishing two random coins with heads-probabilities p_0 and p_1. Given multiple samples from one of the coins, Chernoff bounds state that $O(\log(1/\epsilon)/\delta^2)$ samples are enough to tell which coin the samples came from.

Fix an n-bit function f together with an input distribution $\mathcal{D}$. In the *k-selection problem* for $(f, \mathcal{D})$ the input is a random kn-bit string $x = (x^1, \ldots, x^k) \sim \mathcal{D}^k$, and the goal is to output $(i, f(x^i))$ for some $i \in [k]$. That is, the algorithm gets access to k independent samples from $\mathcal{D}$ and it selects one of them to solve. We define

$$\begin{aligned} k\text{-}\mathsf{Sel}_\epsilon(f, \mathcal{D}) &:= \epsilon\text{-error query complexity of } k\text{-selection for } (f, \mathcal{D}), \\ \mathsf{Sel}_\epsilon(f, \mathcal{D}) &:= \min\nolimits_{k \geq 1} k\text{-}\mathsf{Sel}_\epsilon(f, \mathcal{D}), \\ \mathsf{Sel}_\epsilon(f) &:= \max\nolimits_{\mathcal{D}} \mathsf{Sel}_\epsilon(f, \mathcal{D}). \end{aligned}$$

The selection problem is interesting because it, too, is subject to Shaltiel examples: for $(\textsc{Xor}_n, \mathcal{D})$ as described in Subsection 1.1, we have $\mathsf{Sel}(\textsc{Xor}_n, \mathcal{D}) = O(1)$ using the same idea of searching for an easy sample.

The following relates selection to correlated samples; see Section 5 for the proof.

▶ **Theorem 1.3.** *The correlated samples problem is easier than selection:*

1. $\mathsf{Corr}(f, \mathcal{D}) \leq O(\mathsf{Sel}(f, \mathcal{D}))$ *for every* $(f, \mathcal{D})$.
2. *There exists an n-bit $(f, \mathcal{D})$ such that* $\mathsf{Sel}(f, \mathcal{D}) = \Omega(n)$ *but* $\mathsf{Corr}(f, \mathcal{D}) = O(1)$.
3. *Selection does not admit efficient error reduction (as in Fact 1.1).*

Combining the first item of Theorem 1.3 with our main result (Theorem 1.2) we conclude that multiple samples do not help in the selection problem for the hardest distribution.

▶ **Corollary 1.4.** $\mathsf{Sel}(f) = \Theta(\mathsf{R}(f))$ *for any (partial) boolean function f.*

1.4 Application 2: Randomized composition

We give another application of our main result to the *randomized composition conjecture* studied in [7, 3, 9, 6]. In fact, this application is what originally motivated our research project!

For an n-bit function f and an m-bit function g we define their composition

$$f \circ g \colon (\{0,1\}^m)^n \to \{0,1\} \qquad \text{such that} \qquad (f \circ g)(x^1, \ldots, x^n) := f(g(x^1), \ldots, g(x^n)).$$

A *composition theorem* aims to understand the query complexity of $f \circ g$ in terms of f and g. Such theorems are known for deterministic query complexity, $\mathsf{D}(f \circ g) = \mathsf{D}(f)\mathsf{D}(g)$ [17, 19, 14], and quantum query complexity, $\mathsf{Q}(f \circ g) = \Theta(\mathsf{Q}(f)\mathsf{Q}(g))$ [12, 16]. The conjecture in the randomized case is:

▶ **Conjecture 1.5.** $\mathsf{R}(f \circ g) \geq \Omega(\mathsf{R}(f)\mathsf{R}(g))$ *for all boolean functions f and g.*

Gavinsky et al. [9] have shown that the conjecture fails if f is allowed to be a *relation*. They also show $\mathsf{R}(f \circ g) \geq \Omega(\mathsf{R}(f)\mathsf{R}(g)^{1/2})$ for any relation f and partial function g. In a very recent work (concurrent to ours) Ben-David and Blais [5, 6] have found a counterexample to the randomized conjecture for partial f and g, albeit with a tiny query complexity compared to input length; see also Subsection 1.5. The conjecture is still open for total functions.

Fractional block sensitivity. We show a new composition theorem in terms of *fractional block sensitivity* $\mathsf{fbs}(f)$, introduced by [19, 10]; see also [13, 2]. This measure is at most randomized query complexity, $\mathsf{fbs}(f) \leq O(\mathsf{R}(f))$, and it is equivalent to *randomized certificate complexity* [1].

Let us define $\mathsf{fbs}(f)$ for an n-bit f. We say that a block $B \subseteq [n]$ is *sensitive* on input x iff $f(x) \neq f(x^B)$ where x^B is x but with bits in B flipped. Fix an input x and introduce a real weight $w_B \in [0,1]$ for each sensitive block B of x. Define $\mathsf{fbs}(f,x)$ as the optimum value of the following linear program

$$\begin{array}{rll} \max & \sum_B w_B & \\ \textit{subject to} & \sum_{B \ni i} w_B \leq 1, & \forall i \in [n], \\ & w_B \geq 0, & \forall B. \end{array}$$

Finally, define $\mathsf{fbs}(f) := \max_x \mathsf{fbs}(f,x)$. For comparison, the more usual *block sensitivity* $\mathsf{bs}(f)$ [15] is defined the same way except with the integral constraint $w_B \in \{0,1\}$. In particular $\mathsf{bs}(f) \leq \mathsf{fbs}(f)$, and moreover a polynomial gap (power 1.5) between the two is known for a total function [10].

We make progress towards the composition conjecture; see Section 6 for the proof.

▶ **Theorem 1.6.** $\mathsf{R}(f \circ g) \geq \Omega(\mathsf{fbs}(f)\mathsf{R}(g))$ *for any (partial) boolean functions f and g.*

The previous best comparable composition theorem was $\mathsf{R}(f \circ g) \geq \Omega(\mathsf{bs}(f)\mathsf{R}(g))$, a proof of which is virtually the same as for the result that $\mathsf{R}(\textsc{And}_n \circ g) \geq \Omega(n\mathsf{R}(g))$; see [11, §5.1]. In fact, we were originally motivated to consider the correlated samples problem when trying to strengthen this composition result from block sensitivity to fractional block sensitivity.

1.5 Independent work by Ben-David and Blais

In an independent and concurrent work, Ben-David and Blais [5, 6] have also studied the randomized composition conjecture and ways of circumventing Shaltiel examples via improved minimax theorems. They develop a powerful framework for constructing hard *Shaltiel-free* distributions, which is general enough to apply not only to query complexity but also, for instance, to communication complexity. In particular, their framework is able to give an alternative proof of our main result (Theorem 1.2) as well as our fbs-based composition theorem (Theorem 1.6). Their proof techniques involve information theory and analysis; by contrast, our techniques are more elementary and directly tailored to the correlated samples problem (which does not explicitly appear in their work).

1.6 Roadmap

We will prove our main theorem (Theorem 1.2) in Section 3 and Section 4. Before that, we introduce our basic notions regarding decision trees in Section 2. In Section 3, we characterize decision trees as *likelihood boosters*, emphasizing that a good query algorithm must make significant progress in terms of boosting the likelihood of one of the outputs (0 or 1) to much higher than the other, and vice versa. This characterization frees us from considering inputs from both $\mathcal{D}_0$ and $\mathcal{D}_1$ simultaneously: if an algorithm is certain about the output on $\mathcal{D}_1$, then it must also make few errors on $\mathcal{D}_0$. We thus reduce the proof of Theorem 1.2 to bootstrapping decision trees that can make *overall* progress across multiple samples to a decision tree that makes uniform progress. In Section 4, we build such a bootstrapping algorithm and show that it makes satisfactory progress with a careful analysis. Proofs for our two applications are in Section 5 and Section 6.

2 Preliminaries

Let $f : \Sigma^n \to \{0, 1, *\}$ be a partial function for some alphabet Σ (typically $\Sigma = \{0, 1\}$). Let $\mathcal{D}_0, \mathcal{D}_1$ be distributions supported on $f^{-1}(0)$, $f^{-1}(1)$ respectively. For each $x \in \Sigma^n$, let $\mathcal{D}_0(x)$ (resp. $\mathcal{D}_1(x)$) denote the probability mass on x in distribution $\mathcal{D}_0$ (resp. $\mathcal{D}_1$). For a subset $S \subseteq \Sigma^n$, we define $\mathcal{D}_b(S) = \sum_{x\in S} \mathcal{D}_b(x)$ for $b = 0, 1$. If $\mathcal{D}_b(S) > 0$, we define the conditional distribution $\mathcal{D}_b|_S$ by $\mathcal{D}_b|_S(x) = \frac{\mathcal{D}_b(x)}{\mathcal{D}_b(S)}$ when $x \in S$, and $\mathcal{D}_b|_S(x) = 0$ when $x \notin S$. We define the *likelihood-ratio* of S as

$$\mathsf{LR}(S) := \frac{\mathcal{D}_1(S)}{\mathcal{D}_0(S)}.$$

Let T be a deterministic decision tree that takes as input a sample $x \in \Sigma^n$ drawn from either $\mathcal{D}_0$ or $\mathcal{D}_1$. For every vertex v in T, we use $\mathsf{Input}(v) \subseteq \Sigma^n$ to denote the set of strings that can reach v, or equivalently, the set of strings that agree with all the queries made so far. Typically, every non-leaf vertex in T corresponds to a query to a certain position in the sample, but we will allow non-leaf vertices v in T that do not make any query, each of them having only a single child v' with $\mathsf{Input}(v') = \mathsf{Input}(v)$. We abuse notation slightly and use v as a shorthand for $\mathsf{Input}(v)$, so we have $\mathcal{D}_0(v) = \sum_{x\in v} \mathcal{D}_0(x)$, $\mathcal{D}_1(v) = \sum_{x\in v} \mathcal{D}_1(x)$ and

$$\mathsf{LR}(v) = \frac{\mathcal{D}_1(v)}{\mathcal{D}_0(v)}.$$

Note that the likelihood-ratio $\mathsf{LR}(v)$ is non-negative, but could be zero or infinite. We can eliminate the undefined case ($\mathcal{D}_0(v) = \mathcal{D}_1(v) = 0$) by trimming the unreachable parts of the decision tree.

Now if the decision tree T takes as input k samples from Σ^n, it is not hard to see that $\mathsf{Input}(v)$ can be written as a Cartesian product $\mathsf{Input}(v) = \mathsf{Input}_1(v) \times \cdots \times \mathsf{Input}_k(v)$, where $\mathsf{Input}_j(v) \subseteq \Sigma^n$ is the set of strings that agree with all the queries made to the j-th sample so far. Again, we abuse notation slightly and use v_j as a shorthand for $\mathsf{Input}_j(v)$, so we will often write $v = v_1 \times \cdots \times v_k$. We define the *overall likelihood ratio* of v as the product

$$\mathsf{OLR}(v) := \mathsf{LR}(v_1) \cdots \mathsf{LR}(v_k) = \frac{\mathcal{D}_1(v_1)}{\mathcal{D}_0(v_1)} \cdots \frac{\mathcal{D}_1(v_k)}{\mathcal{D}_0(v_k)}.$$

It is often more convenient to consider the logarithm of likelihood ratios. We will use natural logarithm throughout the paper, i.e. $\log(\cdot) = \ln(\cdot)$.

3 Query Algorithms as Likelihood Boosters

Our overarching goal (Theorem 1.2) is to construct an efficient deterministic query algorithm that distinguishes $\mathcal{D}_0$ from $\mathcal{D}_1$, assuming the existence of one that distinguishes $\mathcal{D}_0^k$ from $\mathcal{D}_1^k$. As the starting point, we introduce the notion of *likelihood boosters* as a way of measuring the progress made by a query algorithm T in distinguishing $\mathcal{D}_0$ from $\mathcal{D}_1$. The key idea is that, as more queries are being made, the algorithm narrows down the possibilities of the unknown input, driving the likelihood of one of the output (0 or 1) much higher than the other. In fact, we show that T can distinguish $\mathcal{D}_0$ from $\mathcal{D}_1$ well if and only if a sample drawn from $\mathcal{D}_1$ has a high probability of arriving at a leaf of T where most of the remaining possibilities produce output 1. (Lemma 3.4 and Lemma 3.5).

In the multiple-sample setting, we use the notions of *overall likelihood boosters* and *uniform likelihood boosters*, which have different levels of guarantees, to measure the progress of a query algorithm on simultaneously classifying each of the samples in the input. We show that an efficient query algorithm that distinguishes $\mathcal{D}_0^k$ from $\mathcal{D}_1^k$ is an efficient overall likelihood booster (Corollary 3.6). Moreover, we show that an efficient uniform likelihood booster on multiple samples induces an efficient likelihood booster on a single sample (Lemma 3.7), which in turn implies an efficient query algorithm that distinguishes $\mathcal{D}_0$ from $\mathcal{D}_1$ (Lemma 3.4). These results will enable us to reduce proving Theorem 1.2 to relating overall likelihood boosters to uniform likelihood boosters, which is the focus of Section 4 (see Theorem 4.1).

We now formally define the three types of likelihood boosters mentioned above:

▶ **Definition 3.1.** *We say a deterministic decision tree T is a (δ, M)-*likelihood booster *for $\mathcal{D}_0, \mathcal{D}_1$ if, with probability at least $1 - \delta$, an input sample drawn from $\mathcal{D}_1$ reaches a leaf ℓ of T with likelihood ratio $\mathsf{LR}(\ell) \geq M$.*

▶ **Definition 3.2.** *We say a deterministic decision tree T is a (δ, M)-*overall likelihood booster *for $\mathcal{D}_0^k, \mathcal{D}_1^k$ if, with probability at least $1 - \delta$, an input drawn from $\mathcal{D}_1^k$ consisting of k samples reaches a leaf ℓ of T with* overall *likelihood ratio $\mathsf{OLR}(\ell) \geq M$.*

▶ **Definition 3.3.** *We say a deterministic decision tree T is a (δ, ε, M)-*uniform likelihood booster *for $\mathcal{D}_0^k$ and $\mathcal{D}_1^k$ if, with probability at least $1 - \delta$, an input x drawn from $\mathcal{D}_1^k$ consisting of k samples reaches a leaf $\ell = \ell_1 \times \cdots \times \ell_k$ of T with the property that at least $(1 - \varepsilon)k$ different samples $j \in \{1, \cdots, k\}$ satisfy $\mathsf{LR}(\ell_j) \geq M$.*

Note that the above definitions do not depend on the actual output of the decision tree T. We now show in the following two lemmas that likelihood boosters are in some sense equivalent to query algorithms that distinguish $\mathcal{D}_0$ from $\mathcal{D}_1$.

▶ **Lemma 3.4.** *Suppose T is a (δ, M)-likelihood booster for $\mathcal{D}_0, \mathcal{D}_1$. Consider the deterministic decision tree T' that makes exactly the same queries as T and accepts if and only if a leaf ℓ with $\mathsf{LR}(\ell) \geq M$ is reached. Then T' distinguishes $\mathcal{D}_0$ from $\mathcal{D}_1$ with the following guarantees:*

1. *(Completeness) T' accepts $x \sim \mathcal{D}_1$ with probability at* least *$1 - \delta$.*
2. *(Soundness) T' accepts $x \sim \mathcal{D}_0$ with probability at* most *$1/M$.*

Proof. Completeness follows directly from the definition of likelihood booster. To prove soundness, consider the set U of leaves ℓ with $\mathsf{LR}(\ell) \geq M$. For all $\ell \in U$, we have $\mathcal{D}_0(\ell) \leq \frac{1}{M}\mathcal{D}_1(\ell)$. Therefore, $\sum_{\ell \in U} \mathcal{D}_0(\ell) \leq \frac{1}{M} \sum_{\ell \in U} \mathcal{D}_1(\ell) \leq \frac{1}{M}$. This means that a sample from $\mathcal{D}_0$ reaches leaves in U with probability at most $\frac{1}{M}$, which is exactly the desired soundness. ◀

▶ **Lemma 3.5.** *Suppose a deterministic decision tree T can distinguish $\mathcal{D}_0$ from $\mathcal{D}_1$ with the following guarantees: T accepts $x \sim \mathcal{D}_0$ with probability at most δ_0, and accepts $x \sim \mathcal{D}_1$ with probability at least $1 - \delta_1$. Then T is a $(M\delta_0 + \delta_1, M)$-likelihood booster for any $M > 0$.*

Proof. Let U denote the set of leaves ℓ with $\mathsf{LR}(\ell) < M$. We can partition U as $U = U_0 \cup U_1$, where U_1 corresponds to the leaves at which T accepts. Since T accepts with probability at most δ_0 on $\mathcal{D}_0$, we have $\sum_{\ell \in U_1} \mathcal{D}_0(\ell) \leq \delta_0$. Similarly, we have $\sum_{\ell \in U_0} \mathcal{D}_1(\ell) \leq \delta_1$. Therefore,

$$\sum_{\ell \in U} \mathcal{D}_1(\ell) = \sum_{\ell \in U_0} \mathcal{D}_1(\ell) + \sum_{\ell \in U_1} \mathcal{D}_1(\ell) \leq \sum_{\ell \in U_0} \mathcal{D}_1(\ell) + M \sum_{\ell \in U_1} \mathcal{D}_0(\ell) = \delta_1 + M\delta_0.$$

In other words, a sample from $\mathcal{D}_1$ has probability at most $M\delta_0 + \delta_1$ of reaching a leaf in U, which means that T is a $(M\delta_0 + \delta_1, M)$-likelihood booster. ◀

In the multiple-sample setting, if we view the pair $\mathcal{D}_0^k$ and $\mathcal{D}_1^k$ as $\mathcal{D}_0'$ and $\mathcal{D}_1'$ in the single-sample setting with input length multiplied by k, the definition of overall likelihood ratio coincides with the definition of likelihood ratio in the single-sample setting. Therefore, we have the following corollary of Lemma 3.5, which essentially shows that an efficient query algorithm for the correlated samples problem is an efficient overall likelihood booster:

▶ **Corollary 3.6.** *Suppose a deterministic decision tree T can distinguish $\mathcal{D}_0^k$ from $\mathcal{D}_1^k$ in that T accepts $x \sim \mathcal{D}_0^k$ with probability at most δ_0, and T accepts $x \sim \mathcal{D}_1^k$ with probability at least $1 - \delta_1$. Then T is a $(M\delta_0 + \delta_1, M)$-overall likelihood booster for any $M > 0$.*

To conclude this section, we show that an efficient uniform likelihood booster in the multiple-sample setting implies an efficient likelihood booster in the single-sample setting.

▶ **Lemma 3.7.** *For any* (δ, ε, M)-uniform likelihood booster *T for $\mathcal{D}_0^k$ and $\mathcal{D}_1^k$ and any $C > 0$, there is a $(\delta + \varepsilon + \frac{1}{C}, M)$-likelihood booster T' for $\mathcal{D}_0$ and $\mathcal{D}_1$ with* $\mathrm{depth}(T') \leq C \cdot \frac{\mathrm{depth}(T)}{k}$.

Proof. Define $Q = C \cdot \frac{\mathrm{depth}(T)}{k}$. We first build a randomized query algorithm $\mathcal{A}'$ for $\mathcal{D}_0$ and $\mathcal{D}_1$, and later derandomize it as T'. On input $x_{\mathcal{A}'}$, $\mathcal{A}'$ generates k random samples $(x_1, \dots, x_k) \sim \mathcal{D}_1^k$, selects a uniformly random index j, replaces x_j with $\mathcal{A}'$'s own input $x_{\mathcal{A}'}$, and finally simulates T on the modified k samples $(x_1, \dots, x_{\mathcal{A}'}, \dots, x_k)$. If T attempts to make the $(\lfloor Q \rfloor + 1)$-th query to the j-th (modified) sample, $\mathcal{A}'$ halts.

It is easy to see that the maximum number of queries made by $\mathcal{A}'$ is at most Q. Moreover, by Markov's inequality, if the input $x_{\mathcal{A}'}$ to $\mathcal{A}'$ is drawn from $\mathcal{D}_1$, the probability that $\mathcal{A}'$ halts early because of T making more than Q queries to the j-th sample is at most $\frac{1}{C}$, since the average number of queries T makes to the j-th sample for a uniformly random j is at most $\frac{\mathrm{depth}(T)}{k}$.

We now show that with probability at least $1-(\delta+\varepsilon+\frac{1}{C})$, $\mathcal{A}'$ reaches a leaf $\ell = \ell_1 \times \cdots \times \ell_k$ of T with $\mathsf{LR}(\ell_j) \geq M$ when its own input $x_{\mathcal{A}'}$ is drawn from $\mathcal{D}_1$. By a union bound, we only need to show that this holds with probability at least $1 - (\delta + \varepsilon)$ for the extended version of $\mathcal{A}'$ that doesn't halt early. If we switch the order of randomness so that j is chosen after a leaf of T is reached, this follows easily from the definition of uniform likelihood boosters (Definition 3.3).

Finally, we derandomize $\mathcal{A}'$. Note that the randomness in $\mathcal{A}'$ only comes from the randomness in j and in all the generated samples x_i except the j-th sample. We can simply fix them so that the probability of reaching a leaf ℓ of T with $\mathsf{LR}(\ell_j) \geq M$ is maximized, assuming that the j-th sample is from $\mathcal{D}_1$. Since j and all generated samples other than the j-th sample have been fixed, the decision tree T now "shrinks" to a decision tree T' with only the first $\lfloor Q \rfloor$ queries to the j-th sample remaining, and every leaf ℓ of T that is reachable when we run $\mathcal{A}'$ now becomes a leaf ℓ' of T'. Shrinking the tree doesn't affect the computation history regarding the j-th sample, so we have $\ell' = \mathsf{Input}(\ell') = \mathsf{Input}_j(\ell) = \ell_j$ and $\mathsf{LR}(\ell') = \mathsf{LR}(\ell_j)$. This proves that T' is a $(\delta + \varepsilon + \frac{1}{C}, M)$-likelihood booster. ◀

4 Bootstrapping Overall Booster to Uniform Booster

The results from the previous section (Section 3) reduce proving our main result (Theorem 1.2) to proving relations between overall likelihood boosters and uniform likelihood boosters. In this section, we complete this step with the following result:

▶ **Theorem 4.1.** *Assume that there is a depth-L* $(0.1, 25)$*-overall likelihood booster for* every *distribution pair $\mathcal{D}_0^k, \mathcal{D}_1^k$. Then there is a depth-$O(KL)$* $(0.1, 0.1, 100)$*-uniform likelihood booster for every distribution pair $\mathcal{D}_0^K, \mathcal{D}_1^K$ whenever $K \geq 1000k(|\Sigma| + 1)^n$.*

We first show how to derive Theorem 1.2 from Theorem 4.1:

Proof of Theorem 1.2. We prove the inequality $\mathsf{R}(f) \leq O(\mathsf{Corr}(f))$ (the converse inequality is trivial). Suppose we have a depth-L deterministic decision tree that solves the correlated samples problem on $\frac{1}{2}\mathcal{D}_0^k + \frac{1}{2}\mathcal{D}_1^k$ with success probability at least 0.999 (recall that the success probability can be amplified by Fact 1.1). That is, the decision tree accepts inputs drawn from $\mathcal{D}_1^k$ with probability at *least* 0.998 and accepts inputs drawn from $\mathcal{D}_0^k$ with probability at *most* 0.002. By Corollary 3.6, it is a $(0.1, 25)$-overall likelihood booster for $\mathcal{D}_0^k$ and $\mathcal{D}_1^k$.

By Theorem 4.1, for any pair of distributions $\mathcal{D}_0^K, \mathcal{D}_1^K$, there is a $(0.1, 0.1, 100)$-uniform likelihood booster with depth $O(KL)$. Then by Lemma 3.7, there is a $(1/3, 100)$-likelihood booster with depth $O(L)$ for $\mathcal{D}_0$ and $\mathcal{D}_1$, which by Lemma 3.4 implies a query algorithm for $\mathcal{D}_0$ and $\mathcal{D}_1$ with success probability at least 1/3. By the arbitrariness of $\mathcal{D}_0$ and $\mathcal{D}_1$, we have $\mathsf{R}_{1/3}(f) = O(L)$ via Yao's minimax, as desired. ◀

The rest of this section is dedicated to proving Theorem 4.1. We construct the desired uniform likelihood booster $T_{\text{bootstrap}}$, described in Subsection 4.1, by applying different overall likelihood boosters to appropriate sets of samples at different phases of computation. To quantify the progress made by $T_{\text{bootstrap}}$, we design a measure based on a "truncated" log likelihood ratio which handles samples that $T_{\text{bootstrap}}$ is confident about with special care. As the technical core of the proof, we show that under our carefully constructed measure, $T_{\text{bootstrap}}$ in expectation makes *positive* and *constant* progress during each phase of computation (Lemmas 4.2 and 4.3). Therefore, $T_{\text{bootstrap}}$ is able to achieve the desired guarantees after sufficiently many phases.

4.1 Bootstrapping algorithm

We describe our depth-$O(KL)$ $(0.1, 0.1, 100)$-uniform likelihood booster $T_{\text{bootstrap}}$ taking $K \geq 1000k(|\Sigma| + 1)^n$ samples. Recall that each vertex v of $T_{\text{bootstrap}}$ can be written as a Cartesian product $v = v_1 \times \cdots \times v_K$, where $v_j \subseteq \Sigma^n$ is the set of strings that are consistent with the queries made to the j-th sample so far. We say that the j-th sample is *settled* at v if

$$\mathsf{LR}(v_j) = \frac{\mathcal{D}_1(v_j)}{\mathcal{D}_0(v_j)} \notin [e^{-100}, e^{100}].$$

Note that it is possible for a sample to be settled in the wrong direction (e.g. $\mathsf{LR}(v_j) < e^{-100}$ on input drawn from $\mathcal{D}_1^K$), but we will show that this is not a serious issue.

The query algorithm $T_{\text{bootstrap}}$ proceeds in at most $C \cdot K$ phases (for some large constant $C > 0$). Each phase consists of at most L queries and is described as follows:

Phase $s = 1, \cdots, C \cdot K$:

1. If fewer than $k(|\Sigma| + 1)^n$ out of the K samples are unsettled, halt.
2. Else, since each v_j is determined by a string v_* in $(\Sigma \cup \{*\})^n$ recording the queries made so far to the j-th sample, by the Pigeonhole Principle there exist k unsettled samples $j_1, \cdots, j_k$ with $v_{j_1} = \cdots = v_{j_k} = v_*$.
3. Run the depth-L $(0.1, 25)$-overall likelihood booster $A^{(v_*)}$, assumed in Theorem 4.1 to exist, relative to the input-distribution pair

$$(\mathcal{D}_0|_{v_*})^k \ , \quad (\mathcal{D}_1|_{v_*})^k$$

on the samples

$$(x^{j_1}, \ldots, x^{j_k}) \ .$$

If any query causes one of these samples to become settled (i.e. $\mathsf{LR}(v_{j_i}) \notin [e^{-100}, e^{100}]$ for some $i \in \{1, \cdots, k\}$), halt $A^{(v_*)}$ and go to the next Phase. Otherwise we proceed to the next Phase after $A^{(v_*)}$ terminates. If fewer than L queries are made in the current phase, insert dummy vertices that do not make any query (see Section 2) to $T_{\text{bootstrap}}$ so that each phase corresponds to a path in $T_{\text{bootstrap}}$ with length exactly L.

4.2 Sub-martingale property of progress measure

It's not hard to see that the overall likelihood ratio (OLR) is *not* an effective measure of progress for $T_{\text{bootstrap}}$: OLR can rocket to infinity even when there is only one settled sample. In this subsection, we introduce a better progress measure: *overall truncated log likelihood ratio* (OTLLR), and show that it is a sub-martingale along the computation path of *any* decision tree (Lemma 4.2). In other words, $T_{\text{bootstrap}}$ always makes *non-negative* progress in expectation. We will show that each phase of $T_{\text{bootstrap}}$ makes *positive* expected progress in the next subsection (Subsection 4.3).

Let T be a deterministic decision tree that takes as input K samples. For every vertex $v = v_1 \times \cdots \times v_K$ of T, we define the *truncated log likelihood ratio* of v_j as

$$\mathsf{TLLR}(v_j) := \begin{cases} \log(\mathsf{LR}(v_j)), & \text{if } |\log(\mathsf{LR}(v_j))| \le 100, \\ 500, & \text{otherwise.} \end{cases}$$

Note that if $\log(\mathsf{LR}(v_j))$ slightly exceeds the upper threshold 100, we set TLLR to a much higher value 500. Also, when $\log(\mathsf{LR}(v_j))$ drops below the lower threshold -100, we also set TLLR to 500. Thus, the j-th sample is *settled* at v if and only if $\mathsf{TLLR}(v_j) = 500$.

We define the *overall truncated log-likelihood-ratio* of v as the sum

$$\mathsf{OTLLR}(v) := \sum_{j=1}^{K} \mathsf{TLLR}(v_j).$$

The input x to T determines a computation path from the root of T to a leaf: $v^0 \to v^1 \to \cdots \to v^q$. The randomness in x transfers to the randomness in the path, so the path is a stochastic process. We now show that $\mathsf{OTLLR}(v^t)$ along the path is a sub-martingale when x is drawn from $\mathcal{D}_1^K$:

▶ **Lemma 4.2.** *Assume that T never queries a settled sample. Assume that the input x to T is drawn from $\mathcal{D}_1^K$, v is a non-leaf vertex with distance t from the root, and v is reachable (i.e.* $\Pr[v^t = v] > 0$ *on $\mathcal{D}_1^K$). Define $\Delta^t := \mathsf{OTLLR}(v^{t+1}) - \mathsf{OTLLR}(v^t)$. Then we have*

$$\mathbb{E}[\Delta^t | v^t = v] \ge 0.001 \cdot \mathbb{E}[(\Delta^t)^2 | v^t = v] \ge 0.$$

Proof. Let us condition on $v^t = v$ in the whole proof. If v is a dummy vertex that does not make any query, then $\Delta^t = 0$ deterministically and the lemma holds trivially. We assume that v is not a dummy vertex henceforth.

Suppose sample j is queried at vertex v. We have $\mathsf{OTLLR}(v^{t+1}) - \mathsf{OTLLR}(v^t) = \mathsf{TLLR}(v_j^{t+1}) - \mathsf{TLLR}(v_j^t)$. Since T never queries a settled sample, we know $\mathsf{TLLR}(v_j^t) = \log \frac{\mathcal{D}_1(v_j^t)}{\mathcal{D}_0(v_j^t)} \in [-100, 100]$.

Let $\sigma \in \Sigma$ denote the random outcome of the query, and let $p_0(\sigma), p_1(\sigma)$ denote the probability that the outcome to the query is σ under $\mathcal{D}_0|_{v_j^t}, \mathcal{D}_1|_{v_j^t}$, respectively. Let $H \subseteq \Sigma$ denote the set of $\sigma \in \Sigma$ with $|\mathsf{TLLR}(v_j^t) + \log \frac{p_1(\sigma)}{p_0(\sigma)}| > 100$. Note that $\mathcal{D}_0(v_j^{t+1}) = \mathcal{D}_0(v_j^t) p_0(\sigma)$ and $\mathcal{D}_1(v_j^{t+1}) = \mathcal{D}_1(v_j^t) p_1(\sigma)$, so

$$\mathsf{TLLR}(v_j^{t+1}) = \begin{cases} \mathsf{TLLR}(v_j^t) + \log \frac{p_1(\sigma)}{p_0(\sigma)}, & \sigma \notin H, \\ 500, & \sigma \in H. \end{cases}$$

Thus, H is precisely the set of outcomes $\sigma \in \Sigma$ that make sample j settled at v^{t+1}. Let $W = W(\sigma)$ denote the difference $\mathsf{TLLR}(v_j^{t+1}) - \mathsf{TLLR}(v_j^t)$. Our goal is to prove $\mathbb{E}[W] \ge 0.001 \cdot \mathbb{E}[W^2]$.

Note that $W(\sigma) \in [400, 600]$ when $\sigma \in H$ and $W(\sigma) = \log \frac{p_1(\sigma)}{p_0(\sigma)} \in [-200, 200]$ when $\sigma \notin H$. We have

$$\begin{aligned}\mathbb{E}[W] \geq& 400 \sum_{\sigma\in H} p_1(\sigma) + \sum_{\sigma\notin H} p_1(\sigma) \log \frac{p_1(\sigma)}{p_0(\sigma)}\\ =& 400 \sum_{\sigma\in H} p_1(\sigma) + \sum_{\sigma\notin H} p_0(\sigma) \cdot \frac{p_1(\sigma)}{p_0(\sigma)} \log \frac{p_1(\sigma)}{p_0(\sigma)}. \qquad (1)\end{aligned}$$

By a helper lemma (Lemma 4.4) proved in Subsection 4.4, we know that

$$\frac{p_1(\sigma)}{p_0(\sigma)} \log \frac{p_1(\sigma)}{p_0(\sigma)} \geq \left(\frac{p_1(\sigma)}{p_0(\sigma)} - 1\right) + \frac{1}{400} \cdot \frac{p_1(\sigma)}{p_0(\sigma)} \left(\log \frac{p_1(\sigma)}{p_0(\sigma)}\right)^2.$$

Plugging this into (1), we have

$$\begin{aligned}\mathbb{E}[W] \geq& 400 \sum_{\sigma\in H} p_1(\sigma) + \sum_{\sigma\notin H} p_1(\sigma) - \sum_{\sigma\notin H} p_0(\sigma) + \frac{1}{400} \sum_{\sigma\notin H} p_1(\sigma) \left(\log \frac{p_1(\sigma)}{p_0(\sigma)}\right)^2\\ \geq& 400 \sum_{\sigma\in H} p_1(\sigma) + \left(\sum_{\sigma\notin H} p_1(\sigma) - 1\right) + \frac{1}{400} \sum_{\sigma\notin H} p_1(\sigma) \left(\log \frac{p_1(\sigma)}{p_0(\sigma)}\right)^2\\ =& 400 \sum_{\sigma\in H} p_1(\sigma) - \sum_{\sigma\in H} p_1(\sigma) + \frac{1}{400} \sum_{\sigma\notin H} p_1(\sigma) \left(\log \frac{p_1(\sigma)}{p_0(\sigma)}\right)^2\\ =& 399 \sum_{\sigma\in H} p_1(\sigma) + \frac{1}{400} \sum_{\sigma\notin H} p_1(\sigma) \left(\log \frac{p_1(\sigma)}{p_0(\sigma)}\right)^2\\ =& 399 \sum_{\sigma\in H} p_1(\sigma) + \frac{1}{400} \sum_{\sigma\notin H} p_1(\sigma)(W(\sigma))^2\\ \geq& \frac{1}{1000} \sum_{\sigma\in H} p_1(\sigma)(W(\sigma))^2 + \frac{1}{400} \sum_{\sigma\notin H} p_1(\sigma)(W(\sigma))^2\\ \geq& \frac{1}{1000}\mathbb{E}[W^2].\end{aligned}$$

◀

4.3 Bounding the conditional expectation of progress

In the previous subsection, we showed that OTLLR, as a progress measure, is a sub-martingale. Now we refine our progress measure to also include the natural measure *number of settled samples*, and show that each phase of $T_{\text{bootstrap}}$ makes *positive* progress in expectation.

Recall that we inserted dummy vertices in $T_{\text{bootstrap}}$ to ensure that each phase corresponds to a computation path of length exactly L. Therefore, an entire computation path of $T_{\text{bootstrap}}$ must have length divisible by L: $v^0 \to \cdots \to v^{qL}$. The sub-path $v^{tL} \to \cdots \to v^{(t+1)L}$ is the computation path of the $(t+1)$-th phase.

Define $\mathsf{S}(v)$ as the number of settled samples at vertex v. Our new measure of progress is

$$\mathsf{P}(v^t) := \mathsf{S}(v^t) + \mathsf{OTLLR}(v^t).$$

▶ **Lemma 4.3.** *Assume that the input x to $T_{\text{bootstrap}}$ is drawn from $\mathcal{D}_1^K$, v is a non-leaf vertex with distance tL from the root, and v is reachable (i.e.* $\Pr[v^{tL} = v] > 0$ *on $\mathcal{D}_1^K$). Then we have*

$$\mathbb{E}[\mathsf{P}(v^{(t+1)L}) - \mathsf{P}(v^{tL}) | v^{tL} = v] \geq 0.001.$$

Before proving the lemma, we first show how it implies Theorem 4.1.

Proof of Theorem 4.1. We consider an extended version of $T_{\text{bootstrap}}$ that always halts after exactly $C \cdot K$ phases: whenever it would halt at line 1, it instead enters dummy phases and increases its total progress P by 0.001 per phase (so that now $\mathsf{P} = \mathsf{S} + \mathsf{OTLLR} + 0.001 \cdot$ number of dummy phases). By Lemma 4.3, the extended algorithm finishes with expected total progress $\mathbb{E}[\mathsf{P}] \geq 0.001C \cdot K$ on input drawn from $\mathcal{D}_1^K$. However, P can never grow too large: before any dummy phase, P is at most $501K$, and there are at most $C \cdot K$ dummy phases, so $\mathsf{P} \leq 501K + 0.001C \cdot K$. By Markov's inequality on the non-negative random variable $(501K + 0.001C \cdot K) - \mathsf{P}$, we have $\Pr[\mathsf{P} \leq 501K] \leq \frac{501K}{0.001C \cdot K} = \frac{501}{0.001C}$. If we choose a large enough C, we know that with probability at least 0.99, the total progress exceeds $501K$, which means that the extended algorithm enters dummy phases before halting, and the original algorithm halts at line 1 with all but 0.001 fraction of the samples settled.

It now suffices to show that the fraction of samples settled in the wrong direction (i.e., the likelihood ratio drops below e^{-100}) is at most 0.01 with probability at least 0.99. We first fix j and show that the probability that the j-th sample is settled in the wrong direction is at most e^{-100}, and then use the linearity of expectation and Markov's inequality to bound the overall wrong settlement.

Conditioning on all but the j-th sample, $T_{\text{bootstrap}}$ becomes a deterministic decision tree T' on a single sample. Let U denote the set of leaves ℓ of T' with $\mathsf{LR}(\ell) \leq e^{-100}$. We have $\sum_{\ell \in U} \mathcal{D}_1(\ell) \leq e^{-100} \sum_{\ell \in U} \mathcal{D}_0(\ell) \leq e^{-100}$. This means that the probability that a sample from $\mathcal{D}_1$ reaches leaves in U is at most e^{-100}. Thus the probability of wrong settlement for sample j in $T_{\text{bootstrap}}$ is at most e^{-100}.

By the linearity of expectation, the expected fraction of samples settled in the wrong direction is at most e^{-100}. Then by Markov's inequality, with probability at least 0.99, the fraction of wrong settlement is at most 0.01. ◀

Proof of Lemma 4.3. $\mathsf{S}(v^{(t+1)L}) - \mathsf{S}(v^{tL})$ is either 0 or 1, depending on whether or not a sample becomes settled in phase $t + 1$.

In the case where $\Pr[\mathsf{S}(v^{(t+1)L}) - \mathsf{S}(v^{tL}) = 1 | v^{tL} = v] \geq 0.001$, we have $\mathbb{E}[\mathsf{S}(v^{(t+1)L}) - \mathsf{S}(v^{tL}) | v^{tL} = v] \geq 0.001$, and by Lemma 4.2 we have $\mathbb{E}[\mathsf{OTLLR}(v^{(t+1)L}) - \mathsf{OTLLR}(v^{tL}) | v^{tL} = v] \geq 0$. Summing these two inequalities up proves the lemma.

From now on, we consider the harder case where $\Pr[\mathsf{S}(v^{(t+1)L}) - \mathsf{S}(v^{tL}) = 1 | v^{tL} = v] < 0.001$. We first prove that

$$\Pr[\mathsf{OTLLR}(v^{(t+1)L}) - \mathsf{OTLLR}(v^{tL}) \geq 3 | v^{tL} = v] \geq 0.8. \tag{2}$$

Recall that in this phase $T_{\text{bootstrap}}$ runs the $(0.1, 25)$-overall likelihood booster $A^{(v_*)}$ for $(\mathcal{D}_0|_{v_*})^k$ and $(\mathcal{D}_1|_{v_*})^k$ on the samples $j_1, \ldots, j_k$. If $\mathsf{S}(v^{(t+1)L}) - \mathsf{S}(v^{tL}) = 0$, i.e. no sample becomes settled in this phase, then

$$\mathsf{OTLLR}(v^{(t+1)L}) - \mathsf{OTLLR}(v^{tL}) = \sum_{s=1}^{k} \left(\log \frac{\mathcal{D}_1(v_{j_s}^{(t+1)L})}{\mathcal{D}_0(v_{j_s}^{(t+1)L})} - \log \frac{\mathcal{D}_1(v_{j_s}^{tL})}{\mathcal{D}_0(v_{j_s}^{tL})} \right).$$

Conditioning on $v^{tL} = v$, we have $v_{j_s}^{tL} = v_*$, since $v_{j_1} = \cdots = v_{j_k} = v_*$. From $\frac{\mathcal{D}_b(v_{j_s}^{(t+1)L})}{\mathcal{D}_b(v_*)} = \mathcal{D}_b|_{v_*}(v_{j_s}^{(t+1)L})$, we see that

$$\mathsf{OTLLR}(v^{(t+1)L}) - \mathsf{OTLLR}(v^{tL}) = \log \prod_{s=1}^{k} \frac{\mathcal{D}_1|_{v_*}(v_{j_s}^{(t+1)L})}{\mathcal{D}_0|_{v_*}(v_{j_s}^{(t+1)L})}.$$

Recall that $A^{(v_*)}$ halts early only when a new sample becomes settled, which happens with probability < 0.001. Therefore, in order to prove (2) by a union bound, we only need to prove that the extended version of phase $t+1$ where $A^{(v_*)}$ gets to run without early halting achieves $\prod_{s=1}^{k} \frac{\mathcal{D}_1|_{v_*}(v_{j_s}^{(t+1)L})}{\mathcal{D}_0|_{v_*}(v_{j_s}^{(t+1)L})} \geq e^3$ with probability at least 0.9. This is indeed true because $A^{(v_*)}$ is a $(0.1, 25)$-overall likelihood booster for $(\mathcal{D}_0|_{v_*})^k$ and $(\mathcal{D}_1|_{v_*})^k$.

We now prove $\mathbb{E}[\mathsf{OTLLR}(v^{(t+1)L}) - \mathsf{OTLLR}(v^{tL})|v^{tL} = v] \geq 0.001$. We prove it by contradiction. Suppose $\mathbb{E}[\mathsf{OTLLR}(v^{(t+1)L}) - \mathsf{OTLLR}(v^{tL})|v^{tL} = v] < 0.001$. For $tL \leq s < (t+1)L$, define $\Delta(v^s)$ as the conditional expectation $\mathbb{E}[\mathsf{OTLLR}(v^{s+1}) - \mathsf{OTLLR}(v^s)|v^s]$ and $\Delta_2(v^s)$ as the conditional variance $\mathbb{E}[(\mathsf{OTLLR}(v^{s+1}) - \mathsf{OTLLR}(v^s) - \Delta(v^s))^2|v^s]$. Note that

$$\begin{aligned}\Delta_2(v^s) &= \mathbb{E}[((\mathsf{OTLLR}(v^{s+1}) - \mathsf{OTLLR}(v^s))^2|v^s] - (\Delta(v^s))^2\\ &\leq \mathbb{E}[((\mathsf{OTLLR}(v^{s+1}) - \mathsf{OTLLR}(v^s))^2|v^s].\end{aligned}$$

Thus by Lemma 4.2, we know that $\Delta(v^s) \geq 0.001 \cdot \Delta_2(v^s) \geq 0$. Now we have

$$\begin{aligned}0.001 >&\mathbb{E}[\mathsf{OTLLR}(v^{(t+1)L}) - \mathsf{OTLLR}(v^{tL})|v^{tL} = v]\\ =& \sum_{tL \leq s < (t+1)L} \mathbb{E}[\Delta(v^s)|v^{tL} = v].\end{aligned}$$

By Markov's inequality, we have $\Pr\left[\sum_{tL \leq s<(t+1)L} \Delta(v^s) \geq 1|v^{tL} = v\right] \leq 0.001$. Now by a union bound with (2), we have

$$\begin{aligned}&\mathbb{E}\left[\left(\sum_{tL\leq s<(t+1)L} (\mathsf{OTLLR}(v^{s+1}) - \mathsf{OTLLR}(v^s) - \Delta(v^s))\right)^2 \middle| v^{tL} = v\right]\\ =&\mathbb{E}\left[\left((\mathsf{OTLLR}(v^{(t+1)L}) - \mathsf{OTLLR}(v^{tL})) - \sum_{tL\leq s<(t+1)L} \Delta(v^s)\right)^2 \middle| v^{tL} = v\right]\\ \geq&(0.8 - 0.001) \times (3-1)^2\\ >&3. \end{aligned} \tag{3}$$

Since $\mathbb{E}\left[\mathsf{OTLLR}(v^{s+1}) - \mathsf{OTLLR}(v^s) - \Delta(v^s)|v^s\right] = 0$, we have

$$\begin{aligned}&\mathbb{E}\big[(\mathsf{OTLLR}(v^{s_1+1}) - \mathsf{OTLLR}(v^{s_1}) - \Delta(v^{s_1}))\cdot\\ &\quad(\mathsf{OTLLR}(v^{s_2+1}) - \mathsf{OTLLR}(v^{s_2}) - \Delta(v^{s_2}))|v^{tL} = v\big] = 0\end{aligned}$$

whenever $s_1 < s_2$ by further conditioning on v^{s_2}. Thus expanding (3) we have

$$\begin{aligned}&\sum_{tL\leq s<(t+1)L} \mathbb{E}[\Delta_2(v^s)|v^{tL} = v]\\ =&\mathbb{E}\left[\sum_{tL\leq s<(t+1)L} (\mathsf{OTLLR}(v^{s+1}) - \mathsf{OTLLR}(v^s) - \Delta(v^s))^2 \middle| v^{tL} = v\right] \geq 3.\end{aligned}$$

Since $\Delta(v^s) \geq 0.001 \cdot \Delta_2(v^s)$, we have

$$0.001 > \sum_{tL\leq s<(t+1)L} \mathbb{E}[\Delta(v^s)|v^{tL} = v] \geq 0.001 \cdot \sum_{tL\leq s<(t+1)L} \mathbb{E}[\Delta_2(v^s)|v^{tL} = v] \geq 0.001 \times 3,$$

a contradiction.

Now we have shown $\mathbb{E}[\mathsf{OTLLR}(v^{(t+1)L}) - \mathsf{OTLLR}(v^{tL})|v^{tL} = v] \geq 0.001$. Adding it to the trivial inequality $\mathbb{E}[\mathsf{S}(v^{(t+1)L}) - \mathsf{S}(v^{tL})|v^{tL} = v] \geq 0$ proves the lemma. ◀

4.4 A helper inequality

▶ **Lemma 4.4.** *For all $M \geq 0, t \in (0, e^M]$, we have*

$$t \ln t - (t-1) \geq \frac{1}{M+2} \cdot t \ln^2 t.$$

Proof. Define function $h(t) = t \ln t - (t-1) - \frac{1}{M+2} \cdot t \ln^2 t$ on the interval $t \in (0, e^M]$. Our goal is to show $h(t) \geq 0$. Note that $h(1) = 0$, so we only need to show $h'(t) \geq 0$ for $t \geq 1$ and $h'(t) \leq 0$ for $t \leq 1$. We prove this by calculating $h'(t)$:

$$h'(t) = \ln t - \frac{1}{M+2} \cdot \ln^2 t - \frac{2}{M+2} \cdot \ln t = \left(1 - \frac{(\ln t) + 2}{M+2}\right) \ln t.$$

Note that $1 - \frac{(\ln t)+2}{M+2} \geq 0$ because $\ln t \leq M$. Therefore $h'(t) \geq 0$ when $t \geq 1$ and $h'(t) \leq 0$ when $t \leq 1$, as desired. ◀

5 Application 1: Selection Problem

5.1 Bi-correlated samples

To establish a relationship between correlated samples and selection, we first define an intermediate problem. The *bi-correlated samples problem* is defined by (here $\mathcal{D}_{ab} := \mathcal{D}_a \times \mathcal{D}_b$):

$$\begin{aligned} \mathsf{biCorr}_\epsilon(f, \mathcal{D}) &:= \min\nolimits_{k \geq 1} \mathsf{D}_\epsilon(f^{2k}, \tfrac{1}{2}\mathcal{D}_{01}^k + \tfrac{1}{2}\mathcal{D}_{10}^k), \\ \mathsf{biCorr}_\epsilon(f) &:= \max\nolimits_{\mathcal{D}} \mathsf{biCorr}_\epsilon(f, \mathcal{D}). \end{aligned}$$

That is, the task is to decide whether f^{2k} outputs $(01)^k$ or $(10)^k$ as $k \to \infty$. We show this is as hard as correlated samples:

▶ **Lemma 5.1.** $\mathsf{Corr}(f, \mathcal{D}) = \Theta(\mathsf{biCorr}(f, \mathcal{D}))$.

Proof. It is obvious that $\mathsf{biCorr}(f, \mathcal{D}) \leq \mathsf{Corr}(f, \mathcal{D})$, so we focus on the converse, $\mathsf{Corr}(f, \mathcal{D}) \leq O(\mathsf{biCorr}(f, \mathcal{D}))$. The proof is via a hybrid argument. Let $T\colon (\{0,1\}^n)^{2k} \to \{0,1\}$ be an optimal algorithm for $\mathsf{biCorr}_{1/3}(f, \mathcal{D})$ that uses k sample pairs. Letting $d(-,-)$ denote the statistical distance between two distributions, the fact that T achieves error $\epsilon := 1/3$ can be written as

$$d(T(\mathcal{D}_{01}^k), T(\mathcal{D}_{10}^k)) \;\geq\; 1 - 2\epsilon.$$

By the triangle inequality,

$$d(T(\mathcal{D}_{01}^k), T(\mathcal{D}_{00}^k)) \;+\; d(T(\mathcal{D}_{00}^k), T(\mathcal{D}_{10}^k)) \;\geq\; 1 - 2\epsilon.$$

Either the first or the second term is $\geq (1-2\epsilon)/2$. Say the first (second case is similar):

$$d(T(\mathcal{D}_{01}^k), T(\mathcal{D}_{00}^k)) \;\geq\; (1-2\epsilon)/2 \;=\; 1 - 2\epsilon' \qquad \text{where } \epsilon' := 1/4 + \epsilon/2 = 5/12.$$

This means we can turn T into an 5/12-error algorithm for the correlated k-samples problem: the odd numbered input samples of T the algorithm can generate from $\mathcal{D}_0$ on its own; the even numbered input samples of T are taken from the input to the correlated k-samples problem. Finally, the error can be reduced to 1/3 via Fact 1.1. ◀

5.2 Proof of Theorem 1.3

First item. The following claim together with Lemma 5.1 implies the first item.

▷ Claim 5.2. $\mathsf{biCorr}_\epsilon(f,\mathcal{D}) \le \mathsf{Sel}_\epsilon(f,\mathcal{D})$.

Proof. Let T_{Sel} be an optimal algorithm for $\mathsf{Sel}_\epsilon(f,\mathcal{D})$ using k samples. We describe an algorithm T_{biCorr} for bi-correlated k-samples with the same error and query cost. Let $x = (x_{ij})$ for $(i,j) \in [k] \times [2]$ be the random input to T_{biCorr}, that is, either *(i)* $x \sim \mathcal{D}_{01}^k$ or *(ii)* $x \sim \mathcal{D}_{10}^k$. The algorithm T_{biCorr} chooses a random string $z \in [2]^k$ and runs T_{Sel} on input $y \coloneqq (x_{iz_i})_{i\in[k]}$. Note that y is distributed as $\mathcal{D}^k$ in both cases *(i)* and *(ii)*. Suppose T_{Sel} outputs some $(i, f(x_{iz_i}))$. Assuming this output is correct for selection, and remembering our choice of z_i, we can deduce which case, *(i)* or *(ii)*, the input x came from, and let T_{biCorr} guess accordingly. Hence algorithm T_{biCorr} is correct every time T_{Sel} is, and so the error parameter is unaffected. ◁

Second and third item. For separating correlated samples from selection, we again consider the n-bit $\textsc{Xor}_n$ function. Define $x \sim \mathcal{D}$ by the following process:

1. Sample z uniformly from $\{0,1\}^{n-2}$ and let $a \coloneqq \textsc{Xor}_{n-2}(z)$.
2. Sample b uniformly from $\{0,1\}$.
3. With probability $\epsilon \coloneqq 1\%$, output $x \coloneqq aaz$; with probability $1-\epsilon = 99\%$, output $x \coloneqq bbz$.

Note that the first two bits of $x \sim \mathcal{D}$ are identical and hence $\textsc{Xor}_n(x) = \textsc{Xor}_{n-2}(z)$. Moreover, the first bit is ϵ-correlated with the function value $\textsc{Xor}_n(x)$. This makes $(\textsc{Xor}_n,\mathcal{D})$ easy for the correlated samples problem: The 1-query algorithm that guesses the function value based on the first bit of the first sample has error $\le 1/2 - \epsilon/2$, and this error can be reduced to $1/3$ via Fact 1.1. This shows that $\mathsf{Corr}(\textsc{Xor}_n,\mathcal{D}) = O(1)$.

Next we prove the lower bound $\mathsf{Sel}(\textsc{Xor}_n,\mathcal{D}) = \Omega(n)$, which also proves the third item. Suppose for contradiction that T is a height-$(n-3)$ deterministic decision tree for k-selection for $(\textsc{Xor}_n,\mathcal{D})$. Consider any leaf ℓ that claims the i-th sample evaluates to $b \in \{0,1\}$. If we condition $\mathcal{D}^k$ by the $\le n-3$ queries made by ℓ, we note that the function value is still only slightly biased away from $1/2$, that is, $\mathbb{E}_{x\sim\mathcal{D}^k|\ell}[\textsc{Xor}_n(x_i)] \in 1/2 \pm \epsilon$. Hence no leaf of T can compute selection to within error $\le 1/3$. This concludes the proof of Theorem 1.3.

6 Application 2: Randomized Composition

Goal. In this section we prove Theorem 1.6, namely $\mathsf{R}(f\circ g) \ge \Omega(\mathsf{fbs}(f)\mathsf{R}(g))$. By Theorem 1.2 and Lemma 5.1 (from Subsection 5.1) it suffices to show

$$\mathsf{biCorr}(g) \;\le\; O(\mathsf{R}(f\circ g)/\mathsf{fbs}(f)).$$

Let T be an optimal $1/10$-error algorithm for $f\circ g$ making $q \coloneqq O(\mathsf{R}(f\circ g))$ queries. Our goal is, given any balanced input distribution $\mathcal{D} \coloneqq \frac{1}{2}\mathcal{D}_0 + \frac{1}{2}\mathcal{D}_1$ to the inner function g, to build a bounded-error algorithm T' solving the bi-correlated samples problem for $(g,\mathcal{D})$.

Rarely queried block. By the definition of $\mathsf{fbs}(f)$, there is an input $y \in \{0,1\}^n$ to f (say, $f(y) = 0$) with sensitive blocks $B_1,\cdots,B_N \subseteq [n]$ and weights $w_1,\cdots,w_N \in [0,1]$ such that

$$\textstyle\sum_{j\in[N]} w_j \;=\; \mathsf{fbs}(f), \tag{4}$$

$$\textstyle\sum_{j:B_j\ni i} w_j \;\le\; 1, \qquad \forall i \in [n]. \tag{5}$$

For any $z \in \{0,1\}^n$, define $\mathcal{D}_z$ as the distribution over $(x_1, \cdots, x_n) \in (\{0,1\}^m)^n$ where each x_i is drawn independently from $\mathcal{D}_{z_i}$. Hence we have $g^n(x) = z$ for $x \sim \mathcal{D}_z$. We define

$$q_j \;\coloneqq\; \text{expected \# of queries } T \text{ makes to block } B_j \text{ on input } \mathcal{D}_y.$$

That is, if we denote by $i_t \in [n]$ the block that T queries at time t, then q_j is the expected number of time steps t with $i_t \in B_j$. By linearity of expectation and (5), we have

$$\textstyle\sum_{j\in[N]} w_j q_j \;=\; \mathbb{E}\Big[\sum_{j\in[N]} w_j \sum_{t:i_t\in B_j} 1\Big] \;=\; \mathbb{E}\Big[\sum_{t\in[q]} \sum_{j:B_j\ni i_t} w_j\Big] \;\le\; \mathbb{E}\Big[\sum_{t\in[q]} 1\Big] \;\le\; q.$$

Combining this with (4), we know there exists $j \in [N]$, say $j = 1$ for simplicity, such that

$$q_1 \;\le\; \frac{q}{\mathsf{fbs}(f)}.$$

Truncated *T*. Next we modify T so that it makes at most $5q_1$ queries to block B_1 for *every* input (not just on average over $\mathcal{D}_y$). Namely, if T makes more than $5q_1$ queries to block B_1, we simply let T halt and output 1; otherwise its behavior is unchanged. We denote this "truncated" algorithm by T^{tr}. We claim that T^{tr} still computes $f \circ g$ correctly on average over both $\mathcal{D}_y$ and $\mathcal{D}_{y^{B_1}}$ (recall that y^{B_1} is y but with the block B_1 flipped; note that $f(y^{B_1}) = 1$ and hence $(f \circ g)(x) = 1$ for each $x \sim \mathcal{D}_{y^{B_1}}$)

$$\begin{aligned}
\textit{Correct for } x \sim \mathcal{D}_{y^{B_1}}\textit{:} \qquad \Pr[T^{\text{tr}}(x) = 1] \;&\ge\; \Pr[T(x) = 1] \\
&\ge\; 4/5. && (6)
\end{aligned}$$

$$\begin{aligned}
\textit{Correct for } x \sim \mathcal{D}_y\textit{:} \qquad \Pr[T^{\text{tr}}(x) = 0] \;&\ge\; \Pr[T(x) = 0] \\
&\quad - \Pr[T(x) \text{ makes} > 5q_1 \text{ queries to } B_1] && (7)\\
&\ge\; 4/5 - 1/5 && (8)\\
&=\; 3/5, && (9)
\end{aligned}$$

where (7) uses the Union Bound and (8) uses the Markov Bound.

Algorithm *T′*. We are ready to define the algorithm T' for the bi-correlated samples problem for $(g, \mathcal{D})$. The random input to this problem is $z = (z_{ij})$, $(i,j) \in [n] \times \{0,1\}$, sampled either from *(i)* $\mathcal{D}_{01}^n$ or *(ii)* $\mathcal{D}_{10}^n$. On input z the algorithm T' simply runs T^{tr} on the input $(x_1, \ldots, x_n) \in (\{0,1\}^m)^n$ defined by

$$x_i \;\coloneqq\; \begin{cases} z_{iy_i} & \text{for } i \in B_1, \\ \sim \mathcal{D}_{y_i} & \text{for } i \notin B_1. \end{cases}$$

That is, for $i \in B_1$ the algorithm T' simply copies its input bits in z to the bits of x. For $i \notin B_1$ the algorithm T' uses its own randomness to generate an independent sample from either $\mathcal{D}_0$ or $\mathcal{D}_1$. The key observation is that in case *(i)* we have $x \sim \mathcal{D}_y$, and in case *(ii)* we have $x \sim \mathcal{D}_{y^{B_1}}$. But T^{tr} can distinguish these two cases to within bounded error by (6) and (9). Hence T' is a bounded-error algorithm for bi-correlated samples with query cost $5q_1 \le O(q/\mathsf{fbs}(f))$. This completes the proof of Theorem 1.6.

References

1 Scott Aaronson. Quantum certificate complexity. *Journal of Computer and System Sciences*, 74(3):313–322, 2008. doi:10.1016/j.jcss.2007.06.020.

2 Andris Ambainis, Martins Kokainis, Krišjānis Prūsis, and Jevgēnijs Vihrovs. All classical adversary methods are equivalent for total functions. In *Proceedings of the 35th Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 96, pages 8:1–8:14, 2018. doi:10.4230/LIPIcs.STACS.2018.8.

3 Anurag Anshu, Dmitry Gavinsky, Rahul Jain, Srijita Kundu, Troy Lee, Priyanka Mukhopadhyay, Miklos Santha, and Swagato Sanyal. A composition theorem for randomized query complexity. In *Proceedings of the 37th Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 93, pages 10:1–10:13, 2018. doi:10.4230/LIPIcs.FSTTCS.2017.10.

4 Amos Beimel, Sebastian Ben Daniel, Eyal Kushilevitz, and Enav Weinreb. Choosing, agreeing, and eliminating in communication complexity. *Computational Complexity*, 23:1–42, 2014. doi:10.1007/s00037-013-0075-7.

5 Shalev Ben-David and Eric Blais. A new minimax theorem for randomized algorithms. *arXiv preprint*, 2020. arXiv:2002.10802.

6 Shalev Ben-David and Eric Blais. A tight composition theorem for the randomized query complexity of partial functions. *arXiv preprint*, 2020. arXiv:2002.10809.

7 Shalev Ben-David and Robin Kothari. Randomized query complexity of sabotaged and composed functions. In *Proceedings of the 43rd International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 55, pages 60:1–60:14, 2016. doi:10.4230/LIPIcs.ICALP.2016.60.

8 Harry Buhrman and Ronald de Wolf. Complexity measures and decision tree complexity: a survey. *Theoretical Computer Science*, 288(1):21–43, 2002. Complexity and Logic. doi:10.1016/S0304-3975(01)00144-X.

9 Dmitry Gavinsky, Troy Lee, Miklos Santha, and Swagato Sanyal. A composition theorem for randomized query complexity via max-conflict complexity. In *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 132, pages 64:1–64:13, 2019. doi:10.4230/LIPIcs.ICALP.2019.64.

10 Justin Gilmer, Michael Saks, and Srikanth Srinivasan. Composition limits and separating examples for some boolean function complexity measures. *Combinatorica*, 36(3):265–311, 2016. doi:10.1007/s00493-014-3189-x.

11 Mika Göös, T. S. Jayram, Toniann Pitassi, and Thomas Watson. Randomized communication versus partition number. *ACM Transactions on Computation Theory*, 10(1), 2018. doi:10.1145/3170711.

12 Peter Høyer, Troy Lee, and Robert Špalek. Negative weights make adversaries stronger. In *Proceedings of the 39th Symposium on Theory of Computing (STOC)*, STOC '07, page 526–535, 2007. doi:10.1145/1250790.1250867.

13 Raghav Kulkarni and Avishay Tal. On fractional block sensitivity. *Chicago Journal of Theoretical Computer Science*, 2016(8), 2016. doi:10.4086/cjtcs.2016.008.

14 Ashley Montanaro. A composition theorem for decision tree complexity. *Chicago Journal of Theoretical Computer Science*, 2014(6), 2014. doi:10.4086/cjtcs.2014.006.

15 Noam Nisan. CREW PRAMs and decision trees. *SIAM Journal on Computing*, 20(6):999–1007, 1991. doi:10.1137/0220062.

16 Ben Reichardt. Reflections for quantum query algorithms. In *Proceedings of the 22nd Symposium on Discrete Algorithms (SODA)*, pages 560–569, 2011.

17 Petr Savický. On determinism versus unambiquous nondeterminism for decision trees. Technical Report TR02-009, Electronic Colloquium on Computational Complexity (ECCC), 2002. URL: http://eccc.hpi-web.de/report/2002/009/.

18 Ronen Shaltiel. Towards proving strong direct product theorems. *Computational Complexity*, 12(1/2):1–22, 2004. doi:10.1007/s00037-003-0175-x.

19 Avishay Tal. Properties and applications of boolean function composition. In *Proceedings of the 4th Conference on Innovations in Theoretical Computer Science (ITCS)*, pages 441–454, 2013. doi:10.1145/2422436.2422485.

20 Andrew Yao. Probabilistic computations: Toward a unified measure of complexity. In *Proceedings of the 18th Symposium on Foundations of Computer Science (SFCS 1977)*, pages 222–227, October 1977. doi:10.1109/SFCS.1977.24.

Medians in Median Graphs and Their Cube Complexes in Linear Time

Laurine Bénéteau
Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France
laurine.beneteau@lis-lab.fr

Jérémie Chalopin
Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France
jeremie.chalopin@lis-lab.fr

Victor Chepoi
Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France
victor.chepoi@lis-lab.fr

Yann Vaxès
Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France
yann.vaxes@lis-lab.fr

Abstract

The median of a set of vertices P of a graph G is the set of all vertices x of G minimizing the sum of distances from x to all vertices of P. In this paper, we present a linear time algorithm to compute medians in median graphs, improving over the existing quadratic time algorithm. We also present a linear time algorithm to compute medians in the ℓ_1-cube complexes associated with median graphs. Median graphs constitute the principal class of graphs investigated in metric graph theory and have a rich geometric and combinatorial structure. Our algorithm is based on the majority rule characterization of medians in median graphs and on a fast computation of parallelism classes of edges (Θ-classes or hyperplanes) via Lexicographic Breadth First Search (LexBFS). To prove the correctness of our algorithm, we show that any LexBFS ordering of the vertices of G satisfies the following *fellow traveler property* of independent interest: the parents of any two adjacent vertices of G are also adjacent.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Theory of computation → Computational geometry

Keywords and phrases Median Graph, CAT(0) Cube Complex, Median Problem, Linear Time Algorithm, LexBFS

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.10

Category Track A: Algorithms, Complexity and Games

Related Version A full version of this paper is available at `https://arxiv.org/abs/1907.10398`.

Funding The research on this paper was supported by the ANR project DISTANCIA (ANR-17-CE40-0015).

Acknowledgements The authors are grateful to the anonymous referees for useful remarks that helped improving the presentation of this work.

1 Introduction

The median problem (also called the Fermat-Torricelli problem or the Weber problem) is one of the oldest optimization problems in Euclidean geometry [43]. The *median problem* can be defined for any metric space (X, d): given a finite set $P \subset X$ of points with positive weights, compute the points x of X minimizing the sum of the distances from x to the points

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 10; pp. 10:1–10:17
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

of P multiplied by their weights. The median problem in graphs is one of the principal models in network location theory [35, 62] and is equivalent to finding nodes with largest closeness centrality in network analysis [14, 15, 57]. It also occurs in social group choice as the Kemeny median. In the consensus problem in social group choice, given individual rankings one has to compute a consensual group decision. By Arrow's impossibility theorem, there is no consensus function satisfying natural "fairness" axioms. It is also well-known that the majority rule leads to Condorcet's paradox, i.e., to the existence of cycles in the majority relation. In this respect, the Kemeny median [39, 40] is an important consensus function and corresponds to the median problem in graphs of permutahedra.

The median problem in Euclidean spaces cannot be solved in a symbolic form [6], but it can be solved numerically by Weiszfeld's algorithm [66] and its convergent modifications (see e.g. [51]) and can be approximated in nearly linear time with arbitrary precision [26]. For the ℓ_1-metric the median problem becomes much easier and can be solved by the majority rule on coordinates, i.e., by taking as median a point whose ith coordinate is the median of the list consisting of ith coordinates of the points of P. This kind of majority rule was used in [38] to define centroids of trees (which coincide with their medians [32, 62]). For graphs with n vertices, m edges, and standard graph distance, the median problem can be trivially solved in $O(nm)$ time by solving the All Pairs Shortest Paths (APSP) problem. One may ask if APSP is necessary to compute the median. However, by [1] APSP and median problem are equivalent under subcubic reductions. It was also shown in [2] that computing the medians of sparse graphs in subquadratic time refutes the HS (Hitting Set) conjecture.

In this paper, we show that the medians in median graphs can be computed in optimal $O(m)$ time (i.e., without solving APSP). Median graphs are the graphs in which each triplet u, v, w of vertices has a unique median, i.e., a vertex metrically lying between u and v, v and w, and w and u. They originally arise in universal algebra [4, 18] and their properties have been first investigated in [45, 49]. It was shown in [24, 54] that the cube complexes of median graphs are exactly the CAT(0) cube complexes, i.e., cube complexes of global non-positive curvature. CAT(0) cube complexes, introduced and nicely characterized in [33] in a local-to-global way, are now one of the principal objects of investigation in geometric group theory [59]. Median graphs also occur in Computer Science: by [3, 13] they are exactly the domains of event structures (one of the basic abstract models of concurrency) [50] and median-closed subsets of hypercubes are exactly the solution sets of 2-SAT formulas [48, 60]. The bijections between median graphs, CAT(0) cube complexes, and event structures have been used in [20, 21, 25] to disprove three conjectures in concurrency [56, 63, 64]. Finally, median graphs, viewed as median closures of sets of vertices of a hypercube, contain all most parsimonious (Steiner) trees [12] and as such have been extensively applied in human genetics. For a survey on median graphs and their connections with other discrete and geometric structures, see the books [36, 42], the surveys [10, 41], and the paper [22].

As we noticed, median graphs have strong geometric and metric properties. For the median problem, the concept of Θ-classes is essential. Two edges of a median graph G are called opposite if they are opposite in a common square of G. The equivalence relation Θ is the reflexive and transitive closure of this oppositeness relation. Each equivalence class of Θ is called a Θ-class (Θ-classes correspond to hyperplanes in CAT(0) cube complexes [58] and to events in event structures [50]). Removing the edges of a Θ-class, the graph G is split into two connected components which are convex and gated. Thus they are called halfspaces of G. The convexity of halfspaces implies via [29] that any median graph G isometrically embeds into a hypercube of dimension equals to the number q of Θ-classes of G.

Our results and motivation. In this paper, we present a linear time algorithm to compute medians in median graphs and in ℓ_1-cube complexes associated to median graphs. Our main motivation and technique stem from the majority rule characterization of medians in median graphs and the unimodality of the median function [8, 61]. Even if the majority rule is simple to state and is a commonly approved consensus method, its algorithmic implementation is less trivial if one has to avoid the computation of the distance matrix. On the other hand, the unimodality of the median function implies that one may find the median set by using local search. More generally, consider a partial orientation of the input median graph G, where an edge uv is transformed into the arc $\overrightarrow{uv}$ iff the median function at u is larger than the median function at v (in case of equality we do not orient the edge uv). Then the median set is exactly the set of all the sinks in this partial orientation of G. Therefore, it remains to compare for every edge uv the median function at u and at v in constant time. For this we use the partition of the edge-set of a median graph G into Θ-classes and; for every Θ-class, the partition of the vertex-set of G into complementary halfspaces. It is easy to notice that all edges of the same Θ-class are oriented in the same way because for any such edge uv the difference between the median functions at u and at v, respectively, can be expressed as the sum of weights of all vertices in the same halfspace as v minus the sum of weights of all vertices in the same halfspace as u.

Our main technical contribution is a new method for computing the Θ-classes of a median graph G with n vertices and m edges in linear $O(m)$ time. For this, we prove that Lexicographic Breadth First Search (LexBFS) [55] produces an ordering of the vertices of G satisfying the following *fellow traveler property*: for any edge uv, the parents of u and v are adjacent. With the Θ-classes of G at hand and the majority rule for halfspaces, we can compute the weights of halfspaces of G in optimal time $O(m)$, leading to an algorithm of the same complexity for computing the median set. We adapt our method to compute in linear time the median of a finite set of points in the ℓ_1-cube complex associated with G. The method can be applied to compute the total distance (the Wiener index) in optimal $O(m)$ time and the distance matrix of G in optimal $O(n^2)$ time (see the full version [16]).

Related work. The study of medians in median graphs originated in [8, 61] and continued in [7, 44, 46, 47, 53]. Using different techniques and extending the majority rule for trees [32], the following *majority rule* has been established in [8, 61]: a halfspace H of a median graph G contains at least one median iff H contains at least one half of the total weight of G; moreover, the median of G coincides with the intersection of halfspaces of G containing strictly more than half of the total weight. Hence the median set is always convex. It was shown in [61] that the median function of a median graph is weakly convex (an analog of a discrete convex function). This convexity property characterizes all graphs in which all local medians are global [9]. A nice axiomatic characterization of medians of median graphs via three basic axioms has been obtained in [47]. More recently, [53] characterized median graphs as *closed Condorcet domains*, i.e., as sets of linear orders with the property that, whenever the preferences of all voters belong to the set, their majority relation has no cycles and also belongs to the set. In the full version [16] we show that the median graphs are the bipartite graphs in which the medians are characterized by the majority rule.

Prior to our work, the best algorithm to compute the Θ-classes of a median graph G has complexity $O(m \log n)$ [34]. It was used in [34] to recognize median graphs in subquadratic time. The previous best algorithm for the median problem in a median graph G with n vertices and q Θ-classes has complexity $O(qn)$ [7] which is quadratic in the worst case. Indeed q may be linear in n (as in the case of trees) and is always at least $d(\sqrt[d]{n}-1)$ as shown below

(d is the largest dimension of a hypercube which is an induced subgraph of G). Additionally, [7] assumes that an isometric embedding of G in a q-hypercube is given. The description of such an embedding has already size $O(qn)$. The Θ-classes of a median graph G correspond to the coordinates of the smallest hypercube in which G isometrically embeds (this is called the *isometric dimension* of G [36]). Thus one can define Θ-classes for all partial cubes, i.e., graphs isometrically embeddable into hypercubes. An efficient computation (in $O(n^2)$ time) of all Θ-classes was the main step of the $O(n^2)$ algorithm of [30] for recognizing partial cubes. The fellow-traveler property (which is essential in our computation of Θ-classes) is a notion coming from geometric group theory [31] and is a main tool to prove the (bi)automaticity of a group. In a slightly stronger form it allows to establish the dismantlability of graphs (see [19, 23, 24] for examples of classes of graphs in which a fellow traveler order was obtained by BFS or LexBFS). LexBFS has been used to solve optimally several algorithmic problems in different classes of graphs, in particular for their recognition (for a survey, see [28]).

Cube complexes of median graphs with ℓ_1-metric have been investigated in [65]. The same complexes but endowed with the ℓ_2-metric are exactly the CAT(0) cube complexes. As we noticed above, they are of great importance in geometric group theory [59]. The paper [17] established that the space of trees with a fixed set of leaves is a CAT(0) cube complex. A polynomial-time algorithm to compute the ℓ_2-distance between two points in this space was proposed in [52]. This result was recently extended in [37] to all CAT(0) cube complexes. A convergent numerical algorithm for the median problem in CAT(0) spaces was given in [5].

2 Preliminaries

All graphs $G = (V, E)$ in this paper are finite, undirected, simple, and connected; V is the vertex-set and E is the edge-set of G. We write $u \sim v$ if $u, v \in V$ are adjacent. The *distance* $d(u, v) = d_G(u, v)$ between two vertices u and v is the length of a shortest (u, v)-path, and the *interval* $I(u, v) = \{x \in V : d(u, x) + d(x, v) = d(u, v)\}$ consists of all the vertices on shortest (u, v)–paths. A set H (or the subgraph induced by H) is *convex* if $I(u, v) \subseteq H$ for any two vertices u, v of H; H is a *halfspace* if H and $V \setminus H$ are convex. Finally, H is *gated* if for every vertex $v \in V$, there exists a (unique) vertex $v' \in V(H)$ (the *gate* of v in H) such that for all $u \in V(H)$, $v' \in I(u, v)$. The *k-dimensional hypercube* Q_k has all subsets of $\{1, \dots, k\}$ as the vertex-set and $A \sim B$ iff $|A \triangle B| = 1$. A graph G is called *median* if $I(x, y) \cap I(y, z) \cap I(z, x)$ is a singleton for each triplet x, y, z of vertices; this unique intersection vertex $m(x, y, z)$ is called the *median* of x, y, z. Median graphs are bipartite and do not contain induced $K_{2,3}$. The *dimension* $d = \dim(G)$ of a median graph G is the largest dimension of a hypercube of G. In G, we refer to the 4-cycles as *squares*, and the hypercube subgraphs as *cubes*.

A map $w : V \to \mathbb{R}^+ \cup \{0\}$ is called a *weight function*. For a vertex $v \in V$, $w(v)$ denotes the weight of v (for a set $S \subseteq V$, $w(S) = \sum_{x \in S} w(x)$ denotes the weight of S). Then $F_w(x) = \sum_{v \in V} w(v) d(x, v)$ is called the *median function* of the graph G and a vertex x minimizing F_w is called a *median vertex* of G. Finally, $\mathrm{Med}_w(G) = \{x \in V : x \text{ is a median of } G\}$ is called the *median set* (or simply, the *median*) of G with respect to the weight function w.

3 Facts about median graphs

We recall the principal properties of median graphs used in the algorithms. Some of those results are a part of folklore for the people working in metric graph theory and some other results can be found in the papers [45, 46] by Mulder.

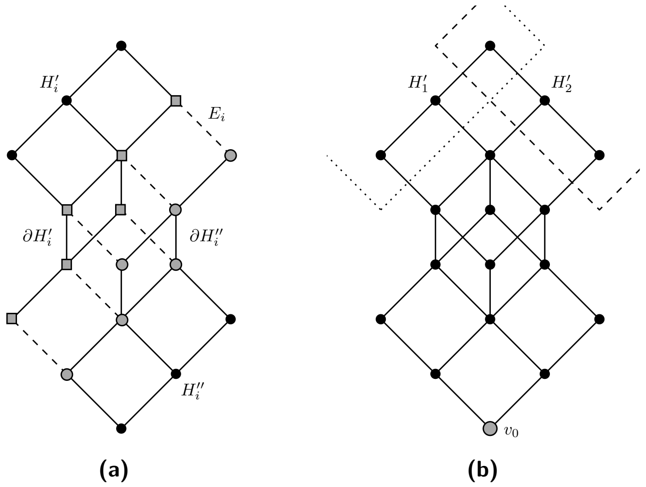


Figure 1 (a) In dashed, the Θ-class E_i of D, its two complementary halfspaces H'_i and H''_i and their boundaries $\partial H'_i$ and $\partial H''_i$, (b) two peripheral halfspaces of D, and (c) a LexBFS ordering of D.

From now on, $G = (V, E)$ is a median graph with n vertices and m edges. The first three properties follow from the definition.

▶ **Lemma 1** (Quadrangle Condition). *For any vertices u, v, w, z of G such that $v, w \sim z$ and $d(u, v) = d(u, w) = d(u, z) - 1 = k$, there is a unique vertex $x \sim v, w$ such that $d(u, x) = k - 1$.*

▶ **Lemma 2** (Cube Condition). *Any three squares of G, pairwise intersecting in three edges and all three intersecting in a single vertex, belong to a 3-dimensional cube of G.*

▶ **Lemma 3** (Convex=Gated). *A subgraph of G is convex if and only if it is gated. Each convex subgraph S of G is the intersection of all halfspaces containing S.*

Two edges uv and $u'v'$ of G are in relation Θ_0 if $uvv'u'$ is a square of G and uv and $u'v'$ are opposite edges of this square. Let Θ denote the reflexive and transitive closure of Θ_0. Denote by $E_1, \ldots, E_q$ the equivalence classes of Θ and call them Θ-*classes* (see Fig. 1(a)).

▶ **Lemma 4.** [45] (Halfspaces and Θ-classes). *For any Θ-class E_i of G, the graph $G_i = (V, E \setminus E_i)$ consists of exactly two connected components H'_i and H''_i that are halfspaces of G; all halfspaces of G have this form. If $uv \in E_i$, then H'_i and H''_i are the subgraphs of G induced by $W(u, v) = \{x \in V : d(u, x) < d(v, x)\}$ and $W(v, u) = \{x \in V : d(v, x) < d(u, x)\}$.*

Two Θ-classes E_i and E_j are *crossing* if each halfspace of the pair $\{H'_i, H''_i\}$ intersects each halfspace of the pair $\{H'_j, H''_j\}$; otherwise, E_i and E_j are called *laminar*.

▶ **Lemma 5** (Crossing Θ-classes). *Any vertex $v \in V(G)$ and incident edges $vv_1 \in E_1, \ldots, vv_k \in E_k$ belong to a single cube of G if and only if $E_1, \ldots, E_k$ are pairwise crossing.*

The *boundary* $\partial H'_i$ of a halfspace H'_i is the subgraph of H'_i induced by all vertices v' of H'_i having a neighbor v'' in H''_i. A halfspace H'_i of G is *peripheral* if $\partial H'_i = H'_i$ (See Fig. 1(b)).

▶ **Lemma 6** (Boundaries). *For any Θ-class E_i of G, $\partial H'_i$ and $\partial H''_i$ are isomorphic and gated.*

From now on, we suppose that G is rooted at an arbitrary vertex v_0 called the *basepoint*. For any Θ-class E_i, we assume that v_0 belongs to the halfspace H''_i. Let $d(v_0, H'_i) = \min\{d(v_0, x) : x \in H'_i\}$. Since H'_i is gated, the gate of v_0 in H'_i is the unique vertex of H'_i

at distance $d(v_0, H_i')$ from v_0. Since median graphs are bipartite, the choice of v_0 defines a canonical orientation of the edges of G: $uv \in E$ is oriented from u to v (notation $\overrightarrow{uv}$) if $d(v_0, u) < d(v_0, v)$. Let $\overrightarrow{G}_{v_0}$ denote the resulting oriented pointed graph.

▶ **Lemma 7.** [46] (Peripheral Halfspaces). *Any halfspace H_i' maximizing $d(v_0, H_i')$ is peripheral.*

For a vertex v, all vertices u such that $\overrightarrow{uv}$ is an edge of $\overrightarrow{G}_{v_0}$ are called *predecessors* of v and are denoted by $\Lambda(v)$. Equivalently, $\Lambda(v)$ consists of all neighbors of v in the interval $I(v_0, v)$. A median graph G satisfies the *downward cube property* if any vertex v and all its predecessors $\Lambda(v)$ belong to a single cube of G.

▶ **Lemma 8.** [45] (Downward Cube Property). *G satisfies the downward cube property.*

Lemma 8 immediately implies the following upper bound on the number of edges of G:

▶ **Corollary 9.** *If G has dimension d, then $m \leq dn \leq n \log n$.*

We give a sharp lower bound on the number q of Θ-classes, which is new to our knowledge.

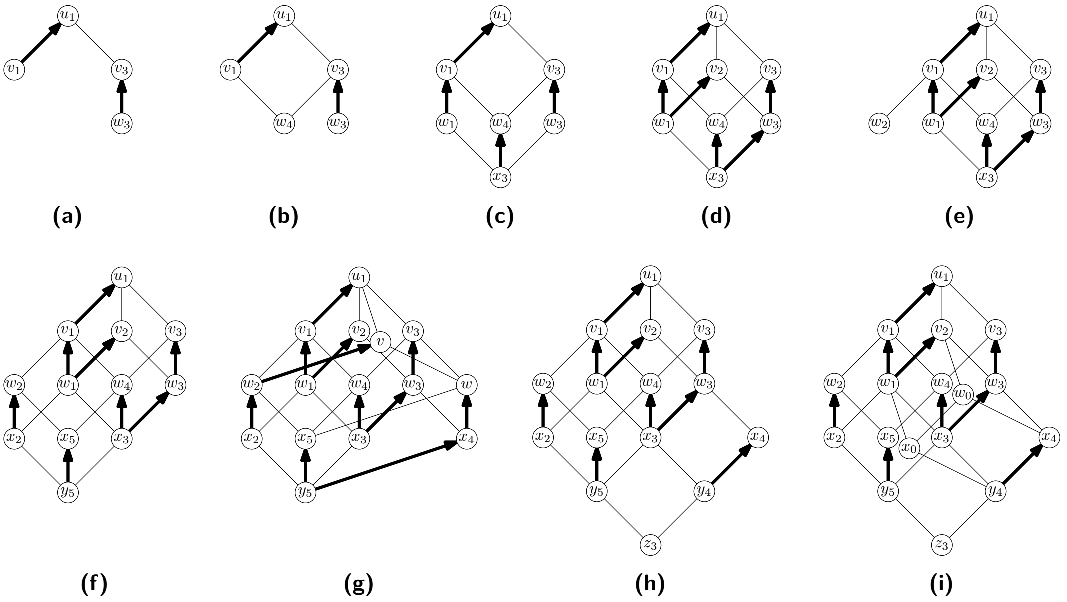
▶ **Proposition 10.** *If G has q Θ-classes and dimension d, then $q \geq d(\sqrt[d]{n} - 1)$ and this bound is tight.*

Proof. Let $\Gamma(G)$ be the crossing graph $\Gamma(G)$ of G: $V(\Gamma(G))$ is the set of Θ-classes of G and two Θ-classes are adjacent in $\Gamma(G)$ if they are crossing. Note that $|V(\Gamma(G))| = q$. Let $X(\Gamma(G))$ be the clique complex of $\Gamma(G)$. By the characterization of median graphs among ample classes [11, Proposition 4], the number of vertices of G is equal to the number $|X(\Gamma(G))|$ of simplices of $X(\Gamma(G))$. Since G is of dimension d, by [11, Proposition 4], $\Gamma(G)$ does not contain cliques of size $d + 1$. By Zykov's theorem [68] (see also [67]), the number of k-simplices in $X(\Gamma(G))$ is at most $\binom{d}{k}\left(\frac{q}{d}\right)^k$. Hence $n = |V(G)| = |X(\Gamma(G))| \leq \sum_{k=0}^{d} \binom{d}{k}\left(\frac{q}{d}\right)^k = \left(1 + \frac{q}{d}\right)^d$ and thus $q \geq d(\sqrt[d]{n} - 1)$. Let now G be the Cartesian product of d paths of length $(\sqrt[d]{n} - 1)$. Then G has $(\sqrt[d]{n} - 1 + 1)^d = n$ vertices and $d(\sqrt[d]{n} - 1)$ Θ-classes (each Θ-class of G corresponds to an edge of one of factors). ◀

4 Computation of the Θ-classes via LexBFS

The *Breadth-First Search (BFS)* refines the basepoint order and defines the same orientation $\overrightarrow{G}_{v_0}$ of G. BFS uses a queue Q and the insertion in Q defines a total order $<$ on the vertices of G: $x < y$ iff x is inserted in Q before y. When a vertex u arrives at the head of Q, it is removed from Q and all not yet discovered neighbors v of u are inserted in Q; u becomes the *parent* $f(v)$ of v; for any vertex $v \neq v_0$, $f(v)$ is the smallest predecessor of v. The arcs $\overrightarrow{f(v)v}$ define the *BFS-tree* of G. The *Lexicographic Breadth-First Search (LexBFS)*, proposed in [55], is a refinement of BFS. In BFS, if u and v have the same parent, then the algorithm order them arbitrarily. Instead, the LexBFS chooses between u and v by considering the ordering of their second-earliest predecessors. If only one of them has a second-earliest predecessor, then that one is chosen. If both u and v have the same second-earliest predecessor, then the tie is broken by considering their third-earliest predecessor, and so on (See Fig. 1(c)). The LexBFS uses a set partitioning data structure and can be implemented in linear time [55]. In median graphs, the next lemma shows that it suffices to consider only the earliest and second-earliest predecessors, leading to a simpler implementation of LexBFS:

▶ **Lemma 11.** *If u and v are two vertices of a median graph G, then $|\Lambda(u) \cap \Lambda(v)| \leq 1$.*



(a) **(b)** **(c)** **(d)** **(e)**

(f) **(g)** **(h)** **(i)**

Figure 2 Animated proof of Theorem 13.

During the execution of BFS (or LexBFS), one can assume that for each vertex v the set $\Lambda(v)$ of its predecessors is computed as an ordered list $\Lambda_<(v)$ (ordered by $<$). By Lemma 8, $|\Lambda_<(v)| \leq d := \dim(G)$. Note that $<$ also gives rise to a total order on the edges of G: for two edges uv and $u'v'$ with $u < v$ and $u' < v'$ we have $uv < u'v'$ iff $u < u'$ or if $u = u'$ and $v < v'$. The next lemma characterizes the first edge in the order $<$ of each Θ-class E_i.

▶ **Lemma 12.** *An edge $uv \in E_i$ with $d(v_0, u) < d(v_0, v)$ is the first edge of E_i iff $\Lambda_<(v) = \{u\}$.*

A graph G satisfies the *fellow-traveler property* if for any LexBFS ordering of the vertices of G, for any edge uv with $v_0 \notin \{u, v\}$, the parents $f(u)$ and $f(v)$ are adjacent.

▶ **Theorem 13.** *Any median graph G satisfies the fellow-traveler property.*

Proof. Let $<$ be an arbitrary LexBFS order of the vertices of G and f be its parent map. Since any LexBFS order is a BFS order, $<$ and f satisfy the following properties of BFS:

(BFS1) if $u < v$, then $f(u) \leq f(v)$;
(BFS2) if $f(u) < f(v)$, then $u < v$;
(BFS3) if $v \neq v_0$, then $f(v) = \min_<\{u : u \sim v\}$;
(BFS4) if $u < v$ and $v \sim f(u)$, then $f(v) = f(u)$.

Notice also the following simple but useful property:

▶ **Lemma 14.** *If $abcd$ is a square of G with $d(v_0, c) = k$, $d(v_0, b) = d(v_0, d) = k+1$, $d(v_0, a) = k + 2$ and $f(a) = b$, and the edge ad satisfies the fellow-traveler property, then $f(d) = c$.*

We prove the fellow-traveler property by induction on the total order on the edges of G defined by $<$. The proof is illustrated by several figures (the arcs of the parent map are represented in bold). We use the following convention: all vertices having the same distance to the basepoint v_0 will be labeled by the same letter but will be indexed differently; for example, w_1 and w_2 are two vertices having the same distance to v_0.

Suppose by way of contradiction that $e = u_1v_3$ with $v_3 < u_1$ is the first edge in the order $<$ such that the parents $f(u_1)$ and $f(v_3)$ of u_1 and v_3 are not adjacent. Then necessarily $f(u_1) \neq v_3$. Set $v_1 = f(u_1)$ and $w_3 = f(v_3)$ (Fig. 2a). Since $d(v_0, v_1) = d(v_0, v_3)$ and $u_1 \sim v_1, v_3$, by the quadrangle condition v_1 and v_3 have a common neighbor at distance

$d(v_0, v_1) - 1$ from v_0. This vertex cannot be w_3, otherwise $f(u_1)$ and $f(v_3)$ would be adjacent. Therefore there is a vertex $w_4 \sim v_1, v_3$ at distance $d(v_0, v_1) - 1$ from v_0 (Fig. 2b). By induction hypothesis, the parent $x_3 = f(w_4)$ of w_4 is adjacent to $w_3 = f(v_3)$. Since $u_1 \sim v_1 = f(u_1), v_3$ and $v_3 \sim w_3 = f(v_3), w_4$, by (BFS3) we conclude that $v_1 < v_3$ and $w_3 < w_4$. By (BFS2), $f(v_1) \leq f(v_3)$, whence $f(v_1) \leq w_3$ and since $f(v_1) \neq f(v_3)$ (otherwise, $f(u_1) \sim f(v_3)$), we deduce that $f(v_1) < w_3 < w_4$. Hence $f(v_1) \neq w_4$. Set $w_1 = f(v_1)$. By the induction hypothesis, $f(v_1) = w_1$ is adjacent to $f(w_4) = x_3$ (Fig. 2c). By the cube condition applied to the squares $w_4v_1w_1x_3$, $w_4v_1u_1v_3$, and $w_4v_3w_3x_3$ there is a vertex v_2 adjacent to u_1, w_1, and w_3. Since $u_1 \sim v_2$ and $f(u_1) = v_1$, by (BFS3) we obtain $v_1 < v_2$. Since v_2 is adjacent to w_1 and $w_1 = f(v_1)$, by (BFS4) we obtain $f(v_2) = f(v_1) = w_1$, and by (BFS2), $v_2 < v_3$. Since $f(v_2) = w_1$, by Lemma 14 for $v_2w_1x_3w_3$, we obtain $f(w_3) = x_3$ (Fig. 2d). Since $v_1 < v_2$, $f(v_1) = f(v_2) = w_1$, and $v_2 \sim w_1, w_3$, by LexBFS v_1 is adjacent to a predecessor different from w_1 and smaller than w_3. Since $w_3 < w_4$, this predecessor cannot be w_4. Denote by w_2 the second smallest predecessor of v_1 (Fig. 2e) and note that $w_1 < w_2 < w_3 < w_4$.

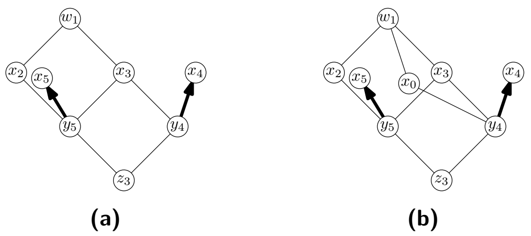
By the quadrangle condition, w_2 and w_4 are adjacent to a vertex x_5, which is necessarily different from x_3 because G is $K_{2,3}$-free. By the induction hypothesis, $f(w_2)$ and $f(v_1) = w_1$ are adjacent. Then $f(w_2) \neq x_3, x_5$, otherwise we obtain a forbidden $K_{2,3}$. Set $f(w_2) = x_2$. Analogously, $f(x_5) = y_5$ and $f(w_2) = x_2$ are adjacent as well as $f(x_5) = y_5$ and $f(w_4) = x_3$ (Fig. 2f). By (BFS1), $x_2 = f(w_2) < f(w_3) = x_3$ and by (BFS3), $x_3 = f(w_4) < x_5$. Since $w_3 < w_4$ with $f(w_3) = f(w_4)$ and w_4 is adjacent to x_5, by LexBFS w_3 must have a predecessor different from x_3 and smaller than x_5. This vertex cannot be x_2 by (BFS3) since $f(w_3) = x_3$. Denote this predecessor of w_3 by x_4 and observe that $x_2 < x_3 < x_4 < x_5$. By the induction hypothesis, the parent of x_4 is adjacent to $f(w_3) = x_3$. Let $y_4 = f(x_4)$.

If $y_4 = y_5$, applying the cube condition to the squares $x_3w_3x_4y_5$, $x_3w_4x_5y_5$, and $x_3w_4v_3w_3$ we find a vertex w adjacent to x_4, v_3, and x_5. Applying the cube condition to the squares $w_4v_3wx_5$, $w_4v_1w_2x_5$, and $w_4v_1u_1v_3$ we find a vertex v adjacent to u_1, w_2, and w. Since $v \sim w_2$, by (BFS3) $f(v) \leq w_2 < w_3 = f(v_3)$, hence by (BFS2) we obtain $v < v_3$. Therefore we can apply the induction hypothesis, and by Lemma 14 for $u_1v_1w_2v$, we deduce that $f(v) = w_2$. By Lemma 14 for $v_3w_3x_4w$, we deduce that $f(w) = x_4$ (Fig. 2g). Applying the induction hypothesis to the edge vw we have that $f(v) = w_2$ is adjacent to $f(w) = x_4$, yielding a forbidden $K_{2,3}$ induced by v, x_5, x_4, w, w_2 (Fig. 2g). All this shows that $y_4 \neq y_5$. By the quadrangle condition, y_5 and y_4 have a common neighbor z_3 (Fig. 2h).

Recall that $x_2 < x_3 < x_4 < x_5$, and note that by (BFS1), $y_4 = f(x_4) < f(x_5) = y_5$. We denote by H the subgraph of G induced by the vertices $V' = \{w_1, x_2, x_3, x_4, x_5, y_4, y_5, z_3\}$. The set of edges of H is $E' = \{z_3y_4, z_3y_5, y_4x_3, y_4x_4, y_5x_2, y_5x_3, y_5x_5, x_2w_1, x_3w_1\}$. To conclude the proof, we use the following technical lemma.

▶ **Lemma 15.** *Let $H = (V', E')$ (Fig. 3a) be an induced graph of G, where $d(v_0, w_1) = d(v_0, x_2) + 1 = \cdots = d(v_0, x_5) + 1 = d(v_0, y_4) + 2 = d(v_0, y_5) + 2 = d(v_0, z_3) + 3$ and $f(x_5) = y_5$ and $f(x_4) = y_4$, such that $x_2 < x_3 < x_4 < x_5$ and $y_4 < y_5$. If G satisfies the fellow-traveler property up to distance $d(v_0, w_1)$, then there exists a vertex x_0 such that $x_0 < x_2$ and $x_0 \sim w_1, y_4$ (Fig. 3b).*

Since G contains a subgraph H satisfying the conditions of Lemma 15, there exists a vertex x_0 such that $x_0 < x_2$ and $x_0 \sim w_1, y_4$ (Fig. 2i). By the cube condition applied to the squares $x_3w_1x_0y_4$, $x_3w_1v_2w_3$, and $x_3w_3x_4y_4$, there exists $w_0 \sim x_0, v_2, x_4$ (Fig. 2i). Since x_0 is adjacent to w_0, by (BFS3) $f(w_0) \leq x_0 < x_2 = f(w_2)$. By (BFS2), $w_0 < w_2$. Recall that $f(v_1) = w_1 = f(v_2)$ and that w_2 is the second-earliest predecessor of v_1. Since $w_0 < w_2$ and w_0 is a predecessor of v_2, by LexBFS we deduce that $v_2 < v_1$. Since v_1 and v_2 are



(a) **(b)**

Figure 3 The induced subgraph H in Lemma 15.

both adjacent to u_1 we obtain a contradiction with $f(u_1) = v_1$. This contradiction shows that any median graph G satisfies the fellow-traveller property. This finishes the proof of Theorem 13. ◀

Now we use Theorem 13 to compute the Θ-classes of G. We run LexBFS and return a LexBFS-ordering of $V(G)$ and $E(G)$ and the ordered lists $\Lambda_<(v), v \in V$. Then consider the edges of G in the LexBFS-order. Pick the first unprocessed edge uv and suppose that $u \in \Lambda_<(v)$. If $\Lambda_<(v) = \{u\}$, by Lemma 12, uv is the first edge of its Θ-class, thus we create a new Θ-class E_i and insert uv as the first edge of E_i. We call uv the *root* of E_i and keep $d(v_0, v)$ as the distance from v_0 to H'_i. Now suppose $|\Lambda_<(v)| \geq 2$. We consider two cases: (i) $u \neq f(v)$ and (ii) $u = f(v)$. For (i), by Theorem 13, uv and $f(u)f(v)$ are opposite edges of a square. Therefore uv belongs to the Θ-class of $f(u)f(v)$ (which was already computed because $f(u)f(v) < uv$). In order to recover the Θ-class of the edge $f(u)f(v)$ in constant time, we use a (non-initialized) matrix A whose rows and columns correspond to the vertices of G such that $A[x, y]$ contains the Θ-class of the edge xy when x and y are adjacent and the Θ-class of xy has already been computed and $A[x, y]$ is undefined if x and y are not adjacent or if the Θ-class of xy has not been computed yet. For (ii), pick any $x \in \Lambda_<(v), x \neq u$. By Theorem 13, $uv = f(v)v$ and $f(x)x$ are opposite edges of a square. Since $f(x)x$ appears before uv in the LexBFS order, the Θ-class of $f(x)x$ has already been computed, and the algorithm inserts uv in the Θ-class of $f(x)x$. Each Θ-class E_i is totally ordered by the order in which the edges are inserted in E_i. Consequently, we obtain:

▶ **Theorem 16.** *The Θ-classes of a median graph G can be computed in $O(m)$ time.*

5 The median of G

We use Theorem 16 to compute the median set $\text{Med}_w(G)$ of a median graph G in $O(m)$ time. We also use the existence of peripheral halfspaces and the majority rule.

5.1 Peripheral peeling

The order $E_1, E_2, \ldots, E_q$ in which the Θ-classes E_i of G are constructed corresponds to the order of the distances from v_0 to H'_i: if $i < j$ then $d(v_0, H'_i) \leq d(v_0, H'_j)$ (recall that $v_0 \in H''_i$). By Lemma 7, the halfspace H'_q of E_q is peripheral. If we contract all edges of E_q (i.e., we identify the vertices of $H'_q = \partial H'_q$ with their neighbors in $\partial H''_q$) we get a smaller median graph $\widetilde{G} = H''_q$; $\widetilde{G}$ has $q-1$ Θ-classes $\widetilde{E}_1, \ldots, \widetilde{E}_{q-1}$, where $\widetilde{E}_i$ consists of the edges of E_i in $\widetilde{G}$. The halfspaces of $\widetilde{G}$ have the form $\widetilde{H}'_i = H'_i \cap H''_q$ and $\widetilde{H}''_i = H''_i \cap H''_q$. Then $\widetilde{E}_1, \ldots, \widetilde{E}_{q-1}$ corresponds to the ordering of the halfspaces $\widetilde{H}'_1, \ldots, \widetilde{H}'_{q-1}$ of $\widetilde{G}$ by their distances to v_0. Hence the last halfspace $\widetilde{H}'_{q-1}$ is peripheral in $\widetilde{G}$. Thus the ordering $E_q, E_{q-1}, \ldots, E_1$ of the Θ-classes of G provides us with a set $G_q = G, G_{q-1} = \widetilde{G}, \dots, G_0$ of median graphs such that G_0 is a single vertex and for each $i \geq 1$, the Θ-class E_i defines a peripheral halfspace in the graph G_i obtained after the successive contractions of the peripheral halfspaces of $G_q, G_{q-1}, \dots, G_{i+1}$ defined by $E_q, E_{q-1}, \dots, E_{i+1}$. We call $G_q, G_{q-1}, \dots, G_0$ a *peripheral peeling* of G. Since each vertex of G and each Θ-class is contracted only once, we do not need to explicitly compute the restriction of each Θ-class of G to each G_i. For this it is enough to keep for each vertex v a variable indicating whether this vertex belongs to an already contracted peripheral halfspace or not. Hence, when the ith Θ-class must be contracted, we simply traverse the edges of E_i and select those edges whose both ends are not yet contracted.

5.2 Computing the weights of the halfspaces of G

We use a peripheral peeling $G_q, G_{q-1}, \dots, G_0$ of G to compute the weights $w(H'_i)$ and $w(H''_i)$, $i = 1, \dots, q$ of all halfspaces of G. As above, let $\widetilde{G}$ be obtained from G by contracting the Θ-class E_q. Consider the weight function $\widetilde{w}$ on $\widetilde{G} = H''_q$ defined as follows:

$$\widetilde{w}(v'') = \begin{cases} w(v'') + w(v') & \text{if } v'' \in \partial H''_q, v' \in H'_q, \text{ and } v'' \sim v', \\ w(v'') & \text{if } v'' \in H''_q \setminus \partial H''_q. \end{cases} \tag{5.1}$$

▶ **Lemma 17.** *For any Θ-class $\widetilde{E}_i$ of $\widetilde{G}$, $\widetilde{w}(\widetilde{H}'_i) = w(H'_i)$ and $\widetilde{w}(H''_i) = w(H''_i)$.*

By Lemma 17, to compute all $w(H'_i)$ and $w(H''_i)$, it suffices to compute the weight of the peripheral halfspace of E_i in the graph G_i, set it as $w(H'_i)$, and set $w(H''_i) := w(G) - w(H'_i)$.

Let G be the current median graph, let H'_q be a peripheral halfspace of G, and $\widetilde{G} = H''_q$ be the graph obtained from G by contracting the edges of E_q. To compute $w(H'_q)$, we traverse the vertices of H'_q (by considering the edges of E_q). Set $w(H''_q) = w(G) - w(H'_q)$. Let $\widetilde{w}$ be the weight function on $\widetilde{G}$ defined by Equation 5.1. Clearly, $\widetilde{w}$ can be computed in $O(|V(H'_q)|) = O(|E_q|)$ time. Then by Lemma 17 it suffices to recursively apply the algorithm to the graph $\widetilde{G}$ and the weight function $\widetilde{w}$. Since each edge of G is considered only when its Θ-class is contracted, the algorithm has complexity $O(m)$.

5.3 The median $\mathrm{Med}_w(G)$

We start with a simple property of the median function F_w that follows from Lemma 4:

▶ **Lemma 18.** *If $xy \in E_i$ with $x \in H'_i$ and $y \in H''_i$, then $F_w(x) - F_w(y) = w(H''_i) - w(H'_i)$.*

A halfspace H of G is *majoritary* if $w(H) > \frac{1}{2}w(G)$, *minoritary* if $w(H) < \frac{1}{2}w(G)$, and *egalitarian* if $w(H) = \frac{1}{2}w(G)$. Let $\mathrm{Med}^{\mathrm{loc}}_w(G) = \{v \in V : F_w(v) \leq F_w(u), \forall u \sim v\}$ be the set of local medians of G. We continue with the majority rule:

▶ **Proposition 19.** [8, 61]. *$\mathrm{Med}_w(G)$ is the intersection of all majoritary halfspaces and $\mathrm{Med}_w(G)$ intersects all egalitarian halfspaces. If H'_i and H''_i are egalitarian halfspaces, then $\mathrm{Med}_w(G)$ intersects both H'_i and H''_i. Moreover, $\mathrm{Med}_w(G) = \mathrm{Med}^{\mathrm{loc}}_w(G)$.*

We use Proposition 19 and the weights of halfspaces computed above to derive $\mathrm{Med}_w(G)$. For this, we define a new orientation of the edges $v'v''$ of each Θ-class E_i of G as follows. If $v' \in H'_i$ and $v'' \in H''_i$, then we direct $v'v''$ from v' to v'' if $w(H''_i) > w(H'_i)$ and from v'' to v' if $w(H'_i) > w(H''_i)$. If $w(H'_i) = w(H''_i)$, then the edge $v'v''$ is not directed. We denote this partially directed graph by $\overrightarrow{G}$. A vertex u of G is a *sink* of $\overrightarrow{G}$ if there is no edge uv directed in $\overrightarrow{G}$ from u to v. From Lemma 18, u is a sink of $\overrightarrow{G}$ if and only if u is a local median of G.

By Proposition 19, $\mathrm{Med}^{\mathrm{loc}}_w(G) = \mathrm{Med}_w(G)$ and thus $\mathrm{Med}_w(G)$ coincides with the set $S(\overrightarrow{G})$ of sinks of $\overrightarrow{G}$. Note that in the graph induced by $\mathrm{Med}_w(G)$, all edges are non-oriented in $\overrightarrow{G}$. Once all $w(H'_i)$ and $w(H''_i)$ have been computed, the orientation $\overrightarrow{G}$ of G can be constructed in $O(m)$ by traversing all Θ-classes E_i of G. The graph induced by $S(\overrightarrow{G})$ can then be found in $O(m)$.

▶ **Theorem 20.** *The median* $\mathrm{Med}_w(G)$ *of a median graph* G *can be computed in* $O(m)$ *time.*

▶ **Corollary 21.** *If* $w(G) > 0$*, we can find* $u, v \in V(G)$ *in* $O(m)$ *such that* $\mathrm{Med}_w(G) = I(u, v)$*.*

6 The median problem in the cube complex of G

We describe a linear time algorithm to compute medians in cube complexes of median graphs.

6.1 The main result

The problem. Let $\mathcal{G}$ be the *cube complex* of a median graph G obtained by replacing each graphic cube of G by a unit solid cube and by isometrically identifying common subcubes. We refer to $\mathcal{G}$ as to the *geometric realization* of G (see Fig. 4(a)). We suppose that $\mathcal{G}$ is endowed with the intrinsic ℓ_1-metric d_1. Let P be a finite set of points of $(\mathcal{G}, d_1)$ (called *terminals*) and let w be a weight function on $\mathcal{G}$ such that $w(p) > 0$ if $p \in P$ and $w(p) = 0$ if $p \notin P$. The goal of the *median problem* is to compute the set $\mathrm{Med}_w(\mathcal{G})$ of median points of $\mathcal{G}$, i.e., the set of all points $x \in \mathcal{G}$ minimizing the function $F_w(x) = \sum_{p\in\mathcal{G}} w(p)d_1(x, p) = \sum_{p\in P} w(p)d_1(x, p)$.

The input. The cube complex $\mathcal{G}$ is given by its 1-skeleton G. Each terminal $p \in P$ is given by its coordinates in the smallest cube $Q(p)$ of $\mathcal{G}$ containing p. Namely, we give a vertex $v(p)$ of $Q(p)$ together with its neighbors in $Q(p)$ and the coordinates of p in the embedding of $Q(p)$ as a unit cube in which $v(p)$ is the origin of coordinates. Let δ be the sum of the sizes of the encodings of the points of P. Thus the input of the median problem has size $O(m + \delta)$.

The output. Unlike $\mathrm{Med}_w(G)$ (which is a gated subgraph of G), $\mathrm{Med}_w(\mathcal{G})$ is not a subcomplex of $\mathcal{G}$. Nevertheless we show that $\mathrm{Med}_w(\mathcal{G})$ is a subcomplex of the box complex $\widehat{\mathcal{G}}$ obtained by subdividing $\mathcal{G}$, using the hyperplanes passing via the terminals of P. The output is the 1-skeleton $\widehat{M}$ of $\mathrm{Med}_w(\widehat{G})$ and $\mathrm{Med}_w(\widehat{\mathcal{G}})$, and the local coordinates of the vertices of $\widehat{M}$ in $\mathcal{G}$. We show that the output has linear size $O(m)$.

▶ **Theorem 22.** *The 1-skeleton* $\widehat{M}$ *of* $\mathrm{Med}_w(\mathcal{G})$ *can be computed in linear time* $O(m + \delta)$*.*

6.2 Geometric halfspaces and hyperplanes

In the following, we fix a basepoint v_0 of G. For each point x of $\mathcal{G}$, let $Q(x)$ be the smallest cube of $\mathcal{G}$ containing x and let $v(x)$ be the gate of v_0 in $Q(x)$. For each Θ-class E_i defining a dimension of $Q(x)$, let $\epsilon_i(x)$ be the coordinate of x along E_i in the embedding of $Q(x)$ as a unit cube in which $v(x)$ is the origin. For a Θ-class E_i and a cube Q having E_i as a dimension, the *i-midcube* of Q is the subspace of Q obtained by restricting the E_i-coordinate of Q to $\frac{1}{2}$. A *midhyperplane* $\mathfrak{h}_i$ of $\mathcal{G}$ is the union of all i-midcubes. Each $\mathfrak{h}_i$ cuts $\mathcal{G}$ in two components [58] and the union of each of these components with $\mathfrak{h}_i$ is called a *geometric halfspace* (see Fig. 4(b)). The *carrier* $\mathcal{N}_i$ of E_i is the union of all cubes of $\mathcal{G}$ intersecting $\mathfrak{h}_i$; $\mathcal{N}_i$ is isomorphic to $\mathfrak{h}_i \times [0, 1]$. For a Θ-class E_i and $0 < \epsilon < 1$, the *hyperplane* $\mathfrak{h}_i(\epsilon)$ is the set of all points $x \in \mathcal{N}_i$ such that $\epsilon_i(x) = \epsilon$. Let $\mathfrak{h}_i(0)$ and $\mathfrak{h}_i(1)$ be the respective geometric

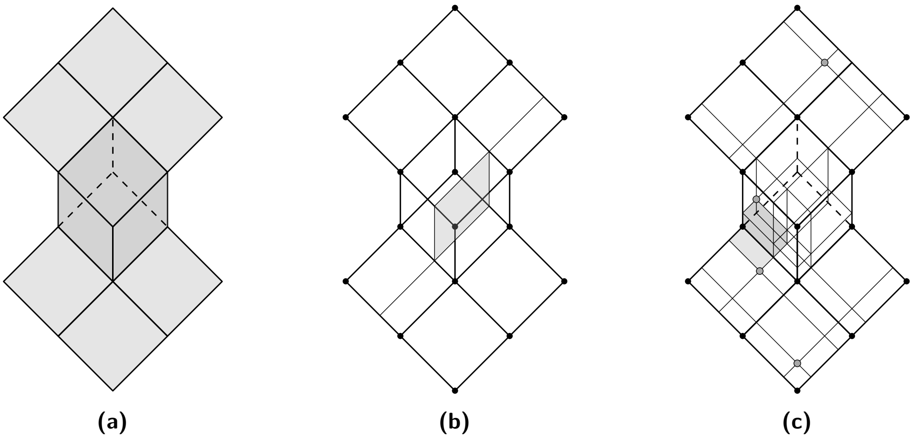


Figure 4 (a) The cube complex $\mathcal{D}$ of D, (b) a hyperplane of $\mathcal{D}$, and (c) the box complex $\widehat{\mathcal{D}}$ and $\mathrm{Med}_w(\mathcal{D})$ (in gray) defined by 4 terminals of weight 1.

realizations of $\partial H''_i$ and $\partial H'_i$. Note that $\mathfrak{h}_i(\epsilon)$ is obtained from $\mathfrak{h}_i$ by a translation. The *open carrier* $\mathcal{N}^\circ_i$ is $\mathcal{N}_i \setminus (\mathfrak{h}_i(0) \cup \mathfrak{h}_i(1))$. We denote by $\mathcal{H}'_i(\epsilon)$ and $\mathcal{H}''_i(\epsilon)$ the geometric halfspaces of $\mathcal{G}$ defined by $\mathfrak{h}_i(\epsilon)$. Let $\mathcal{H}''_i := \mathcal{H}''_i(0)$ and $\mathcal{H}'_i := \mathcal{H}'_i(1)$; they are the geometric realizations of H'_i and H''_i. Note that $\mathcal{G}$ is the disjoint union of $\mathcal{H}'_i$, $\mathcal{H}''_i$, and $\mathcal{N}^\circ_i$.

6.3 The majority rule for $\mathcal{G}$

The box complex $\widehat{\mathcal{G}}$. By [65, Theorem 3.16], $(\mathcal{G}, d_1)$ is a median metric space (i.e., $|I(x,y) \cap I(y,z) \cap I(z,x)| = 1$ $\forall x,y,z \in \mathcal{G}$). For each $p \in P$ and each coordinate $\epsilon_i(p)$, consider the hyperplane $\mathfrak{h}_i(\epsilon_i(p))$. All such hyperplanes subdivide $\mathcal{G}$ into a box complex $\widehat{\mathcal{G}}$ (see Fig. 4(c)). Clearly, $(\widehat{\mathcal{G}}, d_1)$ is a median space. By [65, Theorem 3.13], the 1-skeleton $\widehat{G}$ of $\widehat{\mathcal{G}}$ is a median graph and each point of P corresponds to a vertex of $\widehat{G}$. The Θ-classes of $\widehat{G}$ are subdivisions of the Θ-classes of G. In $\widehat{\mathcal{G}}$, all edges of a Θ-class of $\widehat{G}$ have the same length. Let $\widehat{G}_l$ be the graph $\widehat{G}$ in which the edges have these lengths. $\widehat{G}_l$ is a median space, thus $\mathrm{Med}_w(\widehat{G}_l) = \mathrm{Med}_w(\widehat{G})$ by [61]. By Proposition 19, $\mathrm{Med}_w(\widehat{G}_l)$ is the intersection of the majoritary halfspaces of $\widehat{G}$.

▶ **Proposition 23.** *$\mathrm{Med}_w(\mathcal{G})$ is the subcomplex of $\widehat{\mathcal{G}}$ defined by $\widehat{M} := \mathrm{Med}_w(\widehat{G}_l)$.*

The E_i-median problems. We adapt now Proposition 19 to the continuous setting. For a Θ-class of G, the *E_i-median* is the median of the multiset of points of the segment $[0,1]$ weighted as follows: the weight $w_i(0)$ of 0 is $w(\mathcal{H}''_i)$, the weight $w_i(1)$ of 1 is $w(\mathcal{H}'_i)$, and for each $p \in P \cap \mathcal{N}^\circ_i$, there is a point $\epsilon_i(p)$ of $[0,1]$ of weight $w_i(\epsilon_i(p)) = w(p)$. It is well-known that this median is a segment $[\varrho''_i, \varrho'_i]$ defined by two consecutive points $\varrho''_i \le \varrho'_i$ of $[0,1]$ with positive weights, and for any $p \in P$, $\epsilon_i(p) \le \varrho''_i$ or $\epsilon_i(p) \ge \varrho'_i$. *Majoritary*, *minoritary*, and *egalitarian* geometric halfspaces of $\mathcal{G}$ are defined in the same way as the halfspaces of G.

▶ **Proposition 24.** *Let E_i be a Θ-class of G. Then the following holds:*

1. *$\mathrm{Med}_w(\mathcal{G}) \subseteq \mathcal{H}''_i$ (resp., $\mathrm{Med}_w(\mathcal{G}) \subseteq \mathcal{H}'_i$) if and only if $\mathcal{H}''_i$ is majoritary (resp., $\mathcal{H}'_i$ is majoritary), i.e., $\rho''_i = \rho'_i = 0$ (resp. $\rho''_i = \rho'_i = 1$);*
2. *$\mathrm{Med}_w(\mathcal{G}) \subseteq \mathcal{H}''_i \cup \mathcal{N}^\circ_i$ (resp., $\mathrm{Med}_w(\mathcal{G}) \subseteq \mathcal{H}'_i \cup \mathcal{N}^\circ_i$) and $\mathrm{Med}_w(\mathcal{G})$ intersects each of the sets $\mathcal{H}''_i$ (resp., $\mathcal{H}'_i$) and $\mathcal{N}^\circ_i$ if and only if $\mathcal{H}''_i$ (resp. $\mathcal{H}'_i$) is egalitarian and $\mathcal{H}'_i$ (resp., $\mathcal{H}''_i$) is minoritary, i.e., $0 = \rho''_i < \rho'_i < 1$ (resp. $0 < \rho''_i < \rho'_i = 1$);*
3. *$\mathrm{Med}_w(\mathcal{G}) \subseteq \mathcal{N}^\circ_i$ if and only if $\mathcal{H}'_i$ and $\mathcal{H}''_i$ are minoritary, i.e., $0 < \rho''_i \le \rho'_i < 1$;*
4. *$\mathrm{Med}_w(\mathcal{G})$ intersects the three sets $\mathcal{H}_i, \mathcal{H}''_i$, and $\mathcal{N}^\circ_i$ if and only if $\mathcal{H}'_i$ and $\mathcal{H}''_i$ are egalitarian, i.e., $0 = \rho''_i \le \rho'_i = 1$ (and thus $w(\mathcal{N}^\circ_i) = 0$).*

6.4 The algorithm

Preprocessing the input. We first compute the Θ-classes $E_1, E_2, \ldots, E_q$ of G ordered by increasing distance from v_0 to H_i'. Using this, we can modify the input of the median problem in linear time $O(m+\delta)$ in such a way that for each terminal $p \in P$, $v(p)$ is the gate of v_0 in $Q(p)$. In this way, the local coordinates of the terminals of P coincide with the coordinates $\epsilon_i(p)$ defined in Section 6.2. For each Θ-class E_i, let $P_i = P \cap \mathcal{N}_i^\circ = \{p \in P : 0 < \epsilon_i(p) < 1\}$, and for each point $v \in V(G)$, let $P_v = \{p \in P : v(p) = v\}$. By traversing the points of P, we can compute all sets P_i, $1 \leq i \leq q$ and P_v, $v \in V$ and the weights of these sets in time $O(\delta)$.

Computing the E_i-medians. We first compute the weights $w_i(0) = w(\mathcal{H}_i'')$ and $w_i(1) = w(\mathcal{H}_i')$ of the geometric halfspaces $\mathcal{H}_i'', \mathcal{H}_i'$ of G. For each vertex v of G, let $w_*(v) = w(P_v)$. Note that $w_*(V) = w(P)$. Since $v_0 \in H_i''$, $w_*(H_i') = w(\mathcal{H}_i')$ and $w_*(H_i'') = w(\mathcal{H}_i'') + w(\mathcal{N}_i^o)$ for each Θ-class E_i. We apply the algorithm of Section 5.2 to G with the weight function w_* to compute the weights $w_*(H_i')$ and $w_*(H_i'')$ of all halfspaces of G. Since $w(\mathcal{N}_i^o) = w(P_i)$ is known, we can compute $w(\mathcal{H}_i') = w_*(H_i')$ and $w(\mathcal{H}_i'') = w_*(H_i'') - w(P_i)$. This allows us to complete the definition of each E_i-median problem which altogether can be solved linearly in the size of the input [27, Problem 9.2], i.e., in time $O(\Sigma_{i=1}^q(|P_i|+2)) = O(\delta+m)$.

Computing $\widehat{M}$. To compute the 1-skeleton $\widehat{M}$ of $\mathrm{Med}_w(\mathcal{G})$ in $\widehat{G}$, we orient the edges of E_i according to the weights of $\mathcal{H}_i'$ and $\mathcal{H}_i''$: $v'v'' \in E_i$ with $v' \in \mathcal{H}_i'$ and $v'' \in \mathcal{H}_i''$ is directed from v'' to v' if $\varrho_i' = \varrho_i'' = 1$ ($\mathcal{H}_i'$ is majoritary) and from v' to v'' if $\varrho_i' = \varrho_i'' = 0$ ($\mathcal{H}_i''$ is majoritary), otherwise the edges of E_i are not oriented. Denote this partially directed graph by $\overrightarrow{G}$ and let $S(\overrightarrow{G})$ be the set of sinks of $\overrightarrow{G}$. A non-directed edge $v'v'' \in E_i$ defines a *half-edge with origin* v'' if $\varrho_i'' > 0$ and a *half-edge with origin* v' if $\varrho_i' < 1$ (an edge $v'v''$ such that $0 < \varrho_i'' \leq \varrho_i' < 1$ defines two half-edges).

▶ **Proposition 25.** *For any vertex v of $\overrightarrow{G}$, all half-edges with origin v define a cube Q_v of $\mathcal{G}$.*

Proof. For any vertex v and two Θ-classes E_i, E_j defining half-edges with origin v, let v_i and v_j be the respective neighbors of v in $\widehat{G}$ along the directions E_i and E_j. By Proposition 24, vv_i and vv_j point to two majoritary halfspaces of $\widehat{G}$ (and $\mathcal{G}$). Since those two halfspaces cannot be disjoint, E_i and E_j are crossing. The proposition then follows from Lemma 5. ◀

For any cube Q of $\mathcal{G}$, let $B(Q) \subseteq Q$ be the subcomplex of $\widehat{\mathcal{G}}$ that is the Cartesian product of the E_i-medians $[\varrho_i'', \varrho_i']$ over all Θ-classes E_i wich define dimensions of Q. By the definition of the E_i-medians, $B(Q)$ is a single box of $\widehat{\mathcal{G}}$ and its vertices belong to $\widehat{G}$.

▶ **Proposition 26.** *For any cube Q of $\mathcal{G}$, if $Q \cap \mathrm{Med}_w(\mathcal{G}) \neq \emptyset$, then $B(Q) = \mathrm{Med}_w(\mathcal{G}) \cap Q$.*

Proof. If a vertex x of $B(Q)$ is not a median of $\widehat{G}$, by Proposition 19, x is not a local median of $\widehat{G}$. Thus $F_w(x) > F_w(y)$ for an edge xy of $\widehat{G}$. Suppose that xy is parallel to the edges of E_i of G. Then $\epsilon_i(x)$ coincides with ϱ_i'' or ϱ_i'. Since $F_w(x) > F_w(y)$, the halfspace $W(y,x)$ of $\widehat{G}$ is majoritary, contrary to the assumption that $\epsilon_i(x)$ is an E_i-median point. Thus all vertices of $B(Q)$ belong to $\widehat{M}$ and by Proposition 23, $B(Q) \subseteq \mathrm{Med}_w(\mathcal{G})$. It remains to show that any point of $Q \setminus B(Q)$ is not median. Otherwise, by Proposition 23 and since $\widehat{M}$ is convex, there exists a vertex $y \notin B(Q)$ of $(\widehat{M} \cap Q) \setminus B(Q)$ adjacent to a vertex x of $B(Q)$. Let xy be parallel to E_i. Then $\epsilon_i(x)$ coincides with ϱ_i'' or ϱ_i' and $\epsilon_i(y)$ does not belong to the E_i-median $[\varrho_i'', \varrho_i']$. Hence the halfspace $W(y,x)$ of $\widehat{G}$ is minoritary, contrary to $F_w(y) = F_w(x)$. ◀

For a sink v of $\overrightarrow{G}$, let $g(v)$ be the point of Q_v such that for each Θ-class E_i of Q_v, $\epsilon_i(g(v)) = \varrho'$ if $v \in \mathcal{H}'_i$ and $\epsilon_i(g(v)) = \varrho''$ if $v \in \mathcal{H}''_i$. Note that $g(v)$ is the gate of v in $B(Q_v)$ and $g(v)$ is a vertex of $\widehat{M}$. Conversely, let $x \in \widehat{M}$ and consider the cube $Q(x)$. Since $B(Q(x))$ is a cell of $\widehat{\mathcal{G}}$, for each Θ-class E_i of $Q(x)$, we have $\epsilon_i(x) \in \{\varrho'_i, \varrho''_i\}$. Let $f(x)$ be the vertex of $Q(x)$ such that $f(x) \in \mathcal{H}''_i$ if $\epsilon_i(x) = \varrho''_i$ and $f(x) \in \mathcal{H}'_i$ otherwise.

▶ **Proposition 27.** *For any $v \in S(\overrightarrow{G})$, $g(v)$ is the gate of v in $\widehat{M}$ and* $\mathrm{Med}_w(\mathcal{G})$*. For any $x \in \widehat{M}$, $x = g(f(x))$ is the gate of $f(x)$ in $\widehat{M}$ and* $\mathrm{Med}_w(\mathcal{G})$.

Furthermore, for any edge uv of G with $u, v \in S(\overrightarrow{G})$, either $g(u) = g(v)$ or $g(u)g(v)$ is an edge of $\widehat{M}$. Conversely, for any edge xy of $\widehat{M}$, $f(x)f(y)$ is an edge of G.

Proof. By Proposition 24 applied to $\mathcal{G}$, Proposition 19 applied to $\widehat{G}$, and the definition of sinks of $\overrightarrow{\widehat{G}}$, $g(v)$ is a sink of $\overrightarrow{\widehat{G}}$, thus $g(v)$ is a median of $\widehat{G}$ and $\mathcal{G}$. Since $B(Q_v) = \mathrm{Med}_w(\mathcal{G}) \cap Q_v$ is gated and non-empty, the gate of v in $\mathrm{Med}_w(\mathcal{G})$ belongs to $B(Q_v)$ and thus the gate of v in $\mathrm{Med}_w(\mathcal{G})$ is the gate of v on $B(Q_v)$. Conversely, $\epsilon_i(x) \notin \{0, 1\}$ for any E_i defining a dimension of $Q(x)$, thus there is an E_i-half-edge with origin $f(x)$. Pick now any E_j-edge incident to v such that E_j does not define a dimension of $Q(x)$. Without loss of generality, assume that $f(x) \in \mathcal{H}'_j$. Then $x \in \mathcal{H}'_j$, yielding $w(\mathcal{H}'_j) \geq \frac{1}{2}w(P)$. By Proposition 24, $\varrho'_j = 1$ and thus $f(x)$ is not the origin of an E_j-edge or E_j-half-edge. Consequently, $Q_{f(x)} = Q(x)$ by Proposition 25 and by the definition of $f(x)$ and $g(f(x))$, we have $x = g(f(x))$.

Let $v'v''$ be an E_i-edge between two sinks of $\overrightarrow{G}$ with $v' \in \mathcal{H}'_i$ and $v'' \in \mathcal{H}''_i$. Let $x' = g(v')$ and $x'' = g(v'')$ and assume that $x' \neq x''$. Let u', u'' be the points of $v'v''$ such that $\epsilon_i(u') = \varrho'_i$ and $\epsilon_i(u'') = \varrho''_i$. Note that u' and u'' are adjacent vertices of $\widehat{G}$ and that $u' \in I_{\widehat{G}}(v', x')$ and $u'' \in I_{\widehat{G}}(v'', x'')$. In $\widehat{G}$, x'' is the gate of u'' (and x' is the gate of u') in $\widehat{M}$. Since $d_{\widehat{G}}(u', x') + d_{\widehat{G}}(x', x'') = d_{\widehat{G}}(u', x'') \leq d_{\widehat{G}}(u'', x'') + 1$ and $d_{\widehat{G}}(u'', x'') + d_{\widehat{G}}(x', x'') = d_{\widehat{G}}(u'', x') \leq d_{\widehat{G}}(u', x') + 1$, we obtain that $d_{\widehat{G}}(x', x'') \leq 1$.

Any edge $x'x''$ of $\widehat{M}$ is parallel to a Θ-class E_i of G. For any Θ-class E_j of $Q(x')$ (resp. $Q(x'')$) with $j \neq i$, E_j is a Θ-class of $Q(x'')$ (resp. $Q(x')$) and $\epsilon_j(x') = \epsilon_j(x'')$. By their definition, $f(x')$ and $f(x'')$ can be separated only by E_i, i.e., $d_G(f(x'), f(x'')) \leq 1$. Since f is an injection from $V(\widehat{M})$ to $S(\overrightarrow{G})$, necessarily $f(x')$ and $f(x'')$ are adjacent. ◀

The algorithm computes the set $S(\overrightarrow{G})$ of all sinks of $\overrightarrow{G}$ and for each sink $v \in S(\overrightarrow{G})$, it computes the gate of $g(v)$ of v in $\widehat{M}$ and the local coordinates of $g(v)$ in $\mathcal{G}$. The algorithm returns $\left\{g(v) : v \in S(\overrightarrow{G})\right\}$ as $V(\widehat{M})$ and $\left\{g(u)g(v) : uv \in E \text{ and } u, v \in S(\overrightarrow{G})\right\}$ as $E(\widehat{M})$. Proposition 27 implies that $V(\widehat{M})$ and $E(\widehat{M})$ are correctly computed and that $\widehat{M}$ contains at most n vertices and m edges. Moreover each vertex x of $\widehat{M}$ is the gate $g(f(x))$ of the vertex $f(x)$ of $Q(x)$ that has dimension at most $\deg(f(x))$. Hence the size of the description of the vertices of $\widehat{M}$ is at most $O(m)$. This finishes the proof of Theorem 22.

References

1 A. Abboud, F. Grandoni, and V. Vassilevska Williams. Subcubic equivalences between graph centrality problems, APSP and diameter. In *SODA 2015*, pages 1681–1697. SIAM, 2015. doi:10.1137/1.9781611973730.112.

2 A. Abboud, V. Vassilevska Williams, and J. R. Wang. Approximation and fixed parameter subquadratic algorithms for radius and diameter in sparse graphs. In *SODA 2016*, pages 377–391. SIAM, 2016. doi:10.1137/1.9781611974331.ch28.

3 F. Ardila, M. Owen, and S. Sullivant. Geodesics in CAT(0) cubical complexes. *Adv. in Appl. Math.*, 48(1):142–163, 2012. doi:10.1016/j.aam.2011.06.004.

4 S. P. Avann. Metric ternary distributive semi-lattices. *Proc. Amer. Math. Soc.*, 12(3):407–414, 1961.

5 M. Bacák. Computing medians and means in Hadamard spaces. *SIAM Journal on Optimization*, 24(3):1542–1566, 2014. doi:10.1137/140953393.

6 C. Bajaj. The algebraic degree of geometric optimization problems. *Discrete Comput. Geom.*, 3(2):177–191, 1988. doi:10.1007/BF02187906.

7 K. Balakrishnan, B. Brešar, M. Changat, S. Klavžar, M. Kovše, and A. R. Subhamathi. Computing median and antimedian sets in median graphs. *Algorithmica*, 57(2):207–216, 2010. doi:10.1007/s00453-008-9200-4.

8 H.-J. Bandelt and J.-P. Barthélémy. Medians in median graphs. *Discrete Appl. Math.*, 8(2):131–142, 1984. doi:10.1016/0166-218X(84)90096-9.

9 H.-J. Bandelt and V. Chepoi. Graphs with connected medians. *SIAM J. Discrete Math.*, 15(2):268–282, 2002. doi:10.1137/S089548019936360X.

10 H.-J. Bandelt and V. Chepoi. Metric graph theory and geometry: a survey. In J. E. Goodman, J. Pach, and R. Pollack, editors, *Surveys on Discrete and Computational Geometry: Twenty Years Later*, volume 453 of *Contemp. Math.*, pages 49–86. Amer. Math. Soc., Providence, RI, 2008. doi:10.1090/conm/453/08795.

11 H.-J. Bandelt, V. Chepoi, A. W. M. Dress, and J. H. Koolen. Combinatorics of lopsided sets. *European J. Combin.*, 27(5):669–689, 2006. doi:10.1016/j.ejc.2005.03.001.

12 H.-J. Bandelt, P. Forster, and A. Röhl. Median-joining networks for inferring intraspecific phylogenies. *Mol. Biol. Evol.*, 16(1):37–48, 1999. doi:10.1093/oxfordjournals.molbev.a026036.

13 J.-P. Barthélemy and J. Constantin. Median graphs, parallelism and posets. *Discrete Math.*, 111(1-3):49–63, 1993. Graph Theory and Combinatorics (Marseille-Luminy, 1990). doi:10.1016/0012-365X(93)90140-O.

14 A. Bavelas. Communication patterns in task-oriented groups. *J. Acoust. Soc. Am.*, 22(6):725–730, 1950. doi:10.1121/1.1906679SMASH.

15 M. A. Beauchamp. An improved index of centrality. *Behavorial Science*, 10:161–163, 1965.

16 L. Bénéteau, J. Chalopin, V. Chepoi, and Y. Vaxès. Medians in median graphs in linear time. *arXiv preprint*, 1907.10398, 2019. arXiv:1907.10398.

17 L. J. Billera, S. P. Holmes, and K. Vogtmann. Geometry of the space of phylogenetic trees. *Adv. in Appl. Math.*, 27(4):733–767, 2001. doi:10.1006/aama.2001.0759.

18 G. Birkhoff and S. A. Kiss. A ternary operation in distributive lattices. *Bull. Amer. Math. Soc.*, 53:749–752, 1947. doi:10.1090/S0002-9904-1947-08864-9.

19 B. Brešar, J. Chalopin, V. Chepoi, T. Gologranc, and D. Osajda. Bucolic complexes. *Adv. Math.*, 243:127–167, 2013. doi:10.1016/j.aim.2013.04.009.

20 J. Chalopin and V. Chepoi. A counterexample to Thiagarajan's conjecture on regular event structures. In *ICALP 2017*, volume 80 of *LIPIcs*, pages 101:1–101:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.101.

21 J. Chalopin and V. Chepoi. 1-safe Petri nets and special cube complexes: Equivalence and applications. *ACM Trans. Comput. Log.*, 20(3):17:1–17:49, 2019. doi:10.1145/3322095.

22 J. Chalopin, V. Chepoi, H. Hirai, and D. Osajda. Weakly modular graphs and nonpositive curvature. *Mem. Amer. Math. Soc.*, 2017. To appear.

23 V. Chepoi. On distance-preserving and domination elimination orderings. *SIAM J. Discrete Math.*, 11(3):414–436, 1998. doi:10.1137/S0895480195291230.

24 V. Chepoi. Graphs of some CAT(0) complexes. *Adv. in Appl. Math.*, 24(2):125–179, 2000. doi:10.1006/aama.1999.0677.

25 V. Chepoi. Nice labeling problem for event structures: a counterexample. *SIAM J. Comput.*, 41(4):715–727, 2012. doi:10.1137/110837760.

26 M. B. Cohen, Y. T. Lee, G. L. Miller, J. Pachocki, and A. Sidford. Geometric median in nearly linear time. In *STOC 2016*, pages 9–21. ACM, 2016. doi:10.1145/2897518.2897647.

27 T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press, 3rd edition, 2009.

28 D. G. Corneil. Lexicographic breadth first search – A survey. In *WG 2004*, volume 3353 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2004. doi:10.1007/978-3-540-30559-0_1.

29 D. Ž. Djoković. Distance-preserving subgraphs of hypercubes. *J. Combin. Theory Ser. B*, 14(3):263–267, 1973. doi:10.1016/0095-8956(73)90010-5.

30 David Eppstein. Recognizing partial cubes in quadratic time. *J. Graph Algorithms Appl.*, 15(2):269–293, 2011. doi:10.7155/jgaa.00226.

31 D. B. A. Epstein, J. W. Cannon, D. F. Holt, S. V. F. Levy, M. S. Paterson, and W. P. Thurston. *Word Processing in Groups.* Jones and Bartlett, Boston, MA, 1992.

32 A. J. Goldman and C. J. Witzgall. A localization theorem for optimal facility placement. *Transp. Sci.*, 4(4):406–409, 1970. doi:10.1287/trsc.4.4.406.

33 M. Gromov. Hyperbolic groups. In S. M. Gersten, editor, *Essays in Group Theory*, volume 8 of *Math. Sci. Res. Inst. Publ.*, pages 75–263. Springer, New York, 1987. doi:10.1007/978-1-4613-9586-7_3.

34 J. Hagauer, W. Imrich, and S. Klavžar. Recognizing median graphs in subquadratic time. *Theoret. Comput. Sci.*, 215(1-2):123–136, 1999. doi:10.1016/S0304-3975(97)00136-9.

35 S. L. Hakimi. Optimum locations of switching centers and the absolute centers and medians of a graph. *Oper. Res.*, 12(3):450–459, 1964. doi:10.1287/opre.12.3.450.

36 R. Hammack, W. Imrich, and S. Klavšar. *Handbook of Product Graphs.* Discrete Math. Appl. CRC press, Boca Raton, 2nd edition, 2011. doi:10.1201/b10959.

37 K. Hayashi. A polynomial time algorithm to compute geodesics in CAT(0) cubical complexes. *Discrete Comput. Geom.*, (to appear), 2019. doi:10.1007/s00454-019-00154-2.

38 C. Jordan. Sur les assemblages de lignes. *J. Reine Angew. Math.*, 70:185–190, 1869. doi:10.1515/crll.1869.70.185.

39 J. G. Kemeny. Mathematics without numbers. *Daedalus*, 88(4):577–591, 1959.

40 J. G. Kemeny and J. L. Snell. *Mathematical models in the social sciences.* MIT Press, Cambridge, Mass.-London, 1978. Reprint of the 1962 original.

41 S. Klavžar and H. M. Mulder. Median graphs: Characterizations, location theory and related structures. *J. Combin. Math. Combin. Comput.*, 30:103–127, 1999.

42 D. E. Knuth. *The Art of Computer Programming, Volume 4A, Fascicle 0: Introduction to Combinatorial Algorithms and Boolean Functions.* Addison-Wesley, 2008.

43 R. F. Love, J. G. Morris, and G. O. Wesolowsky. *Facilities location: Models and methods*, volume 7 of *Publ. Oper. Res. Ser.* North Holland, Amsterdam, 1988.

44 F. R. McMorris, H. M. Mulder, and F. S. Roberts. The median procedure on median graphs. *Discrete Appl. Math.*, 84(1-3):165–181, 1998. doi:10.1016/S0166-218X(98)00003-1.

45 H. M. Mulder. *The Interval Function of a Graph*, volume 132 of *Mathematical Centre tracts.* Mathematisch Centrum, Amsterdam, 1980.

46 H. M. Mulder. The expansion procedure for graphs. In *Contemporary methods in graph theory*, pages 459–477. Bibliographisches Inst., Mannheim, 1990.

47 H. M. Mulder and B. Novick. A tight axiomatization of the median procedure on median graphs. *Discrete Appl. Math.*, 161(6):838–846, 2013. doi:10.1016/j.dam.2012.10.027.

48 H. M. Mulder and A. Schrijver. Median graphs and Helly hypergraphs. *Discrete Math.*, 25(1):41–50, 1979. doi:10.1016/0012-365X(79)90151-1.

49 L. Nebeský. Median graphs. *Comment. Math. Univ. Carolinae*, 12:317–325, 1971.

50 M. Nielsen, G. D. Plotkin, and G. Winskel. Petri nets, event structures and domains, part I. *Theoret. Comput. Sci.*, 13(1):85–108, 1981. doi:10.1016/0304-3975(81)90112-2.

51 L. M. Ostresh. On the convergence of a class of iterative methods for solving the Weber location problem. *Oper. Res.*, 26(4):597–609, 1978. doi:10.1287/opre.26.4.597.

52 M. Owen and J. Scott Provan. A fast algorithm for computing geodesic distances in tree space. *IEEE/ACM Trans. Comput. Biology Bioinform.*, 8(1):2–13, 2011. doi:10.1109/TCBB.2010.3.

53 C. Puppe and A. Slinko. Condorcet domains, median graphs and the single-crossing property. *Econom. Theory*, 67(1):285–318, 2019. doi:10.1007/s00199-017-1084-6.

54 M. Roller. Poc sets, median algebras and group actions. Technical report, Univ. of Southampton, 1998.

55 D. J. Rose, R. E. Tarjan, and G. S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM J. Comput.*, 5(2):266–283, 1976. doi:10.1137/0205021.

56 B. Rozoy and P. S. Thiagarajan. Event structures and trace monoids. *Theoret. Comput. Sci.*, 91(2):285–313, 1991. doi:10.1016/0304-3975(91)90087-I.

57 G. Sabidussi. The centrality index of a graph. *Psychometrika*, 31(4):581–603, 1966. doi:10.1007/BF02289527.

58 M. Sageev. Ends of group pairs and non-positively curved cube complexes. *Proc. London Math. Soc.*, s3-71(3):585–617, 1995. doi:10.1112/plms/s3-71.3.585.

59 M. Sageev. CAT(0) cube complexes and groups. In M. Bestvina, M. Sageev, and K. Vogtmann, editors, *Geometric Group Theory*, volume 21 of *IAS/Park City Math. Ser.*, pages 6–53. Amer. Math. Soc., Inst. Adv. Study, 2012. doi:10.1090/pcms/021/02.

60 T. J. Schaefer. The complexity of satisfiability problems. In *STOC 1978*, pages 216–226. ACM, 1978. doi:10.1145/800133.804350.

61 P. S. Soltan and V. D. Chepoĭ. Solution of the Weber problem for discrete median metric spaces. *Trudy Tbiliss. Mat. Inst. Razmadze Akad. Nauk Gruzin. SSR*, 85:52–76, 1987.

62 B. C. Tansel, R. L. Francis, and T. J. Lowe. Location on networks: A survey. Part I: The p-center and p-median problems. *Manag. Sci.*, 29(4):482–497, 1983. doi:10.1287/mnsc.29.4.482.

63 P. S. Thiagarajan. Regular event structures and finite Petri nets: A conjecture. In *Formal and Natural Computing*, volume 2300 of *Lecture Notes in Comput. Sci.*, pages 244–256. Springer, 2002. doi:10.1007/3-540-45711-9_14.

64 P. S. Thiagarajan and S. Yang. Rabin's theorem in the concurrency setting: a conjecture. *Theoret. Comput. Sci.*, 546:225–236, 2014. doi:10.1016/j.tcs.2014.03.010.

65 M. van de Vel. *Theory of Convex Structures*, volume 50 of *North-Holland Math. Library*. North-Holland Publishing Co., 1993.

66 E. Weiszfeld. Sur le point pour lequel la somme des distances de n points donnés est minimum. *Tohoku Math. J.*, 43:355–386, 1937. (English transl.: *Ann. Oper. Res.*, 167:7–41, 2009).

67 D. R. Wood. On the maximum number of cliques in a graph. *Graphs Combin.*, 23(3):337–352, 2007. doi:10.1007/s00373-007-0738-8.

68 A. A. Zykov. On some properties of linear complexes. *Mat. Sb. (N.S.)*, 24(66)(2):163–188, 1949. (English transl.: *Amer. Math. Soc. Transl.* 79, 1952).

Graph Coloring via Degeneracy in Streaming and Other Space-Conscious Models

Suman K. Bera
University of California at Santa Cruz, CA, USA

Amit Chakrabarti
Dartmouth College, Hanover, NH, USA
http://www.cs.dartmouth.edu/~ac

Prantar Ghosh
Dartmouth College, Hanover, NH, USA

Abstract

We study the problem of coloring a given graph using a small number of colors in several well-established models of computation for big data. These include the data streaming model, the general graph query model, the massively parallel communication (MPC) model, and the CONGESTED-CLIQUE and the LOCAL models of distributed computation. On the one hand, we give algorithms with sublinear complexity, for the appropriate notion of complexity in each of these models. Our algorithms color a graph G using $\kappa(G) \cdot (1 + o(1))$ colors, where $\kappa(G)$ is the degeneracy of G: this parameter is closely related to the arboricity $\alpha(G)$. As a function of $\kappa(G)$ alone, our results are close to best possible, since the optimal number of colors is $\kappa(G) + 1$. For several classes of graphs, including real-world "big graphs," our results improve upon the number of colors used by the various $(\Delta(G) + 1)$-coloring algorithms known for these models, where $\Delta(G)$ is the maximum degree in G, since $\Delta(G) \geqslant \kappa(G)$ and can in fact be arbitrarily larger than $\kappa(G)$.

On the other hand, we establish certain lower bounds indicating that sublinear algorithms probably cannot go much further. In particular, we prove that any randomized coloring algorithm that uses at most $\kappa(G) + O(1)$ colors would require $\Omega(n^2)$ storage in the one pass streaming model, and $\Omega(n^2)$ many queries in the general graph query model, where n is the number of vertices in the graph. These lower bounds hold even when the value of $\kappa(G)$ is known in advance; at the same time, our upper bounds do not require $\kappa(G)$ to be given in advance.

2012 ACM Subject Classification Theory of computation → Streaming models; Theory of computation → Dynamic graph algorithms; Theory of computation → Sketching and sampling; Theory of computation → Lower bounds and information complexity

Keywords and phrases Data streaming, Graph coloring, Sublinear algorithms, Massively parallel communication, Distributed algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.11

Category Track A: Algorithms, Complexity and Games

Related Version A full version of the paper is available at https://arxiv.org/pdf/1905.00566.pdf.

Funding This work was supported in part by NSF under award CCF-190773 and by the Simons Institute for the Theory of Computing, which hosted the second author while the work was in progress.

Acknowledgements The authors gratefully acknowledge several helpful discussions with Sepehr Assadi (especially those that called to their attention a nuance with the Congested Clique algorithm) and Deeparnab Chakrabarty.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 11; pp. 11:1–11:21
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Graph coloring is a fundamental topic in combinatorics and the corresponding algorithmic problem of coloring an input graph with a small number of colors is a basic and heavily studied problem in computer science. It has numerous applications, e.g., in scheduling [67, 49, 48], air traffic flow management [15], frequency assignment in wireless networks [8, 56], and register allocation [21, 26, 22]. More recently, vertex coloring has been used to compute seed vertices in social networks that are then expanded to detect community structures in the network [54].

Given an n-vertex graph $G = (V, E)$, the task is to assign colors to the vertices in V so that no two adjacent vertices get the same color. Doing so with the minimum possible number of colors – called the chromatic number, $\chi(G)$ – is famously hard: it is NP-hard to even approximate $\chi(G)$ to a factor of $n^{1-\varepsilon}$ for any constant $\varepsilon > 0$ [31, 68, 45]. In the face of this hardness, it is algorithmically interesting to color G with a possibly suboptimal number of colors depending upon tractable parameters of G. One such simple parameter is Δ, the maximum degree: a trivial greedy algorithm colors G with $\Delta + 1$ colors in linear time.

We study graph coloring in a number of space-constrained and data-access-constrained settings, including the data streaming model, a query model, and certain distributed computing models. As expected, coloring using the optimal number of colors is hard in these models. Abboud et al. [1] show that coloring an n-vertex graph G with $\chi(G)$ colors in the p-pass streaming setting requires $\Omega(n^2/p)$ space, and checking c-colorability for $3 \leqslant c < n$ requires $\Omega((n-c)^2/p)$ space. In such constrained settings, even finding a coloring with "about Δ" colors is a fairly nontrivial problem that has been studied from various angles in a flurry of research over the last decade [5, 10, 23, 24, 40, 59]. In a recent breakthrough (awarded Best Paper at SODA 2019), Assadi, Chen, and Khanna [5] gave sublinear algorithms for $(\Delta + 1)$-coloring an input graph in such models.

In this work, we focus on colorings that use "about κ" colors, where $\kappa = \kappa(G)$ is the *degeneracy* of G, a parameter that improves upon Δ. It is defined as follows: $\kappa = \min\{k :$ every induced subgraph of G has a vertex of degree at most $k\}$. Every graph is $(\kappa + 1)$-colorable. Clearly, $\kappa \leqslant \Delta$ and can be much smaller than Δ for sparse graphs and real-world graphs; see Table 2. Thus, our aim is to use fewer colors when $(\Delta + 1)$-coloring might be wasteful. A closely related parameter is $\alpha(G)$, the arboricity of the graph, which is the minimum number of forests into which the set of edges of G can be partitioned. It is known that $\alpha \leqslant \kappa < 2\alpha$. A number of works has studied the $O(\alpha)$-coloring problem in distributed computing models [11, 12, 35, 36, 47]. However, to the best of our knowledge, such coloring algorithms were unknown in the data streaming and graph query models.

There is a simple greedy algorithm that runs in linear time and produces a $(\kappa+1)$-coloring; see Section 3. However, just as before, when processing a massive graph under the constraints of either the space-bounded streaming model or certain distributed computing models, the inherently sequential nature of the greedy algorithm makes it infeasible. We overcome this barrier with a very simple framework: decompose the graph into smaller subgraphs so as to store all the blocks in our limited memory, and then run the greedy algorithm on each block. We show that this basic framework (with careful implementation in the respective models) suffices for obtaining the colorings we seek.

On the other hand, we give a number of lower bounds showing that, despite its simplicity, our algorithmic framework does about as good a job as sublinear algorithms can. In particular, no randomized algorithm can achieve $(\kappa + O(1))$-colorings without spending $\Omega(n^2)$ space in the streaming model or $\Omega(n^2)$ queries in the query model.

1.1 Our Results and Techniques

Algorithms. We give new graph coloring algorithms, parametrized by the degeneracy κ, in the following models:

(1) the data streaming model, where the input is a stream of edge insertions and deletions (i.e., a dynamic graph stream) resulting in the eventual graph to be colored and we are limited to a work space of $\widetilde{O}(n)$ bits[1], the so-called semi-streaming setting [32];

(2) the general graph query model [38], where we may access the graph using only neighbor queries (what is the ith neighbor of x?) and pair queries (are x and y adjacent?);

(3) the massively parallel communication (MPC) model, where each of a large number of memory-limited processors holds a sublinear-sized portion of the input data and computation proceeds using rounds of communication;

(4) the congested clique model of distributed computation, where there is one processor per vertex holding that vertex's neighborhood information and each round allows each processor to communicate $O(\log n)$ bits to a specific other processor; and

(5) the LOCAL model of distributed computation, where there is one processor per vertex holding that vertex's neighborhood information and each round allows each processor to send an arbitrary amount of information to all its neighbors.

Table 1 Summary of our algorithmic results and basic comparison with most related previous work. In the result marked ($\star$), we require that $\kappa = \omega(\log^2 n)$.

Model	Colors	Complexity Parameters		Reference
Streaming	$\Delta + 1$	$\widetilde{O}(n)$ space,	$\widetilde{O}(n\sqrt{\Delta})$ post-processing time	[5]
(one pass)	$\kappa + o(\kappa)$	$\widetilde{O}(n)$ space,	$\widetilde{O}(n)$ post-processing time	this paper
Query	$\Delta + 1$		$\widetilde{O}(n^{3/2})$ queries	[5]
	$\kappa + o(\kappa)$		$\widetilde{O}(n^{3/2})$ queries	this paper
MPC	$\Delta + 1$	$O(1)$ rounds,	$O(n \log^3 n)$ bits per processor	[5]
	$\kappa + o(\kappa)$	$O(1)$ rounds,	$O(n \log^2 n)$ bits per processor	this paper
Congested	$\Delta + 1$		$O(1)$ rounds	[23]
Clique	$O(\kappa)$		$O(1)$ rounds	[36]
	$\kappa + o(\kappa)^\star$		$O(1)$ rounds	this paper
LOCAL	$O(\kappa n^{1/k})$	$O(k)$ rounds,	for $k \in \left[\omega(\log\log n),\, O(\sqrt{\log n})\right]$	[47]
	$O(\kappa n^{1/k} \log n)$	$O(k)$ rounds,	for $k \in \left[\omega(\sqrt{\log n}),\, o(\log n)\right]$	this paper

Table 1 summarizes our algorithmic results and provides, in each case, a basic comparison with the most related result from prior work. We give an elaborate account of the related works in Section 2.

As we have noted, $\kappa \leqslant \Delta$ in every case; indeed, κ could be arbitrarily better than Δ as shown by the example of a star graph, where $\kappa = 1$ whereas $\Delta = n - 1$. From a practical standpoint, it is notable that in many real-world large graphs drawn from various application domains – such as social networks, web graphs, and biological networks – the parameter κ is often *significantly* smaller than Δ. See Table 2 for some concrete numbers. That said, $\kappa + o(\kappa)$ is, in general, mathematically incomparable with $\Delta + 1$.

[1] The $\widetilde{O}(\cdot)$ notation hides factors polylogarithmic in n.

Table 2 Statistics of several large real-world graphs taken from the application domains of social networks, web graphs, and biological networks, showing that the degeneracy, κ, is often significantly smaller than the maximum degree, Δ. Source: `http://networkrepository.com` [61].

Graph	$\lvert V\rvert$	$\lvert E\rvert$	Δ	κ
soc-friendster	66M	2B	5K	305
fb-uci-uni	59M	92M	5K	17
soc-livejournal	4M	28M	3K	214
soc-orkut	3M	106M	27K	231
web-baidu-baike	2M	18M	98K	83
web-hudong	2M	15M	62K	529
web-wikipedia2009	2M	5M	3K	67
web-google	916K	5M	6K	65
bio-mouse-gene	43K	14M	8K	1K
bio-human-gene1	22K	12M	8K	2K
bio-human-gene2	14K	9M	7K	2K
bio-WormNet-v3	16K	763K	1K	165

A key contribution here is a conceptual idea and a corresponding technical lemma underlying all our algorithms. We show that every graph admits a “small” sized *low degeneracy partition* (LDP), which is a partition of its vertex set into “few” blocks such that the subgraph induced by each block has low degeneracy, roughly logarithmic in n. Moreover, such an LDP can be computed by a very simple and distributed randomized algorithm: for each vertex, choose a “color” independently and uniformly at random from a suitable-sized palette (this is not to be confused with the eventual graph coloring we seek; this random assignment is most probably not a proper coloring of the graph). The resulting color classes define the blocks of such a partition, with high probability. Theorem 8, the LDP Theorem, makes this precise. Given an LDP, a generic graph coloring algorithm is to run the aforementioned greedy algorithm on each block, using distinct palettes for the distinct blocks. We obtain algorithms achieving our claimed results by suitably implementing this generic algorithm in each computational model.

Lower Bounds. Recall that a graph with degeneracy κ admits a proper $(\kappa + 1)$-coloring. As Table 1 makes clear, there are several space-conscious $(\Delta + 1)$-coloring algorithms known; perhaps we could aim for improved algorithms that provide $(\kappa + 1)$-colorings? We prove that this is not possible in sublinear complexity in either the streaming or the query model. In fact, we prove more. We show that distinguishing n-vertex graphs of degeneracy κ from those with chromatic number $\kappa + 2$ requires $\Omega(n^2)$ space in the streaming model and $\Omega(n^2)$ queries in the general graph query model. This shows that it is hard to produce a $(\kappa + 1)$-coloring and in fact even to determine the value of κ. These results generalize to the problems of producing a $(\kappa + \lambda)$-coloring or estimating the degeneracy up to $\pm\lambda$; the corresponding lower bounds are $\Omega(n^2/\lambda^2)$. Furthermore, the streaming lower bounds hold even in the insertion-only model, where the input stream is simply a listing of the graph’s edges in some order; compare this with our upper bound, which works even for dynamic graph streams.

A possible criticism of the above lower bounds for coloring is that they seem to depend on it being hard to *estimate* the degeneracy κ. Perhaps the coloring problem could become easier if κ was given to the algorithm in advance? We prove two more lower bounds showing that this is not so: the same $\Omega(n^2/\lambda^2)$ bounds hold even with κ known *a priori.*

In the full version of this paper [18], we present a “combinatorial” lower bound that addresses a potential criticism of our main algorithmic technique: the LDP. Perhaps a more sophisticated graph-theoretic result, such as the Palette Sparsification Theorem of Assadi et al. (see Section 2), could improve the quality of the colorings obtained? We prove that this is not so: there is no analogous theorem for colorings with “about κ” colors.

2 Related Work and Comparisons

Streaming and Query Models. Assadi et al. [5] give a one-pass streaming $(\Delta+1)$-coloring algorithm that uses $\widetilde{O}(n)$ space (i.e., is semi-streaming) and works on dynamic graph streams. They also give a $(\Delta+1)$-coloring algorithm in the general graph query model that makes $\widetilde{O}(n^{3/2})$ queries. For these, they establish a beautiful *Palette Sparsification Theorem*: a combinatorial result saying that choosing a random $O(\log n)$-sized palette from $\{1,\ldots,\Delta+1\}$ for each vertex allows a compatible list coloring. While we do not get a similarly tight combinatorial result – there isn't one, as we just noted – we do achieve faster post-processing time ($\widetilde{O}(n)$ versus their $\widetilde{O}(n\sqrt{\Delta})$) in the streaming setting and have a simpler post-processing algorithm (greedy offline versus matching-based) in the query setting, while keeping other complexity parameters the same (Table 1). These wins come at the price of a less tight result – $(1+o(1))\kappa$ colors instead of the combinatorially optimal $\kappa+1$ – but of course our streaming and query lower bounds show that such slack is necessary. Also, as noted before, we often have $\kappa \ll \Delta$ (Table 2).

For streaming lower bounds, Abboud et al. [1] show that coloring a graph G with $\chi(G)$ colors requires $\Omega(n^2/p)$ space in p passes. They also show that deciding c-colorability for $3 \leqslant c < n$ (that might be a function of n) takes $\Omega((n-c)^2/p)$ space in p passes. Furthermore, any streaming algorithm that distinguishes between $\chi(G) \leqslant 3c$ and $\chi(G) \geqslant 4c$ must use $\Omega(n^2/pc^2)$ space. Another recent work on coloring in the streaming model is Radhakrishnan et al. [60], which studies the problem of 2-coloring an n-uniform hypergraph.

In the query model, there are a number of works studying basic graph problems [39, 57, 25] but, to the best of our knowledge, Assadi et al. were the first to study graph coloring in this sense. Also, to the best of our knowledge, there was no previously known algorithm for $O(\alpha)$-coloring in the semi-streaming and graph query settings, whereas here we obtain $(\kappa+o(\kappa))$-colorings; recall the bound $\kappa \leqslant 2\alpha-1$.

MPC and Congested Clique Models. The MapReduce framework [27] is extensively used in distributed computing to process massive data sets. Beame, Koutris, and Suciu [16] defined the Massively Parallel Communication (MPC) model to abstract out key theoretical features of MapReduce; it has since become a widely used setting for designing and analyzing big data algorithms, especially for graph problems. Another well studied model for distributed graph algorithms is Congested Clique [50]. Behnezhad et al. [17] show that Congested Clique is equivalent to the "semi-MPC model," defined as MPC with $O(n\log n)$ bits of memory per machine, thanks to simulations in both directions preserving the round complexity.

Harvey et al. [41] gave a $(\Delta+o(\Delta))$-coloring algorithm in the MapReduce model; it can be simulated in MPC using $O(1)$ rounds and $O(n^{1+c})$ space per machine for some constant $c>0$. The aforementioned paper of Assadi et al. [5] gives an $O(1)$-round MPC algorithm for $(\Delta+1)$-coloring using $O(n\log^3 n)$ bits of space per machine. Because this space usage is $\omega(n\log n)$, the equivalence result of Behnezhad et al. [17] does not apply and this doesn't lead to an $O(1)$-round Congested Clique algorithm. In contrast, our MPC algorithm can be made to use only $O(n\log n)$ bits per machine and $\kappa+o(\kappa)$ colors for graphs with $\kappa=\omega(\log^2 n)$, and therefore leads to such a Congested Clique algorithm. Chang et al. [23] gave an $O(\sqrt{\log\log n})$-round MPC algorithm with $o(n)$ space per machine and $\widetilde{O}(m)$ space in total. Using the improved network decomposition results by Rozhon and Ghaffari [62], this round complexity can be reduced to $O(\log\log\log n)$. We, however, focus on the quasi-linear memory per machine regime.

Graph coloring has recently garnered considerable attention in the Congested Clique model. Parter [58] gave a $(\Delta+1)$-coloring algorithm using $O(\log\log\Delta \cdot \log^\star \Delta)$ rounds, later improved to $O(\log^\star \Delta)$ by Parter and Su [59]. Chang et al. [23] have recently improved this to $O(1)$ rounds. They use similar but more involved graph partitioning techniques than us, as is probably necessary for a stringent $(\Delta+1)$-coloring. For low-degeneracy graphs, our algorithm uses fewer colors than all these algorithms while achieving the best possible asymptotic round complexity ($O(1)$). Parallel to our work, Ghaffari and Sayyadi [36] gave an $O(1)$-round algorithm for the $O(\alpha)$-coloring problem. Their analysis suggests that they obtain a $(c\alpha)$-coloring algorithm, where the constant $c > 10$. On the other hand, we get a tight $(1+o(1))\kappa$-coloring. Recall, again, that $\kappa \leqslant 2\alpha - 1$ (Fact 2). Hence, we have an arguably simpler algorithmic framework achieving better results. The main novelty in our techniques lies in choosing degeneracy as the key parameter (instead of arboricity, which could lead to results looser by a factor of 2) and in the careful analysis that gives very sharp – not just asymptotic – bounds on the number of colors. Our algorithm (only the Congested Clique implementation), however, needs $\kappa = \omega(\log^2 n)$ or $\kappa = O(1)$ to keep the round complexity constant.

The LOCAL Model. There is a deep body of work on graph coloring in this model. Indeed, graph coloring is one of *the* most central "symmetry breaking" problems in distributed computing. We refer the reader to the monograph by Barenboim and Elkin [13] for an excellent overview of the state of the art. Here, we shall briefly discuss only a few results closely related to our contribution.

There is a long line of work on fast $(\Delta+1)$-coloring in the LOCAL model, in the deterministic as well as the randomized setting [55, 10, 33, 51, 44, 3, 63, 14] culminating in sublogarithmic time solutions due to Harris [40] and Chang et al. [24]. Barenboim and Elkin [11, 12] studied fast distributed coloring algorithms that may use far fewer than Δ colors: in particular, they gave algorithms that use $O(\alpha)$ colors and run in $O(\alpha^\varepsilon \log n)$ time on graphs with arboricity at most α. Recall again that $\kappa \leqslant 2\alpha - 1$, so that a 2α-coloring always *exists*. They also gave a faster $O(\log n)$-time algorithm using $O(\alpha^2)$ colors. Further, they gave a family of algorithms that produce an $O(t\alpha^2)$-coloring in $O(\log_t n + \log^\star n)$, for every t such that $2 \leqslant t \leqslant O(\sqrt{n/\alpha})$. Our algorithm for the LOCAL model builds on this latter result.

Kothapalli and Pemmaraju [47] focused on arboricity-dependant coloring using very few rounds. They gave a randomized $O(k)$-round algorithm that uses $O(\alpha n^{1/k})$ colors for $2\log\log n \leqslant k \leqslant \sqrt{\log n}$ and $O(\alpha^{1+1/k} n^{1/k+3/k^2} 2^{-2^k})$ colors for $k < 2\log\log n$. We extend their result to the range $k \in \left[\omega(\sqrt{\log n}),\, o(\log n)\right]$, using $O(\alpha n^{1/k} \log n)$ colors.

Ghaffari and Lymouri [35] gave a randomized $O(\alpha)$-coloring algorithm that runs in time $O(\log n \cdot \min\{\log\log n, \log\alpha\})$ as well as an $O(\log n)$-time algorithm using $\min\{(2+\varepsilon)\alpha + O(\log n \log\log n),\, O(\alpha\log\alpha)\}$ colors, for any constant $\varepsilon > 0$. However, their technique does not yield a sublogarithmic time algorithm, even at the cost of a larger palette.

The LDP Technique. As mentioned earlier, our algorithmic results rely on the concept of a low degeneracy partition (LDP) that we introduce in this work. Some relatives of this idea have been considered before. Specifically, Barenboim and Elkin [13] define a d-defective (resp. b-arbdefective) c-coloring to be a vertex coloring using palette $[c]$ such that every color class induces a subgraph with maximum degree at most d (resp. arboricity at most b). Obtaining such improper colorings is a useful first step towards obtaining proper colorings. They give deterministic algorithms to obtain good arbdefective colorings [12]. However, their algorithms are elaborate and are based on construction of low outdegree acyclic partial orientations of the graph's edges: an expensive step in our space-conscious models.

Elsewhere (Theorem 10.5 of Barenboim and Elkin [13]), they note that a useful defective (not arbdefective) coloring is easily obtained by randomly picking a color for each vertex; this is then useful for computing an $O(\Delta)$-coloring.

Our LDP technique can be seen as a simple randomized method to produce an arbdefective coloring. Crucially, we parametrize our result using degeneracy instead of arboricity and give sharp – not just asymptotic – bounds on the degeneracy of each color class.

The Degeneracy Parameter. The parameter has been studied under several other names, such as *width* [34], *linkage* [46] and *Szekeres-Wilf number* [66]. For a graph G, the number $\kappa(G) + 1$ is often called the *coloring number* of G [28, 65]. It has also been extensively studied as *k-core number* in different areas such as data streaming and parallel computing [29], distributed systems [4], data mining [53], protein networks [6], and social networks[20]. Farach-Colton and Tsai [30] studied the parameter in the streaming model, and gave a one-pass semi-streaming algorithm that approximates the degeneracy of an input graph within a multiplicative factor of $1 + \varepsilon$. Our lower bounds complement this result as we show that computing the degeneracy κ exactly or more generally within a multiplicative factor of $(1 + \kappa^{-(1/2+\gamma)})$, for some constant γ, is not possible in the one-pass semi-streaming setting.

Other Related Work. Other work considers coloring in the setting of *dynamic graph algorithms*: edges are inserted and deleted over time and the goal is to *maintain* a valid vertex coloring of the graph that must be updated quickly after each modification. Unlike in the streaming setting, there is no space restriction. Bhattacharya et al. [19] gave a randomized algorithm that maintains a $(\Delta + 1)$-coloring with $O(\log \Delta)$ expected amortized update time, later improved to $O(1)$ by Henzinger and Peng [43]. Solomon and Wein [64] studied the problem for low-arboricity graphs and gave an $O(\alpha \log^2 n)$-coloring algorithm with $O(\text{poly}(\log \log n))$ update time. Recently, Henzinger et al. [42] designed an $O(\alpha \log n)$-coloring algorithm with $O(\log^2 n)$ update time. Barba et al. [9] gave tradeoffs between the number of colors used and update time. However, the techniques in these works do not seem to apply in the streaming setting due to fundamental differences in the models.

Estimating the arboricity of a graph in the streaming model is a well studied problem. McGregor et al. [52] gave a one pass $(1+\varepsilon)$-approximation algorithm to estimate the arboricity of graph using $\widetilde{O}(n)$ space. Bahmani et al. [7] gave a matching lower bound. Our lower bounds for estimating degeneracy are quantitatively much larger but they call for much tighter estimates.

3 Preliminaries

Throughout this paper, graphs are simple, undirected, and unweighted. In considering a graph coloring problem, the input graph will usually be called G and we will put $n = |V(G)|$. The notation "$\log x$" stands for $\log_2 x$. For an integer k, we denote the set $\{1, 2, \ldots, k\}$ by $[k]$.

For a graph G, we define $\Delta(G) = \max\{\deg(v) : v \in V(G)\}$. We say that G is k-degenerate if every induced subgraph of G has a vertex of degree at most k. For instance, every forest is 1-degenerate and an elementary theorem says that every planar graph is 5-degenerate. The *degeneracy* $\kappa(G)$ is the smallest k such that G is k-degenerate. The *arboricity* $\alpha(G)$ is the smallest r such that the edge set $E(G)$ can be partitioned into r forests. When the graph G is clear from the context, we simply write Δ, κ, and α, instead of $\Delta(G)$, etc.

We note two useful facts: the first is immediate and the second is an easy exercise.

▶ **Fact 1.** *If an n-vertex graph has degeneracy κ, then it has at most κn edges.*

▶ **Fact 2.** *In every graph, the degeneracy κ and arboricity α satisfy $\alpha \leqslant \kappa \leqslant 2\alpha - 1$.*

In analyzing our algorithms, it will be useful to consider certain *vertex orderings* of graphs and their connection with the notion of degeneracy, given by Lemma 5 below. Although the lemma is folklore, it is crucial to our analysis, so we include a proof for completeness.

▶ **Definition 3.** *An* ordering *of G is a list consisting of all its vertices (equivalently, a total order on $V(G)$). Given an ordering $\lhd$, for each $v \in V(G)$, the* ordered neighborhood *$N_{G,\lhd}(v) := \{w \in V(G) : \{v, w\} \in E(G), v \lhd w\}$, i.e., the set of neighbors of v that appear* after *v in the ordering. The* ordered degree *$\operatorname{odeg}_{G,\lhd}(v) := |N_{G,\lhd}(v)|$.*

▶ **Definition 4.** *A* degeneracy ordering *of G is an ordering produced by the following algorithm: starting with an empty list, pick a minimum degree vertex v (breaking ties arbitrarily), append v to the end of the list, and recurse on $G - v$ if it is nonempty.*

▶ **Lemma 5.** *G is k-degenerate iff there exists an ordering $\lhd$ such that $\operatorname{odeg}_{G,\lhd}(v) \leqslant k$ for all $v \in V(G)$.*

Proof. Suppose that G is k-degenerate. Let $\lhd = (v_1, \ldots, v_n)$ be a degeneracy ordering. Then, for each i, $\operatorname{odeg}_{G,\lhd}(v_i)$ is the degree of v_i in the induced subgraph $H := G \setminus \{v_1, \ldots, v_{i-1}\}$. By definition, H has a vertex of degree at most k, so v_i, being a minimum degree vertex in H, must have degree at most k.

On the other hand, suppose that G has an ordering $\lhd$ such that $\operatorname{odeg}_{G,\lhd}(v) \leqslant k$ for all $v \in V(G)$. Let H be an induced subgraph of G. Let v be the leftmost (i.e., smallest) vertex in $V(H)$ according to $\lhd$. Then all neighbors of v in H in fact lie in $N_{G,\lhd}(v)$, so $\deg_H(v) \leqslant \operatorname{odeg}_{G,\lhd}(v) \leqslant k$. Therefore, G is k-degenerate. ◀

A c-coloring of a graph G is a mapping $\psi\colon V(G) \to [c]$; it is said to be a *proper coloring* if it makes no edge monochromatic: $\psi(u) \neq \psi(v)$ for all $\{u, v\} \in E(G)$. The smallest c such that G has a proper c-coloring is called the *chromatic number* $\chi(G)$. By considering the vertices of G one at a time and coloring greedily, we immediately obtain a proper $(\Delta + 1)$-coloring. This idea easily extends to degeneracy-based coloring.

▶ **Lemma 6.** *Given unrestricted ("offline") access to an input graph G, we can produce a proper $(\kappa + 1)$-coloring of G in linear time.*

Proof. Construct a degeneracy ordering $(v_1, \ldots, v_n)$ of G and then greedily color the vertices one by one in the order $(v_n, \ldots, v_1)$. Given a palette of size $\kappa + 1$, by the "only if" direction of Lemma 5, there will always be a free color for a vertex. ◀

Of course, the simple algorithm above is not implementable directly in "sublinear" settings, such as space-bounded streaming algorithms, query models, or distributed computing models. Nevertheless, we shall use it on suitably constructed subgraphs of our input graph.

We shall use the following form of the Chernoff bound.

▶ **Fact 7.** *Let X be a sum of mutually independent indicator random variables. Let μ and δ be real numbers such that $\mathbb{E}X \leqslant \mu$ and $0 \leqslant \delta \leqslant 1$. Then, $\Pr[X \geqslant (1 + \delta)\mu] \leqslant \exp(-\mu\delta^2/3)$.*

4 A Generic Framework for Coloring Algorithms

In this section, we give a generic framework for graph coloring that we later instantiate in various computational models. As a reminder, our focus is on graphs G with a nontrivial upper bound on the degeneracy $\kappa = \kappa(G)$. Each such graph *admits* a proper $(\kappa + 1)$-coloring; our focus will be on obtaining a proper $(\kappa + o(\kappa))$-coloring efficiently.

As a broad outline, our framework calls for coloring G in two phases. The first phase produces a *low degeneracy partition* (LDP) of G: it partitions $V(G)$ into a "small" number of parts, each of which induces a subgraph that has "low" degeneracy. This step can be thought of as preprocessing and it is essentially free (in terms of complexity) in each of our models. The second phase properly colors each part, using a small number of colors, which is possible because the degeneracy is low. In later sections, we shall see that the low degeneracy allows this second phase to be efficient in each of the models we consider.

4.1 A Low Degeneracy Partition

In this phase of our coloring framework, we assign each vertex a color chosen uniformly at random from $[\ell]$, these choices being mutually independent, where ℓ is a suitable parameter. For each $i \in [\ell]$, let G_i denote the subgraph of G induced by vertices colored i. We shall call each G_i a *block* of the vertex partition given by $(G_1, \ldots, G_\ell)$. The next theorem, our main technical tool, provides certain guarantees on this partition given a suitable choice of ℓ.

▶ **Theorem 8** (LDP Theorem). *Let G be an n-vertex graph with degeneracy κ. Let $k \in [1, n]$ be a "guess" for the value of κ and let $s \geqslant Cn \log n$ be a sparsity parameter, where C is a sufficiently large universal constant. Put*

$$\ell = \left\lceil \frac{2nk}{s} \right\rceil , \qquad \lambda = 3\sqrt{\kappa \ell \log n}\,, \tag{1}$$

and let $\psi\colon V(G) \to [\ell]$ be a uniformly random coloring of G. For $i \in [\ell]$, let G_i be the subgraph induced by $\psi^{-1}(i)$. Then, the partition $(G_1, \ldots, G_\ell)$ has the following properties.

(i) *If $k \leqslant 2\kappa$, then w.h.p., for each i, the degeneracy $\kappa(G_i) \leqslant (\kappa + \lambda)/\ell$.*

(ii) *W.h.p., for each i, the block size $|V(G_i)| \leqslant 2n/\ell$.*

(iii) *If $\kappa \leqslant k \leqslant 2\kappa$, then w.h.p., the number of monochromatic edges $|E(G_1) \cup \cdots \cup E(G_\ell)| \leqslant s$.*

In each case, "w.h.p." means "with probability at least $1 - 1/\operatorname{poly}(n)$."

Proof. Notice that when $k \leqslant (C/2) \log n$, the condition $s \geqslant Cn \log n$ results in $\ell = 1$, so the vertex partition is the trivial one-block partition, which obviously satisfies all the properties in the theorem. Thus, in our proof, we may assume that $k > (C/2) \log n$.

We start with Item ii, which is the most straightforward. From Equation (1), we have $\ell \leqslant 4nk/s$, so

$$\frac{n}{\ell} \geqslant \frac{s}{4k} \geqslant \frac{Cn \log n}{4k} \geqslant \frac{C \log n}{4}\,.$$

Each block size $|V(G_i)|$ has binomial distribution $\operatorname{Bin}(n, 1/\ell)$, so a Chernoff bound gives

$$\Pr\left[|V(G_i)| > \frac{2n}{\ell}\right] \leqslant \exp\left(-\frac{n}{3\ell}\right) \leqslant \exp\left(-\frac{C \log n}{12}\right) \leqslant \frac{1}{n^2}\,,$$

for sufficiently large C. By a union bound over the at most n blocks, Item ii fails with probability at most $1/n$.

Items i and iii include the condition $k \leqslant 2\kappa$, which we shall assume for the rest of the proof. By Equation (1) and the bounds $s \geqslant Cn \log n$ and $k > (C/2) \log n$,

$$\ell \leqslant \left\lceil \frac{2k}{C \log n} \right\rceil \leqslant \frac{4k}{C \log n} \leqslant \frac{8\kappa}{C \log n}\,,$$

whence, for sufficiently large C,

$$\lambda \leqslant 3\sqrt{\kappa \cdot \frac{8\kappa}{C \log n} \cdot \log n} \leqslant \kappa \,. \tag{2}$$

We now turn to establishing Item i. Let $\lhd$ be a degeneracy ordering for G. For each $i \in [\ell]$, let $\lhd_i$ be the restriction of $\lhd$ to $V(G_i)$. Consider a particular vertex $v \in V(G)$ and let $j = \psi(v)$ be its color. We shall prove that, w.h.p., $\operatorname{odeg}_{G,\lhd_j}(v) \leqslant (\kappa+\lambda)/\ell$.

By the "only if" direction of Lemma 5, we have $\operatorname{odeg}_{G,\lhd}(v) = |N_{G,\lhd}(v)| \leqslant \kappa$. Now note that

$$\operatorname{odeg}_{G_j,\lhd_j}(v) = \sum_{u \in N_{G,\lhd(v)}} \mathbb{1}_{\{\psi(u)=\psi(v)\}}$$

is a sum of mutually independent indicator random variables, each of which has expectation $1/\ell$. Therefore, $\mathbb{E}\operatorname{odeg}_{G_j,\lhd_j}(v) = \operatorname{odeg}_{G,\lhd}(v)/\ell \leqslant \kappa/\ell$. Since $\lambda \leqslant \kappa$ by Equation (2), we may use the form of the Chernoff bound in Fact 7, which gives us

$$\Pr\left[\operatorname{odeg}_{G_j,\lhd_j}(v) > \frac{\kappa+\lambda}{\ell}\right] \leqslant \exp\left(-\frac{\kappa}{\ell}\frac{\lambda^2}{3\kappa^2}\right) = \exp\left(-\frac{9\kappa\ell\log n}{3\kappa\ell}\right) \leqslant \frac{1}{n^3}\,,$$

where the equality follows from Equation (1). In words, with probability at least $1-1/n^3$, the vertex v has ordered degree at most $(\kappa+\lambda)/\ell$ within its own block. By a union bound, with probability at least $1-1/n^2$, all n vertices of G satisfy this property. When this happens, by the "if" direction of Lemma 5, it follows that $\kappa(G_i) \leqslant (\kappa+\lambda)/\ell$ for every i.

Finally, we take up Item iii, which is now straightforward. Assume that the high probability event in Item i occurs. Then, by Fact 1,

$$|E(G_1) \cup \cdots \cup E(G_\ell)| \leqslant \sum_{i=1}^{\ell} \kappa(G_i)\,|V(G_i)| \leqslant \frac{\kappa+\lambda}{\ell}\sum_{i=1}^{\ell}|V(G_i)| = \frac{n(\kappa+\lambda)}{\ell} \leqslant \frac{2n\kappa}{\ell} \leqslant s\,,$$

where the final inequality uses the condition $\kappa \leqslant k$ and Equation (1). ◀

It will be convenient to encapsulate the guarantees of this theorem in a definition.

▶ **Definition 9.** *Suppose graph G has degeneracy κ. A vertex partition $(G_1,\ldots,G_\ell)$ simultaneously satisfying the degeneracy bound in Item i, the block size bound in Item ii, and the (monochromatic) edge sparsity bound in Item iii in Theorem 8 is called an (ℓ,s,λ)-LDP of G.*

It will turn out that an (ℓ,s,λ)-LDP leads to a proper coloring of G using at most $\kappa+\lambda+\ell$ colors. An instructive setting of parameters is $s = \Theta((n\log n)/\varepsilon^2)$, where ε is either a small constant or a slowly vanishing function of n, such as $1/\log n$. Then, a quick calculation shows that when an accurate guess $k \in [\kappa, 2\kappa]$ is made, Theorem 8 guarantees an LDP that has edge sparsity $s = \widetilde{O}(n)$ and that leads to an eventual proper coloring using $(1+O(\varepsilon))\kappa$ colors. When $\varepsilon = o(1)$, this number of colors is $\kappa + o(\kappa)$.

Recall that the second phase of our coloring framework involves coloring each G_i separately, exploiting its low degeneracy. Indeed, given an (ℓ,s,λ)-LDP, each block G_i *admits* a proper $(\kappa(G_i)+1)$-coloring. Suppose we use a distinct palette for each block; then the total number of colors used is

$$\sum_{i=1}^{\ell}(\kappa(G_i)+1) \leqslant \ell\left(\frac{\kappa+\lambda}{\ell}+1\right) = \kappa+\lambda+\ell\,, \tag{3}$$

as claimed above. Of course, even if our first phase random coloring ψ yields a suitable LDP, we still have to collect each block G_i or at least enough information about each block so as to produce a proper $(\kappa(G_i) + 1)$-coloring. How we do this depends on the precise model of computation; see Sections 5 and 6 here and the full version [18] for further instantiations.

4.2 Applications in Various Models

We now turn to the application of the framework in designing graph coloring algorithms in specific models of computation for big data, where the focus is on utilizing space sublinear in the size of the massive input graph. Such models are sometimes termed *space-conscious*.

In Section 5, we discuss the simulation of our framework in the streaming model. We present a semi-streaming algorithm in the dynamic model, meaning that edges can be both inserted and deleted. Our main result is captured in Theorem 10.

▶ **Theorem 10.** *Set $s = \lceil \varepsilon^{-2} n \log n \rceil$, where $\varepsilon > 0$ is a parameter. There is a one-pass algorithm that processes a dynamic (i.e., turnstile) graph stream using $O(\varepsilon^{-2} n \log^4 n)$ bits of space and, with high probability, produces a proper coloring using at most $(1 + O(\varepsilon))\kappa$ colors. In particular, taking $\varepsilon = 1/\log n$, it produces a $(\kappa + o(\kappa))$-coloring using $\widetilde{O}(n)$ space. Each edge update is processed in $\widetilde{O}(1)$ time and end-of-stream post-processing takes $\widetilde{O}(n)$ time.*

In Section 6, applying our framework to the general graph query model, we obtain:

▶ **Theorem 11.** *Given query access to a graph G, there is a randomized algorithm that, with high probability, produces a proper coloring of G using $\kappa + o(\kappa)$ colors. The algorithm's worst-case query complexity, running time, and space usage are all $\widetilde{O}(n^{3/2})$.*

Besides this, we obtain algorithmic results in certain distributed models of computation, namely MPC, Congested-Clique, and LOCAL models, where graph coloring is one of the most heavily studied problems. Our results are stated below. See the full version of our paper [18] for the corresponding discussions and proofs.

▶ **Theorem 12.** *There is a randomized $O(1)$-round MPC algorithm that, given an n-vertex graph G, outputs a $(\kappa + o(\kappa))$-coloring of G with high probability. The algorithm uses n processors, each with $O(n \log^2 n)$ bits of memory.*

▶ **Theorem 13.** *There is a randomized $O(1)$-round algorithm in the Congested Clique model that, given a graph G, w.h.p. finds a $(\kappa + O(\kappa^{3/4} \log^{1/2} n))$-coloring. For $\kappa = \omega(\log^2 n)$, this gives a $(\kappa + o(\kappa))$-coloring.*

▶ **Theorem 14.** *There is a randomized distributed algorithm in the LOCAL model that, given an n-vertex graph G, an estimate of its arboricity α up to a constant factor, and a parameter t such that $2 < t \leqslant O(\sqrt{n/\log n})$, produces an $O(t\alpha \log n)$-coloring of G in time $O(\log_t n + \log^\star n)$.*

5 Streaming Model

We turn to the most intensely studied space-conscious model: the data streaming model. For graph problems, in the basic model, the input is a stream of non-repeated edges that define the input graph G: this is called the *insertion-only* model, since it can be thought of as building up G through a sequence of edge insertions. In the more general *dynamic graph model* or *turnstile model*, the stream is a sequence of edge updates, each update being either an insertion or a deletion: the net effect is to build up G. Our algorithm below will work in this more general model. Later, in we shall give a corresponding lower bound that will hold even in the insertion-only model (for a lower bound, this is a strength).

We assume that the vertex set $V(G) = [n]$ and the input is a stream σ of at most $m = \text{poly}(n)$ updates to an initially empty graph. An update is a triple (u, v, c), where $u, v \in V(G)$ and $c \in \{-1, 1\}$: when $c = 1$, this token represents an insertion of edge $\{u, v\}$ and when $c = -1$, it represents a deletion. Let $N = \binom{n}{2}$ and $[[m]] = \mathbb{Z} \cap [-m, m]$. It is convenient to imagine a vector $\mathbf{x} \in [[m]]^N$ of edge multiplicities that starts at zero and is updated entrywise with each token. The input graph G described by the stream will be the underlying simple graph, i.e., $E(G)$ will be the set of all edges $\{u, v\}$ such that $x_{u,v} \neq 0$ at the end. We shall say that σ *builds up* $\mathbf{x}$ and G.

Our algorithm makes use of two data streaming primitives, each a *linear sketch*. (We can do away with these sketches in the insertion-only setting; see the end of this section.) The first is a sketch for *sparse recovery* given by a matrix A (say): given a vector $\mathbf{x} \in [[m]]^N$ with sparsity $\|\mathbf{x}\|_0 \leqslant t$, there is an efficient algorithm to reconstruct $\mathbf{x}$ from $A\mathbf{x}$. The second is a sketch for ℓ_0 *estimation* given by a random matrix B (say): given a vector $\mathbf{x} \in [[m]]^N$, there is an efficient algorithm that takes $B\mathbf{x}$ and computes from it an estimate of $\|\mathbf{x}\|_0$ that, with probability at least $1 - \delta$, is a $(1 + \gamma)$-multiplicative approximation. It is known that there exists a suitable $A \in \{0, 1\}^{d \times N}$, where $d = O(t \log(N/t))$, where A has column sparsity $O(\log(N/t))$; see, e.g., Theorem 9 of Gilbert and Indyk [37]. It is also known that there exists a suitable distribution over matrices giving $B \in \{0, 1\}^{d' \times N}$ with $d' = O(\gamma^{-2} \log \delta^{-1} \log N(\log \gamma^{-1} + \log \log m))$. Further, given an update to the ith entry of $\mathbf{x}$, the resulting updates in $A\mathbf{x}$ and $B\mathbf{x}$ can be effected quickly by generating the required portion of the ith columns of A and B.

Algorithm 1 One-Pass Streaming Algorithm for Graph Coloring via Degeneracy.

```
procedure COLOR(stream $\sigma$, integer $k$)   ▷ $\sigma$ builds up $\mathbf{x}$ and $G$;  $k \in [1, n]$ is a guess for $\kappa(G)$
    choose $s$, $\ell$ as in Equation (1) and $t, d, d', A, B$ as in the above discussion
    initialize $\mathbf{y} \in [[m]]^d$ and $\mathbf{z} \in [[m]]^{d'}$ to zero
    foreach $u \in [n]$ do $\psi(u) \leftarrow$ uniform random color in $[\ell]$
    foreach token $(u, v, c)$ in $\sigma$ do
        if $\psi(u) = \psi(v)$ then $\mathbf{y} \leftarrow \mathbf{y} + cA_{u,v}$;  $\mathbf{z} \leftarrow \mathbf{z} + cB_{u,v}$
    if estimate of $\|\mathbf{w}\|_0$ obtained from $\mathbf{z}$ is $> 5s/4$ then abort
    $\mathbf{w}' \leftarrow$ result of $t$-sparse recovery from $\mathbf{y}$                ▷ we expect that $\mathbf{w}' = \mathbf{w}$
    foreach $i \in [\ell]$ do
        $G_i \leftarrow$ simple graph induced by $\{\{u, v\} : w'_{u,v} \neq 0$ and $\psi(u) = \psi(v) = i\}$
        color $G_i$ using palette $\{(i, j) : 1 \leqslant j \leqslant \kappa(G_i) + 1\}$; cf. Lemma 6 ▷ net effect is to color $G$
```

In our description of Algorithm 1, we use $A_{u,v}$ (resp. $B_{u,v}$) to denote the column of A (resp. B) indexed by $\{u, v\}$. The algorithm's logic results in sketches $\mathbf{y} = A\mathbf{w}$ and $\mathbf{z} = B\mathbf{w}$, where $\mathbf{w}$ corresponds to the subgraph of G consisting of ψ-monochromatic edges only (cf. Theorem 8), i.e., $\mathbf{w}$ is obtained from $\mathbf{x}$ by zeroing out all entries except those indexed by $\{u, v\}$ with $\psi(u) = \psi(v)$. We choose the parameter $t = 2s$, where $s \geqslant Cn \log n$ is the sparsity parameter from Theorem 8, which gives $d = O(s \log n)$; we choose $\gamma = 1/4$ and $\delta = 1/n$, giving $d' = O(\log^3 n)$.

Notice that Algorithm 1 requires a guess for $\kappa := \kappa(G)$, which is not known in advance. Our final one-pass algorithm runs $O(\log n)$ parallel instances of COLOR(σ, k), using geometrically spaced guesses $k = 2, 4, 8 \ldots$. It outputs the coloring produced by the non-aborting run that uses the smallest guess. This leads to this section's main result (restated from Section 4.2).

▶ **Theorem 10.** *Set $s = \lceil \varepsilon^{-2} n \log n \rceil$, where $\varepsilon > 0$ is a parameter. There is a one-pass algorithm that processes a dynamic (i.e., turnstile) graph stream using $O(\varepsilon^{-2} n \log^4 n)$ bits of space and, with high probability, produces a proper coloring using at most $(1 + O(\varepsilon))\kappa$ colors. In particular, taking $\varepsilon = 1/\log n$, it produces a $(\kappa + o(\kappa))$-coloring using $\widetilde{O}(n)$ space. Each edge update is processed in $\widetilde{O}(1)$ time and end-of-stream post-processing takes $\widetilde{O}(n)$ time.*

Proof. The coloring produced is obviously proper. Let us bound the number of colors used. One of the parallel runs of Color(σ, k) in Algorithm 1 will use a value $k = k^\star \in (\kappa, 2\kappa]$. We shall prove that, w.h.p., (a) every non-aborting run with $k \leqslant k^\star$ will use at most $(1 + O(\varepsilon))\kappa$ colors, and (b) the run with $k = k^\star$ will not abort.

We start with (a). Consider a particular run using $k \leqslant k^\star$. By Item i of Theorem 8, each G_i has degeneracy at most $(\kappa + \lambda)/\ell$; so if $\mathbf{w}$ is correctly recovered by the sparse recovery sketch (i.e., $\mathbf{w}' = \mathbf{w}$ in Algorithm 1), then each G_i is correctly recovered and the run uses at most $\kappa + \lambda + \ell$ colors, as in Equation (3). Using the values from Equation (1), this number is at most $(1 + O(\varepsilon))\kappa$. Now, if the run does not abort, then the estimate of the sparsity $\|\mathbf{w}\|_0$ is at most $5s/4$. By the guarantees of the ℓ_0-estimation sketch, the true sparsity is at most $(5/4)(5s/4) < 2s = t$, so, w.h.p., $\mathbf{w}$ is indeed t-sparse and, by the guarantees of the sparse recovery sketch, $\mathbf{w}' = \mathbf{w}$. Taking a union bound over all $O(\log n)$ runs, the bound on the number of colors holds for all required runs simultaneously, w.h.p..

We now take up (b). Note that $\|\mathbf{w}\|_0$ is precisely the number of ψ-monochromatic edges in G. By Item iii of Theorem 8, we have $\|\mathbf{w}_0\| \leqslant s$ w.h.p. By the accuracy guarantee of the ℓ_0-estimation sketch, in this run the estimate of $\|\mathbf{w}\|_0$ is at most $5s/4$ w.h.p., so the run does not abort.

The space usage of each parallel run is dominated by the computation of $\mathbf{y}$, so it is $O(d \log m) = O(s \log n \log m) = O(\varepsilon^{-2} n \log^3 n)$, using our setting of s and the assumption $m = \text{poly}(n)$. The claims about the update time and post-processing time follow directly from the properties of a state-of-the-art sparse recovery scheme, e.g., the scheme based on expander matching pursuit given in Theorem 9 of Gilbert and Indyk [37]. ◀

6 Query Model

We now turn to the *general graph query model*, a standard model of space-conscious algorithms for big graphs where the input graph is random-accessible but the emphasis is on the examining only a tiny (ideally, sublinear) portion of it; for general background see Chapter 10 of Goldreich's book [38]. In this model, the algorithm starts out knowing the vertex set $[n]$ of the input graph G and can access G only through the following types of queries.

- A *pair query* Pair$(\{u, v\})$, where $u, v \in [n]$. The query returns 1 if $\{u, v\} \in E(G)$ and 0 otherwise. For better readability, we shall write this query as Pair(u, v).
- A *neighbor query* Neighbor(u, j), where $u \in [n]$ and $j \in [n-1]$. The query returns $v \in [n]$ where v is the jth neighbor of u in some underlying fixed ordering of vertex adjacency lists; if $\deg(v) < j$, so that there does not exist a jth neighbor, the query returns $\perp$.

Naturally, when solving a problem in this model, the goal is to do so while minimizing the number of queries.

6.1 Sublinear Algorithm

By adapting the combinatorial machinery from their semi streaming algorithm, Assadi et al. [5] gave an $\widetilde{O}(n^{3/2})$-query algorithm for finding a $(\Delta + 1)$-coloring. Our LDP framework gives a considerably simpler algorithm using $\kappa + o(\kappa)$ colors, where $\kappa := \kappa(G)$. We remark here that $\widetilde{O}(n^{3/2})$ query complexity is optimal (up to polylogarithmic actors), as Assadi et al. [5] proved a matching lower bound for any $(c \cdot \Delta)$-coloring algorithm, for any constant $c > 1$.

▶ **Theorem 11.** *Given query access to a graph G, there is a randomized algorithm that, with high probability, produces a proper coloring of G using $\kappa + o(\kappa)$ colors. The algorithm's worst-case query complexity, running time, and space usage are all $\widetilde{O}(n^{3/2})$.*

Proof. The algorithm proceeds in two stages. In the first stage, it attempts to extract all edges in G through neighbor queries alone, aborting when "too many" queries have been made. More precisely, it loops over all vertices v and, for each v, issues queries $\mathrm{Neighbor}(v, 1), \mathrm{Neighbor}(v, 2), \ldots$ until a query returns $\perp$. If this stage ends up making $3n^{3/2}$ queries (say) without having processed every vertex, then it aborts and the algorithm moves on to the second stage. By Fact 1, if $\kappa \leqslant \sqrt{n}$, then this stage will not abort and the algorithm will have obtained G completely; it can then $(\kappa + 1)$-color G (as in Lemma 6) and terminate, skipping the second stage.

In the second stage, we know that $\kappa > \sqrt{n}$. The algorithm now uses a random coloring ψ to construct an (ℓ, s, λ)-LDP of G using the "guess" $k = \sqrt{n}$, with $s = \Theta(\varepsilon^{-2} n \log n)$ and ℓ, λ given by Equation (1). To produce each subgraph G_i in the LDP, the algorithm simply makes all possible queries $\mathrm{Pair}(u, v)$ where $\psi(u) = \psi(v)$. W.h.p., the number of queries made is at most

$$\frac{1}{2} \sum_{i \in [\ell]} |V(G_i)|^2 \leqslant \frac{\ell}{2} \left(\frac{2n}{\ell} \right)^2 \leqslant \frac{2n^2 s}{4nk} = \Theta\left(\frac{n^{3/2} \log n}{\varepsilon^2} \right),$$

where the first inequality uses Item ii of Theorem 8. We can enforce this bound in the worst case by aborting if it is violated.

Clearly, $k \leqslant 2\kappa$, so Item i of Theorem 8 applies and by the discussion after Definition 9, the algorithm uses $(1 + O(\varepsilon))\kappa$ colors. Setting $\varepsilon = 1/\log n$, this number is at most $\kappa + o(\kappa)$ and the overall number of queries remains $\widetilde{O}(n^{3/2})$, as required. ◀

7 Lower Bounds

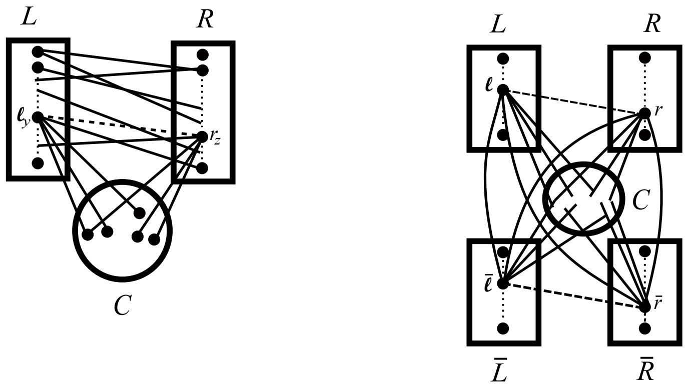
Can we improve the guarantees of our algorithms so that they use at most $\kappa + 1$ colors, rather than $\kappa + o(\kappa)$? After all, every graph G does have a proper $(\kappa(G) + 1)$-coloring. Our lower bounds answer this with a strong "No" in the data streaming and query models. If we insist on a coloring that good, we would incur the worst possible space or query complexity: $\Omega(n^2)$. In fact, this holds even if κ is known to the algorithm in advance. Moreover, all our streaming lower bounds hold even if the input stream consists of edge insertions alone.

Our lower bounds generalize to the problem of producing a $(\kappa + \lambda)$-coloring. We show that this requires $\Omega(n^2/\lambda^2)$ space or query complexity. Such generalizations are based on the following Blow-Up Lemma.

▶ **Definition 15.** *Let G be a graph and λ a positive integer. The* blow-up graph *G^λ is obtained by replacing each vertex of G with a copy of the complete graph K_λ and replacing each edge of G with a complete bipartite graph between the copies of K_λ at its endpoints. More succinctly, G^λ is the lexicographical product $G[K_\lambda]$.*

▶ **Lemma 16** (Blow-Up Lemma). *For all graphs G and positive integers λ, c, if G has a c-clique, then G^λ has a $(c\lambda)$-clique. Also, $\kappa(G^\lambda) \leqslant (\kappa(G) + 1)\lambda - 1$.*

Proof. The claim about cliques is immediate. The bound on $\kappa(G^\lambda)$ follows by taking a degeneracy ordering of G and replacing each vertex v by a list of vertices of the clique that replaces v in G^λ, ordering vertices within the clique arbitrarily. ◀



(a) Gadget graph for Lemma 19. **(b)** Gadget graph for Theorem 21.

Figure 1 Gadget constructions for lower bounds.

Streaming Lower Bounds. Our streaming lower bounds use reductions from the INDEX problem in communication complexity. In the INDEX_N problem, Alice is given a vector $\mathbf{x} = (x_1, \ldots, x_N) \in \{0,1\}^N$ and Bob is given an index $k \in [N]$. The goal is for Alice to send Bob a (possibly random) c-bit message that enables Bob to output x_k with probability at least $2/3$. The smallest c for which such a protocol exists is called the one-way randomized communication complexity, $\mathrm{R}^{\to}(\text{INDEX}_N)$. As is well known, $\mathrm{R}^{\to}(\text{INDEX}_N) = \Omega(N)$ [2].

We shall in fact consider instances of INDEX_N where $N = p^2$, for an integer p. Using a canonical bijection between $[N]$ and $[p] \times [p]$, we reinterpret $\mathbf{x}$ as a matrix with entries $(x_{ij})_{i,j \in [p]}$, and Bob's input as $(y, z) \in [p] \times [p]$. We further interpret this matrix $\mathbf{x}$ as the bipartite adjacency matrix of a $(2p)$-vertex balanced bipartite graph $H_{\mathbf{x}}$. Such graphs $H_{\mathbf{x}}$ will be key gadgets in the reductions to follow.

▶ **Definition 17.** *For $\mathbf{x} \in \{0,1\}^{p \times p}$, a realization of $H_{\mathbf{x}}$ on a list $(\ell_1, \ldots, \ell_p, r_1, \ldots, r_p)$ of distinct vertices is a graph on these vertices whose edge set is $\{\{\ell_i, r_j\} : x_{ij} = 1\}$.*

First Flavor: Degeneracy Not Known in Advance. To prove lower bounds of the first flavor, we start by demonstrating the hardness of the abstract problem GRAPH-DIST.

▶ **Definition 18** (GRAPH-DIST **problem**)**.** *Consider two graph families: $\mathcal{G}_1 := \mathcal{G}_1(n, q, \lambda)$, consisting of n-vertex graphs with chromatic number $\chi \geqslant (q+1)\lambda$, and $\mathcal{G}_2 := \mathcal{G}_2(n, q, \lambda)$, consisting of n-vertex graphs with $\kappa \leqslant q\lambda - 1$. Then GRAPH-DIST$(n, q, \lambda)$ is the problem of distinguishing $\mathcal{G}_1$ from $\mathcal{G}_2$ (note that $\mathcal{G}_1 \cap \mathcal{G}_2 = \varnothing$): given an input graph G on n vertices, the problem is to decide whether $G \in \mathcal{G}_1$ or $G \in \mathcal{G}_2$, with success probability at least $2/3$.*

We shall prove that GRAPH-DIST is "hard" in the insertion-only streaming setting and in the query setting, thereby establishing that in these models it is hard to produce a $(\kappa+\lambda)$-coloring. In fact, our proofs will show that it is just as hard to estimate the parameter κ; this goes to show that the hardness of the coloring problem is not just because of the large output size.

▶ **Lemma 19.** *Solving GRAPH-DIST(n, q, λ) in one pass requires $\Omega(n^2/\lambda^2)$ space. More precisely, there is a constant $c > 0$ such that for every integer $\lambda \geqslant 1$ and every sufficiently large integer q, there is a setting $n = n(q, \lambda)$ for which every randomized one-pass streaming algorithm for GRAPH-DIST(n, q, λ) requires at least cn^2/λ^2 bits of space.*

Proof. Put $p = q - 1$. We reduce from INDEX_N, where $N = p^2$, using the following plan. Starting with an empty graph on $n = 3\lambda p$ vertices, Alice adds certain edges based on her input $\mathbf{x} \in \{0,1\}^{p \times p}$ and then Bob adds certain other edges based on his input

$(y, z) \in [p] \times [p]$. By design, solving GRAPH-DIST(n, q, λ) on the resulting final graph reveals the bit x_{yz}, implying that a one-pass streaming algorithm for GRAPH-DIST requires at least $\mathrm{R}^{\to}(\text{INDEX}_N) = \Omega(N) = \Omega(p^2) = \Omega(n^2/\lambda^2)$ bits of memory. The details follow.

First, consider $\lambda = 1$. We use the vertex set $L \uplus R \uplus C$ ("$\uplus$" denotes a disjoint union), where $L = \{\ell_1, \ldots, \ell_p\}$, $R = \{r_1, \ldots, r_p\}$, and $|C| = p$. Alice introduces the edges of the gadget graph $H_{\mathbf{x}}$ (from def. 17), realized on the vertices $(\ell_1, \ldots, \ell_p, r_1, \ldots, r_p)$. Bob introduces all possible edges within $C \cup \{\ell_y, r_z\}$, except for $\{\ell_y, r_z\}$. Let G be the resulting graph (see Figure 1a).

If $x_{yz} = 1$, then G contains a clique on $C \cup \{\ell_y, r_z\}$, whence $\chi(G) \geqslant p + 2$. If, on the other hand, $x_{yz} = 0$, then we claim that $\kappa(G) \leqslant p$. By Lemma 5, the claim will follow if we exhibit a vertex ordering $\lhd$ such that $\mathrm{odeg}_{G,\lhd}(v) \leqslant p$ for all $v \in V(G)$. We use an ordering where $L \cup R \setminus \{\ell_y, r_z\} \lhd \ell_y \lhd \{r_z\} \cup C$ and the ordering within each set is arbitrary. By construction of $H_{\mathbf{x}}$, each vertex in $L \cup R \setminus \{\ell_y, r_z\}$ has *total* degree at most p. For each vertex $v \in \{r_z\} \cup C$, we trivially have $\mathrm{odeg}_{G,\lhd}(v) \leqslant p$ because $|C| = p$. Finally, since $x_{yz} = 0$, the vertex r_z is not a neighbor of ℓ_y; so $\mathrm{odeg}_{G,\lhd}(\ell_y) = |C| = p$. This proves the claim.

When $\lambda \geqslant 1$, Alice and Bob introduce edges so as to create the blow-up graph G^λ, as in Definition 15. By Lemma 16, if $x_{yz} = 1$, then G^λ has a $(p + 2)\lambda$-clique, whereas if $x_{yz} = 0$, then $\kappa(G^\lambda) \leqslant (p+1)\lambda - 1$. In the former case, $\chi(G^\lambda) \geqslant (p+2)\lambda = (q+1)\lambda$, so that $G^\lambda \in \mathcal{G}_1(n, q, \lambda)$; cf. Definition 18. In the latter case, $\kappa(G^\lambda) \leqslant q\lambda - 1$, so that $G^\lambda \in \mathcal{G}_2(n, q, \lambda)$. Thus, solving GRAPH-DIST(n, q, λ) on G^λ reveals x_{yz}. ◀

Our coloring lower bounds are straightforward consequences of the above lemma.

▶ **Theorem 20.** *Given a single randomized pass over a stream of edges of an n-vertex graph G, succeeding with probability at least $2/3$ at either of the following tasks requires $\Omega(n^2/\lambda^2)$ space, where $\lambda \geqslant 1$ is an integer parameter:*

(i) *produce a proper $(\kappa + \lambda)$-coloring of G;*

(ii) *produce an estimate $\hat{\kappa}$ such that $|\hat{\kappa} - \kappa| \leqslant \lambda$.*

Furthermore, if we require $\lambda = O\big(\kappa^{\frac{1}{2}-\gamma}\big)$, where $\gamma > 0$, then neither task admits a semi-streaming algorithm.

Proof. An algorithm for either task i and or task ii immediately solves GRAPH-DIST with appropriate parameters, implying the $\Omega(n^2/\lambda^2)$ bounds, thanks to Lemma 19. For the "furthermore" statement, note that the graphs in the family $\mathcal{G}_2$ constructed in the proof of Lemma 19 have $\kappa = \Theta(n)$, so performing either task with the stated guarantee on λ would require $\Omega(n^{1+2\gamma})$ space, which is not in $\widetilde{O}(n)$. ◀

▶ Remark. Together, Theorem 10 and Theorem 20 say that producing a $(\kappa + \kappa/\log^{O(1)} n)$-coloring is possible in semi-streaming space whereas producing a $(\kappa + O\big(\kappa^{\frac{1}{2}-O(1)}\big))$-coloring is not. We leave open the question of whether this gap can be tightened.

Second Flavor: Degeneracy Known in Advance. We now show that the coloring problem remains just as hard even if the algorithm knows the degeneracy of the graph before seeing the edge stream.

▶ **Theorem 21.** *Given as input an integer κ, followed by a stream of edges of an n-vertex graph G with degeneracy κ, a randomized one-pass algorithm that produces a proper $(\kappa + \lambda)$-coloring of G requires $\Omega(n^2/\lambda^2)$ bits of space. Furthermore, if we require $\lambda = O\big(\kappa^{\frac{1}{2}-\gamma}\big)$, where $\gamma > 0$, then the task does not admit a semi-streaming algorithm.*

Proof. We reduce from $\textsc{index}_N$, where $N = p^2$, using a plan analogous to the one used in proving Lemma 19. Alice and Bob will construct a graph on $n = 5\lambda p$ vertices, using their respective inputs $\mathbf{x} \in \{0,1\}^{p\times p}$ and $(y, z) \in [p] \times [p]$.

First, we consider the case $\lambda = 1$. We use the vertex set $L \uplus R \uplus \overline{L} \uplus \overline{R} \uplus C$, where $L = \{\ell_1, \dots \ell_p\}$, $R = \{r_1, \dots, r_p\}$, $\overline{L} = \{\overline{\ell}_1, \dots, \overline{\ell}_p\}$, $\overline{R} = \{\overline{r}_1, \dots, \overline{r}_p\}$, and $|C| = p$. Let $\overline{\mathbf{x}}$ be the bitwise complement of $\mathbf{x}$. Alice introduces the edges of the gadget graph $H_{\mathbf{x}}$ (from Definition 17), realized on $L \cup R$, and the edges of $H_{\overline{\mathbf{x}}}$ realized on $\overline{L} \cup \overline{R}$. For ease of notation, put $\ell := \ell_y$, $r := r_z$, $\overline{\ell} := \overline{\ell}_y$, $\overline{r} := \overline{r}_z$, and $S := C \cup \{\ell, r, \overline{\ell}, \overline{r}\}$. Bob introduces all possible edges within S, except for $\{\ell, r\}$ and $\{\overline{\ell}, \overline{r}\}$. Let G be the resulting graph (see Figure 1b).

We claim that the degeneracy $\kappa(G) = p + 2$. To prove this, we consider the case $x_{yz} = 1$ (the other case, $x_{yz} = 0$, is symmetric). By construction, G contains a clique on the $p + 3$ vertices in $C \cup \{\ell, r, \overline{\ell}\}$; therefore, by definition of degeneracy, $\kappa(G) \geqslant p + 2$. To show that $\kappa(G) \leqslant p + 2$, it will suffice to exhibit a vertex ordering $\lhd$ such that $\mathrm{odeg}_{G,\lhd}(v) \leqslant p + 2$ for all $v \in V(G)$. To this end, consider an ordering where $V(G) \setminus S \lhd \overline{\ell} \lhd S \setminus \{\overline{\ell}\}$ and the ordering within each set is arbitrary. Each vertex $v \in V(G) \setminus S$ has $\mathrm{odeg}_{G,\lhd}(v) \leqslant \deg(v) \leqslant p$ and each vertex $v \in S \setminus \{\overline{\ell}\}$ has $\mathrm{odeg}_{G,\lhd}(v) \leqslant \big|S \setminus \{\overline{\ell}\}\big| - 1 = p + 2$. As for the vertex $\overline{\ell}$, since $\overline{x}_{yz} = 1 - x_{yz} = 0$, by the construction in Definition 17, $\overline{r}$ is not a neighbor of $\overline{\ell}$; therefore, $\mathrm{odeg}_{G,\lhd}(\overline{\ell}) \leqslant \big|S \setminus \{\overline{\ell}, \overline{r}\}\big| = p + 2$.

Let $\mathcal{A}$ be a streaming algorithm that behaves as in the theorem statement. Recall that we are considering $\lambda = 1$. Since $\kappa(G) = p + 2$ for every instance of $\textsc{index}_N$, Alice and Bob can simulate $\mathcal{A}$ on their constructed graph G by first feeding it the number $p + 2$, then Alice's edges, and then Bob's. When $\mathcal{A}$ succeeds, the coloring it outputs is a proper $(p + 3)$-coloring; therefore it must repeat a color inside S, as $|S| = p + 4$. But S has exactly one pair of non-adjacent vertices: the pair $\{\ell, r\}$ if $x_{yz} = 0$, and the pair $\{\overline{\ell}, \overline{r}\}$ if $x_{yz} = 1$. Thus, an examination of which two vertices in S receive the same color reveals x_{yz}, solving the $\textsc{index}_N$ instance. It follows that $\mathcal{A}$ must use at least $\mathrm{R}^{\to}(\textsc{index}_N) = \Omega(N) = \Omega(p^2)$ bits of space.

Now consider an arbitrary λ. Alice and Bob proceed as above, except that they simulate $\mathcal{A}$ on the blow-up graph G^λ. Since G always has a $(p + 3)$-clique and $\kappa(G) = p + 2$, the two halves of Lemma 16 together imply $\kappa(G^\lambda) = (p + 3)\lambda - 1$. So, when $\mathcal{A}$ succeeds, it properly colors G^λ using at most $(p + 4)\lambda - 1$ colors. For each $A \subseteq V(G)$, abusing notation, let A^λ denote its corresponding set of vertices in G^λ (cf. Definition 15). Since $|S^\lambda| = (p + 4)\lambda$, there must be a color repetition within S^λ. Reasoning as above, this repetition must occur within $\{\ell, r\}^\lambda$ when $x_{yz} = 0$ and within $\{\overline{\ell}, \overline{r}\}^\lambda$ when $x_{yz} = 1$. Therefore, Bob can examine the coloring to solve $\textsc{index}_N$, showing that $\mathcal{A}$ must use $\Omega(N) = \Omega(p^2) = \Omega(n^2/\lambda^2)$ space.

The "furthermore" part follows by observing that $\kappa(G^\lambda) = \Theta\big(|V(G^\lambda)|\big)$. ◀

Query Lower Bounds. We prove lower bounds of the above two flavors for the graph query model as well. We describe the proofs in the full version of the paper [18].

▶ **Theorem 22.** *Given query access to an n-vertex graph G, succeeding with probability at least $2/3$ at either of the following tasks requires $\Omega(n^2/\lambda^2)$ queries, where $\lambda \geqslant 1$ is an integer:*

(i) *produce a proper $(\kappa + \lambda)$-coloring of G;*

(ii) *produce an estimate $\hat{\kappa}$ such that $|\hat{\kappa} - \kappa| \leqslant \lambda$.*

▶ **Theorem 23.** *Given an integer κ and query access to an n-vertex graph G with $\kappa(G) = \kappa$, an algorithm that, with probability $2/3$, produces a proper $(\kappa + \lambda)$-coloring of G must make $\Omega(n^2/\lambda^2)$ queries.*

Combinatorial Lower Bound. Finally, we prove a combinatorial result that shows that unlike $(\Delta + 1)$-coloring, an analogous Palette Sparsification Theorem (as in Assadi et al. [5]) doesn't exist for degeneracy-based coloring. Moreover, our result implies that an algorithm based on such a technique must use at least as many colors as Algorithm 1. We discuss this in detail in the full version of our paper [18].

References

1 Amir Abboud, Keren Censor-Hillel, Seri Khoury, and Ami Paz. Smaller cuts, higher lower bounds. *CoRR*, abs/1901.01630, 2019. `arXiv:1901.01630`.

2 Farid Ablayev. Lower bounds for one-way probabilistic communication complexity and their application to space complexity. *Theor. Comput. Sci.*, 175(2):139–159, 1996.

3 Noga Alon, László Babai, and Alon Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *Journal of algorithms*, 7(4):567–583, 1986.

4 Sabeur Aridhi, Martin Brugnara, Alberto Montresor, and Yannis Velegrakis. Distributed k-core decomposition and maintenance in large dynamic graphs. In *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems, DEBS '16, Irvine, CA, USA, June 20 - 24, 2016*, pages 161–168, 2016.

5 Sepehr Assadi, Yu Chen, and Sanjeev Khanna. Sublinear algorithms for $(\Delta + 1)$ vertex coloring. In *Proc. 30th Annual ACM-SIAM Symposium on Discrete Algorithms*, page To Appear, 2019.

6 Gary D. Bader and Christopher WV Hogue. An automated method for finding molecular complexes in large protein interaction networks. *BMC Bioinformatics*, 4(1):2, January 2003.

7 Bahman Bahmani, Ravi Kumar, and Sergei Vassilvitskii. Densest subgraph in streaming and mapreduce. *International Conference on Very Large Data Bases*, 5(5):454–465, 2012.

8 Balabhaskar Balasundaram and Sergiy Butenko. Graph domination, coloring and cliques in telecommunications. In *Handbook of Optimization in Telecommunications*, pages 865–890. Springer, 2006.

9 Luis Barba, Jean Cardinal, Matias Korman, Stefan Langerman, André van Renssen, Marcel Roeloffzen, and Sander Verdonschot. Dynamic graph coloring. *Algorithmica*, 81(4):1319–1341, 2019.

10 Leonid Barenboim. Deterministic $(\Delta + 1)$-coloring in sublinear (in Δ) time in static, dynamic, and faulty networks. *Journal of the ACM (JACM)*, 63(5):47, 2016.

11 Leonid Barenboim and Michael Elkin. Sublogarithmic distributed mis algorithm for sparse graphs using nash-williams decomposition. *Distributed Computing*, 22(5-6):363–379, 2010.

12 Leonid Barenboim and Michael Elkin. Deterministic distributed vertex coloring in polylogarithmic time. *Journal of the ACM (JACM)*, 58(5):23, 2011.

13 Leonid Barenboim and Michael Elkin. *Distributed Graph Coloring: Fundamentals and Recent Developments*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2013.

14 Leonid Barenboim, Michael Elkin, Seth Pettie, and Johannes Schneider. The locality of distributed symmetry breaking. *Journal of the ACM (JACM)*, 63(3):20, 2016.

15 Nicolas Barnier and Pascal Brisset. Graph coloring for air traffic flow management. *Annals of operations research*, 130(1-4):163–178, 2004.

16 Paul Beame, Paraschos Koutris, and Dan Suciu. Communication steps for parallel query processing. *Journal of the ACM (JACM)*, 64(6):40, 2017.

17 Soheil Behnezhad, Mahsa Derakhshan, and Mohammad Taghi Hajiaghayi. Brief announcement: Semi-mapreduce meets congested clique. *CoRR*, abs/1802.10297, 2018.

18 Suman K. Bera, Amit Chakrabarti, and Prantar Ghosh. Graph coloring via degeneracy in streaming and other space-conscious models. *CoRR*, abs/1905.00566, 2019. `arXiv:1905.00566`.

19 Sayan Bhattacharya, Deeparnab Chakrabarty, Monika Henzinger, and Danupon Nanongkai. Dynamic algorithms for graph coloring. In *Proc. 39th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1–20, 2018.

20 K. Bhawalkar, J. Kleinberg, K. Lewi, T. Roughgarden, and A. Sharma. Preventing unraveling in social networks: The anchored k-core problem. *SIAM Journal on Discrete Mathematics*, 29(3):1452–1475, 2015.

21 Gregory J Chaitin. Register allocation & spilling via graph coloring. In *ACM Sigplan Notices*, volume 17, pages 98–105, 1982.

22 Gregory J Chaitin, Marc A Auslander, Ashok K Chandra, John Cocke, Martin E Hopkins, and Peter W Markstein. Register allocation via coloring. *Computer languages*, 6(1):47–57, 1981.

23 Yi-Jun Chang, Manuela Fischer, Mohsen Ghaffari, Jara Uitto, and Yufan Zheng. The complexity of (δ+1) coloring in congested clique, massively parallel computation, and centralized local computation. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, PODC '19, page 471–480, 2019.

24 Yi-Jun Chang, Wenzheng Li, and Seth Pettie. An optimal distributed (Δ+ 1)-coloring algorithm. In *Proc. 50th Annual ACM Symposium on the Theory of Computing*, pages 445–456, 2018.

25 Bernard Chazelle, Ronitt Rubinfeld, and Luca Trevisan. Approximating the minimum spanning tree weight in sublinear time. *SIAM Journal on computing*, 34(6):1370–1379, 2005.

26 Fred C Chow and John L Hennessy. The priority-based coloring approach to register allocation. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 12(4):501–536, 1990.

27 Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *6th Symposium on Operating System Design and Implementation (OSDI 2004), San Francisco, California, USA, December 6-8, 2004*, pages 137–150, 2004.

28 P. Erdős and A. Hajnal. On chromatic number of graphs and set-systems. *Acta Mathematica Academiae Scientiarum Hungarica*, 17(1):61–99, March 1966.

29 Hossein Esfandiari, Silvio Lattanzi, and Vahab S. Mirrokni. Parallel and streaming algorithms for k-core decomposition. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 1396–1405, 2018.

30 Martín Farach-Colton and Meng-Tsung Tsai. Tight approximations of degeneracy in large graphs. In *LATIN 2016: Theoretical Informatics*, pages 429–440. Springer Berlin Heidelberg, 2016.

31 Uriel Feige and Joe Kilian. Zero knowledge and the chromatic number. In *Annual IEEE Conference on Computational Complexity*, page 278, 1996.

32 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Theor. Comput. Sci.*, 348(2–3):207–216, 2005. Preliminary version in *Proc. 31st International Colloquium on Automata, Languages and Programming*, pages 531–543, 2004.

33 Pierre Fraigniaud, Marc Heinrich, and Adrian Kosowski. Local conflict coloring. In *Proc. 57th Annual IEEE Symposium on Foundations of Computer Science*, pages 625–634, 2016.

34 Eugene C. Freuder. A sufficient condition for backtrack-free search. *J. ACM*, 29(1):24–32, 1982.

35 Mohsen Ghaffari and Christiana Lymouri. Simple and near-optimal distributed coloring for sparse graphs. In *31st International Symposium on Distributed Computing (DISC 2017)*, page 20, 2017.

36 Mohsen Ghaffari and Ali Sayyadi. Distributed Arboricity-Dependent Graph Coloring via All-to-All Communication. In *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 142:1–142:14, 2019.

37 Anna C. Gilbert and Piotr Indyk. Sparse recovery using sparse matrices. *Proceedings of the IEEE*, 98(6):937–947, 2010.

38 Oded Goldreich. *Introduction to Property Testing.* Cambridge University Press, 2017.

39 Oded Goldreich and Dana Ron. Approximating average parameters of graphs. *Random Structures & Algorithms*, 32(4):473–493, 2008.

40 David G Harris, Johannes Schneider, and Hsin-Hao Su. Distributed (Δ+ 1)-coloring in sublogarithmic rounds. In *Proc. 48th Annual ACM Symposium on the Theory of Computing*, pages 465–478, 2016.

41 Nicholas J. A. Harvey, Christopher Liaw, and Paul Liu. Greedy and local ratio algorithms in the mapreduce model. In *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures, SPAA 2018, Vienna, Austria, July 16-18, 2018*, pages 43–52, 2018.

42 Monika Henzinger, Stefan Neumann, and Andreas Wiese. Explicit and implicit dynamic coloring of graphs with bounded arboricity. *CoRR*, abs/2002.10142, 2020. arXiv:2002.10142.

43 Monika Henzinger and Pan Peng. Constant-time dynamic (Δ+1)-coloring. In *37th International Symposium on Theoretical Aspects of Computer Science, STACS 2020*, volume 154 of *LIPIcs*, pages 53:1–53:18, 2020.

44 Öjvind Johansson. Simple distributed Δ+ 1-coloring of graphs. *Information Processing Letters*, 70(5):229–232, 1999.

45 Subhash Khot and Ashok Kumar Ponnuswami. Better inapproximability results for maxclique, chromatic number and min-3lin-deletion. In *International Colloquium on Automata, Languages and Programming*, pages 226–237, 2006.

46 L. Kirousis and D. Thilikos. The linkage of a graph. *SIAM Journal on Computing*, 25(3):626–647, 1996.

47 Kishore Kothapalli and Sriram Pemmaraju. Distributed graph coloring in a few rounds. In *Proc. 30th ACM Symposium on Principles of Distributed Computing*, pages 31–40, 2011.

48 Frank Thomson Leighton. A graph coloring algorithm for large scheduling problems. *Journal of research of the national bureau of standards*, 84(6):489–506, 1979.

49 Vahid Lotfi and Sanjiv Sarin. A graph coloring algorithm for large scale scheduling problems. *Computers & operations research*, 13(1):27–32, 1986.

50 Zvi Lotker, Boaz Patt-Shamir, Elan Pavlov, and David Peleg. Minimum-weight spanning tree construction in $O(\log \log n)$ communication rounds. *SIAM J. Comput.*, 35(1):120–131, 2005.

51 Michael Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM J. Comput.*, 15(4):1036–1053, 1986.

52 Andrew McGregor, David Tench, Sofya Vorotnikova, and Hoa T Vu. Densest subgraph in dynamic graph streams. In *International Symposium on Mathematical Foundations of Computer Science*, pages 472–482, 2015.

53 Michael Mitzenmacher, Jakub Pachocki, Richard Peng, Charalampos Tsourakakis, and Shen Chen Xu. Scalable large near-clique detection in large-scale networks via sampling. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, 2015.

54 Farnaz Moradi, Tomas Olovsson, and Philippas Tsigas. A local seed selection algorithm for overlapping community detection. In *2014 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2014)*, pages 1–8, 2014.

55 Alessandro Panconesi and Aravind Srinivasan. On the complexity of distributed network decomposition. *Journal of Algorithms*, 20(2):356–374, 1996.

56 Taehoon Park and Chae Y Lee. Application of the graph coloring algorithm to the frequency assignment problem. *Journal of the Operations Research society of Japan*, 39(2):258–265, 1996.

57 Michal Parnas and Dana Ron. Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms. *Theoretical Computer Science*, 381(1-3):183–196, 2007.

58 Merav Parter. (Δ+1) coloring in the congested clique model. In *Proc. 45th International Colloquium on Automata, Languages and Programming*, pages 160:1–160:14, 2018.

59 Merav Parter and Hsin-Hao Su. Randomized (Delta+1)-Coloring in O(log* Delta) Congested Clique Rounds. In *Proc. 32nd International Symposium on Distributed Computing*, pages 39:1–39:18, 2018.

60 Jaikumar Radhakrishnan, Saswata Shannigrahi, and Rakesh Venkat. Hypergraph two-coloring in the streaming model. *arXiv preprint*, 2015. arXiv:1512.04188.

61 Ryan A. Rossi and Nesreen K. Ahmed. The network data repository with interactive graph analytics and visualization. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015. URL: `http://networkrepository.com`.

62 Václav Rozhon and Mohsen Ghaffari. Polylogarithmic-time deterministic network decomposition and distributed derandomization. In *Proc. 52nd Annual ACM Symposium on the Theory of Computing*, 2020. `arXiv:1907.10937`.

63 Johannes Schneider and Roger Wattenhofer. A new technique for distributed symmetry breaking. In *Proc. 29th ACM Symposium on Principles of Distributed Computing*, pages 257–266, 2010.

64 Shay Solomon and Nicole Wein. Improved dynamic graph coloring. In *26th Annual European Symposium on Algorithms, ESA 2018*, volume 112 of *LIPIcs*, pages 72:1–72:16, 2018.

65 Michael Stiebitz and Bjarne Toft. A Brooks type theorem for the maximum local edge connectivity. *Electronic Journal of Combinatorics*, 25, March 2016.

66 George Szekeres and Herbert S. Wilf. An inequality for the chromatic number of a graph. *Journal of Combinatorial Theory*, 4(1):1–3, 1968.

67 Simon Thevenin, Nicolas Zufferey, and Jean-Yves Potvin. Graph multi-coloring for a job scheduling application. *Discrete Applied Mathematics*, 234:218–235, 2018.

68 David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. In *Proc. 38th Annual ACM Symposium on the Theory of Computing*, pages 681–690, 2006.

Improved Bounds for Matching in Random-Order Streams

Aaron Bernstein
Rutgers University, Department of Computer Science, New Brunswick, NJ, USA
https://aaronbernstein.cs.rutgers.edu
bernstei@gmail.com

Abstract

We study the problem of computing an approximate maximum cardinality matching in the semi-streaming model when edges arrive in a *random* order. In the semi-streaming model, the edges of the input graph $G = (V, E)$ are given as a stream $e_1, \ldots, e_m$, and the algorithm is allowed to make a single pass over this stream while using $O(n\text{polylog}(n))$ space ($m = |E|$ and $n = |V|$). If the order of edges is adversarial, a simple single-pass greedy algorithm yields a 1/2-approximation in $O(n)$ space; achieving a better approximation in adversarial streams remains an elusive open question.

A line of recent work shows that one can improve upon the 1/2-approximation if the edges of the stream arrive in a random order. The state of the art for this model is two-fold: Assadi et al. [SODA 2019] show how to compute a $\frac{2}{3}(\sim .66)$-approximate matching, but the space requirement is $O(n^{1.5}\text{polylog}(n))$. Very recently, Farhadi et al. [SODA 2020] presented an algorithm with the desired space usage of $O(n\text{polylog}(n))$, but a worse approximation ratio of $\frac{6}{11}(\sim .545)$, or $\frac{3}{5}(= .6)$ in bipartite graphs.

In this paper, we present an algorithm that computes a $\frac{2}{3}(\sim .66)$-approximate matching using only $O(n\log(n))$ space, improving upon both results above. We also note that for adversarial streams, a lower bound of Kapralov [SODA 2013] shows that any algorithm that achieves a $1 - \frac{1}{e}(\sim .63)$-approximation requires $(n^{1+\Omega(1/\log\log(n))})$ space. Our result for random-order streams is the first to go beyond the adversarial-order lower bound, thus establishing that computing a maximum matching is provably easier in random-order streams.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases Graph Algorithms, Sublinear Algorithms, Matching, Streaming

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.12

Category Track A: Algorithms, Complexity and Games

Funding *Aaron Bernstein*: NSF CAREER Grant 1942010 and Simons Collaboration on Algorithms and Geometry.

Acknowledgements I want to thank Sepehr Assadi for several very helpful discussions.

1 Introduction

Computing a maximum cardinality matching is a classical problem in combinatorial optimization, with a large number of algorithms and applications. Motivated by the rise of massive graphs, much of the recent research on this problem has focused on sub-linear algorithms that are able to compute a matching without storing the entire graph in memory. One of the standard sub-linear models for processing graphs is known as the *semi-streaming* model [17]: the algorithm has access to a sequence of edges (the stream), and is allowed to make a single pass over this sequence while only using only $O(n\text{polylog}(n))$ internal memory, where n is the number of vertices in the graph. Note that the memory used is still significantly smaller than the number of edges in the graph, and that $O(n)$ memory is also necessary if we want the algorithm to output the actual edges of the matching. (One typically assume

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 12; pp. 12:1–12:13
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Table 1 Single-pass semi-streaming algorithms known for the maximum matching when edges arrive in a random order. The space bounds are expressed in terms of $O(\log(n))$-size words, though many existing results do not state the exact polylog(n) term. The result of Gamlath et al. [18] works in weighted graphs; all others are restricted to unweighted graphs.

	Approximation Factor		
	Bipartite graphs	**General graphs**	**Space**
Konrad et al. [28]	0.5005	0.5003	$O(n)$
Gamlath et al. [18]	0.512	0.506	$O(n \cdot \text{polylog}(n))$
Konrad [27]	0.539	-	$O(n \cdot \text{polylog}(n))$
Assadi et al. [3]	0.666	0.666	$O(n^{1.5} \cdot \text{polylog}(n))$
Farhadi et al. [16]	0.6	0.545	$O(n \cdot \text{polylog}(n))$
This paper	**0.666**	**0.666**	$O(n\log(n))$

$O(\log(n))$-size words, so that a single edge can be stored in $O(1)$ space; if one were to express space in terms of the number of bits, all the space bounds in this paper would increase by a $O(\log(n))$ factor.)

If the edges of the stream arrive in an arbitrary order, a simple greedy algorithm can compute a maximal matching – and hence a 1/2-approximate maximum matching – in a single streaming pass and $O(n)$ space. Going beyond a 1/2-approximation with a single pass is considered one of the main open problems in the area. The strongest lower bound is by Kapralov [23], who build upon an earlier lower bound of Goel et al. [20]: any algorithm with approximation ratio $\geq 1 - 1/e \sim .63$ requires $n^{1+\Omega(1/\log\log(n))})$ space [23]. But we still do not know where the right answer lies between $1/2$ and $1 - 1/e$.

To make progress on this intriguing problem, several recent papers studied a more relaxed model, where the graph is still arbitrary, but the edges are assumed to arrive in a *uniformly random* order. Konrad et al. were the first to go beyond a 1/2-approximation in this setting: they showed that in random-order streams, there exists an $O(n)$-space algorithm that computes an .5003-approximate matching, or .5005-approximate for bipartite graphs [28]. This was later improved to .506 in general graphs [18] and .539 in bipartite graphs [27]. Assadi et al. then showed an algorithm with an approximation ratio of $(2/3 - \epsilon) \sim .66$, but their algorithm had a significantly larger space requirement of $O(n^{1.5}\text{polylog}(n))$ [3]. Finally, very recently (SODA 2020), Farhadi et al. achieved the current state of the art for $O(n\text{polylog}(n))$ space; their algorithm achieves an approximation ratio of $6/11 \sim .545$ for general graphs and $3/5 = .6$ for bipartite graphs [16]. A summary of these results can be found in Table 1.

Although this line of work suggests that computing a maximum matching might be fundamentally easier in random-order streams, we note that even in bipartite graphs, none of the previous results go beyond the best known lower bound for adversarial streams mentioned above [23]: the algorithm of Assadi et al. uses too much space ($n^{1.5} \gg n^{1+1/\log\log(n)}$), while the result of Farhadi et al. has an approximation ratio of $.6 < 1 - 1/e$.

Our result is the first to go beyond the adversarial-order lower bound, thus establishing that computing a matching is provably easier in random-order streams.

▶ **Theorem 1** (Our Result). *Given any (possibly non-bipartite) graph G and any approximation parameter $1 > \epsilon > 0$, there exists a deterministic single-pass streaming algorithm that with high probability computes a $(2/3 - \epsilon)$-approximate matching if the edges of G arrive in a uniformly random order. The space usage of the algorithm is $O(n\log(n)poly(\epsilon^{-1}))$.*

Our result significantly improves upon the space requirement of Assadi et al. [3] and the approximation ratio of Farhadi et al. [16]. In fact, our algorithm achieves the best of both those results (see Table 1). On top of that, our result is quite simple; given that it improves upon a sequence of previous results, we see this simplicity as a plus.

Related Work

If the only requirement is to return an approximate estimate of the *size* of the maximum matching, rather than the actual edges, a surprising result by Kapralov et al. shows that one can get away with very little space: given a single pass over a random-order stream, it is possible to estimate the size within a $1/\text{polylog}(n)$ factor using only $\text{polylog}(n)$ space [24]; a very recent improvement reduces the polylog factors to $O(\log^2(n))$ [25]. There is also a line of work that estimates the size of the matching in $o(n)$ space in *adversarial* streams for special classes of graphs as such planar graphs or low-arboricity graphs [14, 8, 30, 11, 31].

There are many one-pass streaming algorithms for computing a maximum matching in *weighted* graphs. For adversarial-order streaming, a long line of work culminated in a $(1/2-\epsilon)$-approximation using $O(n)$ space [17, 29, 13, 12, 32, 19]. Gamlath et al. recently showed that for random-order streams, one can achieve an approximation ratio of $1/2 + \Omega(1)$, [18]. See also other related work on weighted graphs in [8].

There are several results on upper and lower bounds for computing a maximum matching in dynamic streams (where edges can also be deleted) [26, 10, 6, 9, 5]. Finally, there are several results that are able to achieve better bounds by allowing the algorithm to make multiple passes over the stream: some results focus on just two or three passes [28, 15, 22, 27], while others seek to compute a $(1-\epsilon)$-approximate matching by allowing a large constant number (or even $\log(n)$) passes [29, 1, 21, 2].

Overview of Techniques

The basic greedy algorithm trivially achieves a 1/2-approximate matching in adversarial streams; in fact, Konrad et al later showed that the ratio remains $1/2 + o(1)$ even in random-order streams [28]. Existing algorithms for improving the 1/2 ratio in random-order streams generally fall into two categories. The algorithms in [28, 27, 18, 16] use the randomness of the stream to compute some fraction of short augmenting paths, thus going beyond the 1/2-approximation of a maximal matching. The result in [3] instead shows that one can obtain a large matching by constructing a subgraph that obeys certain degree-properties.

Our result follows the framework of [3]. Given any graph G, an earlier result of Bernstein and Stein for fully dynamic matching defined the notion of an edge-degree constrainted subgraph (denoted EDCS), which is a sparse subgraph $H \subseteq G$ that obeys certain degree-properties [7]. They showed that any EDCS H always contains a $(2/3-\epsilon)$-approximate matching. The streaming result of Assadi et al. [3] then showed that given a random-order stream, it is possible to compute an EDCS H in $O(n^{1.5})$ space; returning the maximum matching in H yields a $(2/3-\epsilon)$-approximate matching in G.

Our result also takes the EDCS as its starting point, but it is unclear how to compute an EDCS H of G using less than $O(n^{1.5})$ space. Our algorithm requires two new contributions. Firstly, we show that it is sufficient for H to satisfy a somewhat relaxed set of properties. Our main contribution is then to use an entirely different construction of this relaxed subgraph, which uses the randomness of the stream more aggressively to compute H using low space.

2 Notation and Preliminaries

Consider any graph $H = (V_H, E_H)$. We define $\deg_H(v)$ to be the degree of v in H and we define the degree of an edge (u, v) to be $\deg_H(u) + \deg_H(v)$. A matching M in H is a set of vertex-disjoint edges. All graphs in this paper are unweighted and undirected. We use $\mu(H)$ to denote the size of the maximum matching in H. Unless otherwise indicated, we let $G = (V, E)$ refer to the input graph and let $n = |V|$ and $m = |E|$. We note that every graph referred to in the paper has the same vertex V as the input graph; when we refer to subgraphs, we are always referring to a subset of edges on this same vertex set.

The input graph $G = (V, E)$ is given as a stream of edges $S = \langle e_1, \dots, e_m \rangle$. We assume that the permutation $(e_1, \dots, e_m)$ of the edges is chosen *uniformly at random* among all permutations of E. We use $S_{[i,j]}$ to denote the substream $\langle e_i, \dots, e_j \rangle$, and we use $G_{>i} \subseteq G$ to denote the subgraph of G containing all edges in $\{e_{i+1}, \dots, e_m\}$.

Our analysis will apply concentration bounds to segments $S_{[i,j]}$ of the stream. Observe that because the stream is a random permutation, any segment $S_{[i,j]}$ is equivalent to sampling $j - 1 + 1$ edges from the stream without replacement. We can thus apply the Chernoff bound for negatively associated variables (see e.g. the primer in [33]).

▶ **Theorem 2** (Chernoff). *Let $X_1, \dots X_n$ be negatively associated random variables taking values in $[0, 1]$. Let $X = \sum X_i$ and let $\mu = \mathbb{E}[X]$. Then, for any $0 < \delta < 1$ we have*

$$\Pr[X \le \mu(1-\delta)] \le \exp(\frac{-\mu \cdot \delta^2}{2}),$$

and

$$\Pr[X \ge \mu(1+\delta)] \le \exp(\frac{-\mu \cdot \delta^2}{3})$$

The early and late sections of the stream

Our algorithm will use the first ϵm edges of the stream to learn about the graph and will effectively ignore them for the purposes of analyzing the maximum matching. Thus, we only approximate the maximum matching in the later $(1-\epsilon)m$ edges of stream; because the stream is random, these edges still contain a large fraction of the maximum matching. We use the following definitions and lemmas to formalize this intuition.

▶ **Definition 3.** *We Let E^{early} denote the first ϵm edges of the stream, and E^{late} denote the rest: that is, $E^{early} = \{e_1, \dots, e_{\epsilon m}\}$, and $E^{late} = \{e_{\epsilon m+1}, \dots, e_m\}$. Define $G^{early} = (V, E^{early})$ and $G^{late} = (V, E^{late}) = G_{>\epsilon m}$.*

For the probability bounds to work out, we need to assume that $\mu(G) \ge 20 \log(n)\epsilon^{-2}$. We justify this assumption by observing that every graph G satisfies $m \le 2n\mu(G)$, so if $\mu(G) < 20\log(n)\epsilon^{-2}$, then the algorithm can trivially return an exact maximum matching by simply storing every edge using only $O(m) = O(\log(n)\epsilon^{-2})$ space. This justifies the following:

▷ Claim 4 (Assumption). We can assume for the rest of the paper that $\mu(G) \ge 20\log(n)\epsilon^{-2}$.

Combining Claim 4 with Chernoff bound we get the following lemma, which allows us to focus our analysis on the edges in G^{late}.

▶ **Lemma 5.** *Assuming that $\epsilon < 1/2$, we have that $\Pr[\mu(G^{late}) \ge (1-2\epsilon)\mu(G)] \ge 1 - n^{-5}$.*

Proof. Fix some maximum matching $M = (f_1, ..., f_{\mu(G)})$ of G. Define X_i to be the indicator variable that edge $f_i \in M$ appears in G^{late}. Since the stream is random, and since G^{late} contains exactly $(1-\epsilon)m$ edges, we have that $\mathbb{E}[X_i] = (1-\epsilon)$ and $\sum \mathbb{E}[X_i] = (1-\epsilon)\mu(G)$. It is also easy to see that the X_i are negatively associated, since these variables correspond to sampling $(1-\epsilon)m$ edges without replacement. Recall from Claim 4 that we assume $\mu(G) \geq 20\log(n)\epsilon^{-2}$. Applying the Chernoff Bound in Theorem 2 completes the proof. ◀

Existing Work on EDCS

We now review the basic facts about the edge-degree constrained subgraph (EDCS), which was first introduced in [7].

▶ **Definition 6.** *Let $G = (V, E)$ be a graph, and $H = (V, E_H)$ a subgraph of G. Given any parameters $\beta \geq 2$ and $\lambda < 1$, we say that H is a (β, λ)-EDCS of G if H satisfies the following properties:*

- [Property P1]. *For any edge $(u, v) \in H$, $\deg_H(u) + \deg_H(v) \leq \beta$*
- [Property P2]. *For any edge $(u, v) \in G \setminus H$, $\deg_H(u) + \deg_H(v) \geq \beta(1-\lambda)$.*

The crucial fact about the EDCS is that it always contains a (almost) 2/3-approximate matching. The simplest proof of Lemma 7 below is in Lemma 3.2 of [4].

▶ **Lemma 7** ([4])**.** *Let $G(V, E)$ be any graph and $\epsilon < 1/2$ be some parameter. Let λ, β be parameters with $\lambda \leq \frac{\epsilon}{64}$, $\beta \geq 8\lambda^{-2}\log(1/\lambda)$. Then, for any $(\beta, \lambda) - EDCS$ H of G, we have that $\mu(H) \geq (\frac{2}{3} - \epsilon)\mu(G)$. (Note that the final guarantee is stated slightly differently than in Lemma 3.2 of [4], and to ensure the two are equivalent, we set λ to be a factor of two smaller than in Lemma 3.2 of [4].)*

3 Our Modified Subgraph

Unlike the algorithm of [3], we do not actually construct an EDCS of G, as we do not know how to do this in less than $O(n^{1.5})$ space. We instead rely on a more relaxed set of properties, which we analyze using Lemma 7 as a black-box. We now introduce some of the basic new tools used by our algorithm. Note that graph G in the lemma and definitions below crucially refers to any arbitrary graph G, and not necessarily the main input graph of the streaming algorithm.

▶ **Definition 8.** *We say that a graph H has bounded edge-degree β if for every edge $(u, v) \in H$, $\deg_H(u) + \deg_H(v) \leq \beta$.*

▶ **Definition 9.** *Let G be any graph, and let H be a subgraph of G with bounded edge-degree β. For any parameter $\lambda < 1$, we say that an edge $(u, v) \in G \setminus H$ is (G, H, β, λ)-underfull if $\deg_H(u) + \deg_H(v) < \beta(1-\lambda)$*

The two definitions above effectively separate the two EDCS properties: any subgraph H of G with bounded edge-degree β automatically satisfies property P1 of an EDCS, and underfull edges are then those that violate property P2. We now show that one can always construct a large matching from the combination of these two parts.

▶ **Lemma 10.** *Let $\epsilon < 1/2$ be any parameter, and let λ, β be parameters with $\lambda \leq \frac{\epsilon}{128}$, $\beta \geq 16\lambda^{-2}\log(1/\lambda)$. Consider any graph G, and any subgraph H with bounded edge-degree β. Let X contain all edges in $G \backslash H$ that are (G, H, β, λ)-underfull. Then $\mu(X \cup H) \geq (2/3 - \epsilon)\mu(G)$*

Proof. Note that it is NOT necessarily the case that $H \cup X$ is an EDCS of G, because adding the edges of X to H will increase vertex and edge degrees in H, so $H \cup X$ might not satisfy property P1 of an EDCS. We thus need a more careful argument.

Let M_G be the maximum matching in G, let $M_G^H = M_G \cap H$ and let $M_G^{G \setminus H} = M_G \cap (G \setminus H)$. Let $X^M = X \cap M_G^{G \setminus H}$. Note that by construction, $M_G \subseteq H \cup M_G^{G \setminus H}$, so $\mu(H \cup M_G^{G \setminus H}) = \mu(G)$.

We now complete the proof by showing that $H \cup X^M$ is a $(\beta + 2, 2\lambda)$-EDCS of $H \cup M_G^{G \setminus H}$. Let us start by showing property P2. Recall that X contains all edges (u, v) in $G \setminus H$ for which $\deg_H(u) + \deg_H(v) < \beta(1 - \lambda)$, so by construction X^M contains all such edges in $M_G^{G \setminus H}$. Thus, every edge $(u, v) \in (H \cup M_G^{G \setminus H}) \setminus (H \cup X^M) = M_G^{G \setminus H} \setminus X^M$ must have $\deg_H(u) + \deg_H(v) \geq \beta(1 - \lambda) \geq (\beta + 2)(1 - 2\lambda)$, where the last inequality is just rearranging the algebra to fit Property P2 for our new EDCS parameters of $\beta + 2, 2\lambda$.

For property P1, note that $X^M \subseteq M_G^{G \setminus H}$ is a matching, so for every vertex v we have $\deg_H(v) \leq \deg_{H \cup X^M}(v) \leq \deg_H(v) + 1$. Now, for $(u, v) \in H$ we had $\deg_H(u) + \deg_H(v) \leq \beta$ (by property P1 of H), and for $(u, v) \in X^M \subseteq X$ we had $\deg_H(u) + \deg_H(v) < \beta$ (by definition of X). Thus, for every $(u, v) \in H \cup X^M$ we have that $\deg_{H \cup X^M}(u) + \deg_{H \cup X^M}(v) \leq \deg_H(u) + \deg_H(v) + 2 \leq \beta + 2$.

Note that because of how we set the parameters, $\beta' = \beta + 2 < 2\beta$ and $\lambda' = 2\lambda$ satisfy the requirements of Lemma 7. We thus have that $\mu(H \cup X) \geq \mu(H \cup X^M) \geq (2/3 - \epsilon)\mu(H \cup M_G^{G \setminus H}) = (2/3 - \epsilon)\mu(G)$. ◀

4 The Algorithm

4.1 The Two Phases

Our algorithm will proceed in two phases. Once phase I terminates, the algorithm proceeds to phase II and never returns to phase I. The goal of phase I is to construct a suitable subgraph H of G. We now state the formal properties that will be guaranteed by phase I.

▶ **Definition 11** (parameters). *Throughout this section we use the following parameters. Let $\epsilon < 1/2$ be the final approximation parameter we are aiming for. Set $\lambda = \frac{\epsilon}{128}$ and set $\beta = 16\lambda^{-2} \log(1/\lambda)$; note that λ and β are $O(poly(1/\epsilon))$. Set $\alpha = \frac{\epsilon m}{n\beta^2 + 1} = O(\frac{m}{n} poly(1/\epsilon))$ and $\gamma = 5 \log(n) \frac{m}{\alpha} = O(n \log(n) poly(1/\epsilon))$.*

▶ **Lemma 12.** *Phase I uses $O(n\beta) = O(npoly(1/\epsilon))$ space and constructs a subgraph H of G. The phase satisfies the following properties:*

1. *Phase I terminates within the first ϵm edges of the stream. That is, Phase I terminates at the end of processing some edge e_i with $i \leq \epsilon m$.*
2. *When Phase I terminates at the end of processing some edge e_i, the subgraph $H \subseteq G$ constructed during this phase satisfies the following properties:*
   a. *H has bounded edge-degree β. As a corollary, H has $O(n\beta)$ edges.*
   b. *With probability at least $1 - n^{-3}$, the total number of $(G_{>i}, H, \beta, \lambda)$-underfull edges in $G_{>i} \setminus H$ is at most γ. (Recall that $G_{>i}$ denotes the subgraph of G that contains all edges in $\{e_{i+1}, \ldots, e_m\}$.)*

We now show that if we can ensure the properties of Lemma 12, our main result follows.

Proof of Theorem 1. Let us say that Phase I terminates after edge e_i and let H be the subgraph constructed by Phase I. Phase II of the algorithm proceeds as follows. It initializes an empty set X. Then, for every edge (u, v) in $S_{[i+1,m]}$, if $\deg_H(u) + \deg_H(v) < \beta(1 - \lambda)$ (that is, if (u, v) is $(G_{>i}, H, \beta, \lambda)$-underfull), the algorithm adds edge (u, v) to X. After the algorithm completes the stream, it then returns the maximum matching in $H \cup X$.

Let us now analyze the approximation ratio. By property 1 of Lemma 12, $G_{>i} \subseteq G^{late}$; thus, X contains all $(G^{late}, H, \beta, \lambda)$-underfull edges. By property 2a, H has bounded edge-degree β. Thus, applying Lemma 10, we have that $\mu(H) \geq (2/3 - \epsilon)\mu(G^{late})$. Combining this with Lemma 5, we get that $\mu(H) \geq (2/3 - \epsilon)(1 - 2\epsilon)\mu(G) \geq (2/3 - 3\epsilon)\mu(G)$; using $\epsilon' = \epsilon/3$ thus yields the desired approximation ratio.

For the space analysis, we know from Lemma 12 that Phase I requires $O(n\beta)$ space, which is the space needed to store subgraph H. By Property 2b, the size of X in Phase II is at most $O(n \log(n))$. The overall space is thus $O(n\log(n) + n\beta) = O(n\log(n) + n\text{poly}(1/\epsilon))$.

Finally, note that the only two probabilistic claims are Lemma 5 and Property 2b of Lemma 12, both of which hold with probability $\geq 1 - n^{-3}$. A union bound thus yields an overall probability of success $\geq 1 - 2n^{-3}$. ◀

4.2 Description of Phase I

All we have left is to describe Phase I and prove Lemma 12. See Algorithm 1 for pseudocode of the entire algorithm. Recall the parameters $\epsilon, \beta, \lambda, \alpha, \gamma$ from Definition 11. Phase I is split into epochs, each containing exactly α edges from the stream. So in epoch i, the algorithm looks at $S_{[(i-1)\alpha+1, i\alpha]}$.

Phase I initializes the graph $H = \emptyset$. In epoch i, the algorithm goes through the edges of $S_{[(i-1)\alpha+1,i\alpha]}$ one by one. For edge (u,v), if $\deg_H(u) + \deg_H(v) < (1-\lambda)\beta$, then the algorithm adds edge (u,v) to H (Line 5). (Note that the algorithm changes H over time, so $\deg_H(u)+\deg_H(v)$ always refers to the degrees in H at the time edge (u,v) is being examined.) After each edge insertion to H, the algorithm runs procedure *RemoveOverfullEdges(H)* (Line 7); this procedure repeatedly picks an edge (x,y) with $\deg_H(x) + \deg_H(y) > \beta$ until no such edge remains. Note that as a result, our algorithm preserves the invariant that H always has bounded edge-degree β.

In each epoch, the algorithm also has a single boolean FoundUnderfull, which is set to True if the algorithm ever adds an edge to H during that epoch. At the end of the epoch, if FoundUnderfull is set to True, then the algorithm simply proceeds to the next epoch. If FoundUnderfull is False, then the algorithm permanently terminates Phase I and proceeds to Phase II. (The intuition is that since the ordering of the stream is random, if the algorithm failed to find an underfull edge in an entire epoch, then there must be relatively few underfull edges left in the stream, so Property 2b of Lemma 12 will be satisfied.)

Note that FoundUnderfull being false is the only way Phase I can terminate (Line 9); we prove in the analysis that this deterministically occurs within the first ϵm edges of the stream.

4.3 Analysis

We now turn to proving Lemma 12. The hardest part is proving Property 1. Observe that every epoch that doesn't terminate Phase I must add at least one edge to H. To prove Property 1, we use an auxiliary lemma that bounds the total number of changes made to H.

▶ **Lemma 13.** *Fix any parameter $\beta > 2$. Let $H = (V_H, E_H)$ be a graph, with E_H initially empty. Say that an adersary adds and removes edges from H using an arbitrary sequence of two possible moves*

- [Deletion Move]. *Remove an edge (u,v) from H for which $\deg_H(u) + \deg_H(v) > \beta$*
- [Insertion Move]. *Add an edge (u,v) to H for some pair $u,v \in V$ for which $\deg_H(u) + \deg_H(v) < \beta - 1$.*

Then, after $n\beta^2$ moves, no legal move remains.

Algorithm 1 The algorithm for computing a matching in a random-order stream. After initialization, the algorithm goes to Phase I. Once the algorithm exits Phase I, it moves on to Phase II and never returns to Phase I. Line 9 is the only place where the algorithm can exit Phase I.

```
Procedure Initilization
    Initialize $H = \emptyset$        /* $H$ is a global variable modified by Phase I */
    Let $\epsilon < 1/2$ be the main approximation parameter
    Set $\lambda = \frac{\epsilon}{128}$, $\beta = 16\lambda^{-2}\log(1/\lambda)$, $\alpha = \frac{\epsilon m}{n\beta^2+1}$, $\gamma = 5\log(n)\frac{m}{\alpha}$ (Definition 11).
    Go To Phase I
Procedure Phase I
    Do Until Termination  /* each iteration corresponds to one epoch */
        (1) FoundUnderfull ← FALSE
        (2) for $\alpha$ Iterations: do  /* each epoch looks at exactly $\alpha$ edges.  */
            (3) Let $(u, v)$ be the next edge in the stream
            (4) if $\deg_H(u) + \deg_H(v) < \beta(1-\lambda)$ then
                (5) Add edge $(u, v)$ to $H$   /* note: this increases $\deg_H(u)$ and
                  $\deg_H(v)$.  */
                (6) FoundUnderfull ← TRUE
                (7) RemoveOverfullEdges(H)
        (8) if FoundUnderfull = FALSE then
            (9) Go To Phase II        /* permanently exit Phase I. */ ;
        /* Else, will move on to the next epoch of Phase I.          */
Procedure RemoveOverfullEdges(H)
    (1) while there exists $(u, v) \in H$ such that $\deg_H(u) + \deg_H(v) > \beta$ do
        (2) Remove $(u, v)$ from $H$         /* note: this decreases $\deg_H(u)$ and
          $\deg_H(v)$ */
     /* note: when the while loop terminates, $H$ is guaranteed to have
    bounded edge-degree $\beta$.  */
Procedure Phase II
    (1) Initialize $X \leftarrow \emptyset$       /* all underfull edges will be added to $X$ */ ;
    (2) foreach remaining edge $(u, v)$ in the stream do
        (3) if $\deg_H(u) + \deg_H(v) < \beta(1-\lambda)$ then
            (4) Add edge $(u, v)$ to $X$        /* note: this does NOT change any
              $\deg_H(v)$.  */
    (5) Return the maximum matching in $H \cup X$ ;
```

Proof. The proof is similar to that of Proposition 2.4 in [4]. Define the following potential functions $\Phi_1(H) = (\beta - 1/2) \cdot \sum_{v \in V_H} \deg_H(v)$, $\Phi_2(H) = \sum_{(u,v) \in E_H} \deg_H(u) + \deg_H(v)$, and the main potential function $\Phi(H) = \Phi_1(H) - \Phi_2(H)$. Note that initially H is empty so $\Phi(H) = 0$. We claim that at all times $\Phi(H) \leq \Phi_1(H) \leq n\beta^2$. To see this, note that every vertex $v \in V_H$ always has $\deg_H(v) \leq \beta$, because as long as $\deg_H(v) = \beta$, the adversary cannot perform any insertion moves incident to v. In the rest of the proof, we show that every Insertion/Deletion move increases $\Phi(H)$ by at least 1; combined with the fact that at all times $0 \leq \Phi(H) \leq n\beta^2$, we get that there are at most $n\beta^2$ moves in total.

Consider any Deletion Move of edge (u,v). Clearly $\Phi_1(v)$ decreases by exactly $2\beta - 1$. We now show that $\Phi_2(v)$ decreases by at least 2β. One the one hand, $\Phi_2(v)$ decreases by at least $\beta + 1$ because edge (u,v) no longer participates in the sum, and $\deg_H(u) + \deg_H(v)$ was $> \beta$ before the deletion. But at the same time, since $\deg_H(u) + \deg_H(v) \geq \beta + 1$ before the deletion, there are at least $\beta - 1$ edges other than (u,v) incident to u or v, and each of their edge degrees decrease by 1 in the sum for $\Phi_2(H)$. Thus, $\Phi_2(H)$ decreases by at least $\beta + 1 + (\beta - 1) = 2\beta$, while $\Phi_1(H)$ decreases by exactly $2\beta - 1$, so overall $\Phi(H) = \Phi_1(H) - \Phi_2(H)$ increases by at least one.

Similarly, consider any Insertion Move of edge (u,v). Clearly $\Phi_1(v)$ increases by exactly $2\beta-1$. We now show that $\Phi_2(v)$ increases by at most $2\beta-2$. Recall that $\deg_H(u)+\deg_H(v) \leq \beta - 2$ before the insertion, so after the insertion we have that $\deg_H(u) + \deg_H(v) \leq \beta$, so the edge (u,v) itself contributes at most β to the sum in Φ_2. There are also at most $\beta - 2$ edges other than (u,v) incident to u or v, each of whose edge degrees increases by 1. Thus, overall, $\Phi_2(H)$ increases by at most $\beta + (\beta - 2) = 2\beta - 2$, so $\phi(H)$ increases by at least $(2\beta - 1) - (2\beta - 2) = 1$. ◀

Proof of Lemma 12. Property 2a is clearly satisfied by construction, because after any insertion to H the algorithm runs *RemoveOnderfullEdges(H)* (line 7) to ensure that H has bounded edge-degree β. As a result, we clearly have that every *vertex* degree is at most β, so Phase I needs only $O(n\beta)$ space to store H.

For the proof of Property 1, observe that any changes the algorithm makes to H follow the rules for Insertion/Deletion moves from Lemma 13, so Algorithm 1 makes at most $n\beta^2$ changes to H. (Line 5 of Phase I corresponds to deletion moves in Lemma 13, while line 2 of *RemoveOverfullEdges(H)* corresponds to insertion moves. Note that line 5 of phase I actually obeys an even stronger inequality than deletion moves, since $\beta(1-\lambda) < \beta - 1$.) Each epoch that does not terminate Phase I makes at least one change to H, so phase I goes through at most $n\beta^2 + 1$ epochs before termination. Each epoch contains α edges, so overall Phase I goes through at most $\alpha(n\beta^2 + 1) = \epsilon m$ edges, as desired.

All that remains is to prove Property 2b. As mentioned above, the intuition is simple: the algorithm only exits Phase I if it fails to find a single underfull edge in the entire epoch (Line 9), and since the stream is random, such an event implies that there are probably relatively few underfull edges left in the stream. We now formalize this intuition.

Let $\mathcal{A}_i$ be the event that FoundUnderfull is set to FALSE in epoch i. Recall that epoch i ends on edge $e_{i\alpha}$; let $\mathcal{B}_i$ be the event that the number of $(G_{>i\alpha}, H, \beta, \lambda)$-underfull edges is more than γ. Note that Property 2b fails to hold if and only if we have $\mathcal{A}_i \wedge \mathcal{B}_i$ for some i, so we now upper bound $\Pr[A_i \wedge B_i]$. Our bound relies on the randomness of the stream. Let E_i^r contain all edges in the graph that have not yet appeared in the stream at the *beginning* of epoch i (r for remaining). Let E_i^e be the edges that appear in epoch i (e for epoch), and note that E_i^e is a subset of size α chosen uniformly at random from E_i^r. Define H_i to be the subgraph H at the beginning of epoch i, and define $E_i^u \subseteq E_i^r$ to be the set $\{(u,v) \in E_i^r \mid \deg_{H_i}(u) + \deg_{H_i}(v) < \beta(1-\lambda)\}$ (u for underfull). Observe that because of event $\mathcal{A}_i$, the graph H does not change throughout epoch i, so an edge that is underfull at any point during the epoch will be underfull at the end as well. Thus, $\mathcal{A}_i \wedge \mathcal{B}_i$ is equivalent to the event that $|E_i^u| > \gamma$ but $E_i^u \cap E_i^e = \emptyset$.

Let $\mathcal{A}_i^k$ be the event that the kth edge of epoch i is not in E_i^u. We have that

$$\Pr[\mathcal{B}_i \wedge \mathcal{A}_i] \leq \Pr[\mathcal{A}_i \mid \mathcal{B}_i] = \Pr[\mathcal{A}_i^1 \mid \mathcal{B}_i] \prod_{k=2}^{\alpha} \Pr[\mathcal{A}_i^k \mid \mathcal{B}_i, \mathcal{A}_i^1, \ldots, \mathcal{A}_i^{k-1}].$$

Now, observe that

$$\Pr[\mathcal{A}_i^1 \mid \mathcal{B}_i] < 1 - \frac{\gamma}{m}$$

because the first edge of the epoch is chosen uniformly at random from the set of $\leq m$ remaining edges, and the event fails if the chosen edge is in E_i^u, where $|E_i^u| > \gamma$ by definition of $\mathcal{B}_i$. Similarly, for any k,

$$\Pr[\mathcal{A}_i^k \mid \mathcal{B}_i, \mathcal{A}_i^1, \ldots, \mathcal{A}_i^{k-1}] < 1 - \frac{\gamma}{m}$$

because conditioning on the previous events $\mathcal{A}_i^j$ implies that no edge from E_i^u has yet appeared in this epoch, so there are still at least γ edges from E_i^u left in the stream.

Recall from Definition 11 that $\gamma = 5\log(n) \cdot \frac{m}{\alpha}$. Combining the three above equations yields that $\Pr[\mathcal{B}_i \wedge \mathcal{A}_i] \leq (1 - \frac{\gamma}{m})^\alpha = (1 - \frac{5\log(n)}{\alpha})^\alpha \leq n^{-5}$. There are clearly at most n^2 epochs, so union bounding over all of them shows that Property 2b fails with probability at most n^{-3}, as desired. ◀

5 Open Problems

We presented a new single-pass streaming algorithm for computing a maximum matching in a *random-order* stream. The algorithm achieves a $(2/3 - \epsilon)$-approximation using $O(n\log(n))$ space; these bounds improve upon all previous results for the problem.

But while 2/3 is a natural boundary, there is no reason to believe it is the best possible. Is there an algorithm with approximation ratio $2/3 + \Omega(1)$? Is it possible to compute a $(1 - \epsilon)$-approximate matching in random-order streams? A lower bound of $1 - \Omega(1)$ in this setting would also be extremely interesting.

Another natural open problem is get improved bounds for weighted graphs. Gamlath et al. [18] recently broke through the barrier of 1/2 and presented an algorithm for weighted graphs that computes a .506-approximation (or .512 in bipartite graphs) in random-order streams. Can we improve the approximation ratio to 2/3 in weighted graphs? To $(1 - \epsilon)$?

References

1 Kook Jin Ahn and Sudipto Guha. Linear programming in the semi-streaming model with application to the maximum matching problem. *Inf. Comput.*, 222:59–79, 2013.

2 Kook Jin Ahn and Sudipto Guha. Access to data and number of iterations: Dual primal algorithms for maximum matching under resource constraints. In *Proceedings of the 27th ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA 2015, Portland, OR, USA, June 13-15, 2015*, pages 202–211, 2015.

3 Sepehr Assadi, MohammadHossein Bateni, Aaron Bernstein, Vahab S. Mirrokni, and Cliff Stein. Coresets meet EDCS: algorithms for matching and vertex cover on massive graphs. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1616–1635, 2019.

4 Sepehr Assadi and Aaron Bernstein. Towards a unified theory of sparsification for matching problems. In *2nd Symposium on Simplicity in Algorithms, SOSA@SODA 2019, January 8-9, 2019 - San Diego, CA, USA*, pages 11:1–11:20, 2019.

5 Sepehr Assadi, Sanjeev Khanna, and Yang Li. On estimating maximum matching size in graph streams. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 1723–1742, 2017.

6 Sepehr Assadi, Sanjeev Khanna, Yang Li, and Grigory Yaroslavtsev. Maximum matchings in dynamic graph streams and the simultaneous communication model. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1345–1364. SIAM, 2016.

7 Aaron Bernstein and Cliff Stein. Fully dynamic matching in bipartite graphs. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, pages 167–179, 2015.

8 Marc Bury and Chris Schwiegelshohn. Sublinear estimation of weighted matchings in dynamic data streams. In *Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, pages 263–274, 2015.

9 Rajesh Chitnis, Graham Cormode, Hossein Esfandiari, MohammadTaghi Hajiaghayi, Andrew McGregor, Morteza Monemizadeh, and Sofya Vorotnikova. Kernelization via sampling with applications to finding matchings and related problems in dynamic graph streams. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1326–1344, 2016.

10 Rajesh Hemant Chitnis, Graham Cormode, Mohammad Taghi Hajiaghayi, and Morteza Monemizadeh. Parameterized streaming: Maximal matching and vertex cover. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 1234–1251, 2015.

11 Graham Cormode, Hossein Jowhari, Morteza Monemizadeh, and S. Muthukrishnan. The sparse awakens: Streaming algorithms for matching size estimation in sparse graphs. In *25th Annual European Symposium on Algorithms, ESA 2017, September 4-6, 2017, Vienna, Austria*, pages 29:1–29:15, 2017.

12 Michael Crouch and Daniel S. Stubbs. Improved streaming algorithms for weighted matching, via unweighted matching. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2014, September 4-6, 2014, Barcelona, Spain*, pages 96–104, 2014. doi:10.4230/LIPIcs.APPROX-RANDOM.2014.96.

13 Leah Epstein, Asaf Levin, Julián Mestre, and Danny Segev. Improved approximation guarantees for weighted matching in the semi-streaming model. *SIAM J. Discrete Math.*, 25(3):1251–1265, 2011.

14 Hossein Esfandiari, MohammadTaghi Hajiaghayi, Vahid Liaghat, Morteza Monemizadeh, and Krzysztof Onak. Streaming algorithms for estimating the matching size in planar graphs and beyond. *ACM Trans. Algorithms*, 14(4):48:1–48:23, 2018.

15 Hossein Esfandiari, MohammadTaghi Hajiaghayi, and Morteza Monemizadeh. Finding large matchings in semi-streaming. In Carlotta Domeniconi, Francesco Gullo, Francesco Bonchi, Josep Domingo-Ferrer, Ricardo Baeza-Yates, Zhi-Hua Zhou, and Xindong Wu, editors, *IEEE International Conference on Data Mining Workshops, ICDM Workshops 2016, December 12-15, 2016, Barcelona, Spain*, pages 608–614. IEEE Computer Society, 2016.

16 Alireza Farhadi, Mohammad Taghi Hajiaghayi, Tung Mai, Anup Rao, and Ryan A. Rossi. Approximate maximum matching in random streams. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1773–1785, 2020.

17 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Theor. Comput. Sci.*, 348(2-3):207–216, 2005. doi:10.1016/j.tcs.2005.09.013.

18 Buddhima Gamlath, Sagar Kale, Slobodan Mitrovic, and Ola Svensson. Weighted matchings via unweighted augmentations. In Peter Robinson and Faith Ellen, editors, *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*, pages 491–500. ACM, 2019.

19 Mohsen Ghaffari and David Wajc. Simplified and space-optimal semi-streaming (2+epsilon)-approximate matching. In Jeremy T. Fineman and Michael Mitzenmacher, editors, *2nd*

Symposium on Simplicity in Algorithms, SOSA@SODA 2019, January 8–9, 2019 – San Diego, CA, USA, volume 69 of *OASICS*, pages 13:1–13:8. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2019.

20 Ashish Goel, Michael Kapralov, and Sanjeev Khanna. On the communication and streaming complexity of maximum bipartite matching. In *Proceedings of the Twenty-third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '12, pages 468–485. SIAM, 2012. URL: `http://dl.acm.org/citation.cfm?id=2095116.2095157`.

21 Venkatesan Guruswami and Krzysztof Onak. Superlinear lower bounds for multipass graph processing. In *Proceedings of the 28th Conference on Computational Complexity, CCC 2013, K.lo Alto, California, USA, 5-7 June, 2013*, pages 287–298, 2013.

22 Sagar Kale and Sumedh Tirodkar. Maximum matching in two, three, and a few more passes over graph streams. In Klaus Jansen, José D. P. Rolim, David Williamson, and Santosh S. Vempala, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2017, August 16-18, 2017, Berkeley, CA, USA*, volume 81 of *LIPIcs*, pages 15:1–15:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

23 Michael Kapralov. Better bounds for matchings in the streaming model. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 1679–1697, 2013. `doi:10.1137/1.9781611973105.121`.

24 Michael Kapralov, Sanjeev Khanna, and Madhu Sudan. Approximating matching size from random streams. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 734–751, 2014. `doi:10.1137/1.9781611973402.55`.

25 Michael Kapralov, Slobodan Mitrovic, Ashkan Norouzi-Fard, and Jakab Tardos. Space efficient approximation to maximum matching size from uniform edge samples. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1753–1772, 2020.

26 Christian Konrad. Maximum matching in turnstile streams. In *Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, pages 840–852, 2015.

27 Christian Konrad. A simple augmentation method for matchings with applications to streaming algorithms. In Igor Potapov, Paul G. Spirakis, and James Worrell, editors, *43rd International Symposium on Mathematical Foundations of Computer Science, MFCS 2018, August 27-31, 2018, Liverpool, UK*, volume 117 of *LIPIcs*, pages 74:1–74:16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.

28 Christian Konrad, Frédéric Magniez, and Claire Mathieu. Maximum matching in semi-streaming with few passes. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 15th International Workshop, APPROX 2012, and 16th International Workshop, RANDOM 2012, Cambridge, MA, USA, August 15-17, 2012. Proceedings*, pages 231–242, 2012.

29 Andrew McGregor. Finding graph matchings in data streams. In *Approximation, Randomization and Combinatorial Optimization, Algorithms and Techniques, 8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2005 and 9th InternationalWorkshop on Randomization and Computation, RANDOM 2005, Berkeley, CA, USA, August 22-24, 2005, Proceedings*, pages 170–181, 2005. `doi:10.1007/11538462_15`.

30 Andrew McGregor and Sofya Vorotnikova. Planar matching in streams revisited. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2016, September 7-9, 2016, Paris, France*, pages 17:1–17:12, 2016.

31 Andrew McGregor and Sofya Vorotnikova. A simple, space-efficient, streaming algorithm for matchings in low arboricity graphs. In *1st Symposium on Simplicity in Algorithms, SOSA 2018, January 7-10, 2018, New Orleans, LA, USA*, pages 14:1–14:4, 2018.

32 Ami Paz and Gregory Schwartzman. A (2 + epsilon)-approximation for maximum weight matching in the semi-streaming model. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 2153–2161, 2017.

33 David Wajc. Negative association: definition, properties, and applications. URL: `https://www.cs.cmu.edu/~dwajc/notes/Negative%20Association.pdf`.

An Optimal Algorithm for Online Multiple Knapsack

Marcin Bienkowski
Institute of Computer Science, University of Wrocław, Poland
marcin.bienkowski@cs.uni.wroc.pl

Maciej Pacut
Faculty of Computer Science, University of Vienna, Austria
maciej.pacut@univie.ac.at

Krzysztof Piecuch
Institute of Computer Science, University of Wrocław, Poland
kpiecuch@cs.uni.wroc.pl

Abstract

In the online multiple knapsack problem, an algorithm faces a stream of items, and each item has to be either rejected or stored irrevocably in one of n bins (knapsacks) of equal size. The gain of an algorithm is equal to the sum of sizes of accepted items and the goal is to maximize the total gain.

So far, for this natural problem, the best solution was the 0.5-competitive algorithm FirstFit (the result holds for any $n \geq 2$). We present the first algorithm that beats this ratio, achieving the competitive ratio of $1/(1 + \ln(2)) - O(1/n) \approx 0.5906 - O(1/n)$. Our algorithm is deterministic and optimal up to lower-order terms, as the upper bound of $1/(1 + \ln(2))$ for randomized solutions was given previously by Cygan et al. [TOCS 2016].

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases online knapsack, multiple knapsacks, bin packing, competitive analysis

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.13

Category Track A: Algorithms, Complexity and Games

Related Version A full version of the paper is available at `https://arxiv.org/abs/2002.04543`.

Funding Research supported by Polish National Science Centre grants 2016/22/E/ST6/00499 and 2016/23/N/ST6/03412.

1 Introduction

Knapsack problems have been studied in theoretical computer science for decades [13, 14]. In particular, in the *multiple knapsack* problem [2, 5, 6, 7, 10, 12, 18], items of given sizes and profits have to be stored in n bins (knapsacks), each of capacity 1. The goal is to find a subset of all items that maximizes the total profit and can be feasibly packed into bins without exceeding their capacities. We consider an *online scenario*, where an online algorithm is given a sequence of items of unknown length. When an item is presented to an algorithm, it has to either irrevocably reject the item or accept it to a chosen bin (which cannot be changed in the future). The actions of an online algorithm have to be made without the knowledge of future items.

Proportional case. In this paper, we focus on the most natural, *proportional variant* (sometimes called *uniform*), where item profits are equal to item sizes and the goal is to maximize the sum of profits of all accepted items.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 13; pp. 13:1–13:17
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The single-bin case ($n = 1$) has been fully resolved: no deterministic online algorithm can be competitive [16], and the best randomized algorithm ROne by Böckenhauer et al. [4] achieves the optimal competitive ratio of 0.5.[1]

Less is known for multiple-bin case ($n \geq 2$). Cygan et al. [7] showed that the FirstFit algorithm is 0.5-competitive and proved that no algorithm (even a randomized one) can achieve a competitive ratio greater than R, where

$$R = 1/(1 + \ln 2) \approx 0.5906.$$

Other variants. Some authors focused on the variant, where the goal is to maximize the *maximum* profit over all bins, instead of the sum of the profits. For this objective, optimal competitive ratios are already known: 0.5-competitive deterministic algorithm was given by Böckenhauer et al. [4], and the upper bound of 0.5 holding even for randomized solutions were presented by Cygan et al. [7].

The multiple knapsack problem can be generalized in another direction: profits and sizes may be unrelated. However, already *the unit variant*, where the profit of each item is equal to 1, does not admit any competitive solutions (even randomized ones) [5].

These results together mean that the proportional case studied in this paper is the only variant, whose online complexity has not been fully resolved yet.

1.1 Our results

The main result of this paper is an $(R - O(1/n))$-competitive deterministic online algorithm for the proportional variant of the multiple knapsack problem. We give insights for our construction in Section 1.3 below and the definition of our algorithm later in Section 2. Given the upper bound of R for randomized solutions [7], our result is optimal up to lower-order terms also for the class of randomized solutions.

It is possible to show that for deterministic algorithms, the term $O(1/n)$ in the competitive ratio is inevitable: in the full version of the paper, we show how the upper bound construction given in [7] can be tweaked and extended to show that the competitive ratio of any deterministic algorithm is at most $R - O(1/n)$.

1.2 Related work

Some previous papers focused on a *removable* scenario, where an accepted item can be removed afterwards from its bin [2, 7, 8, 10, 11]. Achievable competitive ratios are better than their non-removable counterparts; in particular, the proportional variant admits constant-competitive deterministic algorithms even for a single bin [10].

The online knapsack problem has been also considered in relaxed variants: with resource augmentation, where the bin capacities of an online algorithm are larger than those of the optimal offline one [11, 17], with a resource buffer [9], or in the variant where an algorithm may accept fractions of items [17].

The hardness of the variants with arbitrary profits and sizes as well as applications to online auctions motivated another strand of research focused on the so-called random-order model [1, 3, 15, 19]. There, the set of items is chosen adversarially, but the items are presented to an online algorithm in random order.

[1] An online algorithm is called α-competitive if, for any input instance, its total profit is at least fraction α of the optimal (offline) solution. While many papers use the reciprocal of α as the competitive ratio, the current definition is more suited for accounting arguments in our proofs.

1.3 Algorithmic challenges and ideas

Our algorithm splits items into three categories: large (of size greater than 1/2), medium (of size from the interval $[\phi, 1/2]$) and small (of size smaller than ϕ). We defer the actual definition of ϕ.

First, we explain what an online algorithm should do when it faces a stream of large items. Note that no two large items can fit together in a single bin. If an algorithm greedily collects all large items, then the adversary may give n items of size $1/2 + \epsilon$ (accepted by an online algorithm) followed by n items of size 1 (accepted by an optimal offline algorithm Opt), and the resulting competitive ratio is then 0.5. On the other hand, if an algorithm stops after accepting some number of large items, Opt may collect all of them.

Our Rising Threshold Algorithm (Rta) balances these two strategies. It chooses a non-decreasing threshold function $\mathsf{f} : [0, 1] \to [1/2, 1]$ and ensures that the size of the i-th accepted large item is at least $\mathsf{f}(i/n)$. While an actual definition of f is given later, to grasp a general idea, it is worth looking at its plot in Figure 1 (left). A natural adversarial strategy is to give large items meeting these thresholds, and once Rta fills k bins, present n items of sizes slightly smaller than the next threshold $\mathsf{f}((k + 1)/n)$. These items will be rejected by Rta but can be accepted by Opt. Analyzing this strategy and ensuring that the ratio is at least R for any choice of k yields boundary conditions. Analyzing these conditions for n tending to infinity, we obtain a differential equation, whose solution is the function f used in our algorithm.

The actual difficulty, however, is posed by medium items. Rta never proactively rejects them and it keeps a subset of *marked* medium items in their own bins (one item per one bin), while it stacks the remaining, non-marked ones (places them together in the same bin, possibly combining items of similar sizes). This strategy allows Rta to combine a large item with marked medium items later. However, the amount of marked items has to be carefully managed as they do not contribute large gain alone. A typical approach would be to partition medium items into discrete sub-classes, control the number of items in each class, and analyze the gain on the basis of the *minimal* size item in a particular subclass. To achieve optimal competitive ratio, we however need a more fine-grained approach: we use a carefully crafted continuous function ξ to control the number of marked items larger than a given value. Analyzing all possible adversarial strategies gives boundary conditions for ξ. In particular, the value ϕ that separates medium items from small ones was chosen as the minimum value that ensures the existence of function ξ satisfying all boundary conditions.

Finally, we note that simply stacking small items in their own bins would not lead to the desired competitive ratio. Instead, Rta tries to stack them in a single bin, but whenever its load exceeds ϕ, Rta tries to merge them into a single medium item and verify whether such an item could be marked. This allows for combining them in critical cases with large items.

1.4 Preliminaries

We have n bins of capacity 1, numbered from 1 to n. An input is a stream of items from $(0, 1]$, defined by their sizes. Upon seeing an item, an online algorithm has to either reject it or place it in an arbitrary bin without violating the bin's capacity. The *load* of a bin b, denoted $\mathsf{load}(b)$, is the sum of item sizes stored in bin b. We define the load of a set of items as the sum of their sizes and the total load as the load of all items collected by an algorithm. Additionally, for any $x \leq 1/2$, we define $\mathsf{pile}(x) = \max\{2/3, 2x\}$. Note that if we put medium items of sizes at least x (till it is possible) into a bin b, then $\mathsf{load}(b) \geq \mathsf{pile}(x)$.

To simplify calculations, for any set Z of items, we define the *gain* of Z, denoted $\mathsf{g}(Z)$, as their load divided by n; similarly, the total gain is the total load divided by n. Furthermore, we use $\mathsf{min}(Z)$ to denote the minimum size of an item in set Z. If Z is accepted by our online algorithm, $\mathsf{b}(Z)$ denotes the number of bins our algorithm uses to accommodate these items, divided by n. For any value $x \in [0, 1]$, $Z^{\geq x}$ is the set of all items from Z of size greater or equal x. Whenever we use terms $\mathsf{g}(Z)$, $\mathsf{min}(Z)$ or $\mathsf{b}(Z)$ for a set Z that varies during runtime, we mean these values for the set Z after an online algorithm terminates its execution.

For any input sequence σ and an algorithm A, we use $A(\sigma)$ to denote the total gain of A on sequence σ. We denote the optimal offline algorithm by Opt.

1.5 Neglecting lower-order terms

As our goal is to show the competitive ratio $R - O(1/n)$, we introduce a notation that allows to neglect terms of order $1/n$. We say that x is *approximately equal* to y (we write $x \eqsim y$) if $|x - y| = O(1/n)$. Furthermore, we say that x is *approximately greater* than y (we write $x \gtrsim y$) if $x \geq y$ or $x \eqsim y$; we define relation $\lesssim$ analogously. Each of these relations is transitive when composed a constant number of times.

In our analysis, we are dealing with Lipschitz continuous functions (their derivative is bounded by a universal constant). For such function h, (i) the relation $\eqsim$ is preserved after application of h, and (ii) an integral of h can be approximated by a sum, as stated in the following facts, used extensively in the paper.

▶ **Fact 1.** *Fix any Lipschitz continuous function h and values $x \eqsim y$ from its domain. Then, $h(x) \eqsim h(y)$. Furthermore, if h is non-decreasing, then $x \lesssim y$ implies $h(x) \lesssim h(y)$ and $x \gtrsim y$ implies $h(x) \gtrsim h(y)$.*

▶ **Fact 2.** *For any Lipschitz continuous function h and integers a, b satisfying $1 \leq a \leq b \leq n$, it holds that $(1/n) \cdot \sum_{i=a+1}^{b} h(i/n) \eqsim \int_{a/n}^{b/n} h(x)\,\mathrm{d}x$.*

1.6 Roadmap of the proof

We present our algorithm in Section 2. Its analysis consists of three main parts.

- In Section 3, we investigate the gain of Rta on large items and explain the choice of the threshold function f.
- In Section 4, we study properties of medium items, marking routine, function ξ and show how the marked items influence the gain on other non-large items.
- In Section 5, we study the impact of marked items on bins containing large items.

Each of these parts is concluded with a statement that, under certain conditions, Rta is $(R - O(1/n))$-competitive (cf. Lemma 5, Lemma 15, Lemma 20 and Lemma 21). In Section 6, we argue that these lemmas cover all possible outcomes. For succinctness, some technical claims have been moved to Section 7.

2 Rising Threshold Algorithm

We arrange items into three categories: small, medium and large. We say that an item is *large* if its size is in the range $(1/2, 1]$, *medium* if it is in the range $[\phi, 1/2]$, and otherwise it is *small*, where we define

$$\xi_\mathrm{c} = (1 + (2/3) \cdot \ln(4/3)) \cdot R - 2/3 \approx 0.0372 \quad \text{and} \tag{1}$$

$$\phi = (2/3) \cdot \xi_\mathrm{c} \,/\, (2/3 - R + \xi_\mathrm{c}) \approx 0.2191. \tag{2}$$

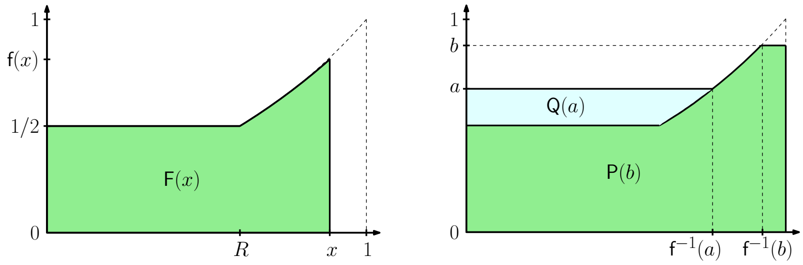


Figure 1 Left: function f and its integral F. The value of $\mathsf{F}(x)$ roughly corresponds to our lower bound on the gain of Rta when it collects $n \cdot x$ large items. Right: functions P and Q used in estimating the gain in Section 4 and Section 5; note that their arguments are marked at Y axis.

We further arrange medium items into subcategories M_2, M_3 and M_4: a medium item belongs to M_i if its size is from range $(1/(i+1), 1/i]$. As we partition only medium items this way, M_4 contains items of sizes from $[\phi, 1/4]$. Note that at most i items of category M_i fit in a single bin.

At some times (defined precisely later) a group of small items of a total load from $[\phi, 2\phi)$ stored in a single bin may become *merged*, and from that point is treated as a single medium item. We ensure that such merging action does not violate invariants of our algorithm.

Our algorithm Rta applies labels to bins; the possible labels are E, A, S_*, M_S, M_2, M_3, M_4 and L_+. Each bin starts as an E-bin, and Rta can relabel it later. The label determines the content of a given bin:

- an E-bin is empty,
- an A-bin (an *auxiliary bin*) contains small items of a total load smaller than ϕ and at most one A-bin exists at any time,
- an S_*-bin contains one or multiple small items,
- an M_S-bin contains a single marked medium item,
- an M_i-bin contains one or more medium items of category M_i,
- an L_+-bin contains a single large item and possibly some other non-large ones.

For any label C, we define a corresponding set, also denoted C, containing all items stored in bins of label C. For instance, L_+ is a set containing all items stored in L_+-bins. Furthermore, we define L as the set of all large items (clearly $L \subseteq L_+$ and $\mathsf{b}(L) = \mathsf{b}(L_+)$) and the set $M_* = M_2 \uplus M_3 \uplus M_4$.

Rta processes a stream of items, and it operates until the stream ends or there are no more empty bins (even if an incoming item could fit in some partially filled bin). Upon the arrival of an item, Rta classifies it by its size and proceeds as described below.

Large items. Whenever a large item arrives, Rta compares its size with the threshold $\mathsf{f}(\mathsf{b}(L) + 1/n)$, and if the item is smaller, Rta rejects it. The function $\mathsf{f} : [0,1] \to [1/2, 1]$ is defined as

$$\mathsf{f}(x) = \begin{cases} 1/2 & \text{if } x \leq R, \\ (2\mathrm{e})^{x-1} & \text{otherwise,} \end{cases} \tag{3}$$

and depicted in Figure 1 (left). If the item meets the threshold, Rta attempts to put it in an M_S-bin with sufficient space left (relabeling it to L_+), and if no such bin exists, Rta puts the item in any empty bin.

Medium items. We fix a continuous and decreasing function ξ that maps medium item sizes to $[0, \xi_c/\phi]$:

$$\xi(x) = \begin{cases} \xi_c/x & \text{if } x \in [\phi, 1/3], \\ 9\xi_c \cdot (1-2x) & \text{if } x \in (1/3, 1/2]. \end{cases} \tag{4}$$

We say that the subset Z of medium items is ξ-dominated if $|Z^{\geq x}|/n \leq \xi(x)$ for any $x \in Z$. Intuitively, it means that if we sort items of Z from largest to smallest, then all points $(i/n, x_i)$ are under or at the plot of ξ^{-1}, see Figure 3 (left).

Rta never proactively rejects medium items, i.e., it always accepts them if it has an empty bin. Some medium items become *marked* upon arrival; we denote the set of medium marked items by D. Large or small items are never marked. Marked medium items are never combined in a single bin with other marked medium items. At all times, Rta ensures that the set D is ξ-dominated. As no two items from D are stored in a single bin, this corresponds to the condition $\mathsf{b}(D^{\geq x}) \leq \xi(x)$ for any $x \in D$. Each marked item is stored either in an M_S-bin (alone) or in an L_+-bin (together with a large item and possibly some other non-marked items). That is, $M_S \subseteq D \subseteq M_S \uplus L_+$.

Whenever a medium item arrives, Rta attempts to put it in an L_+-bin. If it does not fit there, Rta verifies whether marking it (including it in the set D) preserves ξ-domination of D. If so, Rta marks it and stores it in a separate M_S-bin. Otherwise, Rta *fails to mark* the item and the item is stored in an M_i-bin (where i depends on the item size): it is added to an existing bin whenever possible and a new M_i-bin is opened only when necessary.

We emphasize that if Rta puts a large item in an M_S-bin later (and relabel it to L_+), the sole medium item from this bin remains marked (i.e., in the set D). However, if a medium item fits in an L_+-bin at the time of its arrival, it avoids being marked, even though its inclusion might not violate ξ-dominance of the set D. Note also that M_* contains medium items Rta failed to mark.

Small items. Rta never proactively rejects any small item. Whenever a small item arrives, Rta attempts to put this item in an L_+-bin, in an S_*-bin, and in the A-bin, in this exact order. If the item does not fit in any of them (this is possible only if the A-bin does not exist), Rta places it in an empty bin and relabels this bin to A.

If Rta places the small item in an already existing A-bin and in effect its load reaches or exceeds ϕ, Rta attempts to *merge* all its items into a single medium marked item. If the resulting medium item can be marked and included in D without violating its ξ-dominance, Rta relabels the A-bin to M_S and treats its contents as a single marked medium item from now on. Otherwise, it simply changes the label of the A-bin to S_*.

3 Gain on large items

In this section, we analyze the gain of Rta on large items. To this end, we first calculate the integral of function f, denoted F (see Figure 1, left) and list its properties that can be verified by routine calculations.

$$\mathsf{F}(x) = \int_0^x \mathsf{f}(y)\, dy = \begin{cases} x/2 & \text{if } x \leq R, \\ R \cdot (2e)^{x-1} & \text{otherwise.} \end{cases} \tag{5}$$

▶ **Lemma 3.** *The following properties hold for function* F.

1. $\mathsf{f}^{-1}(c) = 1 + R \cdot \ln c$ *and* $\mathsf{F}(\mathsf{f}^{-1}(c)) = R \cdot c$ *for any* $c \in (1/2, 1]$.
2. $\mathsf{F}(x)/\mathsf{f}(x) = \min\{x, R\}$ *for any* $x \in [0, 1]$.
3. $(1/n) \cdot \sum_{i=1}^{\ell} \mathsf{f}(i/n) \approx \int_0^{\ell/n} \mathsf{f}(x)\, dx = \mathsf{F}(\ell/n)$ *for any* $\ell \in \{0, \ldots, n\}$.

Using Lemma 3, we may bound on the gain of Rta on large items L and use this bound to estimate its competitive ratio when it terminates with empty bins.

▶ **Lemma 4.** *It holds that $\mathsf{g}(L) \gtrsim \mathsf{F}(\mathsf{b}(L)) = \mathsf{F}(\mathsf{b}(L_+))$. Moreover, $\mathsf{g}(L) \gtrsim \mathsf{F}(\mathsf{b}(L)) + \mathsf{g}(L^{\geq x}) - x \cdot \mathsf{b}(L^{\geq x})$ for any $x \geq \mathsf{f}(\mathsf{b}(L))$.*

Proof. For the first part of the lemma, we sort large items from L in the order they were accepted by Rta. The size of the i-th large item is at least the threshold $f(i/n)$. Hence, by Lemma 3, $\mathsf{g}(L) \geq (1/n) \cdot \sum_{i=1}^{|L|} f(i/n) \approx \mathsf{F}(|L|/n) = \mathsf{F}(\mathsf{b}(L))$.

To show the second part, we fix any $x \geq \mathsf{f}(\mathsf{b}(L))$ and for each large item of size greater than x we reduce its size to x. The total gain of the removed parts is exactly $\mathsf{g}(L^{\geq x}) - x \cdot \mathsf{b}(L^{\geq x})$. The resulting large item sizes still satisfy acceptance thresholds, and thus the gain on the remaining part of L is approximately greater than $\mathsf{F}(\mathsf{b}(L))$. Summing up yields $\mathsf{g}(L) \gtrsim \mathsf{F}(\mathsf{b}(L)) + \mathsf{g}(L^{\geq x}) - x \cdot \mathsf{b}(L^{\geq x})$. ◀

3.1 When RTA terminates with some empty bins

▶ **Lemma 5.** *If Rta terminates with some empty bins, then it is $(R - O(1/n))$-competitive.*

Proof. Fix an input sequence σ. As Rta terminates with empty bins, it manages to accept all medium and small items from σ. Furthermore, it accepts large items from σ according to the thresholds given by function f. Recall that f is non-decreasing: at the beginning it is equal to $1/2$ (Rta accepts any large item) and the acceptance threshold grows as Rta accepts more large items. Let $x = \mathsf{f}(\mathsf{b}(L) + 1/n)$ be the value of the acceptance threshold for large items when Rta terminates. We consider two cases.

- $\mathsf{b}(L) \leq R - 1/n$. The threshold used for each large item is at most $x \leq \mathsf{f}(R) = 1/2$, i.e., Rta accepts all large items. Then, Rta accepts all items and is 1-competitive.
- $\mathsf{b}(L) > R - 1/n$. Let N be the set of all non-large items accepted by Rta. By Lemma 4,

$$\begin{aligned}\textsc{Rta}(\sigma) = \mathsf{g}(L) + \mathsf{g}(N) &\gtrsim \mathsf{F}(\mathsf{b}(L)) + \mathsf{g}(L^{\geq x}) - x \cdot \mathsf{b}(L^{\geq x}) + \mathsf{g}(N) \\ &\approx R \cdot x + \mathsf{g}(L^{\geq x}) - x \cdot \mathsf{b}(L^{\geq x}) + \mathsf{g}(N).\end{aligned}$$

where for the last relation we used $\mathsf{F}(\mathsf{b}(L)) \approx \mathsf{F}(\mathsf{f}^{-1}(x)) = R \cdot x$ (by Lemma 3).
As Rta takes all non-large items and all large items that are at least x, the input sequence σ contains items taken by Rta and possibly some large items smaller than x. Thus, the gain of Opt on large items is maximized when it takes $L^{\geq x}$ and fills the remaining $n - |L^{\geq x}|$ bins with large items from σ smaller than x. The total gain of Opt is thus at most

$$\textsc{Opt}(\sigma) \leq \mathsf{g}(L^{\geq x}) + x \cdot (1 - \mathsf{b}(L^{\geq x})) + \mathsf{g}(N) = x + \mathsf{g}(L^{\geq x}) - x \cdot \mathsf{b}(L^{\geq x}) + \mathsf{g}(N).$$

Comparing the bounds on gains of Rta and Opt and observing that the term $\mathsf{g}(L^{\geq x}) - x \cdot \mathsf{b}(L^{\geq x}) + \mathsf{g}(N)$ is non-negative, yields $\textsc{Rta}(\sigma) \geq R \cdot \textsc{Opt}(\sigma) - O(1/n)$. As $\textsc{Opt}(\sigma) \geq \mathsf{g}(L) = \Omega(1)$, we obtain $\textsc{Rta}(\sigma) \geq (R - O(1/n)) \cdot \textsc{Opt}(\sigma)$. ◀

As an immediate corollary, we observe that if σ contains large items only, then Rta is $(R - O(1/n))$-competitive: If it terminates with empty bins, then its competitive ratio follows by Lemma 5. Otherwise, it terminates with n large items, and hence, by Lemma 4, $\textsc{Rta}(\sigma) \gtrsim \mathsf{F}(\mathsf{b}(L)) = \mathsf{F}(1) = R$. On the other hand, $\textsc{Opt}(\sigma) \leq 1$, and therefore the competitive ratio is at most $R - O(1/n)$ also in this case.

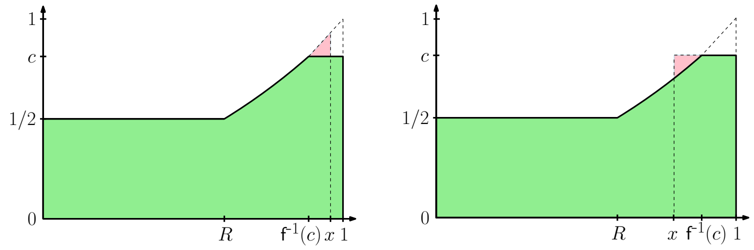


Figure 2 A geometric interpretation of the second property of Lemma 6: using P to lower-bound the sum of $\mathsf{F}(x)$ and the rectangle $c \cdot (1 - x)$ for the case $\mathsf{f}^{-1}(c) \le x$ (left) and $\mathsf{f}^{-1}(c) > x$ (right).

4 Gain on medium items

In the remaining part of the analysis, we make use of the following functions. For any $c \in (1/2, 1]$, let $\mathsf{P}(c) = \int_0^1 \min\{\mathsf{f}(y), c\} \,\mathrm{d}y$ and $\mathsf{Q}(c) = \int_0^1 \max\{c - \mathsf{f}(y), 0\} \,\mathrm{d}y$. Both functions are increasing and depicted in Figure 1 (right). As we show below (cf. the last property of Lemma 6), $\mathsf{P}(c)$ lower-bounds the gain of RTA in the case when its load on non-L_+ bins is at least c.

▶ **Lemma 6.** *Fix any $c \in (1/2, 1]$ and any $x \in [0, 1]$. It holds that*

1. $\mathsf{P}(c) = c - R \cdot c \cdot \ln(2c)$*,*
2. $\mathsf{Q}(c) = R \cdot c \cdot \ln(2c)$*,*
3. $\mathsf{P}(c) + \mathsf{Q}(c) = c$*,*
4. $\mathsf{F}(x) + c \cdot (1 - x) \ge \mathsf{P}(c)$*.*

Proof of Lemma 6. We fix any $c \in (1/2, 1]$ and any $x \in [0, 1]$. For the first property, observe that

$$\mathsf{P}(c) = \int_0^{\mathsf{f}^{-1}(c)} \mathsf{f}(y) \,\mathrm{d}y + \int_{\mathsf{f}^{-1}(c)}^1 c \,\mathrm{d}y = \mathsf{F}(\mathsf{f}^{-1}(c)) + c \cdot (1 - \mathsf{f}^{-1}(c)) = c - R \cdot c \cdot \ln(2c),$$

where for the last equality we used Lemma 3. Similarly, the second property follows as

$$\mathsf{Q}(c) = \int_0^{\mathsf{f}^{-1}(c)} c - \mathsf{f}(y) \,\mathrm{d}y = c \cdot \mathsf{f}^{-1}(c) - \mathsf{F}(\mathsf{f}^{-1}(c)) = R \cdot c \cdot \ln(2c).$$

The third relation, $\mathsf{P}(c) + \mathsf{Q}(c) = c$, follows immediately by the first two. Finally, for the last relation, we use

$$\mathsf{F}(x) + c \cdot (1 - x) = \int_0^x \mathsf{f}(y) \,\mathrm{d}y + \int_x^1 c \,\mathrm{d}y \ge \int_0^x \min\{\mathsf{f}(y), c\} \,\mathrm{d}x + \int_x^1 \min\{f(y), c\} \,\mathrm{d}y = \mathsf{P}(c).$$

See also Figure 2 for a geometric argument. ◀

4.1 Boundary conditions on function ξ

We start with a shorthand notation. Let $\mathsf{T}(a, b) = (a + b - 1/2) \cdot (\xi(b) - \xi(a))$, where $a, b \in [\phi, 1/2]$ (so that the values of $\xi(a)$ and $\xi(b)$ are well defined).

Our choice of function ξ satisfies the conditions below. In fact, for our analysis to hold, function ξ could be replaced by any Lipschitz continuous and non-increasing function mapping $[\phi, 1/2]$ to $[0, 1]$ satisfying these properties.

▶ **Lemma 7.** *The following properties hold for function ξ:*

1. $x \cdot \xi(x)$ *is a non-increasing function of* $x \in [\phi, 1/2]$,
2. $\mathsf{P}(\mathsf{pile}(x)) + x \cdot \xi(x) \geq R$ *for* $x \in [\phi, 1/2]$,
3. $\mathsf{P}(1-\phi) + 2\phi \cdot \xi(2\phi) \geq R$,
4. $2/3 - (2/3 - \phi) \cdot \xi(x) \geq R$ *for* $x \in [\phi, 1/3]$,
5. $\mathsf{P}(1-\phi) + \mathsf{Q}(1-x) + (x+\phi-1) \cdot \xi(x) + \max\{T(x,y), 0\} \geq R$ *for* $x \in [1/3, 1/2]$ *and* $y \in [\phi, 2\phi]$,
6. $\mathsf{P}(\mathsf{pile}(y)) + \mathsf{Q}(1-x) + (x - \mathsf{pile}(y)) \cdot \xi(x) + T(x,y) \geq R$ *for* $x \in [1/3, 1/2]$ *and* $y \in [\phi, x]$,
7. $\mathsf{P}(2x) + \mathsf{Q}(1-x) - x \cdot \xi(x) \geq R$ *for* $x \in [1/3, 1/2]$.

4.2 Marked and tight items

We start with a simple bound on the gain of Rta on M_S-bins. Recall that these bins store single marked items.

▶ **Lemma 8.** *If* Rta *terminates with at least one* M_S*-bin, then* $\mathsf{g}(M_\mathrm{S}) \geq \mathsf{min}(M_\mathrm{S}) \cdot \mathsf{b}(M_\mathrm{S})$*, and* $\mathsf{b}(M_\mathrm{S}) \leq \xi(\mathsf{min}(M_\mathrm{S}))$.

Proof. The first condition follows trivially as each M_S-bin contains a single medium item of size at least $\mathsf{min}(M_\mathrm{S})$. For the second condition, note that $M_\mathrm{S} \subseteq D$, and thus also $M_\mathrm{S} \subseteq D^{\geq \min(M_\mathrm{S})}$. As D is ξ-dominated, $\mathsf{b}(M_\mathrm{S}) \leq \mathsf{b}(D^{\geq \min(M_\mathrm{S})}) \leq \xi(\min(M_\mathrm{S}))$. ◀

We now take a closer look at the marked items and their influence on the gain on other sets of items. We say that a medium marked item $x \in D$ is *tight* if it is on the verge of violating ξ-domination invariant.

▶ **Definition 9.** *An item* $x \in D$ *is tight if* $\mathsf{b}(D^{\geq x}) > \xi(x) - 1/n$.

If an item $x \in D$ is tight, then another item of size x or greater cannot be included in D without violating ξ-domination invariant. Figure 3 (left) illustrates this concept. As D can only grow, once an item becomes tight, it remains tight till the end. We emphasize that items smaller than x are not relevant for determining whether x is tight. If D contains a tight item, then $\mathsf{mt}(D)$ denotes the size of the minimum tight item in D. This important parameter influences the gain both on set D and also on stacking bins M_* and S_*.

▶ **Lemma 10.** *If* D *contains a tight item, then* $\mathsf{g}(D) \gtrsim \mathsf{mt}(D) \cdot \xi(\mathsf{mt}(D))$.

Proof. Fix a tight item $d \in D$ of size $\mathsf{mt}(D)$. By Definition 9, $\mathsf{b}(D^{\geq d}) > \xi(d) - 1/n$, and thus $\mathsf{g}(D) \geq \mathsf{g}(D^{\geq d}) \geq d \cdot \mathsf{b}(D^{\geq d}) \gtrsim d \cdot \xi(d)$. ◀

4.3 Impact of tight items on stacking bins

By Property 1 of Lemma 7, $x \cdot \xi(x)$ is a non-increasing function of x. Therefore, the smaller $\mathsf{mt}(D)$ is, the larger is the lower bound on $\mathsf{g}(D)$ guaranteed by Lemma 10. Now we argue that the larger $\mathsf{mt}(D)$ is, the better is the gain on stacking bins M_* and S_*.

▶ **Lemma 11.** *Assume* Rta *failed to mark a medium item* y*. Then, a tight item exists and* $\mathsf{mt}(D) \lesssim y$.

Proof. Let $D_\mathrm{ext} = D \cup \{y\}$. By the lemma assumption, D_ext is not ξ-dominated, i.e., there exists an item $x \in D_\mathrm{ext}$ such that $\mathsf{b}(D_\mathrm{ext}^{\geq x}) > \xi(x)$. Note that $x \leq y$, as otherwise we would have $D^{\geq x} = D_\mathrm{ext}^{\geq x}$, and thus $\mathsf{b}(D^{\geq x}) > \xi(x)$, which would contradict ξ-domination of D.

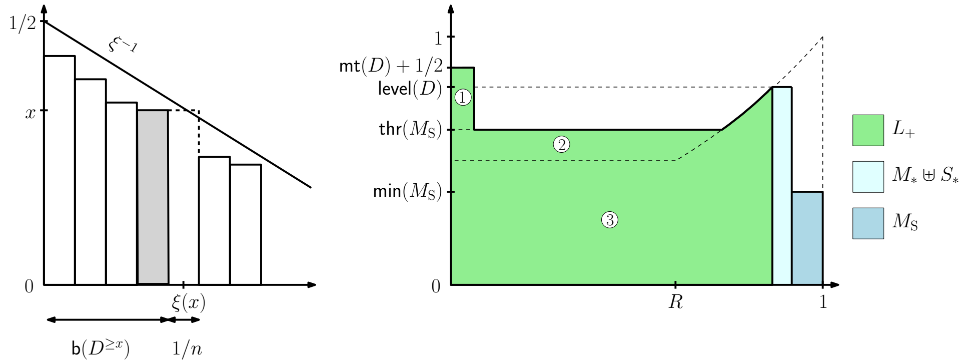


Figure 3 Left: set D of marked items with a tight (gray) item of size x. As x is tight, insertion of another item of size x (with a dashed border) would violate ξ-domination of D. Right: items collected by Rta, when it terminates without empty bins and with M_{S}-bins. Gain on L_+-sets is split into three parts, where the first part corresponds to the gain of T^* (cf. Lemma 17). The minimum guaranteed load in M_{S}-bins is given by Lemma 8 and in the bins of $M_* \uplus S_*$ by Lemma 13.

Let $d \geq x$ be the minimum size of an item from $D^{\geq x}$. Then, $D^{\geq d} = D^{\geq x}$, and thus

$$\mathsf{b}(D^{\geq d}) = \mathsf{b}(D^{\geq x}) = \mathsf{b}(D^{\geq x}_{\text{ext}}) - 1/n > \xi(x) - 1/n \geq \xi(d) - 1/n, \tag{6}$$

where in the last inequality we used monotonicity of ξ. By (6), d is tight. On the other hand, ξ-domination of D implies that $\mathsf{b}(D^{\geq d}) \leq \xi(d)$. This, combined with (6), yields $\xi(d) \approx \xi(x)$, and thus $d \approx x \leq y$. Note that d remains tight till the end of the execution. This concludes the lemma, as the minimum tight item, $\mathsf{mt}(D)$, can be only smaller than d. ◀

▶ **Lemma 12.** *If* Rta *finishes*

- *with at least one M_*-bin, then $\mathsf{mt}(D)$ is defined and $\mathsf{mt}(D) \lesssim \mathsf{min}(M_*)$;*
- *with at least one S_*-bin, then $\mathsf{mt}(D)$ is defined and $\mathsf{mt}(D) \lesssim 2\phi$.*

Proof. For the first part of the lemma, fix a medium item from M_* of size $\mathsf{min}(M_*)$. By the definition of Rta, it failed to mark this item. Hence, by Lemma 11, $\mathsf{mt}(D)$ is defined and $\mathsf{mt}(D) \lesssim \mathsf{min}(M_*)$.

Assume now that Rta finishes with at least one S_*-bin. When the first such S_*-bin was created, Rta placed a small item $s < \phi$ in the already existing A-bin of load $r < \phi$, and the merge action failed, because Rta failed to mark the resulting item of size $s + r$. Thus, again by Lemma 11, $\mathsf{mt}(D)$ is defined and $\mathsf{mt}(D) \lesssim s + r < 2\phi$. ◀

To estimate the gain on M_*-bins and S_*-bins, we define

$$\mathsf{level}_* = \begin{cases} \min\{\mathsf{pile}(\mathsf{mt}(D)), 1-\phi\} & \text{if } D \text{ contains a tight item and } S_* \neq \emptyset, \\ \mathsf{pile}(\mathsf{mt}(D)) & \text{if } D \text{ contains a tight item and } S_* = \emptyset, \\ 1 & \text{if } D \text{ does not contain any tight item.} \end{cases} \tag{7}$$

▶ **Lemma 13.** *It holds that $\mathsf{g}(M_* \uplus S_*) \gtrsim \mathsf{level}_* \cdot \mathsf{b}(M_* \uplus S_*)$.*

Proof. If D does not contain a tight item, then, by Lemma 12, both M_* and S_* are empty, and the lemma follows trivially. Thus, in the following we assume that D contains a tight item and we take a closer look at the contents of S_*-bins and M_*-bins.

Assume that M_* is non-empty. RTA creates a new M_i-bin (for $i \in \{2,3,4\}$) if the incoming medium item of category M_i (of size $(1/(i+1), 1/i]$) does not fit in any of the existing M_i bins. Hence, each M_i-bin (except at most one) has exactly i items, and therefore its load is greater than $i/(i+1) \geq 2/3$ and is also at least $i \cdot \min(M_i) \geq 2 \cdot \min(M_*)$. Thus,

$$\mathsf{g}(M_*) \gtrsim \max\{2/3, 2 \cdot \min(M_*)\} \cdot \mathsf{b}(M_*) = \mathsf{pile}(\min(M_*)) \cdot \mathsf{b}(M_*) \gtrsim \mathsf{pile}(\mathsf{mt}(D)) \cdot \mathsf{b}(M_*), \quad (8)$$

where the second inequality follows by Lemma 12 and by monotonicity of function pile. Note that (8) holds trivially also when there are no M_*-bins.

If there are no S_*-bins, then, $\mathsf{g}(M_* \uplus S_*) = \mathsf{g}(M_*) \gtrsim \mathsf{pile}(\mathsf{mt}(D)) \cdot \mathsf{b}(M_*) = \mathsf{level}_* \cdot \mathsf{b}(M_* \uplus S_*)$, and the lemma follows.

If there are some S_*-bins, recall that RTA creates a new S_*-bin only if the considered small item does not fit in any existing S_*-bin. Thus, the load of each S_*-bin (except at most one) is at least $1 - \phi$, and therefore $\mathsf{g}(S_*) \gtrsim (1-\phi) \cdot \mathsf{b}(S_*)$. Combining this with (8) implies $\mathsf{g}(M_* \uplus S_*) \gtrsim \mathsf{pile}(\mathsf{mt}(D)) \cdot \mathsf{b}(M_*) + (1-\phi) \cdot \mathsf{b}(S_*) \geq \min\{\mathsf{pile}(\mathsf{mt}(D)), 1-\phi\} \cdot \mathsf{b}(M_* \uplus S_*)$. ◀

4.4 When RTA terminates without empty bins and without M_S-bins

Using tight items, we may analyze the case when RTA terminates without empty bins and without M_S-bins, and show that in such case its gain is approximately greater than R. As the gain of OPT is at most 1, this yields the desired competitive ratio.

▶ **Lemma 14.** *If RTA terminates without empty bins, then $\mathsf{b}(L) + \mathsf{b}(M_\mathrm{S}) + \mathsf{b}(M_*) + \mathsf{b}(S_*) \approx 1$.*

Proof. There is at most one A-bin. The remaining bins (at least $n-1$ many) are of classes L_+, M_S, M_* or S_*, and thus $\mathsf{b}(L) + \mathsf{b}(M_\mathrm{S}) + \mathsf{b}(M_*) + \mathsf{b}(S_*) \approx 1$. ◀

▶ **Lemma 15.** *If on input σ, RTA terminates without empty bins and without M_S-bins, then RTA$(\sigma) \gtrsim R$.*

Proof. We analyze the gain of RTA on three disjoint sets: L, D and $M_* \uplus S_*$.

$$\begin{aligned} \textsc{Rta}(\sigma) &\geq \mathsf{g}(L) + \mathsf{g}(M_* \uplus S_*) + \mathsf{g}(D) \\ &\gtrsim \mathsf{F}(\mathsf{b}(L)) + \mathsf{level}_* \cdot (1 - \mathsf{b}(L)) + \mathsf{g}(D) && \text{(by L. 4, L. 13 and L. 14)} \\ &\gtrsim \mathsf{P}(\mathsf{level}_*) + \mathsf{g}(D) && \text{(by L. 6)} \end{aligned}$$

If D does not contain a tight item, then $\mathsf{level}_* = 1$, and thus RTA$(\sigma) \gtrsim \mathsf{P}(1) = R$.

If D contains a tight item, then by Lemma 10, $\mathsf{g}(D) \geq \mathsf{mt}(D) \cdot \xi(\mathsf{mt}(D))$, and therefore RTA$(\sigma) \gtrsim \mathsf{P}(\mathsf{level}_*) + \mathsf{mt}(D) \cdot \xi(\mathsf{mt}(D))$. We consider two cases.

- If $\mathsf{level}_* \geq \mathsf{pile}(\mathsf{mt}(D))$, then RTA$(\sigma) \gtrsim \mathsf{P}(\mathsf{pile}(\mathsf{mt}(D))) + \mathsf{mt}(D) \cdot \xi(\mathsf{mt}(D)) \geq R$, where the last inequality follows by Property 2 of Lemma 7.
- The opposite case, $\mathsf{level}_* < \mathsf{pile}(\mathsf{mt}(D))$, is possible only if S_*-bins exist and $\mathsf{level}_* = 1 - \phi$. By Lemma 12, the existence of S_*-bins implies $\mathsf{mt}(D) \lesssim 2\phi$. As the function $x \cdot \xi(x)$ is non-increasing (cf. Property 1 of Lemma 7), RTA$(\sigma) \gtrsim \mathsf{P}(1-\phi) + 2\phi \cdot \xi(2\phi) \geq R$. The last inequality follows by Property 3 of Lemma 7. ◀

5 Gain on large items revisited

In this section, we assume that RTA terminates without empty bins and with at least one M_S-bin. Recall that Lemma 4 allows us to estimate $\mathsf{g}(L_+)$ by calculating the gain on *large* items alone. Now we show how to improve this bound by taking into account non-large items in L_+. First, we leverage the fact that if a (marked) medium item is in M_S, then Rta must have failed to combine it with a large item, and we obtain a better lower bound on the size of each large item. Second, we show that in some cases marked medium items must be in L_+ which increases its load. If an M_S-bin exists, we define

$$\mathsf{T}^* = \begin{cases} \mathsf{T}(\mathsf{min}(M_\mathrm{S}), \mathsf{mt}(D)) & \text{if } \mathsf{mt}(D) \text{ is defined and } \mathsf{min}(M_\mathrm{S}) > \max\{\mathsf{mt}(D), 1/3\} \\ 0 & \text{otherwise,} \end{cases}$$

$$\mathsf{thr}(M_\mathrm{S}) = \min\{1 - \mathsf{min}(M_\mathrm{S}), 1/2 + \phi\}.$$

Note that T^* is always non-negative. In particular, $\mathsf{T}(\mathsf{min}(M_\mathrm{S}), \mathsf{mt}(D)) = (\mathsf{min}(M_\mathrm{S}) + \mathsf{mt}(D) - 1/2) \cdot (\xi(\mathsf{mt}(D)) - \xi(\mathsf{min}(M_\mathrm{S})) \geq 0$ because $\mathsf{min}(M_\mathrm{S}) + \mathsf{mt}(D) \geq 1/2$ for $\mathsf{min}(M_\mathrm{S}) > 1/3$ and $\mathsf{mt}(D) \geq \phi$.

▶ **Lemma 16.** *Assume Rta terminates with at least one M_S-bin. Then, the load of any L_+-bin is at least $\mathsf{thr}(M_\mathrm{S})$.*

Proof. Consider any L_+-bin b and let y be the large item contained in this bin. If b contains an additional medium item, then its load is greater than $1/2 + \phi$, and the lemma follows. Hence, in the following, we assume that y was not combined with a medium item in a single bin. As Rta finishes with an M_S-bin, we fix a medium item x of size $\mathsf{min}(M_\mathrm{S})$. We consider three cases.

- Item x arrived (or was created by merging some small items) before the arrival of y. Rta did not place y in the M_S-bin containing x, because $y + x > 1$. Thus, $\mathsf{load}(b) \geq y \geq 1 - x = 1 - \min(M_\mathrm{S})$.
- Item x arrived after the arrival of y. (Some small items might be placed together with y prior to the arrival of x.) As Rta placed x in a separate bin, it did not fit in b, i.e., the load of b at the time of the arrival of x was greater than $1 - x = 1 - \min(M_\mathrm{S})$.
- Item x was created by merging small items after the arrival of y. Let $s < x$ be the small item that caused the creation of x. Rta placed s in A-bin, because s did not fit in b, i.e., the load of b at that time was greater than $1 - s > 1 - x = 1 - \min(M_\mathrm{S})$. ◀

▶ **Lemma 17.** *Assume that Rta terminates with at least one M_S-bin. Then, $\mathsf{g}(L_+) \gtrsim \mathsf{T}^* + \int_0^{\mathsf{b}(L)} \max\{\mathsf{f}(y), \mathsf{thr}(M_\mathrm{S})\}\,\mathrm{d}y$.*

Proof. We sort accepted large items by their arrival time and denote the bin containing the i-th large item by b_i. The bin b_i contains a large item of size at least $\mathsf{f}(i/n)$ because of the acceptance threshold, and its load is at least $\mathsf{thr}(M_\mathrm{S})$ by Lemma 16, i.e., $\mathsf{load}(b_i) \geq \max\{\mathsf{f}(i/n), \mathsf{thr}(M_\mathrm{S})\}$.

We now show how to decrease the load in L_+-bins, so that the remaining load in bin b_i remains at least $\max\{\mathsf{f}(i/n), \mathsf{thr}(M_\mathrm{S})\}$ and the change in the total gain is approximately equal to T^*. This claim is trivial for $\mathsf{T}^* = 0$, so we assume $\mathsf{T}^* > 0$. This is possible only if a tight item exists, $\mathsf{min}(M_\mathrm{S}) > \mathsf{mt}(D)$ and $\mathsf{min}(M_\mathrm{S}) > 1/3$. As $\mathsf{min}(M_\mathrm{S}) > \mathsf{mt}(D)$, every marked medium item of size from the interval $[\mathsf{mt}(D), \mathsf{min}(M_\mathrm{S}))$ is in (a separate) L_+-bin; let $\tilde{L}$ be the set of these bins. As $M_\mathrm{S} \subseteq D^{\geq \mathsf{mt}(D)}$, $n \cdot \mathsf{b}(\tilde{L}) = |D^{\geq \mathsf{mt}(D)} \setminus M_\mathrm{S}| = |D^{\geq \mathsf{mt}(D)}| - |M_\mathrm{S}|$. Using the tightness of $\mathsf{mt}(D)$ and Lemma 8, $\mathsf{b}(\tilde{L}) = \mathsf{b}(D^{\geq \mathsf{mt}(D)}) - \mathsf{b}(M_\mathrm{S}) \gtrsim \xi(\mathsf{mt}(D)) - \xi(\mathsf{min}(M_\mathrm{S}))$. From each bin of $\tilde{L}$ we remove a load of $\mathsf{mt}(D) + \mathsf{min}(M_\mathrm{S}) - 1/2$. The induced change in the total gain is then approximately equal to $\mathsf{b}(\tilde{L}) \cdot (\mathsf{mt}(D) + \mathsf{min}(M_\mathrm{S}) - 1/2) = \mathsf{T}^*$.

We now analyze the load of bin b_i after the removal. The original load of bin b_i was at least $\mathsf{f}(i/n) + \mathsf{mt}(D)$, and after removal it is at least $\mathsf{f}(i/n) + \mathsf{min}(M_\mathrm{S}) - 1/2$. This amount is at least $\mathsf{f}(i/n)$ (as $\mathsf{min}(M_\mathrm{S}) \leq 1/2$) and at least $1 - \mathsf{min}(M_\mathrm{S}) \geq \mathsf{thr}(M_\mathrm{S})$ (as $\mathsf{f}(i/n) \geq 1/2$). Hence, the remaining load of b_i is at least $\max\{\mathsf{f}(i/n), \mathsf{thr}(M_\mathrm{S})\}$.

Thus, $\mathsf{g}(L_+) - \mathsf{T}^* \geq \frac{1}{n}\sum_{i=1}^{n\cdot\mathsf{b}(L)} \max\{\mathsf{f}(i/n), \mathsf{thr}(M_\mathrm{S})\} \approx \int_0^{\mathsf{b}(L)} \max\{\mathsf{f}(y), \mathsf{thr}(M_\mathrm{S})\}\,\mathrm{d}y$. Particular subsets of L_+ are depicted in Figure 3 (right); the removed part of gain T^* is depicted as (1). ◀

▶ **Lemma 18.** *Assume that* Rta *terminates with at least one* M_S*-bin. Then,* $\mathsf{g}(L_+) + \eta \cdot (1 - \mathsf{b}(L)) \gtrsim \mathsf{P}(\eta) + \mathsf{Q}(\min\{\mathsf{thr}(M_\mathrm{S}), \eta\}) + \mathsf{T}^*$ *for any* $\eta \in (1/2, 1]$.

Proof. We fix any $\eta \in (1/2, 1]$ and define $h = \min\{\mathsf{thr}(M_\mathrm{S}), \eta\}$. By Lemma 17, $\mathsf{g}(L_+) - \mathsf{T}^* + \eta \cdot (1 - \mathsf{b}(L)) \gtrsim \int_0^{\mathsf{b}(L)} \max\{\mathsf{f}(y), h\}\,\mathrm{d}y + \int_{\mathsf{b}(L)}^1 \eta\,\mathrm{d}y$. We denote this lower bound by $A(\mathsf{b}(L))$ and we analyze it as a function of $\mathsf{b}(L)$. When $\mathsf{b}(L) < \mathsf{f}^{-1}(h)$, then using $h \leq \eta$ we obtain $A(\mathsf{b}(L)) = \int_0^{\mathsf{b}(L)} h\,\mathrm{d}y + \int_{\mathsf{b}(L)}^1 \eta\,\mathrm{d}y \geq \int_0^{\mathsf{f}^{-1}(h)} h\,\mathrm{d}y + \int_{\mathsf{f}^{-1}(h)}^1 \eta\,\mathrm{d}y = A(\mathsf{f}^{-1}(h))$. Therefore, we need to lower-bound the value of $A(\mathsf{b}(L))$ only for $\mathsf{b}(L) \geq \mathsf{f}^{-1}(h)$. In such case,

$$\begin{aligned}
A(\mathsf{b}(L)) &= \int_0^{\mathsf{b}(L)} \max\{h, \mathsf{f}(y)\}\,\mathrm{d}y + \int_{\mathsf{b}(L)}^1 \eta\,\mathrm{d}y \\
&= \int_0^{\mathsf{b}(L)} \mathsf{f}(y)\,\mathrm{d}y + \int_{\mathsf{b}(L)}^1 \eta\,\mathrm{d}y + \int_0^{\mathsf{b}(L)} \max\{h - \mathsf{f}(y), 0\}\,\mathrm{d}y \\
&\geq \int_0^1 \min\{\mathsf{f}(y), \eta\} + \int_0^1 \max\{h - \mathsf{f}(y), 0\}\,\mathrm{d}y = \mathsf{P}(\eta) + \mathsf{Q}(h).
\end{aligned}$$ ◀

5.1 When RTA terminates without empty bins and with some M_S-bins

The following lemma combines our bounds on gains on L_+, M_*, S_* and M_S.

▶ **Lemma 19.** *Assume that* Rta *run on input* σ *terminates without empty bins and with at least one* M_S*-bin. Then, for any* $\eta \in (1/2, \mathsf{level}_*]$,

$$\textsc{Rta}(\sigma) \gtrsim \mathsf{P}(\eta) + \mathsf{Q}(\min\{\eta, \mathsf{thr}(M_\mathrm{S})\}) + \mathsf{T}^* + (\mathsf{min}(M_\mathrm{S}) - \eta) \cdot \xi(\mathsf{min}(M_\mathrm{S})).$$

Proof. By the lemma assumptions,

$$\begin{aligned}
\textsc{Rta}(\sigma) &\geq \mathsf{g}(L_+) + \mathsf{g}(M_* \uplus S_*) + \mathsf{g}(M_\mathrm{S}) \\
&\gtrsim \mathsf{g}(L_+) + \eta \cdot \mathsf{b}(M_* \uplus S_*) + \mathsf{min}(M_\mathrm{S}) \cdot \mathsf{b}(M_\mathrm{S}) && \text{(by L. 13 and L. 8)} \\
&\approx \mathsf{g}(L_+) + \eta \cdot (1 - \mathsf{b}(L_+)) + (\mathsf{min}(M_\mathrm{S}) - \eta) \cdot \mathsf{b}(M_\mathrm{S}). && \text{(by L. 14)}
\end{aligned}$$

Applying the guarantee of Lemma 18 to $\mathsf{g}(L_+) + \eta \cdot (1 - \mathsf{b}(L_+))$ concludes the proof. ◀

▶ **Lemma 20.** *Assume that* Rta *run on input* σ *terminates without empty bins and with at least one* M_S*-bin. If* $\mathsf{min}(M_\mathrm{S}) \leq 1/3$*, then* $\textsc{Rta}(\sigma) \gtrsim R$.

Proof. As $\mathsf{level}_* \geq 2/3$, we may apply Lemma 19 with $\eta = 2/3$. Note that $\mathsf{thr}(M_\mathrm{S}) \geq 2/3$ for $\min(M_\mathrm{S}) \leq 1/3$. Then,

$$\begin{aligned}
\textsc{Rta}(\sigma) &\gtrsim \mathsf{P}(2/3) + \mathsf{Q}(2/3) + (\mathsf{min}(M_\mathrm{S}) - 2/3) \cdot \xi(\mathsf{min}(M_\mathrm{S})) \\
&= 2/3 + (\phi - 2/3) \cdot \xi(\mathsf{min}(M_\mathrm{S})) \geq R. && \text{(by L. 6)}
\end{aligned}$$

The last inequality follows by Property 4 of Lemma 7. ◀

▶ **Lemma 21.** *Assume that* Rta *run on input* σ *terminates without empty bins and with at least one* M_S*-bin. If* $\mathsf{min}(M_\mathrm{S}) > 1/3$*, then* $\textsc{Rta}(\sigma) \gtrsim R$.

Proof. As $\mathsf{min}(M) > 1/3$, $\mathsf{thr}(M_\mathrm{S}) < 1 - \mathsf{min}(M_\mathrm{S}) \geq 2/3$. Lemma 19 applied with any $\eta \in [2/3, \mathsf{level}_*]$ yields

$$\textsc{Rta}(\sigma) \gtrsim \mathsf{P}(\eta) + \mathsf{Q}(1 - \mathsf{min}(M_\mathrm{S})) + \mathsf{T}^* + (\mathsf{min}(M_\mathrm{S}) - \eta) \cdot \xi(\mathsf{min}(M_\mathrm{S})). \tag{9}$$

First, we assume that D has no tight items. Then, $\mathsf{level}_* = 1$, and we may use (9) with $\eta = 2 \cdot \mathsf{min}(M_\mathrm{S})$ obtaining $\textsc{Rta}(\sigma) \geq \mathsf{P}(2 \cdot \mathsf{min}(M_\mathrm{S})) + \mathsf{Q}(1 - \mathsf{min}(M_\mathrm{S})) - \mathsf{min}(M_\mathrm{S}) \cdot \xi(\mathsf{min}(M_\mathrm{S})) \geq R$, where the last inequality follows by Property 7 of Lemma 7.

Second, we assume that D contains a tight item and we consider three cases.

- $\mathsf{level}_* < \mathsf{pile}(\mathsf{mt}(D))$. This relation is possible only when $\mathsf{level}_* = 1 - \phi$ and $\textsc{Rta}$ terminates with at least one S_*-bin. In this case, Lemma 12 implies that $\mathsf{mt}(D) \leq 2\phi$. We apply (9) with $\eta = 1 - \phi$ obtaining $\textsc{Rta}(\sigma) \geq \mathsf{P}(1 - \phi) + \mathsf{Q}(1 - \mathsf{min}(M_\mathrm{S})) + \mathsf{T}^* + (\mathsf{min}(M_\mathrm{S}) + \phi - 1) \cdot \xi(\mathsf{min}(M_\mathrm{S}))$, which is at least R by Property 5 of Lemma 7.
- $\mathsf{level}_* \geq \mathsf{pile}(\mathsf{mt}(D))$ and $\mathsf{min}(M_\mathrm{S}) \leq \mathsf{mt}(D)$. Using monotonicity of pile, $\mathsf{level}_* \geq \mathsf{pile}(\mathsf{min}(M_\mathrm{S})) = 2 \cdot \mathsf{min}(M_\mathrm{S})$. Applying (9) with $\eta = 2 \cdot \mathsf{min}(M_\mathrm{S})$ yields $\textsc{Rta}(\sigma) \gtrsim R$ by Property 7 of Lemma 7.
- $\mathsf{level}_* \geq \mathsf{pile}(\mathsf{mt}(D))$ and $\mathsf{min}(M) > \mathsf{mt}(D)$. In this case, $\mathsf{T}^* = \mathsf{T}(\min(M_\mathrm{S}), \mathsf{mt}(D))$. Applying (9) with $\eta = \mathsf{pile}(\mathsf{mt}(D))$ yields $\textsc{Rta}(\sigma) \gtrsim \mathsf{P}(\mathsf{pile}(\mathsf{mt}(D))) + \mathsf{Q}(1 - \mathsf{min}(M_\mathrm{S})) + \mathsf{T}(\min(M_\mathrm{S}), \mathsf{mt}(D)) + (\mathsf{min}(M_\mathrm{S}) - \mathsf{pile}(\mathsf{mt}(D)) \cdot \xi(\mathsf{min}(M_\mathrm{S}))$, which is at least R by Property 6 of Lemma 7. ◀

6 Competitive ratio of RTA

▶ **Theorem 22.** *The competitive ratio of* $\textsc{Rta}$ *for the multiple knapsack problem is at least* $R - O(1/n)$.

Proof. Fix an input σ. If $\textsc{Rta}(\sigma)$ terminates with some empty bins, then its competitive ratio follows by Lemma 5.

Hence, below we assume that $\textsc{Rta}$ terminates without empty bins. We presented three lemmas that cover all possible cases: there are no M_S-bins (Lemma 15), there are M_S-bins and $\mathsf{min}(M_\mathrm{S}) \leq 1/3$ (Lemma 20), and there are M_S-bins and $\mathsf{min}(M_\mathrm{S}) > 1/3$ (Lemma 21). In all these cases, we proved $\textsc{Rta}(\sigma) \gtrsim R$. As $\textsc{Opt}(\sigma) \leq 1$, the theorem follows. ◀

7 Proof of Lemma 7

We start with technical helper claims.

▶ **Fact 23.** *The functions below are the derivatives of functions* P, Q *and* ξ, *respectively.*

$$\mathsf{P}'(x) = -R \cdot \ln x$$
$$\mathsf{Q}'(x) = 1 + R \cdot \ln x$$
$$\xi'(x) = \begin{cases} -\xi_\mathrm{c}/x^2 & \textit{if } x < 1/3, \\ -18\xi_\mathrm{c} & \textit{if } x > 1/3 \end{cases}$$

▶ **Lemma 24.** $\mathsf{T}(x,y)$ *is a non-increasing function of* y *in the interval* $[\phi, 1/2]$.

Proof. Recall that $\mathsf{T}(x,y) = (x + y - 1/2) \cdot (\xi(y) - \xi(x))$ is defined for $x, y \in [\phi, 1/2]$. As the function $\mathsf{T}(x,y)$ is continuous and differentiable everywhere except $y = 1/3$, it suffices to show that its partial derivative $\partial_y \mathsf{T}(x,y)$ is non-positive (except $y = 1/3$). We have

$$\partial_y \mathsf{T}(x,y) = \xi(y) - \xi(x) + (x + y - 1/2) \cdot \xi'(y) \leq \xi(y) + y \cdot \xi'(y),$$

where for the inequality we used $\xi(x) \geq 0$ and $(x - 1/2) \cdot \xi'(y) \geq 0$.

If $y \leq 1/3$, then $\partial_y \mathsf{T}(x,y) \leq \xi_{\mathrm{c}}/y + y \cdot (-\xi_{\mathrm{c}}/y^2) = 0$. If $y > 1/3$, then $\partial_y \mathsf{T}(x,y) \leq 9\xi_{\mathrm{c}} \cdot (1-2y) - 18\xi_{\mathrm{c}} \cdot y = 9\xi_{\mathrm{c}} \cdot (1-4y) < 0$, which concludes the proof. ◀

▶ **Lemma 25.** *$P(2/3) = R - \xi_{\mathrm{c}}$ and for any $y \in [2/3, 1]$ it holds that $\mathsf{P}(y) \geq R + 3\xi_{\mathrm{c}} \cdot y - 3\xi_{\mathrm{c}}$.*

Proof. For the first part of the lemma, observe that by the definition of ξ_{c} (see (1)),

$$\mathsf{P}(2/3) + \xi_{\mathrm{c}} = 2/3 - R \cdot (2/3) \cdot \ln(4/3) + (1 + (2/3) \cdot \ln(4/3)) \cdot R - 2/3 = R.$$

To show the second relation, note that $\mathsf{P}(2/3) = R - \xi_{\mathrm{c}}$ and $\mathsf{P}(1) = R$. Let $h(y) = R + 3\xi_{\mathrm{c}} \cdot y - 3\xi_{\mathrm{c}}$ be the linear function that coincides with $\mathsf{P}(y)$ for $y = 2/3$ and $y = 1$. As the function P is concave on its whole domain (its second derivative $\mathsf{P}''(y) = -R/y$ is negative), we have $\mathsf{P}(y) \geq h(y)$ for any $y \in [2/3, 1]$. ◀

Proof of Lemma 7. Note that $\xi(1/3) = 3 \cdot \xi_{\mathrm{c}}$ and $\xi(1/2) = 0$, For each property, we define an appropriate function G_i that we analyze; for all properties except the first one, we show that the function value is at least R for an appropriate range of arguments.

Property 1. Let $G_1(x) = x \cdot \xi(x)$. Then $G_1(x) = \xi_{\mathrm{c}}$ for $x \in [\phi, 1/3]$, and for $x \in [1/3, 1/2]$ it holds that $G_1(x) = 9\xi_{\mathrm{c}} \cdot x \cdot (1-2x)$, i.e., the function $G_1(x)$ is decreasing. Hence, $G_1(x)$ is non-increasing in the whole domain $[\phi, 1/2]$.

Property 2. Let $G_2(x) = \mathsf{P}(\mathsf{pile}(x)) + x \cdot \xi(x)$; we want to show that $G_2(x) \geq R$ for any $x \in [\phi, 1/2]$. For $x \in [\phi, 1/3]$, it holds that $G_2(x) = \mathsf{P}(2/3) + \xi_{\mathrm{c}} = R$ (by Lemma 25). For $x \in [1/3, 1/2]$, $G_2(x) = \mathsf{P}(2x) + x \cdot \xi(x)$. Using Lemma 25, we obtain $G_2(x) \geq R + 6\xi_{\mathrm{c}} \cdot x - 3\xi_{\mathrm{c}} + 9\xi_{\mathrm{c}} \cdot (x - 2x^2) = R - 18\xi_{\mathrm{c}} \cdot (x - 1/2) \cdot (x - 1/3) \geq R$. The last inequality follows as for any $x \in [1/3, 1/2]$ the term $(x - 1/2) \cdot (x - 1/3)$ is non-positive.

Property 3. Let $G_3 = \mathsf{P}(1-\phi) + 2\phi \cdot \xi(2\phi)$. It can be verified numerically that $G_3 > 0.593 > R$.

Property 4. Let $G_4(x) = 2/3 - (2/3 - \phi) \cdot \xi(x)$. As the function $\xi(x)$ is decreasing, for any $x \in [\phi, 1/3]$ it holds that $G_4(x) \geq G_4(\phi) = 2/3 - (2/3) \cdot \xi_{\mathrm{c}}/\phi + \xi_{\mathrm{c}}$. Substituting the definition of ϕ (see (2)), we obtain $G_4(x) \geq G_4(\phi) \geq 2/3 - 2/3 + R - \xi_{\mathrm{c}} + \xi_{\mathrm{c}} = R$.

Property 5. Let $G_5(x,y) = \tilde{G}_5(x) + \max\{\mathsf{T}(x,y), 0\}$, where $\tilde{G}_5(x) = \mathsf{P}(1-\phi) + \mathsf{Q}(1-x) + (x + \phi - 1) \cdot \xi(x)$. We want to show that $G_5(x,y) \geq R$ for any $x \in [1/3, 1/2]$ and $y \in [\phi, 2\phi]$.

- If $x \in [1/3, 2\phi]$, then already $\tilde{G}_5(x) \geq R$. To show this relation, we estimate its derivative in the interval $[1/3, 2\phi]$:

$$\begin{aligned}\tilde{G}_5'(x) &= -1 - R \cdot \ln(1-x) + (x + \phi - 1) \cdot \xi'(x) + \xi(x) \\ &= -1 - R \cdot \ln(1-x) - 36\xi_{\mathrm{c}} \cdot x - 18\xi_{\mathrm{c}} \cdot \phi + 27\xi_{\mathrm{c}} \\ &\leq -1 - R \cdot \ln(1-2\phi) - 36\xi_{\mathrm{c}} \cdot (1/3) - 18\xi_{\mathrm{c}} \cdot \phi + 27\xi_{\mathrm{c}} < -0.2479 < 0.\end{aligned}$$

  Hence, $\tilde{G}_5$ is decreasing in the interval $[1/3, 2\phi]$, and thus for any $x \in [1/3, 2\phi]$ it holds that $G_5(x,y) \geq \tilde{G}_5(x) \geq \tilde{G}_5(2\phi) > 0.5997 > R$.
- If $x \in (2\phi, 1/2]$, then $\mathsf{T}(x,y) \geq 0$. As $\mathsf{T}(x,y)$ is a non-increasing function of y in the interval $[\phi, 2\phi]$ (by Lemma 24), it holds that $\mathsf{T}(x,y) \geq \mathsf{T}(x, 2\phi)$. Therefore,

$$G_5(x,y) \geq G_5(x, 2\phi) = \mathsf{P}(1-\phi) + \mathsf{Q}(1-x) - (\phi + 1/2) \cdot \xi(x) + (x + 2\phi - 1/2) \cdot \xi(2\phi).$$

We now estimate its partial derivative for $x \in [2\phi, 1/2]$:

$$\begin{aligned}\partial_x G_5(x, 2\phi) &= -1 - R \cdot \ln(1-x) - (\phi + 1/2) \cdot \xi'(x) + \xi(2\phi) \\ &\le -1 - R \cdot \ln(1/2) + 18\xi_{\mathrm{c}} \cdot (\phi + 1/2) + 9\xi_{\mathrm{c}} \cdot (1 - 4\phi) < -0.0673 < 0.\end{aligned}$$

Therefore $G_5(x, 2\phi)$ is decreasing as a function of x in the interval $[2\phi, 1/2]$. Thus, for the considered range of arguments, $G_5(x, y) \ge G_5(x, 2\phi) \ge G_5(1/2, 2\phi) > 0.5934 > R$.

Property 6. Let

$$G_6(x, y) = \mathsf{P}(\mathsf{pile}(y)) + \mathsf{Q}(1-x) + (x - \mathsf{pile}(y)) \cdot \xi(x) + \mathsf{T}(x, y)$$

Fix any pair $(x, y) \in [1/3, 1/2] \times [\phi, 1/2]$, such that $y \le x$. We prove that $G_6(x, y) \ge R$, by showing each of the following inequalities

$$G_6(x, y) \ge G_6(x, y') \ge \tilde{G}_6(x, y') \ge \tilde{G}_6(1/2, y'') \ge R, \tag{10}$$

where $y' = \max\{y, 1/3\}$ and $y'' = y' + 1/2 - x$. The function $\tilde{G}_6$ is a lower bound on function G_6 created by using Lemma 25 (and defined formally later).

The first inequality of (10) is trivial for $y \ge 1/3$, hence we assume that $y < 1/3$. In such case, using the definition of function pile, we get $G_6(x, y) = \mathsf{P}(2/3) + \mathsf{Q}(1-x) + (x - 2/3) \cdot \xi(x) + \mathsf{T}(x, y)$. As $\mathsf{T}(x, y)$ is a decreasing function of y, we obtain that $\mathsf{T}(x, y) \ge \mathsf{T}(x, y')$ and thus also $G_6(x, y) \ge G_6(x, y')$.

To show the second inequality of (10), we first simplify $G_6(x, y)$ using that $y \ge 1/3$ and the definition of pile:

$$G_6(x, y) = \mathsf{P}(2y) + \mathsf{Q}(1-x) + (x - 2y) \cdot \xi(x) + (x + y - 1/2) \cdot (\xi(y) - \xi(x))$$

Now, using Lemma 25, we have

$$G_6(x, y) \ge R + \mathsf{Q}(1-x) + 6\xi_{\mathrm{c}} \cdot y - 3\xi_{\mathrm{c}} + (1/2 - 3y) \cdot \xi(x) + (x + y - 1/2) \cdot \xi(y).$$

We denote the estimate above by $\tilde{G}_6(x, y)$ and we inspect its two partial derivatives.

$$\begin{aligned}\partial_x \tilde{G}_6(x, y) &= -1 - R \cdot \ln(1-x) + (1/2 - 3y) \cdot \xi'(x) + \xi(y) \\ \partial_y \tilde{G}_6(x, y) &= 6\xi_{\mathrm{c}} - 3 \cdot \xi(x) + \xi(y) + (x + y - 1/2) \cdot \xi'(y)\end{aligned}$$

The directional derivative along the vector $\mathbf{v} = (1, 1)$ is then equal to

$$\begin{aligned}\partial_{\mathbf{v}} \tilde{G}_6(x, y) &= \partial_x \tilde{G}_6(x, y) + \partial_y \tilde{G}_6(x, y) \\ &= -1 - R \cdot \ln(1-x) + 6\xi_{\mathrm{c}} - 3 \cdot \xi(x) + 2 \cdot \xi(y) - 18\xi_{\mathrm{c}} \cdot (x - 2y) \\ &= -1 - R \cdot \ln(1-x) + 36\xi_{\mathrm{c}} \cdot x - 3\xi_{\mathrm{c}} \\ &\le -1 - R \cdot \ln(1/2) + 36\xi_{\mathrm{c}} \cdot (1/2) - 3\xi_{\mathrm{c}} < -0.0322 < 0.\end{aligned}$$

This means that if we take any point $(x, y') \in [1/3, 1/2] \times [1/3, 1/2]$, where $y' \le x$, and move along vector $\mathbf{v}$, to point $(1/2, y'') = (1/2, y + 1/2 - x)$, the value of the function $\tilde{G}_6$ can only decrease. This concludes the proof of the third inequality of (10).

To show the final inequality of (10), we fix any $y'' \in [1/3, 1/2]$. Then $\tilde{G}_6(1/2, y'') = R + 6\xi_{\mathrm{c}} \cdot y - 3\xi_{\mathrm{c}} + y'' \cdot \xi(y'') = R - 18\xi_{\mathrm{c}} \cdot (y'' - 1/2) \cdot (y'' - 1/3) \ge R$, where the last inequality holds as $(y'' - 1/2) \cdot (y'' - 1/3)$ is non-positive.

Property 7. Let $G_7(x) = \mathsf{P}(2x) + \mathsf{Q}(1-x) - x \cdot \xi(x)$. For $x \in [1/3, 1/2]$, it holds that $G_7(x) = G_6(x, x)$. Hence $G_7(x) \ge R$ for $x \in [1/3, 1/2]$ follows by Property 6. ◀

References

1 Susanne Albers, Arindam Khan, and Leon Ladewig. Improved online algorithms for knapsack and GAP in the random order model. In *Proc. 22nd Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, pages 22:1–22:23, 2019. doi:10.4230/LIPIcs.APPROX-RANDOM.2019.22.

2 Yossi Azar, Joan Boyar, Lene M. Favrholdt, Kim S. Larsen, Morten N. Nielsen, and Leah Epstein. Fair versus unrestricted bin packing. *Algorithmica*, 34(2):181–196, 2002. doi:10.1007/s00453-002-0965-6.

3 Moshe Babaioff, Nicole Immorlica, David Kempe, and Robert Kleinberg. A knapsack secretary problem with applications. In *Proc. 11th Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, pages 16–28, 2007. doi:10.1007/978-3-540-74208-1_2.

4 Hans-Joachim Böckenhauer, Dennis Komm, Richard Královic, and Peter Rossmanith. The online knapsack problem: Advice and randomization. *Theoretical Computer Science*, 527:61–72, 2014. doi:10.1016/j.tcs.2014.01.027.

5 Joan Boyar, Lene M. Favrholdt, Kim S. Larsen, and Morten N. Nielsen. The competitive ratio for on-line dual bin packing with restricted input sequences. *Nordic Journal of Computing*, 8(4):463–472, 2001.

6 Chandra Chekuri and Sanjeev Khanna. A polynomial time approximation scheme for the multiple knapsack problem. *SIAM Journal on Computing*, 35(3):713–728, 2005. doi:10.1137/S0097539700382820.

7 Marek Cygan, Łukasz Jeż, and Jirí Sgall. Online knapsack revisited. *Theory of Computing Systems*, 58(1):153–190, 2016. doi:10.1007/s00224-014-9566-4.

8 Xin Han, Yasushi Kawase, and Kazuhisa Makino. Randomized algorithms for online knapsack problems. *Theoretical Computer Science*, 562:395–405, 2015. doi:10.1016/j.tcs.2014.10.017.

9 Xin Han, Yasushi Kawase, Kazuhisa Makino, and Haruki Yokomaku. Online knapsack problems with a resource buffer. In *Proc. 30th Int. Symp. on Algorithms and Computation (ISAAC)*, pages 28:1–28:14, 2019. doi:10.4230/LIPIcs.ISAAC.2019.28.

10 Kazuo Iwama and Shiro Taketomi. Removable online knapsack problems. In *Proc. 29th Int. Colloq. on Automata, Languages and Programming (ICALP)*, pages 293–305, 2002. doi:10.1007/3-540-45465-9_26.

11 Kazuo Iwama and Guochuan Zhang. Online knapsack with resource augmentation. *Information Processing Letters*, 110(22):1016–1020, 2010. doi:10.1016/j.ipl.2010.08.013.

12 Hans Kellerer. A polynomial time approximation scheme for the multiple knapsack problem. In *Proc. 3rd Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, pages 51–62, 1999. doi:10.1007/978-3-540-48413-4_6.

13 Hans Kellerer. Knapsack. In Ming-Yang Kao, editor, *Encyclopedia of Algorithms*, pages 1048–1051. Springer, 2016.

14 Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack Problems.* Springer, 2004.

15 Thomas Kesselheim, Klaus Radke, Andreas Tönnis, and Berthold Vöcking. Primal beats dual on online packing LPs in the random-order model. *SIAM Journal on Computing*, 47(5):1939–1964, 2018. doi:10.1137/15M1033708.

16 Alberto Marchetti-Spaccamela and Carlo Vercellis. Stochastic on-line knapsack problems. *Mathematical Programming*, 68:73–104, 1995. doi:10.1007/BF01585758.

17 John Noga and Veerawan Sarbua. An online partially fractional knapsack problem. In *8th Int. Symp. on Parallel Architectures, Algorithms, and Networks (ISPAN)*, pages 108–112, 2005. doi:10.1109/ISPAN.2005.19.

18 David Pisinger. An exact algorithm for large multiple knapsack problems. *European Journal of Operational Research*, 114(3):528–541, 1999. doi:10.1016/S0377-2217(98)00120-9.

19 Rahul Vaze. Online knapsack problem and budgeted truthful bipartite matching. In *Proc. 36th IEEE Int. Conf. on Computer Communications (INFOCOM)*, pages 1–9, 2017. doi:10.1109/INFOCOM.2017.8057223.

Space Efficient Construction of Lyndon Arrays in Linear Time

Philip Bille
DTU Compute, Technical University of Denmark,
Lyngby, Denmark
phbi@dtu.dk

Jonas Ellert
Department of Computer Science,
Technical University of Dortmund, Germany
jonas.ellert@tu-dortmund.de

Johannes Fischer
Department of Computer Science,
Technical University of Dortmund, Germany
johannes.fischer@cs.tu-dortmund.de

Inge Li Gørtz
DTU Compute, Technical University of Denmark,
Lyngby, Denmark
inge@dtu.dk

Florian Kurpicz
Department of Computer Science,
Technical University of Dortmund, Germany
florian.kurpicz@tu-dortmund.de

J. Ian Munro
Cheriton School of Computer Science,
University of Waterloo, Canada
imunro@uwaterloo.ca

Eva Rotenberg
DTU Compute, Technical University of Denmark,
Lyngby, Denmark
erot@dtu.dk

Abstract

Given a string S of length n, its *Lyndon array* identifies for each suffix $S[i..n]$ the next lexicographically smaller suffix $S[j..n]$, i.e. the minimal index $j > i$ with $S[i..n] \succ S[j..n]$. Apart from its plain $(n \log_2 n)$-bit array representation, the Lyndon array can also be encoded as a succinct parentheses sequence that requires only $2n$ bits of space. While linear time construction algorithms for both representations exist, it has previously been unknown if the same time bound can be achieved with less than $\Omega(n \lg n)$ bits of additional working space. We show that, in fact, $o(n)$ additional bits are sufficient to compute the succinct $2n$-bit version of the Lyndon array in linear time. For the plain $(n \log_2 n)$-bit version, we only need $\mathcal{O}(1)$ additional words to achieve linear time. Our space efficient construction algorithm makes the Lyndon array more accessible as a fundamental data structure in applications like full-text indexing.

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

Keywords and phrases String algorithms, string suffixes, succinct data structures, Lyndon word, Lyndon array, nearest smaller values, nearest smaller suffixes

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.14

Category Track A: Algorithms, Complexity and Games

Related Version `https://arxiv.org/abs/1911.03542`

Supplementary Material `https://github.com/jonas-ellert/nearest-smaller-suffixes`

Funding *Philip Bille*: Supported by the Danish Research Council (DFF-9131-00069B).
Inge Li Gørtz: Partially supported by the Danish Research Council (DFF-8021-002498).
Florian Kurpicz: Supported by the German Research Foundation DFG SPP 1736 "Algorithms for Big Data".
J. Ian Munro: Supported by the Canada Research Chairs Programme and NSERC Discovery Grant 8237.
Eva Rotenberg: Partially supported by Independent Research Fund Denmark grants 2020-2023 (9131-00044B), 2020-2023 (8021-002498), and 2018-2021 (8021-00249B).

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 14; pp. 14:1–14:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction & Related Work

The Lyndon array [5] is a well-known combinatorial object on strings and has gained renewed attention [6, 7, 14, 18, 19, 23] due to its recently discovered central role in combinatorics on strings, e.g., when computing all the runs in a string [2]. It is known [18, 13] that the Lyndon array can be computed in linear time from the list of lexicographically sorted suffixes (the *suffix array*). Baier [1] introduced the first *direct* algorithm for computing the Lyndon array – interestingly as a *preliminary* step for his new suffix sorting algorithm. However, it requires $\Theta(n \lg n)$ bits of additional working space even for just computing the Lyndon array. Other Lyndon array construction algorithms have been introduced [13, 19, 20], but they all have the same space bounds.

The Lyndon array has some structural properties that allow for a more space efficient representation, namely using only $2n + 2$ bits [19]. Thus, it would be desirable to compute this succinct representation using less than $\Theta(n \lg n)$ bits of working space, without sacrificing the linear running time. Previously, no such algorithm was known.

Our Contributions. We introduce the first algorithm that computes the succinct Lyndon array in $\mathcal{O}(n)$ time, using only $\mathcal{O}(n \lg\lg n/ \lg n)$ bits of additional working space. Alternatively, our algorithm can construct the plain ($\mathcal{O}(n \lg n)$-bits) Lyndon array using only $\mathcal{O}(1)$ words of additional working space, i.e., directly without precomputing the suffix array. In practice, our approach is up to 10 times faster than previous algorithms for the Lyndon array.

The rest of the paper is organized as follows: In Section 2 we introduce the notation and definitions that we use throughout the paper. Section 3 provides a new intuitive definition of the succinct Lyndon array. We introduce our construction algorithm for the succinct Lyndon array in Sections 4 and 5, and adapt it such that it computes the plain ($\mathcal{O}(n \lg n)$-bits) Lyndon array in Section 6. Finally, we present experimental results for both versions (Section 7), and summarize our findings (Section 8).

2 Preliminaries

We write $\lg x$ for $\log_2 x$. For $i, j \in \mathbb{N}$, the interval $[i, j]$ represents $\{x \mid x \in \mathbb{N} \wedge i \leq x \leq j\}$. We use the notation $[i, j + 1) = (i - 1, j] = (i - 1, j + 1) = [i, j]$ for open and half-open discrete intervals. Our analysis is performed in the word RAM model [17], where we can perform fundamental operations (logical shifts, basic arithmetic operations etc.) on words of size w bits in constant time. For the input size n of our problems we assume $\lceil \lg n \rceil \leq w$.

A *string* (also called *text*) over the *alphabet* Σ is a finite sequence of *symbols* from the finite and totally ordered set Σ. We say that a string $\mathcal{S}$ has length n and write $|\mathcal{S}| = n$, iff $\mathcal{S}$ is a sequence of exactly n symbols. The i-th symbol of a string $\mathcal{S}$ is denoted by $\mathcal{S}[i]$, while the *substring* from the i-th to the j-th symbol is denoted by $\mathcal{S}[i..j]$. For convenience we use the interval notations $\mathcal{S}[i..j + 1) = \mathcal{S}(i - 1..j] = \mathcal{S}(i - 1..j + 1) = \mathcal{S}[i..j]$. The i-th *suffix* of $\mathcal{S}$ is defined as $\mathcal{S}_i = \mathcal{S}[i..n]$, while the substring $\mathcal{S}[1..i]$ is called *prefix* of $\mathcal{S}$. A prefix or suffix of $\mathcal{S}$ is called *proper*, iff its length is at most $n - 1$. The concatenation of two strings $\mathcal{S}$ and $\mathcal{T}$ is denoted by $\mathcal{S} \cdot \mathcal{T}$. The length of the *longest common prefix (LCP)* between $\mathcal{S}$ and $\mathcal{T}$ is defined as $\textsc{lcp}(\mathcal{S}, \mathcal{T}) = \max\{\ell \mid \mathcal{S}[1..\ell] = \mathcal{T}[1..\ell]\}$. The *longest common extension (LCE)* of indices i and j is the length of the LCP between $\mathcal{S}_i$ and $\mathcal{S}_j$, i.e. $\textsc{lce}(i, j) = \textsc{lcp}(\mathcal{S}_i, \mathcal{S}_j)$. We can simplify the description of our algorithm by introducing a special symbol $\$ \notin \Sigma$ that is smaller than all symbols from Σ. For a string $\mathcal{S}$ of length n we define the 0-*th suffix* $\mathcal{S}_0 = \$$ as well as the $(n + 1)$-*st suffix and position* $\mathcal{S}_{n+1} = \mathcal{S}[n + 1] = \$$. The total order on Σ

induces a total order on the set Σ^* of strings over Σ. Let $\mathcal{S}$ and $\mathcal{T}$ be strings over Σ, and let $\ell = \textsc{lcp}(\mathcal{S}, \mathcal{T})$. We say that $\mathcal{S}$ is lexicographically smaller than $\mathcal{T}$ and write $\mathcal{S} \prec \mathcal{T}$, iff we have $\mathcal{S} \neq \mathcal{T}$ and $\mathcal{S}[\ell+1] < \mathcal{T}[\ell+1]$. Analogously, we say that $\mathcal{S}$ is lexicographically larger than $\mathcal{T}$ and write $\mathcal{S} \succ \mathcal{T}$, iff we have $\mathcal{S} \neq \mathcal{T}$ and $\mathcal{S}[\ell+1] > \mathcal{T}[\ell+1]$.

2.1 The Lyndon Array & Nearest Smaller Suffixes

A *Lyndon word* is a string that is lexicographically smaller than all of its proper suffixes, i.e. $\mathcal{S}$ is a Lyndon word, iff $\forall i \in [2..n] : \mathcal{S}_i \succ \mathcal{S}$ holds [8]. For example, the string `northamerica` is not a Lyndon word because its suffix `america` is lexicographically smaller than itself. On the other hand, its substring `americ` is a Lyndon word. The Lyndon array of $\mathcal{S}$ identifies the longest Lyndon word at each position of $\mathcal{S}$:

▶ **Definition 1** (Lyndon Array). *Given a string $\mathcal{S}$ of length n, the Lyndon array is an array λ of n integers with $\lambda[i] = \max\{\ell \mid \mathcal{S}[i..i+\ell)$ is a Lyndon word$\}$.*

▶ **Definition 2** (Nearest Smaller Suffixes). *Given a string $\mathcal{S}$ and a suffix $\mathcal{S}_i$, the* next smaller suffix of $\mathcal{S}_i$ *is $\mathcal{S}_j$, where j is the smallest index that is larger than i and satisfies $\mathcal{S}_i \succ \mathcal{S}_j$. The* previous smaller suffix of $\mathcal{S}_i$ *is defined analogously. The* next-smaller-suffix array (NSS array) and previous-smaller-suffix array (PSS array) *are arrays of size n defined as follows:*

$$\mathsf{nss}[i] = \min\{j \mid j \in (i, n+1] \wedge \mathcal{S}_i \succ \mathcal{S}_j\} \qquad \mathsf{pss}[i] = \max\{j \mid j \in [0, i) \wedge \mathcal{S}_j \prec \mathcal{S}_i\}$$

The Lyndon array and nearest smaller suffixes are highly related to each other. In fact, the NSS array is merely a different representation of the Lyndon array (Lemma 3), as visualized in Figure 1a. We conclude the preliminaries by showing a slightly weaker connection between the PSS array and Lyndon words (Lemma 4).

▶ **Lemma 3** (Lemma 15 [13]). *The longest Lyndon word at position i ends at the starting position of the NSS of $\mathcal{S}_i$, i.e. $\lambda[i] = \mathsf{nss}[i] - i$.*

▶ **Lemma 4.** *Let $\mathsf{pss}[j] = i > 0$, then $\mathcal{S}[i..j)$ is a Lyndon word.*

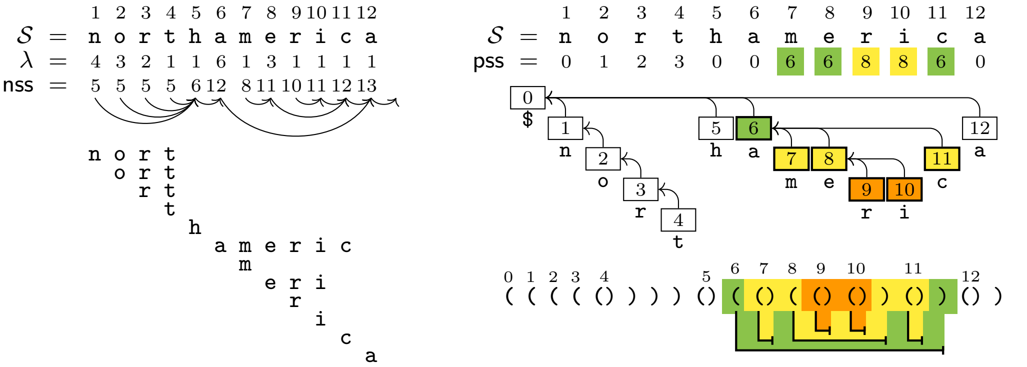
Proof. By definition, the string $\mathcal{S}[i..j)$ is a Lyndon word iff there exists no $k \in (i, j)$ with $\mathcal{S}[k..j) \prec \mathcal{S}[i..j)$. Assume that such a k exists. Since $i = \mathsf{pss}[j]$, we know that (a) $\mathcal{S}_k \succ \mathcal{S}_i$. Now assume there is a mismatching character between $\mathcal{S}[k..j)$ and $\mathcal{S}[i..j)$. Then appending $\mathcal{S}_j$ to both strings preserves this mismatch. This implies that we have $\mathcal{S}[k..j) \prec \mathcal{S}[i..j) \iff \mathcal{S}[k..j) \cdot \mathcal{S}_j \prec \mathcal{S}[i..j) \cdot \mathcal{S}_j$, and thus $\mathcal{S}_k \prec \mathcal{S}_i$, which contradicts (a). Therefore, we know that (b) $\mathcal{S}[k..j) = \mathcal{S}[i..i+(j-k))$. Then

$$\mathcal{S}_k \overset{(a)}{\succ} \mathcal{S}_i \quad \iff \quad \mathcal{S}[k..j) \cdot \mathcal{S}_j \succ \mathcal{S}[i..i+(j-k)) \cdot \mathcal{S}_{i+(j-k)} \quad \overset{(b)}{\iff} \quad \mathcal{S}_j \succ \mathcal{S}_{i+(j-k)}$$

which contradicts the fact that $\mathsf{pss}[j] = i < i + (j-k)$. Hence, the described k cannot exist, and $\mathcal{S}[i..j)$ must be a Lyndon word. ◀

3 Previous-Smaller-Suffix Trees

In this section we introduce the *previous-smaller-suffix tree*, which simulates access to the Lyndon array, the NSS array, and the PSS array. The PSS array inherently forms a tree in which each index i is represented by a node whose parent is $\mathsf{pss}[i]$. The root is the artificial index 0, which is parent of all indices that do not have a PSS (see Figure 1b for an example).



(a) Lyndon array, NSS array, and maximal Lyndon words at all indices of $\mathcal{S}$.

(b) PSS array, PSS tree, and BPS representation of the PSS tree of $\mathcal{S}$. (Best viewed in color.)

Figure 1 Data structures for $\mathcal{S} = \texttt{northamerica}$.

▶ **Definition 5** (Previous-Smaller-Suffix Tree $\mathcal{T}_{\mathsf{pss}}$). *Let $\mathcal{S}$ be a string of length n. The* previous-smaller-suffix tree (PSS tree) *of $\mathcal{S}$ is an ordinal tree $\mathcal{T}_{\mathsf{pss}}$ with nodes $[0, n]$ and root 0. For $i \in [1, n]$, we define* Parent$(i) = \mathsf{pss}[i]$. *The children are arranged in ascending order, i.e. if i is a left-side sibling of j, then $i < j$ holds.*

The PSS tree is highly similar to the Left-to-Right-Minima (LRM) tree [3, 9, 22], which we will briefly explain now. Given an array $A[1, n]$ with artificial minimum $A[0] = -\infty$, let $\mathsf{psv}[i] = \max\{j \mid j \in [0, i) \wedge A[j] < A[i]\}$ be the index of the *previous smaller value (PSV)* of $A[i]$. The LRM tree of A is an ordinal tree in which each index i is a child of $\mathsf{psv}[i]$, and the children are ordered ascendingly (i.e. the only difference to the PSS tree is that we consider previous smaller *values* instead of previous smaller *suffixes*). If A is the inverse suffix array of $\mathcal{S}$, then by definition of the inverse suffix array we have $\forall i, j \in [1, n] : \mathcal{S}_i \prec \mathcal{S}_j \Leftrightarrow A[i] < A[j]$. It follows that the PSS tree of a string is identical to the LRM tree of its inverse suffix array. Consequently, an important property of the LRM tree also applies to the PSS tree:

▶ **Corollary 6** (Lemma 1 [9]). *The nodes of the PSS tree directly correspond to the preorder-numbers, i.e. node i has preorder-number i (if the first preorder-number is 0).*

The corollary allows us to simulate the NSS array with the PSS tree:

▶ **Lemma 7.** *Given the PSS tree, NSS array and Lyndon array of the same string we have* $\mathsf{nss}[i] = i + \textsc{SubtreeSize}(i)$ *and thus* $\lambda[i] = \textsc{SubtreeSize}(i)$.[1]

Proof. Since the nodes directly correspond to the preorder-numbers (Corollary 6), it follows that the descendants of i form a consecutive interval $(i, r]$. Since $i + \textsc{SubtreeSize}(i) = i + (r - i + 1) = r + 1$ holds, we only have to show $\mathsf{nss}[i] = r + 1$. Assume $r = n$, then there is no index larger than i that is not a descendant of i. Clearly, in this case i does not have an NSS, and thus it follows $\mathsf{nss}[i] = n + 1 = r + 1$. Assume $r < n$ instead, then $\mathcal{S}_{r+1}$ must be lexicographically smaller than all suffixes that begin at positions from $[i, r]$, since otherwise $r + 1$ would be a descendant of i. Therefore, $\mathcal{S}_{r+1}$ is the first suffix that starts right of i and is lexicographically smaller than $\mathcal{S}_i$, which means $\mathsf{nss}[i] = r + 1$. ◀

[1] $\textsc{SubtreeSize}(i)$ denotes the number of nodes in the subtree that is rooted in i, including i itself.

3.1 Storing the PSS Tree as a BPS

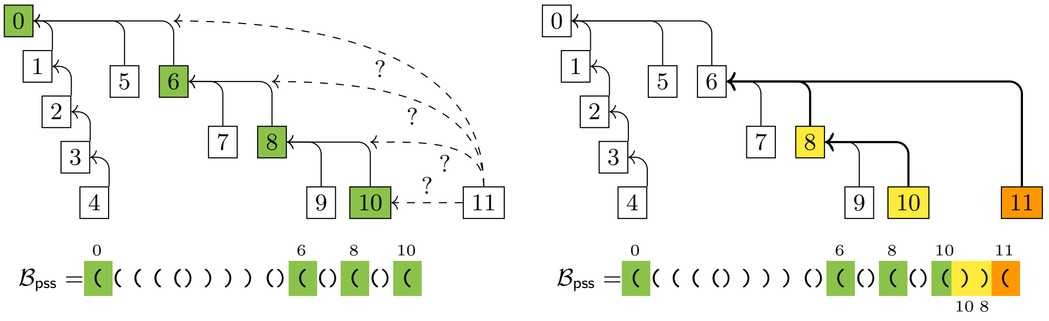
We store the PSS tree as a balanced parentheses sequence (BPS, [21]) of length $2n+2$, which takes $2n+2$ bits. Note that this is less than the $\approx 2.54n$ bits that are necessary to encode previous and next smaller *values* [10] because different suffixes of a text cannot be equal. As a shorthand for the BPS of the PSS tree we write $\mathcal{B}_{\mathsf{pss}}$. The sequence is algorithmically defined by a preorder-traversal of the PSS tree, where we write an opening parenthesis whenever we *walk down* an edge, and a closing one whenever we *walk up* an edge. An example is provided in Figure 1b. Note that the BPS of the PSS tree is identical to the succinct Lyndon array presentation from [19]. While the BPS itself does not support fast tree operations, it can be used to construct the data structure from [22], which takes $\mathcal{O}(n)$ time and $\mathcal{O}(n)$ bits of working space. This data structure is of size $2n + \mathcal{O}(n/\lg^c n)$ bits (for any $c \in \mathbb{N}^+$ of our choice) and supports $\textsc{Parent}(\cdot)$ and $\textsc{SubtreeSize}(\cdot)$ operations in $\mathcal{O}(c^2)$ time, and thus allows us to simulate access to the Lyndon array in $\mathcal{O}(c^2)$ time using Lemma 7.

Operations on a BPS Prefix. Since we will be building $\mathcal{B}_{\mathsf{pss}}$ from left to right, at any given point of the algorithm execution we know a prefix of $\mathcal{B}_{\mathsf{pss}}$. It is crucial that we maintain support for the following queries in constant time:

- Given the index o_i of an opening parenthesis in $\mathcal{B}_{\mathsf{pss}}$, determine the node i that belongs to the parenthesis. We have $i = \text{rank}_{\text{open}}(o_i) - 1$, where $\text{rank}_{\text{open}}(o_i)$ is the number of opening parentheses in $\mathcal{B}_{\mathsf{pss}}[1..o_i]$.
- Given a preorder-number i, find the index o_i of the corresponding opening parenthesis in $\mathcal{B}_{\mathsf{pss}}$. We have $o_i = \text{select}_{\text{open}}(i) = \min\{o \mid \text{rank}_{\text{open}}(o) > i\}$.
- Given an integer $k \geq 1$, find the index $o_{\text{uncl}(k)} = \text{select}_{\text{uncl}}(k)$ of the k-th *unclosed* parenthesis in $\mathcal{B}_{\mathsf{pss}}$. An opening parenthesis is called unclosed, if we have not written the matching closing parenthesis yet. For example, there are five opening parentheses in (()((), but only the first and the third one are unclosed.

There are support data structures of size $\mathcal{O}(n \lg\lg n / \lg n)$ bits that answer $\text{rank}_{\text{open}}$ and $\text{select}_{\text{open}}$ queries in constant time [16]. Since these data structures can be constructed in linear time by scanning the BPS from left to right, clearly we can maintain them with no significant time overhead when writing the BPS in an append-only manner. The structure for $\text{select}_{\text{uncl}}$ is a simple modification of the structure for $\text{select}_{\text{open}}$. Consider the (not necessarily balanced) parentheses sequence $\hat{\mathcal{B}}$ with $\hat{\mathcal{B}}[i] =$ (, iff $\mathcal{B}_{\mathsf{pss}}[i]$ is an unclosed parenthesis, and otherwise $\hat{\mathcal{B}}[i] =$). Clearly, answering $\text{select}_{\text{uncl}}$ on $\mathcal{B}_{\mathsf{pss}}$ is equivalent to answering $\text{select}_{\text{open}}$ on $\hat{\mathcal{B}}$. Thus, if we construct the $\text{select}_{\text{open}}$ data structure by Golynski [16, Section 2.1] for $\hat{\mathcal{B}}$, then we already have a working index for $\text{select}_{\text{uncl}}$ on $\mathcal{B}_{\mathsf{pss}}$. However, this approach comes at the cost of additional $2n$ bits of space because we need to explicitly store $\hat{\mathcal{B}}$. In the following paragraph, we outline how to modify the index such that queries can be answered directly on $\mathcal{B}_{\mathsf{pss}}$, i.e. without $\hat{\mathcal{B}}$. The reader should be familiar with [16, Section 2.1].

Assume that we want to answer $\text{select}_{\text{open}}(i)$ on $\hat{\mathcal{B}}$. Golynski's index conceptually splits $\hat{\mathcal{B}}$ into chunks of size $\lg n - 3\lg\lg n$. At the time we actually need to access $\hat{\mathcal{B}}$, we have already identified the chunk $\hat{C} = \hat{\mathcal{B}}[c_x..c_x + \lg n - 3\lg\lg n)$ and the value p such that the i-th opening parenthesis in $\hat{\mathcal{B}}$ is exactly the p-th opening parenthesis in $\hat{C}$. Answering the query is realized by simply retrieving the index j of the p-th opening parenthesis within $\hat{C}$ from a precomputed lookup table. Then, the result of the query is $c_x + j - 1$. We can answer the query without $\hat{\mathcal{B}}$, if we retrieve the chunk $C = \mathcal{B}_{\mathsf{pss}}[c_x..c_x + \lg n - 3\lg\lg n)$ directly from $\mathcal{B}_{\mathsf{pss}}$ (instead of retrieving the chunk $\hat{C}$ from $\hat{\mathcal{B}}$). We only need to change the precomputed lookup table such that it returns the index of the p-th unclosed parenthesis within the chunk instead of the index of the p-th opening parenthesis.


0
1
5
6
2
7
8
3
4
9
10
11
?
?
?
?
0
6
8
10
$\mathcal{B}_{\mathsf{pss}}$ = ((((( )))) ( ) (( ) (( ) (
0
6
8
10
11
$\mathcal{B}_{\mathsf{pss}}$ = ((((( )))) ( ) (( ) (( ) ( )) (
10 8

(a) Before.

(b) After.

Figure 2 The partial PSS tree before and after processing index 11 of $\mathcal{S} = \texttt{northamerica}$ during the execution of Algorithm 1. We have $p_1 = 10$, $p_2 = 8$, $p_3 = 6$, $p_4 = 0$, and $p_m = p_3$. (Best viewed in color.)

Appending parentheses to $\mathcal{B}_{\mathsf{pss}}$ may invalidate parts of the support data structure for $\text{select}_{\text{uncl}}$. The reason for this is that we essentially emulate $\text{select}_{\text{uncl}}$ on $\mathcal{B}_{\mathsf{pss}}$ by answering $\text{select}_{\text{open}}$ on $\hat{\mathcal{B}}$. Appending a closing parenthesis to $\mathcal{B}_{\mathsf{pss}}$ not only translates to appending a closing parenthesis to $\hat{\mathcal{B}}$, but also means that we have to replace the rightmost opening parenthesis in $\hat{\mathcal{B}}$ with a closing one. Thus, we may have to change previously computed parts of the support data structure. This can easily be handled without significant time overhead by only periodically updating the data structure, and naively keeping track of newly appended parentheses in between updates. We omit the details.

4 Constructing the PSS Tree

In this section we introduce our construction algorithm for the BPS of the PSS tree, which processes the indices of the input text in left-to-right order. Processing index i essentially means that we attach i to a partial PSS tree that is induced by the nodes from $[0, i)$. An example is provided in Figure 2a. But how can we efficiently determine i's parent $\mathsf{pss}[i]$? Consider the nodes on the rightmost path of the partial tree, which starts at $i-1$ and ends at the root 0. We call the set of these nodes *PSS closure* $\mathcal{P}_{i-1}$ of $i-1$ because it contains exactly the nodes that can be obtained by repeated application of the PSS function on $i-1$. For $j \in [1, n]$ we recursively define $\mathcal{P}_0 = \{0\}$ and $\mathcal{P}_j = \{j\} \cup \mathcal{P}_{\mathsf{pss}[j]}$. Interestingly, $\mathsf{pss}[i]$ is a member of $\mathcal{P}_{i-1}$:

▶ **Lemma 8.** *For any index $i \in [1, n]$ we have* $\mathsf{pss}[i] = \max\{j \mid j \in \mathcal{P}_{i-1} \wedge \mathcal{S}_j \prec \mathcal{S}_i\}$.

Proof. If we show $\mathsf{pss}[i] \in \mathcal{P}_{i-1}$, then the correctness of the lemma follows from Definition 2. Assume $\mathsf{pss}[i] \notin \mathcal{P}_{i-1}$, then there is some index $j \in \mathcal{P}_{i-1}$ with $\mathsf{pss}[i] \in (\mathsf{pss}[j], j)$. By Definition 2, this implies $\mathcal{S}_{\mathsf{pss}[i]} \succ \mathcal{S}_j$. However, we also have $j \in (\mathsf{pss}[i], i)$, which leads to the contradiction $\mathcal{S}_{\mathsf{pss}[i]} \prec \mathcal{S}_j$. ◀

Let $p_1 = i - 1, p_2, \ldots, p_k = 0$ be the elements of $\mathcal{P}_{i-1}$ in descending order, then it follows from Lemma 8 that there is some $m \in [1, k]$ with $\mathsf{pss}[i] = p_m$, i.e. node i has to become a child of p_m in the partial PSS tree. In terms of the BPS, we have to append $m - 1$ closing parentheses to the BPS prefix. Then, we can simply write the opening parenthesis of node i. Once again, an example is provided in Figure 2b.

Algorithm 1 BuildPssBps.

```
Input: String $\mathcal{S}$ of length $n$
Output: BPS of the PSS Tree of $\mathcal{S}$
1: $\mathcal{B}_{\mathsf{pss}} \leftarrow ($                                        ▷ Open node 0
2: for $i = 1$ to $n$ do
3:     Let $\mathcal{P}_{i-1} = \{p_1, \dots, p_k\}$ with $\mathsf{pss}[p_x] = p_{x+1}$
4:     Determine $p_m = \mathsf{pss}[i]$
5:     Append $m - 1$ closing parentheses to $\mathcal{B}_{\mathsf{pss}}$      ▷ Close nodes $p_1, \dots, p_{m-1}$
6:     Append one opening parenthesis to $\mathcal{B}_{\mathsf{pss}}$          ▷ Open node $i$
7: Append $|\mathcal{P}_n|$ closing parentheses to $\mathcal{B}_{\mathsf{pss}}$          ▷ Close rightmost path
```

Our construction algorithm for $\mathcal{B}_{\mathsf{pss}}$ is based on this simple idea, as outlined by Algorithm 1. Initially, the BPS only contains the opening parenthesis of the root 0 (line 1). Then, whenever we process an index i, we use $\mathcal{P}_{i-1}$ to determine p_m (lines 3–4) and extend $\mathcal{B}_{\mathsf{pss}}$ by appending $m-1$ closing parentheses and one opening one (lines 5–6). Finally, once all nodes have been added to the PSS tree, we only have to close all remaining unclosed parentheses (line 7). The algorithm has two black boxes: How do we determine $\mathcal{P}_{i-1}$ (line 3), and how do we use it to find p_m (line 4)? The first question is easily answered, since the operations that we support on the BPS prefix at all times (see Section 3.1) are already sufficient to access each element of $\mathcal{P}_{i-1}$ in constant time. Let $p_1, \dots, p_k$ be exactly these elements in descending order. As explained earlier, they directly correspond to the unclosed parentheses of the BPS prefix, such that p_k corresponds to the leftmost unclosed parenthesis, and p_1 to the rightmost one. Therefore, we have $p_x = \text{rank}_{\text{open}}(\text{select}_{\text{uncl}}(k - x + 1)) - 1$. It remains to be shown how to efficiently find p_m.

4.1 Efficiently Computing p_m

Consider the following naive approach for computing p_m: Iterate over the indices $p_1, \dots, p_k$ in descending order (i.e. p_1 first, p_k last). For each index p_x, evaluate whether $\mathcal{S}_{p_x} \prec \mathcal{S}_i$ holds. As soon as this is the case, we have found p_m. The cost of this approach is high: A naive suffix comparison between $\mathcal{S}_{p_x}$ and $\mathcal{S}_i$ takes $\textsc{lce}(p_x, i) + 1$ individual character comparisons, which means that we spend $\mathcal{O}(m + \sum_{x=1}^{m} \textsc{lce}(p_x, i))$ time to determine m. However, the following property will allow us to decrease this time bound significantly:

▶ **Corollary 9** (Bitonic LCE Values). *Let $p_1, \dots, p_k$ be exactly the elements of $\mathcal{P}_{i-1}$ in descending order and let $p_m = \mathsf{pss}[i]$. Furthermore, let $\ell_x = \textsc{lce}(p_x, i)$ for all $x \in [1, k]$. We have $\ell_1 \leq \ell_2 \leq \cdots \leq \ell_{m-1}$ as well as $\ell_m \geq \ell_{m+1} \geq \cdots \geq \ell_k$.*

Proof. Follows from $\mathcal{S}_{p_1} \succ \ldots \succ \mathcal{S}_{p_{m-1}} \succ \mathcal{S}_i \succ \mathcal{S}_{p_m} \succ \ldots \succ \mathcal{S}_{p_k}$ and simple properties of the lexicographical order. ◀

From now on, we continue using the notation $\ell_x = \textsc{lce}(p_x, i)$ from the corollary. Note that the longest LCE between i and any of the p_x occurs either with p_m or with p_{m-1}. Let $\ell_{\max} = \max(\ell_{m-1}, \ell_m)$ be this largest LCE value, then our more sophisticated approach for determining m only takes $\mathcal{O}(m + \ell_{\max})$ time. It consists of two steps: First, we determine a candidate interval $(u, w] \subseteq [1, k]$ of size at most $\ell_{\max}$ that contains m. In the second step we gradually narrow down the borders of the candidate interval until the exact value of m is known.

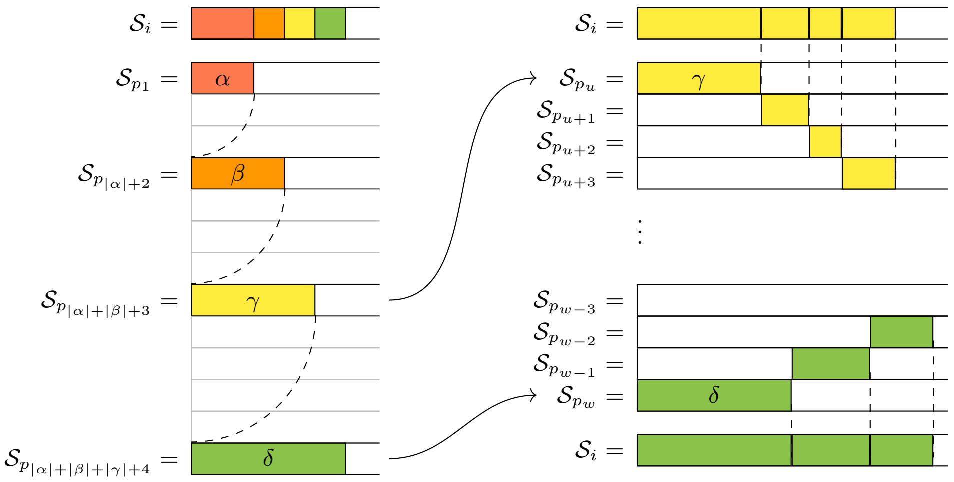

$\mathcal{S}_i =$
$\mathcal{S}_{p_1} =$
α
$\mathcal{S}_{p_{|\alpha|+2}} =$
β
$\mathcal{S}_{p_{|\alpha|+|\beta|+3}} =$
γ
$\mathcal{S}_{p_{|\alpha|+|\beta|+|\gamma|+4}} =$
δ
$\mathcal{S}_i =$
$\mathcal{S}_{p_u} =$
γ
$\mathcal{S}_{p_{u+1}} =$
$\mathcal{S}_{p_{u+2}} =$
$\mathcal{S}_{p_{u+3}} =$
$\mathcal{S}_{p_{w-3}} =$
$\mathcal{S}_{p_{w-2}} =$
$\mathcal{S}_{p_{w-1}} =$
$\mathcal{S}_{p_w} =$
δ
$\mathcal{S}_i =$

Figure 3 Matching character comparisons when determining p_m. On the left we have the suffix $\mathcal{S}_i$ as well as $\mathcal{S}_{p_1}, \mathcal{S}_{p_2}, \ldots, \mathcal{S}_{p_w}$, which are relevant for the first step. Each prefix $\alpha, \beta, \gamma, \delta$ highlights the LCP between the respective suffix $\mathcal{S}_{p_x}$ and $\mathcal{S}_i$. On the right side we have the suffixes $\mathcal{S}_{p_u}, \mathcal{S}_{p_{u+1}}, \ldots, \mathcal{S}_{p_w}$, which are relevant for the second step. (Best viewed in color.)

Step 1: Find a candidate interval. Our goal is to find $(u, w] = (u, u + \ell_u + 1]$ with $m \in (u, w]$. Initially, we naively compute $\ell_1 = \text{LCE}(p_1, i)$, allowing us to evaluate $\mathcal{S}_{p_1} \prec \mathcal{S}_i$ in constant time. If this holds, then we have $m = 1$ and no further steps are necessary. Otherwise, let $u \leftarrow 1$ and (i) let $w \leftarrow u + \ell_u + 1$. We already know that $u < m$ holds. Now we have to evaluate if $m \leq w$ also holds. Therefore, we compute $\ell_w = \text{LCE}(p_w, i)$ naively, which allows us to check in constant time if $\mathcal{S}_{p_w} \prec \mathcal{S}_i$ and decide if $m \leq w$ holds. If this is not the case, then we assign $u \leftarrow w$ as well as $\ell_u \leftarrow \ell_w$ and continue at (i). If however $\mathcal{S}_{p_w} \prec \mathcal{S}_i$ holds, then we have $m \leq w$ and therefore $m \in (u, w]$. Figure 3 (left) outlines the procedure.

Step 2: Narrow down $(u, w]$ to the exact value of m. Now we gradually tighten the borders of the candidate interval. If ℓ_u is smaller than ℓ_w, then we try to increase u by one. Otherwise, we try to decrease w by one.

Assume that we have $\ell_u < \ell_w$, then it follows from Corollary 9 that $\ell_{u+1} \geq \ell_u$ holds. Therefore, when computing ℓ_{u+1} we can simply skip the first ℓ_u character comparisons. Now we use ℓ_{u+1} to evaluate in constant time if $\mathcal{S}_{p_{u+1}} \succ \mathcal{S}_i$ holds. If that is the case, then we have $u + 1 < m$ and thus we can assign $u \leftarrow u + 1$ and start Step 2 from the beginning. If however $\mathcal{S}_{p_{u+1}} \prec \mathcal{S}_i$ holds, then we have $m = u + 1$ and no further steps are necessary. In case of $\ell_u \geq \ell_w$ we proceed analogously. Once again, Figure 3 (right) visualizes the procedure.

Time Complexity. Step 1 is dominated by computing LCE values. Determining the final LCE value ℓ_w takes $\ell_w + 1$ individual character comparisons and thus $\Theta(\ell_w + 1)$ time. Whenever we compute any previous value of ℓ_w, we increase w by $\ell_w + 1$ afterwards. Therefore, the time for computing all LCE values is bound by $\Theta(w + \ell_w) = \Theta(u + \ell_u + \ell_w) \subseteq \mathcal{O}(m + \ell_{\max})$. Since initially $(u, w]$ has size at most $\ell_{\max}$, we call Step 2 at most $\mathcal{O}(\ell_{\max})$ times. With every call we increase ℓ_u or ℓ_w by exactly the number of matching character comparisons that we perform. Therefore, the total number of matching character comparisons is bound by $2\ell_{\max}$.

Thus, the total time needed for Step 2 is bound by $\mathcal{O}(\ell_{\max})$. In sum, processing index i takes $\mathcal{O}(m + \ell_{\max})$ time. For the total processing time of all indices (and thus the execution time of Algorithm 1) we get:

$$\begin{aligned} &\sum_{i=1}^{n} \mathcal{O}(\overbrace{|\mathcal{P}_{i-1} \cap [\mathsf{pss}[i], i]|}^{m}) + \sum_{i=1}^{n} \mathcal{O}(\overbrace{\max_{p_x \in \mathcal{P}_{i-1}} \mathrm{LCE}(p_x, i)}^{\ell_{\max}}) \\ = \quad &\mathcal{O}(n) \qquad\qquad\qquad + \qquad \mathcal{O}(n^2) \end{aligned}$$

(The $\mathcal{O}(m)$-terms sum to $\mathcal{O}(n)$ since $m - 1$ is exactly the number of closing parentheses that we write while processing i, and there are exactly $n+1$ closing parentheses in the entire BPS.) As it appears, the total time bound of the algorithm is still far from linear time. However, it is easy to identify the crucial time component that makes the algorithm too expensive. From now on we call the $\mathcal{O}(m)$ term of the processing time *negligible*, while the $\mathcal{O}(\ell_{\max})$ term is called *critical*.

Clearly, if we could somehow remove the critical terms, we would already achieve linear time. There exists a variety of data structures that could help us to achieve this goal by accelerating suffix comparisons, e.g. the (compressed or sparse) suffix tree, the (compressed) suffix array, or dedicated data structures for fast LCE queries. However, all of theses data structures either require more than $\mathcal{O}(n)$ bits of construction space, or more than $\mathcal{O}(n)$ construction time, or they are non-deterministic, or their efficiency depends on the alphabet or the compressability of the text. For example, there exists a linear time construction algorithm for the sparse suffix tree [15], but it is non-deterministic. This motivates the techniques that we describe in the following sections, which directly remove the critical terms without relying on any of the aforementioned data structures. This way, the execution time of Algorithm 1 decreases to $\mathcal{O}(n)$, while the additional working space remains unchanged.

5 Achieving Linear Time

The critical time component for processing index i is $\ell_{\max} = \max_{p_x \in \mathcal{P}_{i-1}} \mathrm{LCE}(p_x, i)$. When processing i with the technique from Section 4.1, we inherently find out the exact value of $\ell_{\max}$, and we also discover the index $p_{\max}$ for which we have $\mathrm{LCE}(p_{\max}, i) = \ell_{\max}$. From now on, we simply use $\ell = \ell_{\max}$ and $j = p_{\max}$. While discovering a large LCE value ℓ is costly, it yields valuable structural information about the input text: There is a repeating substring of length ℓ with occurrences $\mathcal{S}[j..j + \ell)$ and $\mathcal{S}[i..i + \ell)$. Intuitively, there is also a large repeating structure in the PSS tree, and consequently a repeating substring in $\mathcal{B}_{\mathsf{pss}}$. This motivates the techniques shown in this section, which conceptually alter Algorithm 1 as follows: Whenever we finish processing an index i with critical cost ℓ, we skip the next $\Omega(\ell)$ iterations of the loop by simply extending the BPS prefix with the copy of an already computed part, which means that the amortized critical cost per index becomes constant.

Depending on j and ℓ we choose either the *run extension* (Section 5.1) or the *amortized look-ahead* (Section 5.2) to perform the extension. Algorithm 2 outlines our final construction algorithm on a higher level, and complements the written description by showing when the special cases arise. Before going into detail, we point out that $\mathcal{S}[j..i)$ is a Lyndon word. As mentioned earlier, it follows from Corollary 9 that j equals p_m or p_{m-1}. Since i is the first node that is not a descendant of p_{m-1}, we have $\mathsf{nss}[p_{m-1}] = i$. Therefore, if $j = p_{m-1}$ holds, we have $\mathsf{nss}[j] = i$, which by definition implies that $\mathcal{S}[j..i)$ is a Lyndon word. If however $j = p_m = \mathsf{pss}[i]$ holds, then $\mathcal{S}[j..i)$ is a Lyndon word because of Lemma 4.

Algorithm 2 BuildPssBpsLinear.

```
Input: String $\mathcal{S}$ of length $n$
Output: BPS of the PSS Tree of $\mathcal{S}$
1: $\mathcal{B}_{\mathsf{pss}} \leftarrow$ (
2: for $i = 1$ to $n$ do
3:     Let $\mathcal{P}_{i-1} = \{p_1, \ldots, p_k\}$ with $\mathsf{pss}[p_x] = p_{x+1}$
4:     Determine $p_m = \mathsf{pss}[i]$,
           using the technique from Section 4.1, causing critical cost $\ell$ and
           discovering the index $j$ with $\textsc{lce}(j, i) = \ell$ as described in the
           beginning of Section 5.
5:     Append $m - 1$ closing parentheses to $\mathcal{B}_{\mathsf{pss}}$
6:     Append one opening parenthesis to $\mathcal{B}_{\mathsf{pss}}$
       (For any $x$, let $o_x$ be the opening parenthesis of node $x$.)
7:     if $\ell \geq 2(i - j)$ then
8:         Apply the run extension as described in Section 5.1.
               Let $t = \lfloor \ell/(i - j) \rfloor + 1$. Take $\mathcal{B}_{\mathsf{pss}}(o_j..o_i]$ and append it $(t - 2)$
               times to $\mathcal{B}_{\mathsf{pss}}$. Continue in line 2 with $i \leftarrow i + (t - 2) \cdot (i - j)$.
9:     else
10:        Apply the amortized look-ahead as described in Section 5.2.
               Using Lemma 13, find the largest value $\chi \in [1, \lfloor \ell/4 \rfloor]$ that sat-
               isfies $\mathcal{B}_{\mathsf{pss}}[o_j..o_{j+\chi-1}] = \mathcal{B}_{\mathsf{pss}}[o_i..o_{i+\chi-1}]$, and append a copy of
               $\mathcal{B}_{\mathsf{pss}}(o_j..o_{j+\chi-1}]$ to $\mathcal{B}_{\mathsf{pss}}$. Continue in line 2 with $i \leftarrow i + \chi$. If
               $\chi < \lfloor \ell/4 \rfloor$, then iteration $i + \chi$ will automatically skip additional
               $\Omega(\ell)$ iterations by using the run extension.
11: Append $|\mathcal{P}_n|$ closing parentheses to $\mathcal{B}_{\mathsf{pss}}$
```

5.1 Run Extension

We apply the run extension iff we have $\ell \geq 2(i - j)$. It is easy to see that in this case $\mathcal{S}[j..j+\ell)$ and $\mathcal{S}[i..i+\ell)$ overlap such that the Lyndon word $\mu = \mathcal{S}[j..i)$ repeats itself at least three times, starting at index j. We call the substring $\mathcal{S}[j..i+\ell)$ *Lyndon run with period* $|\mu|$. The number of *repetitions* is $t = \lfloor \ell/|\mu| \rfloor + 1 \geq 3$, and the starting positions of the repetitions are $r_1, \ldots, r_t$ with $r_1 = j$, $r_2 = i$, and generally $r_x = r_{x-1} + |\mu|$. In a moment we will show that in this particular situation the following lemma holds:

▶ **Lemma 10.** *Let o_x be the index of the opening parenthesis of node x in $\mathcal{B}_{\mathsf{pss}}$. Then we have $\mathcal{B}_{\mathsf{pss}}[o_{r_1}..o_{r_2}] = \mathcal{B}_{\mathsf{pss}}[o_{r_2}..o_{r_3}] = \cdots = \mathcal{B}_{\mathsf{pss}}[o_{r_{t-1}}..o_{r_t}]$.*

Expressed less formally, each repetition of μ – except for the last one – induces the same substring in the BPS. Performing the run extension is as easy as taking the already written substring $\mathcal{B}_{\mathsf{pss}}(o_{r_1}..o_{r_2}] = \mathcal{B}_{\mathsf{pss}}(o_j..o_i]$, and appending it $t - 2$ times to $\mathcal{B}_{\mathsf{pss}}$. Afterwards, the last parenthesis that we have written is the opening parenthesis of node r_t, and we continue the execution of Algorithm 1 with iteration $r_t + 1$. Thus, we have skipped the processing of $r_t - i$ indices. Since

$$r_t - i \;=\; (t-2)\cdot|\mu| \;\geq\; \frac{(t-2)\cdot|\mu|}{t\cdot|\mu|}\cdot\ell \;\geq\; \frac{1}{3}\cdot\ell \;=\; \Omega(\ell)\ ,$$

it follows that the average critical cost per index from $[i, r_t]$ is constant.

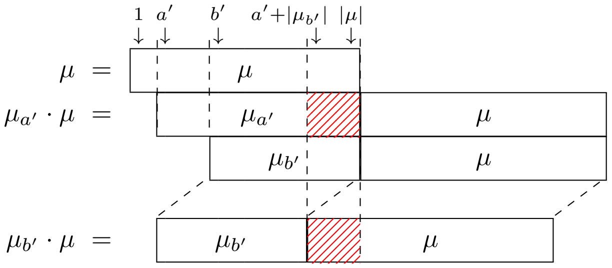
Proving the Lemma. It remains to be shown that Lemma 10 holds. It is sufficient to prove the correctness for $t = 3$, since the correctness for the general case follows by repeatedly applying the lemma with $t = 3$. Therefore, we only have to show $\mathcal{B}_{\mathsf{pss}}[o_{r_1}..o_{r_2}] = \mathcal{B}_{\mathsf{pss}}[o_{r_2}..o_{r_3}]$.

Isomorphic Subtrees. Since μ is a Lyndon word, it is easy to see that the suffixes at the starting positions of repetitions are lexicographically smaller than the suffixes that begin in between the starting positions of repetitions, i.e. we have $\forall x \in (r_1, r_2) : \mathcal{S}_{r_1} \prec \mathcal{S}_x$ and $\forall x \in (r_2, r_3) : \mathcal{S}_{r_2} \prec \mathcal{S}_x$. Consequently, the indices from (r_1, r_2) are descendants of r_1 in the PSS tree, and the indices from (r_2, r_3) are descendants of r_2 in the PSS tree, i.e. each of the intervals $[r_1, r_2)$ and $[r_2, r_3)$ induces a tree.

Next, we show that these trees are actually isomorphic. Clearly, the tree induced by $[r_1, r_2)$ solely depends on the lexicographical order of suffixes that begin within $[r_1, r_2)$, and the tree induced by $[r_2, r_3)$ solely depends on the lexicographical order of suffixes that begin within $[r_2, r_3)$. Assume that the trees are *not* isomorphic, then there must be a suffix comparison that yields different results in each interval, i.e. there must be offsets $a, b \in [0, |\mu|)$ with $a \neq b$ such that $\mathcal{S}_{r_1+a} \prec \mathcal{S}_{r_1+b} \iff \mathcal{S}_{r_2+a} \succ \mathcal{S}_{r_2+b}$ holds. However, this is impossible, as shown by the lemma below.

▶ **Lemma 11.** *For all $a, b \in [0, |\mu|)$ with $a \neq b$ we have $\mathcal{S}_{r_1+a} \prec \mathcal{S}_{r_1+b} \iff \mathcal{S}_{r_2+a} \prec \mathcal{S}_{r_2+b}$.*

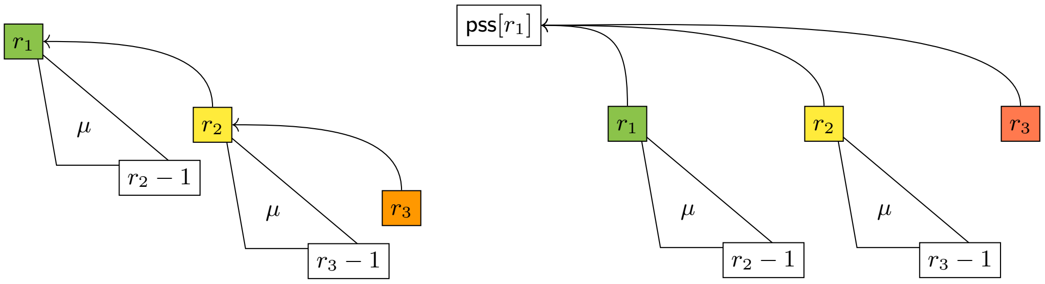
Proof. Assume w.l.o.g. $a < b$, and let $a' = a + 1$ and $b' = b + 1$. We can show that the strings $\mu_{a'} \cdot \mu$ and $\mu_{b'} \cdot \mu$ have a mismatch:



Consider the two hatched areas in the drawing above. The top area highlights the suffix $\mu_{a'+|\mu_{b'}|}$ of μ, which has length $c = |\mu| - (a' + |\mu_{b'}|) + 1$. The bottom area highlights the prefix $\mu[1..c]$ of μ. Since μ is a Lyndon word, there is no proper non-empty suffix of μ that is also a prefix of μ. It follows that the hatched areas cannot be equal, i.e. $\mu_{a'+|\mu_{b'}|} \neq \mu[1..c]$. This guarantees a mismatch between $\mu_{a'} \cdot \mu$ and $\mu_{b'} \cdot \mu$. Therefore, appending an arbitrary string to $\mu_{a'} \cdot \mu$ and $\mu_{b'} \cdot \mu$ does not influence the outcome of a lexicographical comparison. The statement of the lemma directly follows by appending $\mathcal{S}_{r_3}$ and $\mathcal{S}_{r_4}$ respectively:

$$\mu_{a'}\cdot\mu \prec \mu_{b'}\cdot\mu \iff \underbrace{\mu_{a'}\cdot\mu\cdot\mathcal{S}_{r_3}}_{=\ \mathcal{S}_{r_1+a}} \prec \underbrace{\mu_{b'}\cdot\mu\cdot\mathcal{S}_{r_3}}_{=\ \mathcal{S}_{r_1+b}} \iff \underbrace{\mu_{a'}\cdot\mu\cdot\mathcal{S}_{r_4}}_{=\ \mathcal{S}_{r_2+a}} \prec \underbrace{\mu_{b'}\cdot\mu\cdot\mathcal{S}_{r_4}}_{=\ \mathcal{S}_{r_2+b}}$$

◀



(a) Increasing run. **(b)** Decreasing run.

Figure 4 The run of the Lyndon word $\mu = \mathcal{S}[r_1, r_2) = \mathcal{S}[r_2, r_3) = \mathcal{S}[r_3, r_3{+}|\mu|)$ induces isomorphic subtrees in the PSS tree. If $\mathcal{S}_{r_1} \prec \mathcal{S}_{r_2}$, then the roots of the subtrees form a chain **(a)**. Otherwise, they are siblings **(b)**.

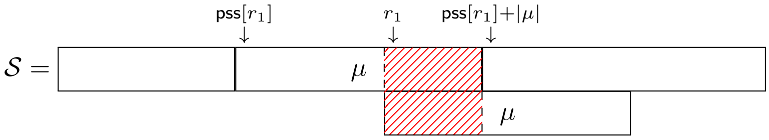
Finally, we show that in the PSS tree the induced isomorphic trees are connected in a way that ultimately implies $\mathcal{B}_{\mathsf{pss}}[o_{r_1}..o_{r_2}] = \mathcal{B}_{\mathsf{pss}}[o_{r_2}..o_{r_3}]$. There are two possible scenarios for this connection, which depend on the so called *direction* of the Lyndon run. We call a run *increasing* iff $\mathcal{S}_{r_1} \prec \mathcal{S}_{r_2}$ holds, and *decreasing* otherwise.

Increasing Runs. First, we focus on increasing runs. It follows from $\mathcal{S}_{r_1} \prec \mathcal{S}_{r_2} \iff \mu \cdot \mathcal{S}_{r_2} \prec \mu \cdot \mathcal{S}_{r_3} \iff \mathcal{S}_{r_2} \prec \mathcal{S}_{r_3}$ that $\mathcal{S}_{r_1} \prec \mathcal{S}_{r_2} \prec \mathcal{S}_{r_3}$. Since μ is a Lyndon word, we have $\forall x \in (r_1, r_2) : \mathcal{S}_{r_2} \prec \mathcal{S}_x$ as well as $\forall x \in (r_2, r_3) : \mathcal{S}_{r_3} \prec \mathcal{S}_x$. Therefore, we have $\mathsf{pss}[r_2] = r_1$ and $\mathsf{pss}[r_3] = r_2$, and the isomorphic subtrees are connected as visualized in Figure 4a. It is easy to see that a preorder-traversal from r_1 to r_2 yields the same sequence of parentheses as a preorder-traversal from r_2 to r_3. Therefore we have $\mathcal{B}_{\mathsf{pss}}[o_{r_1}..o_{r_2}] = \mathcal{B}_{\mathsf{pss}}[o_{r_2}..o_{r_3}]$, which means that Lemma 10 holds for increasing runs.

Decreasing Runs. With the same argument as for increasing runs, we have $\mathcal{S}_{r_1} \succ \mathcal{S}_{r_2} \succ \mathcal{S}_{r_3}$ in decreasing runs. We also have $\forall x \in (r_1, r_2) : \mathcal{S}_{r_2} \prec \mathcal{S}_x$ as well as $\forall x \in (r_2, r_3) : \mathcal{S}_{r_3} \prec \mathcal{S}_x$, which means that $\mathsf{pss}[r_2] \leq \mathsf{pss}[r_1]$ and $\mathsf{pss}[r_3] \leq \mathsf{pss}[r_1]$ hold. In Lemma 12 we will show that in fact $\mathsf{pss}[r_1] = \mathsf{pss}[r_2] = \mathsf{pss}[r_3]$ holds, such that the isomorphic subtrees are connected as visualized in Figure 4b. A preorder-traversal from r_1 to r_2 yields the same sequence of parentheses as a preorder-traversal from r_2 to r_3. Therefore we have $\mathcal{B}_{\mathsf{pss}}[o_{r_1}..o_{r_2}] = \mathcal{B}_{\mathsf{pss}}[o_{r_2}..o_{r_3}]$, which means that Lemma 10 holds for decreasing runs.

▶ **Lemma 12.** *In decreasing runs we have* $\mathsf{pss}[r_1] = \mathsf{pss}[r_2] = \mathsf{pss}[r_3]$.

Proof. As explained previously, we have $\mathsf{pss}[r_2] \leq \mathsf{pss}[r_1]$ and $\mathsf{pss}[r_3] \leq \mathsf{pss}[r_1]$, and thus only need to show $\mathcal{S}_{\mathsf{pss}[r_1]} \prec \mathcal{S}_{r_2}$ and $\mathcal{S}_{\mathsf{pss}[r_1]} \prec \mathcal{S}_{r_3}$. We will show below that μ cannot be a prefix of $\mathcal{S}_{\mathsf{pss}[r_1]}$, from which the statement of the lemma can be deduced easily since the suffixes $\mathcal{S}_{r_2}$ and $\mathcal{S}_{r_3}$ begin with the prefix μ. Assume for the sake of contradiction that μ is a prefix of $\mathcal{S}_{\mathsf{pss}[r_1]}$. If we also assume $\mathsf{pss}[r_1] + |\mu| > r_1$, we get:



As indicated by the hatched area, this implies that there is a proper non-empty suffix of μ that is also a prefix of μ, which is impossible because μ is a Lyndon word. Thus we have $\mathsf{pss}[r_1] + |\mu| \not> r_1$. Also, we cannot have $\mathsf{pss}[r_1] + |\mu| = r_1$, because then $\mathsf{pss}[r_1]$ would be the starting position of another repetition of μ, which would imply $\mathcal{S}_{\mathsf{pss}[r_1]} \succ \mathcal{S}_{r_1}$. It follows $\mathsf{pss}[r_1] + |\mu| < r_1$, i.e. $\mathsf{pss}[r_1] + |\mu| \in (\mathsf{pss}[r_1], r_1)$ and thus $\mathcal{S}_{\mathsf{pss}[r_1]+|\mu|} \succ \mathcal{S}_{r_1}$. However, this leads to a contradiction:

$$\begin{aligned} \mathcal{S}_{\mathsf{pss}[r_1]} \prec \mathcal{S}_{r_1} &\iff \mu \cdot \mathcal{S}_{\mathsf{pss}[r_1]+|\mu|} \prec \mu \cdot \mathcal{S}_{r_2} \\ &\iff \mathcal{S}_{\mathsf{pss}[r_1]+|\mu|} \prec \mathcal{S}_{r_2} \\ &\underset{\mathcal{S}_{r_1} \succ \mathcal{S}_{r_2}}{\implies} \mathcal{S}_{\mathsf{pss}[r_1]+|\mu|} \prec \mathcal{S}_{r_1} \end{aligned}$$ ◀

5.2 Amortized Look-Ahead

Finally, we show how to amortize the critical cost $\mathcal{O}(\ell)$ of processing index i if the run extension is not applicable, i.e. if we have $\ell < 2(i - j)$. Unfortunately, the trees induced by the nodes from $[j, j + \ell)$ and $[i, i + \ell)$ are not necessarily isomorphic. However, we can still identify a sufficiently large isomorphic structure. In a moment we will show that the following lemma holds:

▶ **Lemma 13.** *Let o_x be the index of the opening parenthesis of node x in $\mathcal{B}_{\mathsf{pss}}$. We either have $\mathcal{B}_{\mathsf{pss}}[o_j..o_{j+\lfloor \ell/4 \rfloor -1}] = \mathcal{B}_{\mathsf{pss}}[o_i..o_{i+\lfloor \ell/4 \rfloor -1}]$, or there is an integer $\chi < \lfloor \ell/4 \rfloor$ with $\mathcal{B}_{\mathsf{pss}}[o_j..o_{j+\chi-1}] = \mathcal{B}_{\mathsf{pss}}[o_i..o_{i+\chi-1}]$ and an index $h \in [i, i+\chi)$ such that $\mathcal{S}[h..i+\ell)$ is a Lyndon run of the Lyndon word $\mathcal{S}[h..i+\chi)$. We can determine which case applies, and also determine the value of χ (if applicable) in $\mathcal{O}(\ell)$ time and $\mathcal{O}(1)$ words of additional space.*

When performing the amortized look-ahead we first determine which case of the lemma applies. Then, if $\mathcal{B}_{\mathsf{pss}}[o_j..o_{j+\lfloor \ell/4 \rfloor -1}] = \mathcal{B}_{\mathsf{pss}}[o_i..o_{i+\lfloor \ell/4 \rfloor -1}]$, we extend the known prefix of the BPS by appending a copy of $\mathcal{B}_{\mathsf{pss}}(o_j..o_{j+\lfloor \ell/4 \rfloor -1}]$, and continue the execution of Algorithm 1 with iteration $i + \lfloor \ell/4 \rfloor$. Since this way we skip the processing of $\lfloor \ell/4 \rfloor - 1 = \Omega(\ell)$ indices, the average critical cost per index from $[i, i + \lfloor \ell/4 \rfloor)$ is constant. If, however, the second case applies, then we determine the value of χ and extend the known prefix of the BPS by appending a copy of $\mathcal{B}_{\mathsf{pss}}(o_j..o_{j+\chi-1}]$, allowing us to continue the execution of Algorithm 1 with iteration $i+\chi$. We know that there is some $h \in [i, i+\chi)$ such that $\mathcal{S}[h..i+\ell)$ is a Lyndon run of the Lyndon word $\mu = \mathcal{S}[h..i+\chi)$. This run might even be longer: Let $\ell' = \textsc{lce}(h, i+\chi)$ (computed naively), then $\mathcal{S}[h..i + \chi + \ell')$ is the longest run of μ that starts at index h. If the run is increasing, then $\mathsf{pss}[i + \chi] = h$ holds (see Section 5.1), and the longest LCE that we discover when processing index $i + \chi$ is ℓ'. If the run is decreasing, then $\mathsf{pss}[i + \chi] = \mathsf{pss}[h]$ holds. Also in this case, the longest LCE that we discover when processing index $i + \chi$ is ℓ', since $\textsc{lce}(\mathsf{pss}[i + \chi], i + \chi)$ is less than $|\mu|$ (see proof of Lemma 12). Therefore, the critical cost of processing index $i + \chi$ will be $\mathcal{O}(\ell')$. However, since the Lyndon run has at least three repetitions, we will also skip the processing of $\Omega(\ell')$ indices by using the run extension. The algorithmic procedure for the second case can be summarized as follows: We process index i with critical cost $\mathcal{O}(\ell)$ and skip $\chi - 1$ indices afterwards. Then we process index $i + \chi$ with critical cost $\mathcal{O}(\ell')$ and skip another $\Omega(\ell')$ indices by using the run extension. Since we have $\ell' = \Omega(\ell)$, the total critical cost is $\mathcal{O}(\ell')$, and the total number of processed or skipped indices is $\Omega(\ell')$. Thus, the average critical cost per index is constant.

Proving Lemma 13. It remains to be shown that Lemma 13 holds. For this purpose, assume $\mathcal{B}_{\mathsf{pss}}[o_j..o_{j+\lfloor \ell/4 \rfloor -1}] \neq \mathcal{B}_{\mathsf{pss}}[o_i..o_{i+\lfloor \ell/4 \rfloor -1}]$. From now on we refer to $\mathcal{B}_{\mathsf{pss}}[o_j..o_{j+\lfloor \ell/4 \rfloor -1}]$ and $\mathcal{B}_{\mathsf{pss}}[o_i..o_{i+\lfloor \ell/4 \rfloor -1}]$ as *left* and *right side*, respectively. Consider the first mismatch between the two, where w.l.o.g. we assume that the mismatch has an opening parenthesis on the left

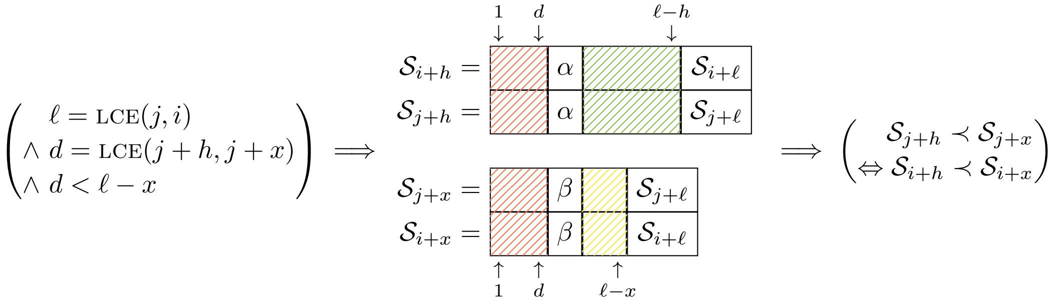


Figure 5 Proving Lemma 13. Equal colors indicate equal substrings. (Best viewed in color.)

side, and a closing one on the right side. On the left side, the opening parenthesis corresponds to a node $j+x$ with $x \in [1, \lfloor \ell/4 \rfloor)$ that is a child of another node $j+h$. Since $\mathcal{S}[j..j+\ell)$ is a Lyndon word, all nodes from $(j, j+\ell)$ are descendants of j. Consequently, we have $h \in [0,x)$. Now we look at the right side: Since we have a closing parenthesis instead of an opening one, we know that $i+x$ is not attached to $i+h$, but to a smaller node, i.e. we have $\mathsf{pss}[i+x] < i+h$. It follows that $\mathcal{S}_{j+h} \prec \mathcal{S}_{j+x}$ and $\mathcal{S}_{i+h} \succ \mathcal{S}_{i+x}$ hold. Let $d = \text{LCE}(j+h, j+x)$ and assume $d \leq \ell - x$. Then due to $\mathcal{S}_{j+h} \prec \mathcal{S}_{j+x}$ we have $\mathcal{S}[j+h+d] < \mathcal{S}[j+x+d]$. However, since we have $\mathcal{S}[j..j+\ell] = \mathcal{S}[i..i+\ell]$, it follows $\text{LCE}(i+h, i+x) = d$ and $\mathcal{S}[i+h+d] < \mathcal{S}[i+x+d]$, which contradicts $\mathcal{S}_{i+h} \succ \mathcal{S}_{i+x}$ (see Figure 5). Thus, it holds $d = \text{LCE}(j+h, j+x) \geq \ell - x$, allowing us to show that $\mathcal{S}[j+h..j+\ell)$ is a Lyndon run with period $x-h$. Since $\mathsf{pss}[j+x] = j+h$ holds, it follows from Lemma 4 that $\mathcal{S}[j+h..j+x)$ is a Lyndon word. Due to $\text{LCE}(j+h, j+x) > \ell - x \geq 3(\ell/4) \geq 3(x-h)$ we know that the Lyndon word repeats at least four times, and the run extends all the way to the end of $\mathcal{S}[j..j+\ell)$. Note that since the opening parenthesis of node $j+x$ causes the first mismatch between $\mathcal{B}_{\mathsf{pss}}[o_j..o_{j+\lfloor \ell/4 \rfloor -1}]$ and $\mathcal{B}_{\mathsf{pss}}[o_i..o_{i+\lfloor \ell/4 \rfloor -1}]$, we have $\mathcal{B}_{\mathsf{pss}}[o_j..o_{j+x-1}] = \mathcal{B}_{\mathsf{pss}}[o_i..o_{i+x-1}]$. Therefore, $\chi \leftarrow x$ already satisfies Lemma 13.

Finally, we explain how to determine $\chi = x$ in $\mathcal{O}(\ell)$ time. As described above, if $\mathcal{B}_{\mathsf{pss}}[o_j..o_{j+\lfloor \ell/4 \rfloor -1}] \neq \mathcal{B}_{\mathsf{pss}}[o_i..o_{i+\lfloor \ell/4 \rfloor -1}]$, then there is some offset $h < \lfloor \ell/4 \rfloor$ such that $\mathcal{S}[j+h..j+\ell)$ is a Lyndon run of at least four repetitions of a Lyndon word μ. Consequently, $\mathcal{S}[j+\lfloor \ell/4 \rfloor..j+\ell)$ has the form $\text{suf}(\mu) \cdot \mu^t \cdot \text{pre}(\mu)$ with $t \geq 2$, where $\text{suf}(\mu)$ and $\text{pre}(\mu)$ are a proper suffix and a proper prefix of μ. A string of this form is called *extended Lyndon run*. In Section 5.2.1 we propose an algorithm that checks whether or not $\mathcal{S}[j+\lfloor \ell/4 \rfloor..j+\ell)$ is an extended Lyndon run in $\mathcal{O}(\ell)$ time and constant additional space. If $\mathcal{S}[j+\lfloor \ell/4 \rfloor..j+\ell)$ is not an extended Lyndon run, then we have $\mathcal{B}_{\mathsf{pss}}[o_j..o_{j+\lfloor \ell/4 \rfloor -1}] = \mathcal{B}_{\mathsf{pss}}[o_i..o_{i+\lfloor \ell/4 \rfloor -1}]$ and no further steps are needed to satisfy Lemma 13. Otherwise, the algorithm from Section 5.2.1 also provides the period $|\mu|$ of the run, as well as $|\text{suf}(\mu)|$. In this case, we try to extend the extended Lyndon run to the left: We are now not only considering $\mathcal{S}[j+\lfloor \ell/4 \rfloor..j+\ell)$, but $\mathcal{S}[j..j+\ell)$. We want to find the leftmost index $j+h$ that is the starting position of a repetition of μ. Given $|\mu|$ and $|\text{suf}(\mu)|$, this can be done naively by scanning $\mathcal{S}[j..j+\lfloor \ell/4 \rfloor]$ from right to left, which takes $\mathcal{O}(\ell)$ time. If we have $h \geq \lfloor \ell/4 \rfloor - |\mu|$, then the first case of Lemma 13 applies and no further steps are necessary. Otherwise, we let $\chi \leftarrow h + |\mu|$. This concludes the proof of Lemma 13 and the description of our construction algorithm.

5.2.1 Detecting Extended Lyndon Runs

In this section, we propose a linear time algorithm that identifies extended Lyndon runs, i.e. strings of the form $\text{suf}(\mu) \cdot \mu^t \cdot \text{pre}(\mu)$ with $t \geq 2$, where $\text{suf}(\mu)$ and $\text{pre}(\mu)$ are a proper suffix and a proper prefix of μ. Our approach exploits properties of the Lyndon factorization, which is defined as follows:

▶ **Lemma 14** (Lyndon Factorization [5]). *Every non-empty string $\mathcal{S}$ can be decomposed into non-empty Lyndon words $s_1, s_2, \dots, s_m$ such that $\mathcal{S} = s_1 \cdot s_2 \cdot \ldots \cdot s_m$ and $\forall i \in [2, m] : s_{i-1} \succeq s_i$. There is exactly one such factorization for each string.*

▶ **Lemma 15.** *Let $\mathcal{S} = \mathrm{suf}(\mu) \cdot \mu^t \cdot \mathrm{pre}(\mu)$ be an extended Lyndon run. Let $x_1, \dots, x_{k_1}$ be the Lyndon factorization of $\mathrm{suf}(\mu)$, and let $y_1, \dots, y_{k_2}$ be the Lyndon factorization of $\mathrm{pre}(\mu)$. Then the Lyndon factorization of $\mathcal{S}$ is $x_1, \dots, x_{k_1}, \underbrace{\mu, \mu, \dots, \mu}_{t \text{ times}}, y_1, \dots, y_{k_2}$.*

Proof. Clearly, the factorization given by the lemma consists solely of Lyndon words. Thus, we only have to show $x_1 \succeq \ldots \succeq x_{k_1} \succeq \mu \succeq y_1 \succeq \ldots \succeq y_{k_2}$. Since we defined $x_1, \dots, x_{k_1}$ and $y_1, \dots, y_{k_2}$ to be the Lyndon factorizations of $\mathrm{suf}(\mu)$ and $\mathrm{pre}(\mu)$ respectively, we already know that $\forall i \in [2, k_1] : x_{i-1} \succeq x_i$ and $\forall i \in [2, k_2] : y_{i-1} \succeq y_i$ hold. It remains to be shown that $x_{k_1} \succeq \mu \succeq y_1$ holds. Since x_{k_1} is a non-empty suffix of $\mathrm{suf}(\mu)$ and thus also a non-empty proper suffix of μ, it follows that $x_{k_1} \succ \mu$ holds. Since y_1 is a prefix of $\mathrm{pre}(\mu)$ and thus also a prefix of μ, it follows (by definition of the lexicographical order) that $\mu \succ y_1$ holds. ◀

Lemma 15 implies that the longest factor of the Lyndon factorization of an extended Lyndon run is exactly the repeating Lyndon word μ. This is the key insight that we use to detect extended Lyndon runs:

▶ **Lemma 16.** *Let $\mathcal{S}$ be a string of length n. If $\mathcal{S}$ is an extended Lyndon run of the form $\mathcal{S} = \mathrm{suf}(\mu) \cdot \mu^t \cdot \mathrm{pre}(\mu)$, then we can determine $|\mu|$ and $|\mathrm{suf}(\mu)|$ in $\mathcal{O}(n)$ time and $\mathcal{O}(1)$ words of additional space.*

Proof. Using Duval's algorithm [8, Algorithm 2.1], we compute the Lyndon factorization of $\mathcal{S}$ in $\mathcal{O}(n)$ time and $\mathcal{O}(1)$ words of additional space. The algorithm computes and outputs the factors one-at-a time and in left-to-right order. Whenever it outputs a factor that is longer than all previous ones, we store its length l and its starting position d. Note that since we investigate each factor individually and then immediately discard it, we never need to store the entire factorization in memory. If $\mathcal{S}$ is an extended Lyndon run, then following Lemma 15 it must have the form $\mathcal{S} = \mathrm{suf}(\mu) \cdot \mu^t \cdot \mathrm{pre}(\mu)$ with $|\mathrm{suf}(\mu)| = d - 1$ and $|\mu| = l$. Since we know d and l, checking whether $\mathcal{S} = \mathrm{suf}(\mu) \cdot \mu^t \cdot \mathrm{pre}(\mu)$ holds can be achieved by performing a simple scan over $\mathcal{S}$. ◀

6 Algorithmic Summary & Adaptation to the Lyndon Array

We now summarize our construction algorithm for the PSS tree. We process the indices from left to right using the techniques from Section 4.1, where processing an index means attaching it to the PSS tree. Whenever the critical time of processing an index is $\mathcal{O}(\ell)$, we skip the next $\Omega(\ell)$ indices by using the run extension (Section 5.1) or the amortized look-ahead (Section 5.2). Thus, the critical time per index is constant, and the total worst-case execution time is $\mathcal{O}(n)$. In terms of working space, we only need $\mathcal{O}(n \lg\lg n / \lg n)$ bits to support the operations described in Section 3.1. The correctness of the algorithm follows from the description. We have shown:

▶ **Theorem 17.** *For a string $\mathcal{S}$ of length n we can compute its succinct Lyndon array $\mathcal{B}_{\mathsf{pss}}$ in $\mathcal{O}(n)$ time using $\mathcal{O}(n \lg\lg n / \lg n)$ bits of working space apart from the space needed for $\mathcal{S}$ and $\mathcal{B}_{\mathsf{pss}}$.*

The algorithm can easily be adapted to compute the Lyndon array instead of the PSS tree. For this purpose, we use a single array $\mathcal{A}$ (which later becomes the Lyndon array), and no further auxiliary data structures. We maintain the following invariant: At the time we start processing index i, we have $\mathcal{A}[j] = \mathsf{pss}[j]$ for $j \in \mathcal{P}_{i-1}$, and $\mathcal{A}[j] = \lambda[j]$ for $j \in [1, i) \setminus \mathcal{P}_{i-1}$. As before, we determine $p_m = \mathsf{pss}[i]$ with the techniques from Section 4.1. In Step 1 and Step 2 we require some access on elements of $\mathcal{P}_{i-1}$, which we can directly retrieve from $\mathcal{A}$. Apart from that, the algorithm remains unchanged. Once we computed p_m, we set $\mathcal{A}[i] \leftarrow p_m$ ($= \mathsf{pss}[i]$). Additionally, it follows that i is the first node that is not a descendant of any of the nodes $p_1, \ldots, p_{m-1}$, which means that we have $\mathsf{nss}[p_x] = i$ for any such node. Therefore, we assign $\mathcal{A}[p_x] \leftarrow i - p_x$ ($= \lambda[p_x]$). The run extension and the amortized look-ahead remain essentially unchanged, with the only difference being that we copy and append respective array intervals instead of BPS substrings (some trivial shifts on copied values are necessary). Once we have processed index n, we have $\mathcal{A}[j] = \mathsf{pss}[j]$ for $j \in \mathcal{P}_n$, and $\mathcal{A}[j] = \lambda[j]$ for $j \in [1, n] \setminus \mathcal{P}_n$. Clearly, all indices $p_x \in \mathcal{P}_n$ do not have a next smaller suffix, and we set $\mathcal{A}[p_x] \leftarrow n - p_x + 1 = \lambda[p_x]$. After this, we have $\mathcal{A} = \lambda$. Since at all times we only use $\mathcal{A}$ and no auxiliary data structures, the additional working space needed (apart from input and output) is constant. The linear execution time and correctness of the algorithm follow from the description. Thus we have shown:

▶ **Theorem 18.** *Given a string $\mathcal{S}$ of length n, we can compute its Lyndon array λ in $\mathcal{O}(n)$ time using $\mathcal{O}(1)$ words of working space apart from the space needed for $\mathcal{S}$ and λ.*

7 Experimental Results

We implemented our construction algorithm for both the succinct and the plain Lyndon array (LA-Succ and LA-Plain). The C++ implementation is publicly available at GitHub[2]. As a baseline we compared the throughput of our algorithms with the throughput of DivSufSort[3], which is known to be the fastest suffix array construction algorithm in practice [12]. Thus, it can be seen as a natural lower bound for any Lyndon array construction algorithm that depends on the suffix array. Additionally we consider LA-ISA-NSV, which builds the Lyndon array by computing next smaller values on the inverse suffix array (see [13], we use DivSufSort to construct the suffix array). For LA-Succ we only construct the succinct Lyndon array without the support data structure for fast queries. All experiments were conducted on the LiDO3 cluster[4], using an Intel Xeon E5-2640v4 processor and 64GiB of memory. We repeated each experiment five times and use the median as the final result. All texts are taken from the Pizza & Chili text corpus[5].

Table 1 shows the throughput of the different algorithms, i.e. the number of input bytes that can be processed per second. We are able to construct the plain Lyndon array at a speed of between 41 MiB/s (`fib41`) and 82 MiB/s (`xml`), which is on average 9.9 times faster than LA-ISA-NSV, and 8.1 times faster than DivSufSort. Even in the worst case, LA-Plain is still 6.8 times faster than LA-ISA-NSV, and 5.2 times faster than DivSufSort (`pitches`). When constructing the succinct Lyndon array we achieve around 86% of the throughput of LA-Plain on average, but never less than 81% (`pitches`). In terms of memory usage, we measured the additional working space needed apart from the space for the text and the

[2] `https://github.com/jonas-ellert/nearest-smaller-suffixes`
[3] `https://github.com/y-256/libdivsufsort`
[4] `https://www.lido.tu-dortmund.de/cms/de/LiDO3/index.html`
[5] `http://pizzachili.dcc.uchile.cl/`

Table 1 Throughput in MiB/s.

	(normal corpus)						(repetitive corpus)			
	`english.1GiB`	`dna`	`pitches`	`proteins`	`sources`	`xml`	`cere`	`einstein.de`	`fib41`	`kernel`
LA-Plain	60.57	50.83	60.58	62.18	66.13	82.10	53.08	59.09	41.71	62.27
LA-Succ	52.81	46.03	49.49	52.77	57.31	68.56	48.20	50.35	35.30	54.42
LA-ISA-NSV	4.61	4.86	9.13	4.40	7.41	7.11	5.44	6.72	3.81	6.79
DivSufSort	5.53	5.76	11.61	5.21	9.25	8.62	6.57	8.45	4.20	8.45

(succinct) Lyndon array. Both LA-Plain and LA-Succ never needed more than 0.002 bytes of additional memory per input character (or 770 KiB of additional memory in total), which is why we do not list the results in detail.

8 Summary

We showed how to construct the succinct Lyndon array in linear time using $\mathcal{O}(n \lg\lg n/\lg n)$ bits of working space. The construction algorithm can also produce the (non-succinct) Lyndon array in linear time using only $\mathcal{O}(1)$ words of working space. There are no other linear time algorithms achieving these bounds. Our algorithm performs also extremely well in practice. We envision applications of these practical algorithms in full-text indexing, such as an improved implementation of Baier's suffix array construction algorithm [1], or as a first step in sparse suffix sorting [11, 4].

References

1 Uwe Baier. Linear-time suffix sorting - A new approach for suffix array construction. In *Proceedings of the 27th Annual Symposium on Combinatorial Pattern Matching (CPM 2016)*, Tel Aviv, Israel, June 2016. `doi:10.4230/LIPIcs.CPM.2016.23`.

2 Hideo Bannai, Tomohiro I, Shunsuke Inenaga, Yuto Nakashima, Masayuki Takeda, and Kazuya Tsuruta. The "Runs" theorem. *SIAM Journal on Computing*, 46(5):1501–1514, 2017. `doi:10.1137/15M1011032`.

3 Jérémy Barbay, Johannes Fischer, and Gonzalo Navarro. LRM-trees: Compressed indices, adaptive sorting, and compressed permutations. In *Proceedings of the 22nd Annual Symposium on Combinatorial Pattern Matching (CPM 2011)*, pages 285–298, Palermo, Italy, June 2011. `doi:10.1007/978-3-642-21458-5_25`.

4 Philip Bille, Johannes Fischer, Inge Li Gørtz, Tsvi Kopelowitz, Benjamin Sach, and Hjalte Wedel Vildhøj. Sparse text indexing in small space. *ACM Transactions on Algorithms*, 12(3):Article No. 39, 2016. `doi:10.1145/2836166`.

5 K. T. Chen, R. H. Fox, and R. C. Lyndon. Free differential calculus, IV. the quotient groups of the lower central series. *Annals of Mathematics*, 68(1):81–95, 1958. `doi:10.2307/1970044`.

6 Maxime Crochemore and Luís M. S. Russo. Cartesian and lyndon trees. *Theor. Comput. Sci.*, 806:1–9, 2020.

7 Jacqueline W. Daykin, Frantisek Franek, Jan Holub, A. S. M. Sohidull Islam, and W. F. Smyth. Reconstructing a string from its lyndon arrays. *Theor. Comput. Sci.*, 710:44–51, 2018.

8 Jean Pierre Duval. Factorizing words over an ordered alphabet. *Journal of Algorithms*, 4(4):363–381, 1983. `doi:10.1016/0196-6774(83)90017-2`.

9 Johannes Fischer. Optimal succinctness for range minimum queries. In *Proceedings of the 9th Latin American Symposium on Theoretical Informatics (LATIN 2010)*, pages 158–169, Oaxaca, Mexico, April 2010. doi:10.1007/978-3-642-12200-2_16.

10 Johannes Fischer. Combined data structure for previous- and next-smaller-values. *Theoretical Computer Science*, 412(22):2451–2456, 2011.

11 Johannes Fischer, Tomohiro I, and Dominik Köppl. Deterministic sparse suffix sorting on rewritable texts. In *Proceedings of the 12th Latin American Theoretical Informatics Symposium (LATIN 2016)*, pages 483–496, Ensenada, México, April 2016. doi:10.1007/978-3-662-49529-2_36.

12 Johannes Fischer and Florian Kurpicz. Dismantling DivSufSort. In *Proceedings of the 25th Prague Stringology Conference (PSC 2017)*, pages 62–76, Prague, Czech Republic, August 2017.

13 Frantisek Franek, A. S. M. Sohidull Islam, Mohammad Sohel Rahman, and William F. Smyth. Algorithms to compute the Lyndon array. In *Proceedings of the 20th Prague Stringology Conference (PSC 2016)*, pages 172–184, Prague, Czech Republic, August 2016.

14 Frantisek Franek, Asma Paracha, and William F. Smyth. The linear equivalence of the suffix array and the partially sorted Lyndon array. In *Proceedings of the 21st Prague Stringology Conference (PSC 2017)*, pages 77–84, Prague, Czech Republic, August 2017.

15 Paweł Gawrychowski and Tomasz Kociumaka. Sparse suffix tree construction in optimal time and space. In *Proceedings of the 28th Annual Symposium on Discrete Algorithms (SODA 2017)*, pages 425–439, Barcelona, Spain, January 2017. doi:10.1137/1.9781611974782.27.

16 Alexander Golynski. Optimal lower bounds for rank and select indexes. *Theoretical Computer Science*, 387(3):348–359, 2007. doi:10.1016/j.tcs.2007.07.041.

17 Torben Hagerup. Sorting and searching on the word ram. In *Proceedings of the 15th Annual Symposium on Theoretical Aspects of Computer Science (STACS 1998)*, pages 366–398, Paris, France, February 1998. doi:10.1007/BFb0028575.

18 Christophe Hohlweg and Christophe Reutenauer. Lyndon words, permutations and trees. *Theoretical Computer Science*, 307(1):173–178, 2003. doi:10.1016/S0304-3975(03)00099-9.

19 Felipe A. Louza, W.F. Smyth, Giovanni Manzini, and Guilherme P. Telles. Lyndon array construction during Burrows–Wheeler inversion. *Journal of Discrete Algorithms*, 50:2–9, May 2018. doi:10.1016/j.jda.2018.08.001.

20 Felipe Alves Louza, Sabrina Mantaci, Giovanni Manzini, Marinella Sciortino, and Guilherme P. Telles. Inducing the Lyndon array. In *Proceedings of the 26th International Symposium on String Processing and Information Retrieval (SPIRE 2019)*, pages 138–151, Segovia, Spain, October 2019. doi:10.1007/978-3-030-32686-9_10.

21 J. Ian Munro and Venkatesh Raman. Succinct representation of balanced parentheses and static trees. *SIAM Journal on Computing*, 31(3):762–776, 2001. doi:10.1137/s0097539799364092.

22 Kunihiko Sadakane and Gonzalo Navarro. Fully-functional succinct trees. In *Proceedings of the 21st Annual Symposium on Discrete Algorithms (SODA 2010)*, pages 134–149, Austin, TX, USA, January 2010. doi:10.1137/1.9781611973075.13.

23 Kazuya Tsuruta, Dominik Köppl, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Grammar-compressed self-index with Lyndon words. *CoRR*, abs/2004.05309, 2020. arXiv:2004.05309.

New Fault Tolerant Subset Preservers

Greg Bodwin
Georgia Tech, Atlanta, GA, USA
greg.bodwin@gmail.com

Keerti Choudhary
Tel Aviv University, Israel
keerti.choudhary@cs.tau.ac.il

Merav Parter
The Weizmann Institute of Science, Rehovot, Israel
merav.parter@weizmann.ac.il

Noa Shahar
The Weizmann Institute of Science, Rehovot, Israel
noa.shahar@weizmann.ac.il

Abstract

Fault tolerant distance preservers are sparse subgraphs that preserve distances between given pairs of nodes under edge or vertex failures. In this paper, we present the first non-trivial constructions of subset distance preservers, which preserve all distances among a subset of nodes S, that can handle either an edge *or* a vertex fault.

- For an n-vertex undirected weighted graph or weighted DAG $G = (V, E)$ and $S \subseteq V$, we present a construction of a subset preserver with $\widetilde{O}(|S|n)$ edges that is resilient to a single fault. In the single pair case ($|S| = 2$), the bound improves to $O(n)$. We further provide a nearly-matching lower bound of $\Omega(|S|n)$ in either setting, and we show that the same lower bound holds conditionally even if attention is restricted to unweighted graphs.
- For an n-vertex directed unweighted graph $G = (V, E)$ and $r \in V, S \subseteq V$,we present a construction of a preserver of distances in $\{r\} \times S$ with $\widetilde{O}(n^{4/3}|S|^{5/6})$ edges that is resilient to a single fault. In the case $|S| = 1$ the bound improves to $O(n^{4/3})$, and for this case we provide another matching conditional lower bound.
- For an n-vertex directed weighted graph $G = (V, E)$ and $r \in V, S \subseteq V$, we present a construction of a preserver of distances in $\{r\} \times S$ with $\widetilde{O}(n^{3/2}|S|^{3/4})$ edges that is resilient to a single vertex fault. (It was proved in [14] that the bound improves to $O(n^{3/2})$ when $|S| = 1$, and that this is conditionally tight.)

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Subset Preservers, Distances, Fault-tolerance

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.15

Category Track A: Algorithms, Complexity and Games

Funding *Greg Bodwin*: Supported in part by NSF awards CCF-1717349, DMS-183932 and CCF-1909756.
Keerti Choudhary: Supported in part by ERC grant no. 803118.
Merav Parter: Supported in part by an ISF grant no. 2084/18.
Noa Shahar: Supported by an ISF grant no. 2084/18.

1 Introduction

Distance Preservers. This paper is about *distance preservers*, a graph-theoretic primitive that appears in work on spanners [16, 32, 23, 17, 2, 1, 24], hopsets [2, 28, 27], shortcutting [27, 2], shortest path algorithms [5, 4, 21, 18], etc.; recently, distance preservers have also been a popular topic in their own right [17, 20, 16, 12, 25, 18].

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 15; pp. 15:1–15:19
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

▶ **Definition 1** (Distance Preservers). *For a graph $G = (V, E)$ and a set $P \subseteq V \times V$ of "demand pairs", a subgraph $H = (V, E_H \subseteq E)$ is a distance preserver of G, P if*

$$\texttt{dist}_G(s,t) = \texttt{dist}_H(s,t) \text{ for all } (s,t) \in P.$$

When $P = S \times S$ for some $S \subseteq V$, we say that H is a subset distance preserver of G, S.

Most often, the goal is to determine the worst-case tradeoff between the number of demand pairs and the number of edges needed for a distance preserver. For example, a classic result in the area is that any p demand pairs in an n-node graph have a distance preserver on $O(np^{1/2})$ edges [20].

The subset distance preserver structure $P = S \times S$ is quite common in the known applications of distance preservers (e.g. [4, 21, 18]), and one of the primary reasons distance preservers were first developed was to address this setting [22]. Despite this, our understanding of subset distance preservers lags far behind our understanding of the general case. There is *no* graph setting in which the structure $P = S \times S$ is known to be useful towards proving sparser distance preservers, and yet our lower bounds get much worse when they are required to have this structure. To illustrate, one of the main questions in the area is whether one can generally have a subset preserver with only a constant number of edges per demand pair:

Is there an absolute constant $c > 0$ such that, for any $|P| = n^{2-c}$ demand pairs in an n-node graph, there is always a distance preserver on $O(|P|)$ edges?

In the *sourcewise* setting $P = S \times V$, the answer is clearly yes (build a shortest path tree rooted at each $s \in S$). In the *pairwise* setting, i.e. P is arbitrary, it was proved by Coppersmith and Elkin [20] that the answer is no (even if attention is restricted to undirected unweighted graphs). But in the intermediate *subset* setting $P = S \times S$, the question is completely open: we can neither prove it for undirected unweighted graphs, nor refute it for directed weighted graphs; all we know is that $c \leq \frac{2}{3}$ [12].

Fault Tolerance. Distance preservers and friends are often applied to networks or distributed systems where parts can spontaneously fail. A natural additional requirement for these applications is *fault-tolerance*, meaning roughly that the preserver is robust to these failures:

▶ **Definition 2** (Fault Tolerant Distance Preservers). *For a graph $G = (V, E)$ and demand pairs P, a distance preserver H of G, P is* fault tolerant (FT) *if for any vertex or edge x we have that $H \setminus \{x\}$ is still a distance preserver of $G \setminus \{x\}$.*[1] *We say H is* vertex (edge) fault tolerant*, abbreviated VFT (EFT), to indicate that x must specifically be a vertex (edge).*

The current literature reflects a world in which vertex faults are harder to analyze than edge faults; many basic questions in the area are closed for EFT but open for (V)FT [19, 9, 8, 13, 15]. To highlight one example:

For a single demand pair (s, t) in an undirected (possibly weighted) graph G, is there always an FT distance preserver on $O(n)$ edges?

In [14], it is proved that the answer is *yes* for the special case of edge failures. The argument leverages a convenient structural fact about edge failures, which may generally explain some of the EFT/VFT discrepancy: a shortest path in $G \setminus \{x\}$ is always the union of two shortest paths in G when x is an edge [3], but nothing like this seems to hold when x is a vertex. Accordingly, the above question is open when the failures can be vertices.

[1] One can also consider a version where several nodes/edges fail at once. However, recent lower bounds have proved that the available preserver quality is quite poor already for $f = 2$ faults [30, 14], so $f = 1$ may be the more applicable setting.

1.1 Our Results

In this work, we show that the $P = S \times S$ structure does actually seem to be useful in the FT setting. For background, it was proved by Bilo et al. [8] that any $|S|$ source nodes in an n-node undirected weighted graph has an EFT preserver on $\widetilde{O}(n|S|)$ edges. Our main results fill in several gaps around this result, extending to the general FT setting and providing corresponding lower bounds.

▶ **Theorem 3** (Undirected Graphs)**.** *For any undirected weighted n-node graph $G = (V, E, w)$ and set of source nodes $S \subseteq V$, there is a general FT subset distance preserver on $\widetilde{O}(n|S|)$ edges. When $|S| = 2$, the bound improves to $O(n)$. Moreover, for any given value of $|S|$ there are examples where $\Omega(n|S|)$ edges are needed.*

We note that Theorem 3 positively resolves the latter open question mentioned above. We think this bound $\widetilde{O}(n|S|)$ is surprising; to explain why, let us compare to the corresponding bounds in the non-faulty setting. Here the current state-of-the-art [20] is that $|S| = \sigma$ source nodes in an n-node graph have a subset distance preserver of size

$$O\left(\min\left\{n + n^{1/2}\sigma^2, n\sigma\right\}\right).$$

In particular, when $\sigma \geq \sqrt{n}$ the bound is $O(n\sigma)$. In other words, following Theorem 3, one can tolerate a fault in this parameter regime essentially *for free*. It is more likely that this reflects the weakness of our current understanding of non-faulty subset distance preservers, rather a world in which fault tolerance is actually free. Still, the non-faulty subset distance preserver bounds have resisted improvement for the last 15 years, so the unintuitive hypothesis of free fault tolerance suggested by Theorem 3 may be hard to refute.

Since Theorem 3 is essentially best possible, for the rest of the paper we investigate to what extent it extends to other graph settings. For example, what if attention is restricted to unweighted graphs? Our lower bound construction fundamentally relies on the use of edge weights, and thus it does not constrain this setting. However, we show that it can be replaced with a *conditional* lower bound: it will not be possible to meaningfully improve Theorem 3 for unweighted graphs without first improving on the non-faulty setting. This result will use a new parameter for distance preservers that we call the *gap*. For a graph G and demand pairs P, the gap is

$$\gamma(P) := \max_{(s,t),(s',t') \in P} \mathtt{dist}_G(s, t) - \mathtt{dist}_G(s', t').$$

Intuitively, the gap measures how close G, P are to a “layered” instance: except for a few degenerate cases, the gap is 0 if and only if G is a layered graph, P is a subset of the first × last layer, and the shortest paths for P cross the layers directly without backtracking. We use the following technical hypothesis for our unweighted lower bound:

▶ **Hypothesis 4.** *For any $\sigma = \sigma_n$, there is an n-node undirected unweighted graph G and demand pairs $P = S \times T$ with*

$$|S| \leq \sigma \qquad \textit{and} \qquad |T| \leq \sqrt{\frac{n\sigma}{(\gamma(P) + 1)}}$$

such that any distance preserver has $\Omega(n\sigma)$ edges.

We remark that this hypothesis is only plausible when $|T| = \Omega(\sigma)$, and hence we need $\gamma(P) = O(n/\sigma)$. But the current understanding of non-faulty distance preservers is compatible with the bounds in Hypothesis 4, even if we were to assume more strongly that $\gamma(P) = 0$.

▶ **Theorem 5.** *Assuming Hypothesis 4, for any $\sigma = \sigma_n$, there are examples of undirected unweighted n-node graphs and node subsets of size $|S| = \sigma$ where any FT distance preserver needs $\Omega(n\sigma)$ edges.*

Thus Theorem 3 is conditionally tight even for unweighted graphs. Next, we ask if it extends to directed graphs. For the special case of DAGs, we observe that any replacement path avoiding a failure can be represented by concatenation of two original shortest paths linked together by an edge (much like the EFT setting). This allows the above content to extend directly to preservers for DAGs.

▶ **Theorem 6** (DAGs)**.** *The upper and lower bound of Theorem 3 both still hold when the input graph G is a weighted DAG. If G is an unweighted DAG, then the lower bound of Theorem 5 still holds as well, assuming that Hypothesis 4 also holds for unweighted DAGs.*

The general directed setting is trickier. The question of single-pair FT preservers for directed weighted graphs was settled in [14]; these need at least $\Omega(n^{4/3})$ and at most $O(n^{3/2})$ edges,[2] and hence Theorem 3 does not extend to this setting. For directed unweighted graphs, we provide the first improvements on this upper bound:

▶ **Theorem 7** (Directed Unweighted Graphs)**.** *For any directed unweighted n-node graph $G = (V, E)$, and demand pairs $(r, S) \in V \times 2^V$ there is an FT distance preserver on $\widetilde{O}(n^{4/3}|S|^{5/6})$ edges. When $|S| = 1$, the bound improves to $O(n^{4/3})$.*

Like before, we show conditional tightness of this bound, although here only in the single-pair setting. Our lower bound needs the following hypothesis:

▶ **Hypothesis 8.** *There is an n-node directed unweighted graph G and demand pairs $P = S \times T$ with*

$$|S|, |T|, \gamma(P) = O(n^{1/3})$$

such that any distance preserver has $\Omega(n^{4/3})$ edges.

Hypothesis 8 is more tenuous than Hypothesis 4, as it is right on the boundary of current techniques. We will discuss the interpretation of these hypotheses more shortly. Still, the point is that one cannot meaningfully improve our single-fault distance preservers without first gaining an improved understanding of the non-faulty case:

▶ **Theorem 9.** *Assuming Hypothesis 8, there are examples of directed unweighted n-node graphs and single demand pairs where any FT distance preserver needs $\Omega(n^{4/3})$ edges.*

We remark that if the goal is just to refute the extension of Theorem 3 to directed graphs, rather than to show exact tightness of Theorem 7, then one can get by with a weaker assumption than Hypothesis 8 (e.g. the preserver lower bound can be $\Omega(n^{1.01})$, or one can trade this off with a relaxation of $|S|, |T|, \gamma(P)$).

Lastly, we mention that our techniques give a nontrivial extension of the single-pair result in [14] to the multi-target setting.

[2] More specifically, it was proved that the correct size bound is exactly that of a non-faulty distance preserver of n demand pairs in an n-node directed weighted graph. Due to [20], this is at least $\Omega(n^{4/3})$ and at most $O(n^{3/2})$.

▶ **Theorem 10** (Directed Weighted Graphs). *For any directed weighted n-node graph $G = (V, E)$ and demand pairs $(r, S) \in V \times 2^V$, there is an FT distance preserver on $\widetilde{O}(n^{3/2}|S|^{3/4})$ edges.*

We recall from [14] that the bound improves to $O(n^{3/2})$ when $|S| = 1$, and that this single-pair bound is unimprovable under the hypothesis that the current bounds for non-faulty distance preservers of directed weighted graphs are tight.

1.2 Interpretation of Hypotheses

Since our lower bounds rely on Hypotheses 4 and 8, we will give them some more context here and discuss how likely they are to be true. Currently, except for a certain tiny range of parameters [12], the state-of-the-art upper bounds for non-faulty $S \times T$ preservers are based entirely on a property called *consistency*:

▶ **Definition 11** (Consistency). *A set of paths Π in a (possibly directed) graph G are* consistent *if, for any $\pi_1, \pi_2 \in \Pi$ with nodes $u < v \in \pi_1, u < v \in \pi_2$, the subpaths $\pi_1[u \rightsquigarrow v], \pi_2[u \rightsquigarrow v]$ are equal.*

In other words, given any set of consistent paths with endpoints in $S \times T$, one can exploit the consistency property to prove upper bounds on the total number of edges contained in the union of all these paths [20, 12]. Interpreting $S \times T$ as demand pairs for a preserver, it is not hard to break shortest path ties in such a way that the chosen paths are consistent, and so these "consistency bounds" yield preserver upper bounds, which are essentially state-of-the-art.

With this in mind, let us imagine a weaker version of Hypotheses 4, where we hypothesize a *consistent* set of $S \times T$ paths with $\Omega(n\sigma)$ edges in their union, the gap $\gamma(S \times T)$ defined as the maximum difference between any two path lengths, and $|S|, |T|$ are bounded as before. From a construction in [16], this hypothesis is true. The similar "consistency-weakened" version of Hypothesis 8 is also true, from an unpublished construction of Bodwin and Reingold.

So, the truth of Hypotheses 4 and 8 essentially depends on how smoothly one can pass from consistent paths to unique shortest paths without destroying the other important properties of the construction, like the edge density and the gap. It is hard to say for sure whether this will be possible. Our guess is that the consistency bounds can be improved – and thus Hypotheses 4, 8 are false – but that this will require major new technical ideas that are not currently in the literature. The main evidence for this is that the consistency bounds have been polynomially improved for general *pairwise* preservers, which do not require the structure $P = S \times T$ [16]. But in the $P = S \times T$ setting the consistency bounds have resisted improvement for the last 15 years [20], despite significant research effort, so we feel that Hypotheses 4, 8 accurately mark the boundaries of current knowledge. Thus we interpret these hypotheses, and the corresponding lower bounds, as proof that no more progress can really be made in the FT setting until the non-faulty setting is understood first.

1.3 Related Work

For an n-vertex unweighted graph G and a set of sources S, Parter and Peleg [31] showed that there exists an $S \times V$ FTP with $O\left(n^{3/2}|S|^{1/2}\right)$ edges. This bound holds also for the case of a single node failure and when the graph is directed. They also showed that this result is existentially optimal, namely, there exist n-vertex graphs and a set of sources S such that any $S \times V$ FTP has at least $\Omega\left(n^{3/2}|S|^{1/2}\right)$ edges.

For the more general case of $P = S \times T$, [8] showed an FTP of size $\widetilde{O}\left(n^{4/3}|S|^{1/3}|T|^{1/3}\right)$ under edge failure. An FTP for a single pair in undirected (possibly weighted) graph of linear size was explicitly presented in [14], but was also implied by previous replacement-paths algorithms, e.g., [29].

Considerably much less is known for the multiple fault setting (even for edges faults). Parter [30] showed an upper bound of $O(n^{5/3})$ edges for $\{s\} \times V$ (edge) f-FTP with $f = 2$, in undirected unweighted graphs. A matching lower bound of $\Omega\left(|S|^{1/(f+1)} \cdot n^{2-1/(f+1)}\right)$ for sourcewise ($P = S \times V$) f-FTP was also provided in [30], for any f. Gupta and Khan [26] extended this result, providing a tight upper bound for sourcewise ($S \times V$) f-FTP with $f = 2$. This bound holds also for the case of a two vertex failures and when the graph is directed. For the more general case of f failures, Bodwin et al. [14] showed an upper bound of$\widetilde{O}(f \cdot |S|^{1/2^f} \cdot n^{2-1/2^f})$ edges for a $S \times V$ f-FTP. This result holds under both edge and vertex faults, and in directed graphs. For weighted graphs, [14] showed that even a single pair f-FTP with $f \geq 2$ has $\Theta(n^2)$ edges.

Baswana and Khanna [7] proved the existence of a $(1+\epsilon)$ multiplicative vertex FT spanner for $P = \{s\} \times V$, with $O(n/\epsilon^3 + n\log n)$ edges, for any $\epsilon > 0$. Recently, Bilo et al. [11] improved this result to $O(n \log n/\epsilon^2)$, for both edge and vertex single failure. In [10], Bilò et al. showed construction of approximate FTP to handle multiple edge failures. They showed that for any $f \geq 1$ and for $P = \{s\} \times V$, we can compute an FTP $O(fn)$ size that after failure of f edge preserves distance up to a multiplicative stretch of $(2f + 1)$.

1.4 Preliminaries and Tools

Graph Notations. We use the following graph notations and definitions in the context of a given undirected graph $G = (V, E, w)$ with $n = |V|$, $m = |E|$ and a weight function $w : E \to \mathbb{R}^+$. To avoid complications due to shortest-paths of the same length, we assume throughout that all shortest path are computed with a consistent tie-breaking function π that guarantees the uniqueness of the shortest-paths[3]. For every $x, y \in V$, and a subgraph $G' \subseteq G$, let $\pi(x, y, G')$ be the (unique) x-y shortest path in G', when $G' = G$ we may simply write $\pi(x, y)$. For any path P, let $P[x, y]$ be the subpath of P between x and y and let $P(x, y) = P[x, y] \setminus \{x, y\}$. For a node $v \in V$, let T_v be the shortest path tree rooted at v. For $v, x \in V$, let $T_v(x)$ the subtree of T_v rooted at x.

For $s, t \in V$ and a failure $x \in V$, the *replacement path* $P_{s,t,x}$ is the unique s-t shortest path in $G \setminus \{x\}$. To avoid cumbersome notation, when s or t are clear from context, we may omit them from the notation. Let $D_{s,t,x}$ be the *detour* segment of the replacement path defined by $P_{s,t,x} \setminus \pi(s, t)$.

For a path P, let $E(P)$ denote its edges and $V(P)$ denote its vertices. Similarly, for a collection of paths $\mathcal{P}$, we denote their edges and vertices with $E(\mathcal{P}) = \bigcup_{P \in \mathcal{P}} E(P)$ and $V(\mathcal{P}) = \bigcup_{P \in \mathcal{P}} V(P)$ respectively. We denote the first and last vertex of a path P by $\mathrm{first}(P)$ and $\mathrm{last}(P)$ respectively. Similarly, for a collection of path $\mathcal{P}$, we denote their sources and terminals with $\mathrm{first}(\mathcal{P}) = \bigcup_{P \in \mathcal{P}} \mathrm{first}(P)$ and $\mathrm{last}(\mathcal{P}) = \bigcup_{P \in \mathcal{P}} \mathrm{last}(P)$. For a tree T and vertices u, v let $\mathrm{LCA}(u, v)$ denote the *least common ancestor* of u and v in T.

Heavy Path Decomposition. For a shortest path tree T_s rooted at s, we use the *heavy-path decomposition* technique devised by Sleator and Tarjan [33] in order to break the tree T_s into vertex-disjoint paths with several desired properties. The following lemma summarizes the main properties of this partitioning scheme (proof can be also found in [7]).

[3] See Def. 11 for a formal definition of a consistent tie-breaking.

▶ **Lemma 12** ([33]). *There exist a linear time algorithm that given an n-vertex tree T computes a path Q in T whose removal breaks T into vertex-disjoint subtrees $T_1, ..., T_\ell$ such that for each $i \leq \ell$:*

- *$|V(T_i)| \leq n/2$ and $V(Q) \cap V(T_i) = \emptyset$,*
- *T_i is connected to Q through some edge.*

The desired decomposition is obtained by recursively applying Lemma 12 (on each of the subtrees T_i), getting a partition of T into vertex disjoint paths $\mathcal{HP}(T)$. A useful property of the decomposition is that for every edge $(u, v) \in T$ that does not appear in any of the paths $\mathcal{HP}(T)$, it holds that $|V(T_s(v))| \leq |V(T_s(u))|/2$. This yields the following useful lemma.

▶ **Lemma 13** ([7]). *For any vertex v, its path to the root in T intersects at most $\log n$ paths in $\mathcal{HP}(T)$.*

2 Fault Tolerant Preservers for Undirected Graphs

2.1 Preservers for Undirected Weighted Graphs

▶ **Theorem 14.** *Any undirected (possibly weighted) $G = (V, E, w)$ and a set $S \subseteq V$ of sources has a $S \times S$ subset (vertex) fault tolerant preserver H with $O(n|S| \log n)$ edges.*

The subgraph H simply contains all the $S \times S$ replacement paths, that is,

$$H = \{P_{s,t,x} \mid s, t \in S,\ x \in \pi(s, t)\}.$$

To prove Theorem 14, we will provide a bound on the size of H.

Size Analysis. A replacement path $P_{s,t,x}$ is *short* if it has at most $n/|S|$ edges; otherwise it is *long*. Throughout this section, we mainly focus on bounding the number of edges in the collection of all short replacement paths, denoted throughout by $\mathcal{P}_{short}$:

▶ **Lemma 15.** $|E(\mathcal{P}_{short})| = O(n|S| \log n)$.

We start by observing that to prove Theorem 14 it is indeed sufficient to consider only the short replacement paths.

▷ Claim 16. Lemma 15 implies Theorem 14.

Proof. Let S^* be a sample of nodes obtained by including each node independently with probability $p = |S|/n$, and let $S' = S^* \cup S$. For each edge e in a long replacement path $P = P_{s,t,x}$, by standard Chernoff bounds, with constant probability there are nodes $s_1, s_2 \in S' \cap P$, separated by $\leq n/|S|$ edges in P, such that e comes between s_1 and s_2 in P. We thus have $e \in P_{s_1,s_2,x}$, so e is now part of a short replacement path. Hence e is counted in Lemma 15 with at least constant probability. Since only $O(n|S| \log n)$ edges may be counted in this way, by linearity of expectation it follows that there are $O(n|S| \log n)$ total edges in long replacement paths. ◁

Road-Map for Proving Lemma 15. From now on, we focus on the collection of short replacement paths, $\mathcal{P} = \mathcal{P}_{short}$. We will show the existence of a subgraph H' with $\widetilde{O}(|S|n)$ edges that contains all the edges of $\mathcal{P}$. This subgraph will be the union of three auxiliary sub-graphs each containing a different subset of replacement paths. For any source $s \in S$, we first show the existence of a subgraph H_s of size $\widetilde{O}(n)$ such that $H_s \cup T(S)$ include *most* of the replacement paths which originate at s, where $T(S) = \cup_s T_s$. Then, letting $H_2 = \bigcup_{s \in S} H_s$, we have that $|H_2| = \widetilde{O}(n|S|)$. Finally, the third subgraph H_3 will include a collection of left-over $O(|S|^2 \log n)$ (short) replacement paths, and thus its size will be bounded by $O(n|S| \log n)$ as well.

Partitioning of Replacement Paths based on Heavy-Path Decomposition. Throughout, we consider a fixed source node $s \in S$ and the set $\mathcal{P}_s = \{P_{s,t,x} \mid P_{s,t,x} \in \mathcal{P}, t \in S\}$ of all its short replacement paths. We then divide these replacement paths into several subsets based on the heavy-path decomposition $\mathcal{HP}(T_s)$ of the shortest-path tree T_s as follows. For every path $Q \in \mathcal{HP}(T_s)$, let

$$\mathcal{P}_s(Q) := \{P_{s,t,x} \mid x \in Q \text{ and } x \neq \mathrm{LCA}(t, \mathrm{last}(Q))\} \quad \text{and} \quad \mathcal{P}'_s = \bigcup_{Q \in \mathcal{HP}(T_s)} \mathcal{P}_s(Q) \ .$$

We will also define a small subset of the *left-over* replacement paths $\mathcal{L}_s = \mathcal{P}_s \setminus \mathcal{P}'_s$. The analysis shows that $|\mathcal{L}_s| = O(|S| \log n)$, and since all these replacement paths are short, the total number of edges in these left-over replacement paths can be bounded by $O(|S| n \log n)$. We next bound the number of edges in each of the subsets $\mathcal{P}_s(Q)$, enjoying the fact that the failures of all the replacement paths in these sets lie on a *single* path.

Bounding the number of edges in $\mathcal{P}_s(Q)$ (failure on a single path). Let $Q = \langle x_0, ..., x_k \rangle$. Following Baswana and Khanna [7], for every $x_i \in Q$, we define the vertex partition of $T_s \setminus \{x_i\}$ into

$$U_i : V(T_s \setminus T_s(x_i)), \quad D_i := V(T_s(x_{i+1})) \text{ and } O_i := V(T_s) \setminus (U_i \cup D_i \cup \{x_i\}).$$

Note that for any $i \neq j$, O_i and O_j are pairwise disjoint. This property is crucial for our construction.

▶ **Observation 17.** *Fix $x_i \in Q$. If $P_{s,t,x_i} \in \mathcal{P}_s(Q)$ and $P_{s,t,x_i} \neq \pi(s,t)$ then $t \in D_i$.*

Proof. Since $U_i \cup O_i \cup D_i = V \setminus \{x_i\}$, we need to rule out two cases. (i) Assume that $t \in U_i$. In this case $x_i \notin \pi(s,t)$, and thus $P_{s,t,x_i} = \pi(s,t)$. (ii) Assume that, $t \in O_i$. In this case, we must also have that $x_i = \mathrm{LCA}(t, \mathrm{last}(Q))$ and thus $P_{s,t,x_i} \notin \mathcal{P}_s(Q)$ (this path will be included in the left-over set $\mathcal{L}_s$). As $U_i \cup O_i \cup D_i = V$, we conclude that $t \in D_i$. ◀

▶ **Lemma 18.** *Fix $x_i \in Q$ and $t \in S \cap D_i$. The replacement path $P = P_{s,t,x_i}$ is of the form $p_1(P) \circ e_1(P) \circ p_2(P) \circ e_2(P) \circ p_3(P)$ where $e_1(P), e_2(P)$ are edges, $p_1(P) \subseteq T_s$, $p_3(P) \subseteq T_t$ and $V(p_2(P)) \subseteq O_i$. Each of the edges $e_1(P), e_2(P)$ (but not both) and some of the paths $p_i(P)$, $i \in \{1,2,3\}$ might be empty.*

Proof. Let $P_i = p_i(P)$ for $i \in \{1,2,3\}$, and $e_j = e_j(P)$ for $j \in \{1,2\}$. Let z be the *first* vertex of the path P (i.e., closest to s) that belongs to D_i. Let u be the *last* vertex of the path P (i.e., closest to t) that belongs to U_i. Let $P_1 = P[s,u]$ and $P_3 = P[z,t]$. We first claim that $P_1 \subset T_s$. To see this, observe that since $u \in U_i$, $x_i \notin \pi(s,u)$ and thus $P_1 = \pi(s,u) \subseteq T_s$. Since $P_1 = \pi(s,u)$, it also implies that u appears strictly before z on the path P.

Next, we show that $P_3 \subset T_t$ by proving that $x_i \notin \pi(z,t)$. Assume towards contradiction otherwise, i.e., that $x_i \in \pi(z,t)$. We have that $\pi(z,t) = \pi(z,x_i) \circ \pi(x_i,t)$. Since (x_i, x_{i+1}) appears on both of the paths $\pi(z,x_i)$ and $\pi(x_i,t)$, we get an alternative z-t path $\pi(z,x_{i+1}) \circ \pi(x_{i+1},t)$ that is strictly shorter, leading to a contradiction. As $x_i \notin \pi(z,t)$, we get that $P_3 = \pi(z,t) \subset T_t$ as required. It remains to consider the path $P(u,z)$. If $P \cap O_i = \emptyset$ then

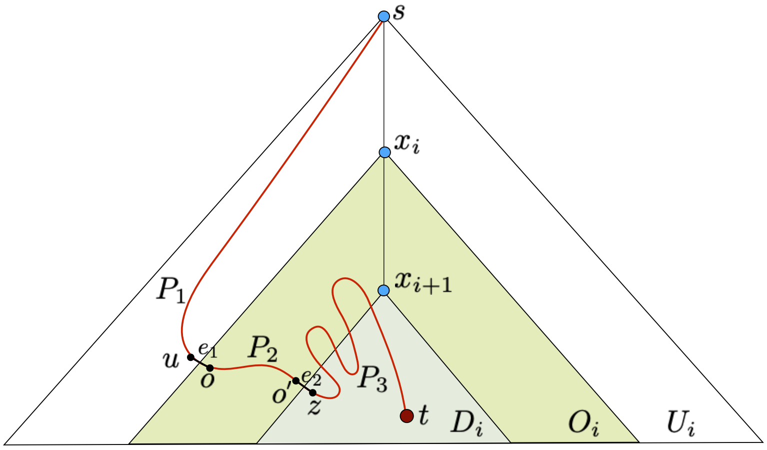


Figure 1 The partition of the path $P = P_{s,t,x_i}$ suggested in Lemma 18 is presented. $P_1 = p_1(P)$ is the portion of P that appears before o, $P_3 = p_3(P)$ is the portion of P from z, and the remaining portion is $P_2 = p_2(P)$. In the figure, $e_1 = e_1(P)$ and $e_2 = e_2(P)$.

we are done. Otherwise, Let o, o' be the first (resp. last) vertices in O_i on the path P. Note that it might be the case that $o = o'$. Letting $e_1 = (u, o)$, $e_2 = (o', z)$, and $P_2 = P[o, o']$, we get that $P = P_1 \circ e_1 \circ P_2 \circ e_2 \circ P_3$ and by definition $V(P_2) \subseteq O_i$. The claim follows. ◀

From now on, for any path $P \in \mathcal{P}_s(Q)$, let $p_i(P), e_j(P)$ denote the partition defined in Lemma 18 (for $i \in [3], j \in [2]$). As $p_1(P), p_3(P) \subseteq T(S)$, it remains to mainly bound the edges contributed by $p_2(P)$ for every $P \in \mathcal{P}_s(Q)$. To do that, we focus on a fixed failure $x_i \in Q$ and define a special graph G_i (which is not necessarily a sub-graph of G) in which $p_2(P)$ is a shortest path. A similar approach has been taken in [7]. For every $x_i \in Q$, define the graph $G_i = (V_i, E_i)$ such that $V_i = O_i \cup \{s\}$ and E_i includes the edges of $G[O_i]$ and the following additional edges. For each $o \in O_i$ with a neighbor in U_i, E_i contains the edge (s, o) with weight $\min_{(u,o)\in E, u\in U_i}\{w(\pi(s, u, G)) + w((u, o))\}$. Letting τ_i denote the shortest paths tree rooted at s in G_i, define

$$H(s, Q) := \left(\bigcup_{i<k}(\tau_i \cap G)\right) \cup \left(\bigcup_{P\in\mathcal{P}_s(Q), j\in\{1,2\}} e_j(P)\right).$$

We show the following:

▶ **Lemma 19.**
(i) *For every $P_{s,t,x_i} \in \mathcal{P}_s(Q)$, it holds that $P_{s,t,x_i} \subseteq T(S) \cup H(s, Q)$.*
(ii) $|E(H(s, Q))| = O(|T_s(\text{first}(Q))| + |\mathcal{P}_s(Q)|)$.

Proof. (i) Since $p_1(P_{s,t,x_i}), p_3(P_{s,t,x_i}) \subseteq T(S)$ and $e_1(P_{s,t,x_i}), e_2(P_{s,t,x_i}) \in H(s, Q)$, it is sufficient to show that $p_2(P_{s,t,x_i}) \subseteq \tau_i \cap G$. Specifically, we show that $p_2(P_{s,t,x_i})$ is a suffix of a shortest path rooted in s in the graph G_i.

Let o, o' be the first (resp., last) nodes in $P_{s,t,x_i} \cap O_i$. First, observe that the existence of an edge $e = (s, o) \in G_i$ with weight $w(e)$ implies that there exists a path in $G[U_i \cup O_i]$ of weight $w(e)$ from s to o, such that all of its vertices are in U_i except the last one. Thus, if there exists a path of weight W from s to $o \in O$ in G_i, it implies that there exists a path in $G[U_i \cup O_i]$ of weight W from s to o. We now claim $p_2(P_{s,t,x_i}) \subseteq \tau_i \cap G$. Let $u \in U_i$ be the vertex preceding o on the path P_{s,t,x_i} such that $e_1(P_{s,t,x_i}) = (u, o)$ (i.e., alternatively, u is the last vertex in U_i on the path P_{s,t,x_i}). By the optimal subpath property,

$P_{s,t,x_i}[s,o']$ is a shortest path in $G \setminus \{x_i\}$, and by Lemma 18 it is in the subgraph $G[U_i \cup O_i]$. Recall that $V(P_{s,t,x_i}[s,u]) \subseteq U_i$ and $V(P_{s,t,x_i}[o,o']) \subseteq O_i$. Thus $e_1(P_{s,t,x_i})$ is the edge that minimizes $w(\pi(s,u,G)) + w((u,o))$, namely, $w((s,o)) = w(\pi(s,u,G)) + w((u,o))$. Therefore $(s,o) \circ P_{s,t,x_i}[o,o']$ is a shortest path from s to o' in G_i, and $p_2(P_{s,t,x_i}) \subseteq \tau_i \cap G$.

We proceed with (ii). As $V(\tau_i \cap G) \subseteq O_i$, by the mutual disjointness of O_i's, it follows that

$$\sum_{i<k} |\tau_i \cap G| \leq \sum_{i<k} |O_i| \leq |T_s(\text{first}(Q))| \ .$$

The claim follows by noting that contribution of $e_1(P)$ and $e_2(P)$ for every $P \in \mathcal{P}_s(Q)$ is at most $2|\mathcal{P}_s(Q)|$. ◀

Interestingly, Lemma 19(ii) yields an immediate linear upper bound for single pair preservers. That is, in the case where $S = \{s,t\}$ and $Q = \pi(s,t)$, the set $\mathcal{P}_s(Q)$ contains all the replacement paths. As $|\mathcal{P}_s(Q)|$ includes at most one path for each failure, its size is bounded by $|\pi(s,t)| = O(n)$.

▶ **Corollary 20** (Single-Pair FT Preservers). *Any undirected (possibly weighted) graph $G = (V,E)$, and a pair of nodes $s,t \in V$ has a single-pair FT Preserver of linear size.*

Bounding the set $\mathcal{P}'_s$. So far, we assume that the failing vertex appears on a fixed path $Q \in \mathcal{HP}(T_s)$. We next bound the total number of edges in the union of all paths $\mathcal{P}'_s = \bigcup_{Q \in \mathcal{HP}(T_s)} \mathcal{P}_s(Q)$. By Lemma 19, every $P_{s,t,x_i} \in \mathcal{P}_s(Q)$ is contained in $T(S) \cup H(s,Q)$. Therefore, it remains to bound the number of edges in the subgraph $H_s = \cup_{Q \in \mathcal{HP}(T_s)} H(s,Q)$.

▶ **Lemma 21.** *$|E(H_s)| = O(n \log n)$.*

Proof. Since $H_s = \cup_{Q \in \mathcal{HP}(T_s)} H(s,Q)$, by Lemma 19 it is sufficient to show that:

- (i) $\Sigma_{Q \in \mathcal{HP}(T_s)} |T_s(\text{first}(Q))| = O(n \log n)$ and
- (ii) $\Sigma_{Q \in \mathcal{HP}(T_s)} |\mathcal{P}_s(Q)| = O(n \log n)$.

Begin with (i). By Lemma 13, for every $t \in S$, the s-t path in T_s intersects with at most $\log n$ paths $Q \in \mathcal{HP}(T_s)$. Since a vertex t appears in $T_s(\text{first}(Q))$ only if $\text{first}(Q) \in \pi(s,t)$, it holds that each vertex belongs to at most $\log n$ such subtrees.

We proceed with (ii) and first show that $\Sigma_{Q \in \mathcal{HP}(T_s)} |\mathcal{P}_s(Q)| \leq |\mathcal{P}'_s|$. Recall that $\mathcal{P}_s(Q)$ includes the replacement paths of $\mathcal{P}'_s$ whose failure appears in Q. Since $\mathcal{HP}(T_s)$ is a partition of T_s, every two paths $Q \neq Q'$ in this partitioning are vertex disjoint. Thus for $Q \neq Q' \in \mathcal{HP}(T_s)$, $\mathcal{P}'_s(Q) \cap \mathcal{P}'_s(Q') = \emptyset$, implying that $\{\mathcal{P}'_s(Q)\}_{Q \in \mathcal{HP}(T_s)}$ is a partition of $\mathcal{P}'_s$ and the claim follows.

Next, we show that $|\mathcal{P}'_s| = O(n \log n)$ by claiming that for every $t \in S$, there are at most $O(n \log n/|S|)$ replacement paths between s and t in $\mathcal{P}'_s$. Fix $t \in S$ and let $\mathcal{P}'_s(t) = \{P_{s,t,x} \mid P_{s,t,x} \in \mathcal{P}'_s\}$. Our goal is to show that $|\mathcal{P}'_s(t)| \leq O(n \log n/|S|)$. Note that in the case where G is unweighted, for every short replacement path $P_{s,t,x}$ with hop-length of $O(n \log n/|S|)$, it holds that the original shortest path $\pi(s,t)$ is short as well, and thus it has at most $O(n \log n/|S|)$ vertex failures that require a replacement path. Consider a shortest path $\pi(s,t)$ in a weighted graph and let $\ell := c \cdot n \log n/|S|$. In the case that $\pi(s,t)$ has hop-length less than 2ℓ, the lemma trivially holds. Otherwise, we assume $|\pi(s,t)| > 2\ell$. Let s' be the ℓ'th vertex on the path from s, namely $\pi(s,s')$ has hop-length of ℓ. Similarly, let t' be the ℓ'th vertex on the path from t, namely $\pi(t',t)$ has hop-length of ℓ. Observe that any short replacement path $P \in \mathcal{P}'_s(t)$ avoids both s', t', otherwise its length is larger than ℓ. This implies that the starting point of any detour of a path in $\mathcal{P}'_s(t)$ is before s' and its ending point is after t'. We show that there is at most one replacement path $P \in \mathcal{P}'_s(t)$ for all the failures in $\pi(s',t')$. Assume towards contradiction that there are two replacement paths $P_v, P_u \in \mathcal{P}'_s(t)$, and $u, v \in \pi(s',t')$. As both P_v, P_u avoid $\pi(s',t')$, both avoid u, v. Thus, we get that both paths are shortest paths in $G \setminus \{u, v\}$, which is a contradiction to the uniqueness of the shortest paths. As any other shortest path must have a failure on $\pi(s,s')$ or $\pi(t',t)$, we have that $|\mathcal{P}'_s(t)| \leq 1 + 2\ell$. Thus, for any $t \in S$, we have that $|\mathcal{P}'_s(t)| = O(n \log n/|S|)$, and $|\mathcal{P}'_s| = O(n \log n)$, as desired. ◀

Bounding the number of edges in the left-over subset $\mathcal{L}_s$. It remains to bound the number of edges contributed by the left-over replacement paths.

▷ Claim 22. $|\mathcal{L}_s| = O(|S| \log n)$ and thus $|E(\mathcal{L}_s)| = O(n \log n)$.

Proof. We first claim that for every fixed $t \in S$, $\mathcal{L}_s$ contains $O(\log n)$ replacement paths between s and t. Thus, $|\mathcal{L}_s| = O(|S| \log n)$. Recall that $P_{s,t,x} \in \mathcal{L}_s$ if and only if there exists a path $Q \in \mathcal{HP}(T_s)$ such that $x \in Q$ and $x = \mathrm{LCA}(t, \mathrm{last}(Q))$. Thus, we have that $x \in \pi(s,t) \cap Q$. According to Lemma 13, there are at most $\log n$ paths from $\mathcal{HP}(T_s)$ intersecting with $\pi(s,t)$ in T_s. Since an LCA is unique for every pair of nodes, for each such path Q there is only one failure x such that $x = \mathrm{LCA}(t, \mathrm{last}(Q))$. Since each path in $\mathcal{L}_s$ is short, i.e., has at most $O(n/|S|)$ edges, the total number of edges in the paths of $\mathcal{L}_s$ is bounded by $O(n|S| \log n)$ as desired. ◁

We are now ready to complete the proof of Lemma 15.

Proof of Lemma 15. The collection of all short replacement paths $\mathcal{P}$ is divided into $\bigcup_s \mathcal{P}'_s$ and the left-over sets $\bigcup_s \mathcal{L}_s$. By Lemma 19(i), $E(\mathcal{P}'_s) \subseteq H_s \cup T(S)$. By Lemma 21, $|E(H_s)| = O(n \log n)$ and thus the total number of edges in $\bigcup_s \mathcal{P}'_s$ is bounded by $O(n|S| \log n)$. By Claim 22, $|E(\mathcal{L}_s)| = O(n \log n)$ and thus $E(\bigcup_s \mathcal{L}_s) = O(n|S| \log n)$. The lemma follows. ◀

2.2 Preservers for Undirected Unweighted Graphs

We next extend our constructions to the $S \times T$ setting. In [8], such an extension has been provided for the case of the single edge failure. We obtain the following theorem.

▶ **Theorem 23.** *For any undirected unweighted $G = (V, E)$ and subsets $S, T \subseteq V$, one can compute a (vertex) fault tolerant $S \times T$ preserver H with $\widetilde{O}(n^{4/3}|S|^{1/3}|T|^{1/3})$ edges.*

Proof. The subgraph H simply contains all the $S \times T$ replacement paths, i.e., $H = \{P_{s,t,x} \mid s, t \in S, \ x \in \pi(s,t)\}$. We will bound size the size of this subgraph, by bounding the size of subgraph H' that contains H. Let R be a random subset of $O(n/L)$ nodes where L is a parameter to be optimized later. Let $\ell = \lceil L \log n \rceil$. The subgraph $H' = H_1 \cup H_2$ is defined as follows.

1. Let H_1 be an $W \times W$ FPT for $W = R \cup S$ obtained by Theorem 14.
2. Let $H_2 = \{\ell\text{-length suffix of } P_{s,t,x} \mid s, t \in S \times T, x \in \pi(s,t), \mathtt{dist}_G(x,t) \leq \ell\}$.

Correctness. Fix $\{s,t\} \in S \times T$ and $x \in \pi(s,t)$. First assume that $|P_{s,t,x}| \leq \ell$. It then implies that also $|\pi(s,t)| \leq \ell$ and thus also that $\mathtt{dist}(x,t) \leq \ell$. Concluding that $P_{s,t,x} \subseteq H_2$. Next, assume that $|P_{s,t,x}| > \ell$. By Chernoff bound, w.h.p. it holds that $|P_{s,t,x} \cap R| \neq \emptyset$. Let $r \in R$ be the closest vertex to t from R on $\pi(s,t)$, we then have that $P_{s,t,x} = P_{s,r,x} \circ P_{r,t,x}$, and $|P_{r,t,x}| \leq \ell$ (as r is the closest vertex to t from R). Since H_1 is an $W \times W$ FPT, we have that $P_{s,r,x} \subset H_1$. It is left to show that $P_{r,t,x}$ is included in either H_1 or H_2. First, assume that $\texttt{dist}(x,t) > \ell$. As $\ell \geq |P_{r,t,x}| \geq |\pi(r,t)|$, it holds that $x \notin \pi(r,t)$ and thus $P_{r,t,x} = \pi(r,x)$. Since the $W \times W$ preservers of Theorem 14 contains all BFS trees of the W nodes, it includes the BFS tree T_r, and $P_{r,t,x} \subset H_1$. Next, assume that $\texttt{dist}(x,t) \leq \ell$. By construction, as $|P_{r,t,x}| \leq \ell$, it holds that $P_{r,t,x} \subset H_2$.

Size Analysis. By Theorem 14, $|E(H_1)| = O\left(n(|S| + n/\ell)\log^2 n\right)$. In addition, $|E(H_2)| = O(|S||T|\ell^2)$ as it includes ℓ edges of ℓ s-t replacement paths for every s,t pair. Letting $\ell = n^{2/3}(|S||T|)^{-1/3}$ gives $O(n^{4/3}(|S||T|)^{1/3}\log^2 n)$. This bound matches the state of the art bound known for one edge failure by [8] (Theorem 11). ◀

3 Fault Tolerant Preservers for General Directed Graphs

We start by defining a couple of notations needed for our constructions. Recall the for each pair of vertices $s,t \in V$, and a node fault $x \in \pi(s,t)$, the detour of replacement path $P_{s,t,x}$ is represented as $D_{s,t,x}$. We denote $\partial D_{s,t,x}$ to denote the partial subpath of $D_{s,t,x}$ obtained by removing the first and last vertex from $D_{s,t,x}$.

Any partial detour $\partial D_{s,t,x}$ is a shortest path in the graph $G \setminus \pi(s,t)$, so we have following lemma.

▶ **Lemma 24.** *For any $s,t \in V$, and $x \in \pi(s,t)$, the path $\partial D_{s,t,x}$ is a shortest path in $G \setminus \pi(s,t)$.*

We now describe the fault-tolerant distance preservers with respect to pair-set $\{r\} \times S$, for a given choice of a root node $r \in V$ and a set $S \subseteq V$. Let T represent the shortest path tree rooted at r in G. We initialize H_0 to T. Further for each $s \in S$ and failure $x \in \pi(r,s)$, we add the two edge present in $E(D_{s,t,x}) \setminus E(\partial D_{s,t,x})$ to H_0. In the process, we add at most $O(n|S|)$ edges to H.

For a directed path Q, we use "H_Q" to represent the minimal sub-graph such that $H_0 + H_Q$ is a 1-FT distance preserver for pairs in $(r \times S)$ when vertex-faults in G are *restricted* to path Q. One of our main-contributions in achieving sparseness for (non-acyclic graphs) is in obtaining tight bound over the size of H_Q. This is captured in the Proposition 25 and Proposition 29.

3.1 Preservers for Directed Weighted Graphs

We obtain the following bound on H_Q for directed weighted graphs.

▶ **Proposition 25.** *For any directed path $Q = (y \rightsquigarrow z)$ in T (between two vertices y, z with y being ancestor of z in T), the graph H_Q, for a directed possibly weighted graph G, requires at most $O(|T(y)|\ |S|^{3/4}\ \sqrt{|Q|})$ edges, where $|Q|$ denotes the number of vertices in path Q.*

Proof. Let Q be a directed $y \rightsquigarrow z$ tree-path in T, for some $y,z \in V$ with y being ancestor of z. Let $k = |S \cap T(y)|$, and α be an integer parameter to be decided later on. For simplicity we assume $K = k/\alpha$ is integer. Let $S_1, \ldots, S_\alpha$ be an arbitrary partition of $S \cap T(y)$, each of size K, satisfying the constraint that for each $(s,s') \in S_i \times S_{i+1}$, LCA$(s,z)$ is either equal to or an ancestor of LCA(s',z). Further let $w_0 = y$, w_i be LCA$(S_i \cup \{z\})$ for $1 \leq i \leq \alpha$, and $w_{\alpha+1} = z$. Observe that for $i \in [1,\alpha]$, w_i is either equal to, or a ancestor of w_{i+1}. Partition Q in consecutive segments (blocks): $B_0 = Q[w_0,w_1]$, $B_1 = Q[w_1,w_2]$, $\ldots$, $B_\alpha = Q[w_\alpha,w_{\alpha+1}]$. Further, let L_i be the number of vertices in B_i, for $0 \leq i \leq \alpha$. (See Figure 2).

We will analyze the failures in each block separately. Fix an index $1 \leq i \leq K$. We distinguish two cases as below.

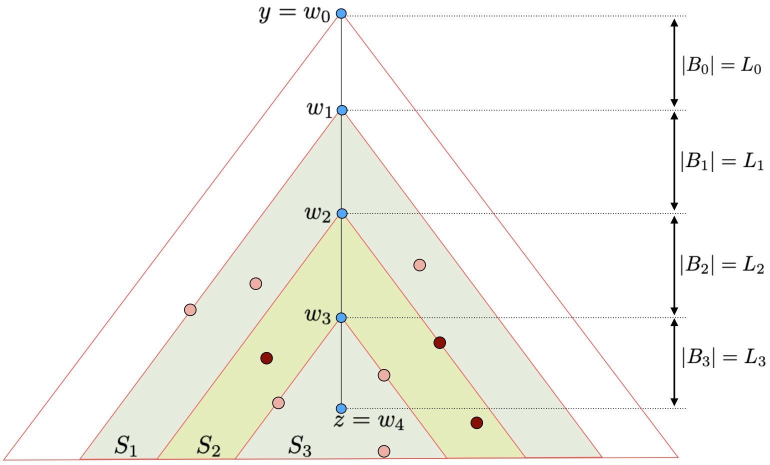


Figure 2 Depiction of block-partitioning of $y \rightsquigarrow z$ tree-path, for $K = \alpha = 3$. Observe that the sets $S_1, S_2, \ldots, S_\alpha$ are each of size $K = 3$.

Analysis of replacement path to vertices in S_i on vertex failure in B_i. Consider a pair $(f, s) \in B_i \times S_i$. Observe that on failure of f, the partial detour $\partial D_{r,s,f}$ (of replacement path $P_{r,s,f}$) is a shortest path in $G \setminus B_i[w_i, \mathrm{LCA}(s, z)]$. (Recall $B_i[w_i, \mathrm{LCA}(s, z)]$ is the sub-path of B_i comprising of those vertices that are ancestor of s in T). Furthermore, it follows from our tie-breaking scheme that none of the vertices of $\partial D_{r,s,f}$ can lie outside $T(y)$. So, for a fixed $s \in S_i$, to compute an (r, s) FT-preserver containing partial-detours corresponding to $f \in B_i$, we can simply use Coppersmith-Elkin's [20] pairwise-distance preserver over graph $G[T(y)] \setminus B_i[w_i, \mathrm{LCA}(s, z)]$, where, $G[T(y)]$ is the graph induced by vertices in subtree $T(y)$.

Since fault f has L_i choices, this gives a bound of $O(|T(y)| \cdot \sqrt{L_i})$ edges, for a single $s \in S_i$. On summing over $|S_i| = K = k/\alpha$ nodes in S_i, and each of the α blocks, we get a bound (say X_1):

$$X_1 = O\Big(|T(y)| \cdot \frac{k}{\alpha} \cdot \sum_{i\in[1,\alpha]} \big(\sqrt{L_i}\big)\Big) \tag{1}$$

Analysis of replacement path to vertices in $\cup_{j>i} S_j$ on vertex failure in B_i. Consider a pair $(f, s) \in B_i \times \cup_{j>i} S_j$. Observe that on failure of f, the partial detour $\partial D_{r,s,f}$ is a shortest path in $G \setminus B_i$, since all the vertices in B_i are ancestor of s. So here we will use a distance preserver over graph $G[T(y)] \setminus B_i$, and the number of pairs (as well as partial-detours) is at most $|S \cap T(y)| \cdot L_i$. Hence, handling failures on B_i incurs us $O(|T(y)| \cdot \sqrt{k\ L_i})$ cost. On summing over all blocks, we get a bound (say X_2):

$$X_2 = O\Big(|T(y)| \cdot \sum_{i\in[0,\alpha]} \big(\sqrt{k \cdot L_i}\big)\Big) \tag{2}$$

For a given α, we have $X_1 \leq O(|T(y)| \cdot (k^2/\alpha)^{1/2} \cdot \sqrt{|Q|})$ and $X_2 \leq O(|T(y)| \cdot (k\alpha)^{1/2} \cdot \sqrt{|Q|})$, where $|Q|$ denotes the number of vertices on path Q. Optimizing over α, we get α must be $\Theta(\sqrt{k})$. This provides a bound of at most $O(|T(y)|k^{3/4}\sqrt{|Q|})$ edges on the size of H_Q. ◀

We are now ready to prove our results for directed weighted graphs. Previously it was know by Bodwin et al. [14] that for a single-pair $(r, s) \in V \times V$, we can compute a FTP with at most $O(n^{1.5})$ edges. A direct implementation of this result over pairs in $\{r\} \times S$ would result in a bound of $O(n^{1.5}|S|)$ edges. However using Proposition 25 and heavy-path decomposition, we are able to obtain a better bound of $o(n^{1.5}|S|)$ size for the $\{r\} \times S$ setting.

Let $Q_1 = (y_1 \rightsquigarrow z_1), \ldots, Q_\gamma = (y_\gamma \rightsquigarrow z_\gamma)$ be the paths in the heavy-path decomposition of T (i.e. $\mathcal{HP}(T)$). Then $\sum_{i=1}^{\gamma} |T(y_i)|$ is $O(n \log n)$ (see Lemma 13). Since $\cup_{i=1}^{\gamma} V(Q_i) = V(T)$, it follows that to handle all failures in T we incur at most $O(n|S|^{3/4}\sqrt{n}\log n)$ cost.

Thus our $\{r\}\times S$ 1-FT distance-preserver for weighted directed graphs require $\widetilde{O}(n^{3/2}|S|^{3/4})$ edges.

▶ **Theorem 26.** *For any n-node directed weighted graph $G = (V, E)$, $(r, S) \in V \times 2^V$, there is a 1-VFT $(r \times S)$ distance preserver H on $\widetilde{O}(n^{3/2}|S|^{3/4})$ edges.*

3.2 Preservers for Directed Unweighted Graphs

Using Proposition 25, together with a deeper insight into shortest-path's structure in unweighted graphs, we provide an alternative and better bound on H_Q.

Let Q be a directed $y \rightsquigarrow z$ tree-path in T, with y being ancestor of z. Let ℓ (a function of Q) be a parameter to be chosen later. For a triplet (r, s, x) we say that $D_{r,s,x}$ is *long* if the partial detour $\partial D_{r,s,x}$ has at least ℓ nodes, and *short* otherwise. We separately analyse the short and long detours.

Long Detours. Let us fix a node $s \in S \cap T(y)$. Let $W_s \subseteq V(T(y))$ be a random sample of nodes obtained by including each node in $T(y) \setminus \pi(r, s)$ independently with probability ℓ^{-1}. For each long detour $D_{r,s,x}$, for $x \in \pi(r, s)$, with constant probability or higher we sample a node $w \in W_s$. Edges of corresponding partial-detour intersecting a node $w \in W_s$ is contained in the union of an in- and out-BFS tree rooted at w. Hence they contain $O(|T(y)|)$ edges. Unioning over all $w \in W_s$, all long detours that intersect W_s contain $O(|T(y)|^2/\ell)$ edges in total. Finally, we note that since each long detour is counted with at least constant probability, there are at most $O(|T(y)|^2/\ell)$ total edges contained in *all* long detours, for a single node s. Summing this over $s \in S$ gives a bound of $O(|S| \cdot |T(y)|^2/\ell)$.

Short Detours. Let $w_0 = y$, and for $i = 1$ to $\alpha = \lfloor |Q|/\ell \rfloor$, let w_i be the descendant of w_{i-1} on Q at a distance ℓ from it. We partition Q into blocks: $B_0, B_1, \ldots, B_\alpha$ such that B_i includes w_i but excludes w_{i+1}, for $i \in [0, \alpha]$. The following lemma presents the disjointness relation for short detours corresponding to non-consecutive blocks.

▶ **Lemma 27.** *For each $s \in S \cap T(y)$ and $x \in B_i$, a short partial detour $\partial\mathcal{D}_{r,s,x}$ lies in $T(w_i) \setminus T(w_{i+2})$.*

Proof. Consider a fault $x \in B_i$ and a node $s \in S \cap T(y)$. Let a, b be respectively the first and last vertices on $\mathcal{D}_{r,s,x}$. As $x \in B_i = Q[w_i, w_{i+1}] \setminus \{w_{i+1}\}$, node a must be at least the grand parent of w_{i+1}. Thus $\texttt{dist}(a, w_{i+2}) \geq \ell + 2$, and moreover, distance from a to all nodes in $T(w_{i+2})$ is also at least $\ell + 2$. Since distance from a to all vertices in $\partial\mathcal{D}_{r,s,x}$ is at most $\ell + 1$, this completes the proof that $\partial\mathcal{D}_{r,s,x}$ is disjoint with $T(w_{i+2})$. ◀

From a more careful implementation of Proposition 29, it follows that the $T(y)$ term in its bound can in-fact be replaced by $T(w_i) \setminus T(w_{i+2})$ term when faults are restricted to B_i (see Lemma 27). Thus the next lemma follows.

▶ **Lemma 28.** *To handle short detours for faults on B_i, for $i \in [0, \alpha]$, we require at most*

$$O\big(|T(w_i) \setminus T(w_{i+2})| \; |S|^{3/4} \; \sqrt{\ell}\big)$$

edges to be added to H_0.

Since $\sum_{i=0}^{\alpha} |T(w_i) \setminus T(w_{i+2})| = O(T(y))$, the total cost of the construction of H_Q is

$$O\big(|S| \cdot |T(y)|^2/\ell + |T(y)|\ |S|^{3/4}\ \sqrt{\ell}\big)\ .$$

Setting $\ell := |S|^{1/6}|T(y)|^{2/3}$ thus proves the following.

▶ **Proposition 29.** *For any directed path $Q = (y \rightsquigarrow z)$ in T (between two vertices y, z with y being ancestor of z in T), the graph H_Q, for a directed unweighted graph G, requires at most $O(|T(y)|^{4/3}\ |S|^{5/6})$ edges.*

As a direct corollary of Proposition 29, we obtain an $O(n^{4/3})$ upper bound for single node-pair preserver. If the input pair is $(r, s) \in V \times V$, then taking $Q = \pi(r, s)$, provides us the following.

▶ **Corollary 30.** *For any n-node directed unweighted graph $G = (V, E)$, $r, s \in V$, there is a 1-VFT (r, s) distance preserver H on $O(n^{4/3})$ edges.*

We now use the technique of heavy path decomposition. Let $Q_i = (y_i \rightsquigarrow z_i)$, $1 \leq i \leq \gamma$ be the paths in the heavy-path decomposition of T (i.e. $\mathcal{HP}(T)$). Then $\sum_{i=1}^{\gamma} |T(y_i)|$ is $O(n \log n)$. So Proposition 29 directly proves the following.

▶ **Theorem 31.** *For any n-node directed unweighted graph $G = (V, E)$, $(r, S) \in V \times 2^V$, there is a 1-VFT $(r \times S)$ distance preserver H on $\widetilde{O}(n^{4/3}|S|^{5/6})$ edges.*

4 Fault Tolerant Preservers for Directed Acyclic Graphs

For DAGs, we first present our result for single node-pair case.

▶ **Theorem 32.** *For every n-node directed graph $G = (V, E)$ and a pair $p \in V \times V$, there exists a FT-vertex preserver $H \subseteq G$ for p with $O(n)$ edges.*

Proof. Let $p = (s, t)$, and $Q = \pi(s, t)$ denote the s-t shortest path. Let T_1 (T_2) be an outgoing (incoming) shortest path tree rooted at s (t) in G such that the s-t path in it overlaps with Q; and let H be initialized to $T_1 \cup T_2$. Consider a vertex failure x on Q. Let y_x be the last vertex on the replacement path $P_{s,t,x}$ lying outside subtree $T_1(x)$, and let z_x be its successor. As G is acyclic the z_x to t shortest path in G cannot pass through x. Thus we may assume $Q[s, y_x]$ is contained in T_1, and $Q[z_x, t]$ is contained in T_2. So, for each $x \in Q$, it only remains to add edge (y_x, z_x) to H, thereby proving the linear size bound. ◀

In above theorem, we showed that for each $x \in \pi(s, t)$, there is exists an edge $e_{x,s,t} = (y_x, z_x)$ such that

1. $\textsc{treepath}_{T_s}(s, y_x)$ doesn't contains x,
2. no shortest-path from z_x to t can contain x, and
3. the concatenated path $\textsc{treepath}_{T_s}(s, y_x) \cdot (y_x, z_x) \cdot \pi_G(z_x, t)$ is an s-t shortest path in $G \setminus x$,

where T_s, for $s \in S$, denotes the shortest path tree rooted at s.

To extend our construction to $S \times S$ setting we proceed as follows. We choose a uniformly random set $\widetilde{S}$ of $\Theta(|S|)$ vertices, and let $R = S \cup \widetilde{S}$ and $L = \frac{n \log n}{|S|}$. Initialize H to $\cup_{r \in R}(T_r)$. Next, for each $s, t \in R$ satisfying $\texttt{dist}_G(s, t) \leq L$ and each $x \in \pi_G(s, t)$, add the edge $e_{x,s,t}$ to H. Observe that in this process, we include at most $O(n|S| \log n)$ edges to H. Thus the size of H is at most $O(n|S| \log n)$.

It remains to prove the correctness of H. Consider a pair $(s,t) \in S \times S$, and a vertex $x \in \pi_G(s,t)$. Let $(s = r_0, r_1, r_2, \ldots, r_\ell = t)$ be the vertices in R lying on $P_{s,t,x}$, in the order they appear. (Recall $P_{s,t,x}$ denote the s to t replacement path in $G \setminus x$). With high probability, length between consecutive r_i's in $P_{s,t,x}$ is at most L. This shows that with high probability each segment $P_{s,t,x}[r_i, r_{i+1}]$, for $i < \ell$, is present in $G \setminus x$.

Therefore, with high probability, H is a $S \times S$ fault-tolerant preserver for the input DAG G, and it comprises of at most $O(n|S|\log n)$ edges.

▶ **Theorem 33.** *Any DAG (possibly weighted) $G = (V, E, w)$ and a set $S \subseteq V$ of sources has a $S \times S$ sourcewise (vertex) fault tolerant preserver H with $O(n|S|\log n)$ edges.*

Using the same analysis as in Theorem 23, the above result can be extended to obtain a $S \times T$ preserver for unweighted DAGs with $O(n^{4/3}(|S||T|)^{1/3}\log^2 n)$ edges.

▶ **Theorem 34.** *For any unweighted DAG $G = (V, E)$ and subsets $S, T \subseteq V$, one can compute a (vertex) fault tolerant $S \times T$ preserver H with $\widetilde{O}(n^{4/3}|S|^{1/3}|T|^{1/3})$ edges.*

5 Lower Bounds for FT Preservers

5.1 Unconditional Lower-Bounds for Weighted Graphs

We show here a construction of a graph G on $O(n)$ vertices with integral edge weights in range $[1, n^c]$ (for some constant c) such that its $(\{s\} \times S)$-distance preserver requires at least $\Omega(n|S|)$ edges. Our lower bound construction is an adaption of $\{s\} \times V$ $(1+\epsilon)$-FTP by [6] to the $\{s\} \times S$ exact-FTP setting.

The vertex set of $V(G)$ constitutes $n + \sigma$ vertices, and is union of disjoint sets $U = \{u_1, u_2, \ldots, u_n\}$ and $W = \{w_1, w_2, \ldots, w_\sigma\}$. The edge set of $E(G)$ (in both directed as well as undirected scenario) is the union of the following two sets.

- $E_U = \{(u_n, u_{n-1}), \ldots, (u_i, u_{i-1}), \ldots, (u_2, u_1)\}$, with $wt(u_i, u_{i-1}) = 1$.
- $E_{U,W} = \{(u_i, w_j) \mid i \in [1, n], j \in [1, \sigma]\}$, with $wt(u_i, w_j) = (i \cdot n^4)$.

Say that $S = W$, and let $s = u_n$ be the designated source vertex. Let T_s be the shortest path tree rooted at s in G. It is easy to verify the set $E_U \cup (\{u_1\} \times W)$ constitute the edges of T_s. Now for any $i \in [1, n]$ and $j \in [1, \sigma]$, let $P_{i,j}$ denote the path $(u_n, u_{n-1}, \ldots, u_i) \circ (u_i, w_j)$. Then for any i, j, $wt(P_{i,j}) = i \cdot n^4 + (n - i)$.

If vertex u_{i-1} fails then the shortest path from s to vertex w_j $(j \in [1, \sigma])$ in G is path $P_{i,j}$. Hence each w_j must keep all its incoming edges in the FT-preserver. This shows there exists graphs whose $\{s\} \times S$ FT-preserver must contain $\Omega(n|S|)$ edges, thereby, implying the following result.

▶ **Theorem 35.** *For any positive integers n and σ $(\leq n)$, there exists an n-vertex (un)directed weighted graph $G = (V, E)$ with pair $(s, S) \in V \times 2^V$ satisfying $|S| = \sigma$ whose $\{s\} \times S$ 1-FT-distance-preserver must contain $\Omega(n|S|)$ edges.*

5.2 Conditional Lower-Bounds for Undirected Unweighted Graphs

▶ **Hypothesis 36** (Gap $S \times T$ Distance Preserver Lower Bounds)**.** *For any $\sigma = \sigma_n$, there is an n-node undirected unweighted graph $G = (V, E)$ and demand pairs $P = S \times T$ with $|S| \leq \sigma, |T| \leq \sqrt{n\sigma/\gamma(P)}$ such that any distance preserver of P has $\Omega(n\sigma)$ edges.*

▶ **Theorem 37.** *Assuming Hypothesis 36, for any $\sigma = \sigma_n$, there are examples of n-node undirected unweighted graphs $G = (V, E)$ and node subsets of size $|S| = \sigma$ where $\Omega(n\sigma)$ edges are needed for an $S \times S$ FT preserver of an undirected unweighted graph.*

We first say the construction, which uses ideas from [31]. Let $\gamma = \gamma(P)$ be the gap of an instance from Hypothesis 36. For $i \in [\sigma]$, we create identical disjoint trees T_i, each on $O(n/\sigma)$ nodes, computed as follows. Let

$$\beta := \sqrt{\frac{n}{\sigma\gamma}},$$

and start with β disjoint paths $L_{i,1}, \dots, L_{i,\beta}$, where $L_{i,j}$ is path on $2j\gamma$ nodes with endpoints $(q_{i,j}, s_{i,j})$. For $j \in [\beta - 1]$, the parent of $q_{i,j}$ is set to $q_{i,j+1}$ by adding an edge; thus the node $q_{i,\beta} =: r_i$ is naturally viewed as the root of T_i, and $\{s_{i,1}, \dots, s_{i,\beta}\}$ are then the leaves. We set

$$\widetilde{T} = \{s_{i,j} \mid i \in [\alpha],\ j \in [\beta]\}$$

and take $\widetilde{S}$ to be a set of $|\widetilde{S}| = \sigma$ new nodes. Noting that $\left|\widetilde{T}\right| = \sigma\beta = \sqrt{n\sigma/\gamma}$, between $\widetilde{S}$ and $\widetilde{T}$ we may plug in an n-node distance preserver lower bound graph H from Hypothesis 36. We then let

$$S := \widetilde{S} \cup \{r_1, \dots, r_\alpha\}.$$

This completes the construction. One immediately counts that the number of vertices is

$$n + \left|\bigcup_{i=1}^{\alpha} T_i\right| = O(n),$$

and $|S| = 2\sigma$, so it remains to argue that an FT subset distance preserver must contain a non-faulty $\widetilde{S} \times \widetilde{T}$ preserver in the copy of H, which thus has $\Omega(n\sigma)$ edges. Consider nodes $t_{i,j} \in \widetilde{T}, s \in \widetilde{S}$. Following the argument in [31], one can verify that on failure of $q_{i,j-1}$, every shortest $r_i \rightsquigarrow s$ path has the form

$$(q_{i,\beta}, \dots, q_{i,j}) \circ L_{i,j} \circ \pi(t_{i,j}, s)$$

where $\pi(t_{i,j}, s)$ is a shortest $t_{i,j} \rightsquigarrow s$ path in H.

5.3 Conditional Lower-Bounds for Directed Unweighted Graphs

For directed unweighted graphs, we prove:

▶ **Hypothesis 38** (Layered $S \times T$ Directed Distance Preserver Lower Bounds). *There is an n-node directed unweighted graph $G = (V, E)$ and demand pairs $S \times T$ with $|S|, |T|, \gamma(S \times T) \leq O(n^{1/3})$ such that any distance preserver of P has $\Omega(n^{4/3})$ edges.*

▶ **Theorem 39.** *Assuming Hypothesis 38, there are examples where $\Omega(n^{4/3})$ edges are needed for a single-pair 1-VFT preserver of a directed unweighted graph.*

To prove Theorem 39, we start with an $s \rightsquigarrow t$ shortest path π of length $\Theta(n^{2/3})$ and an n-node distance preserver lower bound H from Hypothesis 38, and then we add some additional nodes and edges to the graph to carefully connect these two parts of the construction. Then, similar in spirit to our previous lower bounds, we prove that for each $s' \in S, t' \in T$ pair one can fault a particular node on π so that every shortest $s \rightsquigarrow t$ path passes through s', t'. This means one must implicitly keep a (non-faulty) distance preserver of H, which thus has $\Omega(n^{4/3})$ nodes.

References

1 Amir Abboud and Greg Bodwin. The 4/3 additive spanner exponent is tight. *J. ACM*, 64(4):28:1–28:20, 2017.

2 Amir Abboud, Greg Bodwin, and Seth Pettie. A hierarchy of lower bounds for sublinear additive spanners. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 568–576. Society for Industrial and Applied Mathematics, 2017.

3 Yehuda Afek, Anat Bremler-Barr, Haim Kaplan, Edith Cohen, and Michael Merritt. Restoration by path concatenation: fast recovery of MPLS paths. *Distributed Computing*, 15(4):273–283, 2002.

4 Donald Aingworth, Chandra Chekuri, Piotr Indyk, and Rajeev Motwani. Fast estimation of diameter and shortest paths (without matrix multiplication). *SIAM Journal on Computing*, 28(4):1167–1181, 1999.

5 Noga Alon. Testing subgraphs in large graphs. *Random Structures & Algorithms*, 21(3-4):359–370, 2002.

6 Surender Baswana, Keerti Choudhary, Moazzam Hussain, and Liam Roditty. Approximate single source fault tolerant shortest path. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1901–1915, 2018.

7 Surender Baswana and Neelesh Khanna. Approximate shortest paths avoiding a failed vertex: Near optimal data structures for undirected unweighted graphs. *Algorithmica*, 2013.

8 Davide Bilò, Keerti Choudhary, Luciano Gualà, Stefano Leucci, Merav Parter, and Guido Proietti. Efficient oracles and routing schemes for replacement paths. In *35th Symposium on Theoretical Aspects of Computer Science, STACS 2018, February 28 to March 3, 2018, Caen, France*, pages 13:1–13:15, 2018.

9 Davide Bilò, Fabrizio Grandoni, Luciano Gualà, Stefano Leucci, and Guido Proietti. Improved purely additive fault-tolerant spanners. In *Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, pages 167–178, 2015.

10 Davide Bilò, Luciano Gualà, Stefano Leucci, and Guido Proietti. Multiple-edge-fault-tolerant approximate shortest-path trees. In *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016, February 17-20, 2016, Orléans, France*, pages 18:1–18:14, 2016.

11 Davide Bilò, Luciano Gualà, Stefano Leucci, and Guido Proietti. Fault-tolerant approximate shortest-path trees. *Algorithmica*, 80(12):3437–3460, 2018.

12 Greg Bodwin. Linear size distance preservers. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 600–615, 2017.

13 Greg Bodwin, Michael Dinitz, Merav Parter, and Virginia Vassilevska Williams. Optimal vertex fault tolerant spanners (for fixed stretch). In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1884–1900. Society for Industrial and Applied Mathematics, 2018.

14 Greg Bodwin, Fabrizio Grandoni, Merav Parter, and Virginia Vassilevska Williams. Preserving distances in very faulty graphs. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, pages 73:1–73:14, 2017.

15 Greg Bodwin and Shyamal Patel. A trivial yet optimal solution to vertex fault tolerant spanners. In *Proceedings of the 26th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 541–543, 2019.

16 Greg Bodwin and Virginia Vassilevska Williams. Better distance preservers and additive spanners. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 855–872, 2016.

17 Béla Bollobás, Don Coppersmith, and Michael Elkin. Sparse distance preservers and additive spanners. *SIAM Journal on Discrete Mathematics*, 19(4):1029–1055, 2005.

18 Hsien-Chih Chang, Powel Gawrychowski, Shay Mozes, and Oren Weimann. Near-optimal distance preserver for planar graphs. In *Proceedings of the Twenty-Sixth Annual European Symposium on Algorithms*, 2018.

19 Shiri Chechik, Michael Langberg, David Peleg, and Liam Roditty. f-sensitivity distance oracles and routing schemes. *Algorithmica*, 63(4):861–882, 2012.

20 Don Coppersmith and Michael Elkin. Sparse sourcewise and pairwise distance preservers. *SIAM J. Discrete Math.*, 20(2):463–501, 2006.

21 Dorit Dor, Shay Halperin, and Uri Zwick. All-pairs almost shortest paths. *Siam Journal on Computing (SICOMP)*, 29(5):1740–1759, 2000.

22 Michael Elkin. Personal communication.

23 Michael Elkin, Arnold Filtser, and Ofer Neiman. Terminal embeddings. *Theoretical Computer Science*, 697:1–36, 2017.

24 Michael Elkin and Seth Pettie. A linear-size logarithmic stretch path-reporting distance oracle for general graphs. *ACM Transactions on Algorithms (TALG)*, 12(4):50, 2016.

25 Kshitij Gajjar and Jaikumar Radhakrishnan. Distance-Preserving Subgraphs of Interval Graphs. In Kirk Pruhs and Christian Sohler, editors, *25th Annual European Symposium on Algorithms (ESA 2017)*, volume 87 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 39:1–39:13, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ESA.2017.39.

26 Manoj Gupta and Shahbaz Khan. Multiple source dual fault tolerant BFS trees. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, pages 127:1–127:15, 2017. doi:10.4230/LIPIcs.ICALP.2017.127.

27 S.-E. Huang and S. Pettie. Lower bounds on sparse spanners, emulators, and diameter-reducing shortcuts. In *Proceedings 16th Scandinavian Symposium and Workshops on Algorithm Theory*, pages 26:1–26:12, 2018.

28 S.-E. Huang and S. Pettie. Thorup-Zwick emulators are universally optimal hopsets. *Information Processing Letters*, 2018.

29 Enrico Nardelli, Guido Proietti, and Peter Widmayer. A faster computation of the most vital edge of a shortest path. *Information Processing Letters*, 79(2):81–85, 2001.

30 Merav Parter. Dual failure resilient BFS structure. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 21–23, 2015*, pages 481–490, 2015. doi:10.1145/2767386.2767408.

31 Merav Parter and David Peleg. Sparse fault-tolerant BFS structures. *ACM Trans. Algorithms*, 13(1):11:1–11:24, 2016.

32 Seth Pettie. Low distortion spanners. *ACM Transactions on Algorithms (TALG)*, 6(1):7, 2009.

33 Daniel Dominic Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26(3):362–391, 1983.

Bridge-Depth Characterizes Which Structural Parameterizations of Vertex Cover Admit a Polynomial Kernel

Marin Bougeret
LIRMM, Université de Montpellier, CNRS, France
`http://www.lirmm.fr/~bougeret/`
marin.bougeret@lirmm.fr

Bart M. P. Jansen
Eindhoven University of Technology, The Netherlands
b.m.p.jansen@tue.nl

Ignasi Sau
LIRMM, Université de Montpellier, CNRS, France
ignasi.sau@lirmm.fr

Abstract

We study the kernelization complexity of structural parameterizations of the Vertex Cover problem. Here, the goal is to find a polynomial-time preprocessing algorithm that can reduce any instance (G, k) of the Vertex Cover problem to an equivalent one, whose size is polynomial in the size of a pre-determined complexity parameter of G. A long line of previous research deals with parameterizations based on the number of vertex deletions needed to reduce G to a member of a simple graph class $\mathcal{F}$, such as forests, graphs of bounded tree-depth, and graphs of maximum degree two. We set out to find the most general graph classes $\mathcal{F}$ for which Vertex Cover parameterized by the vertex-deletion distance of the input graph to $\mathcal{F}$, admits a polynomial kernelization. We give a complete characterization of the minor-closed graph families $\mathcal{F}$ for which such a kernelization exists. We introduce a new graph parameter called *bridge-depth*, and prove that a polynomial kernelization exists if and only if $\mathcal{F}$ has bounded bridge-depth. The proof is based on an interesting connection between bridge-depth and the size of minimal blocking sets in graphs, which are vertex sets whose removal decreases the independence number.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases vertex cover, parameterized complexity, polynomial kernel, structural parameterization, bridge-depth

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.16

Category Track A: Algorithms, Complexity and Games

Related Version The full version of this paper is permanently available at `https://arxiv.org/abs/2004.12865`.

Funding *Bart M. P. Jansen*: This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 803421, ReduceSearch).
Ignasi Sau: Supported by French projects DEMOGRAPH (ANR-16-CE40-0028) and ESIGMA (ANR-17-CE23-0010).

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 16; pp. 16:1–16:19
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Background and motivation. The NP-complete Vertex Cover problem is one of the most prominent problems in the field of kernelization [3, 8, 13, 16, 26], which investigates provably efficient and effective preprocessing for parameterized problems. A *parameterized problem* is a decision problem in which a positive integer k, called the *parameter*, is associated with every instance x. A *kernelization* for a parameterized problem is a polynomial-time algorithm that reduces any parameterized instance (x, k) to an equivalent instance (x', k') of the same problem whose size is bounded by $f(k)$ for some function f, which is the *size* of the kernelization. Hence a kernelization guarantees that instances which are large compared to their parameter, can be efficiently reduced without changing their answer. Of particular interest are *polynomial kernelizations*, whose size bound f is polynomial.

An instance (G, k) of Vertex Cover asks whether the undirected graph G has a vertex set S of size at most k that contains at least one endpoint of every edge. Using the classic Nemhauser-Trotter theorem [28], one can reduce (G, k) in polynomial time to an instance (G', k') with the same answer, such that $|V(G')| \leq 2k$. Hence when using the size of the desired solution as the parameter, Vertex Cover has a kernelization that reduces to instances of $2k$ vertices, which can be encoded in $\mathcal{O}(k^2)$ bits. While the bitsize of this kernelization is known to be essentially optimal [10] assuming the established conjecture $\mathsf{NP} \not\subseteq \mathsf{coNP/poly}$, this result does not guarantee any effect of the preprocessing for instances whose solution has size at least $|V(G)|/2$. In particular, it does not promise any size reduction when G is simply a path.

To be able to give better preprocessing guarantees, one can use *structural parameters* which take on smaller values than the size of a minimum vertex cover, a quantity henceforth called the *vertex cover number*. Such structural parameterizations can conveniently be described in terms of the vertex-deletion distance to certain graph families $\mathcal{F}$. Note that the vertex cover number $\textsc{vc}(G)$ of G can be defined as the minimum number of vertex deletions needed to reduce G to an edgeless graph. Hence this number will always be at least as large as the *feedback vertex number* $\textsc{fvs}(G)$ of G, which is the vertex-deletion distance of G to a forest. In 2011, it was shown that Vertex Cover even admits a polynomial kernelization when parameterized by the feedback vertex number [21, 22]. This triggered a long line of follow-up research, which aimed to find the most general graph families $\mathcal{F}$ such that Vertex Cover admits a polynomial kernelization when parameterized by vertex-deletion distance to $\mathcal{F}$. Polynomial kernelizations were obtained for the families $\mathcal{F}$ of graphs of maximum degree two [27], of graphs of constant *tree-depth* [5, 23], of the *pseudo-forests* where each connected component has at most one cycle [17], and for *d-quasi-forests* in which each connected component has a feedback vertex set of size at most $d \in \mathcal{O}(1)$ [18]. Note that all these target graph classes are *closed under taking minors*. Using *randomized algorithms* with a small error probability, polynomial kernelizations are also known for several parameterizations by vertex-deletion distance to graph classes that are not minor-closed, such as Kőnig graphs [25], bipartite graphs [25], and parameterizations based on the linear-programming relaxation of Vertex Cover [19, 24]. On the negative side, it is known that Vertex Cover parameterized by the vertex-deletion distance to a graph of treewidth two [9] does *not* have a polynomial kernel, unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$. This long line of research into kernelization for structural parameterizations raises the following question:

> How can we characterize the graph families $\mathcal{F}$ for which Vertex Cover parameterized by vertex-deletion distance to $\mathcal{F}$ admits a polynomial kernel?

We answer this question for all minor-closed families $\mathcal{F}$, by introducing a new graph parameter.

Our results. We introduce a new graph parameter that we call *bridge-depth.* It has a recursive definition similar to that of tree-depth [30] (full definitions follow in Section 3), but deals with bridges in a special way. A graph without vertices has bridge-depth zero. The bridge-depth $\mathsf{bd}(G)$ of a disconnected graph G is simply the maximum bridge-depth of its connected components. The bridge-depth of a connected nonempty graph G is defined as follows. Let G_{CB} denote the graph obtained from G by contracting each edge that is a bridge in G; the order does not matter. Then $\mathsf{bd}(G) := 1 + \min_{v \in V(G_{\text{CB}})} \mathsf{bd}(G_{\text{CB}} \setminus v)$. Intuitively, the bridge-depth of G is given by the depth of an elimination process [6] that reduces G to the empty graph. One step consists of contracting all bridges and removing a vertex; each of the remaining connected components is then recursively eliminated in parallel. From this definition, it is not difficult to see that $\mathsf{bd}(G)$ is at least as large as the *tree-width* of G, but never larger than the tree-depth or feedback vertex number of G. In particular, any forest has bridge-depth one.

Using the notion of bridge-depth, we characterize the minor-closed families $\mathcal{F}$ for which Vertex Cover parameterized by vertex-deletion distance to $\mathcal{F}$ admits a polynomial kernel.

▶ **Theorem 1.1.** *Let $\mathcal{F}$ be a minor-closed family of graphs, and assume $\mathsf{NP} \nsubseteq \mathsf{coNP/poly}$. Vertex Cover parameterized by vertex-deletion distance to $\mathcal{F}$ has a polynomial kernelization if and only if $\mathcal{F}$ has bounded bridge-depth.*

Theorem 1.1 gives a clean and unified explanation for all the minor-closed families $\mathcal{F}$ that were previously considered individually [5, 17, 18, 22, 27], and generalizes these results as far as possible. To the best of our knowledge, Theorem 1.1 captures all known (deterministic) kernelizations for structural parameterizations of Vertex Cover. (There are *randomized* kernelizations [19, 24, 25] which apply for distance to classes $\mathcal{F}$ that are *not* minor-closed, such as bipartite graphs.) For example, we capture the case of $\mathcal{F}$ being a forest [22] since forests have bridge-depth one, and the case of $\mathcal{F}$ being graphs of constant tree-depth [5, 23] since bridge-depth does not exceed tree-depth. In this sense, bridge-depth can be seen as the ultimate common generalization of feedback vertex number and tree-depth (which are incomparable parameters) in the context of polynomial kernels for Vertex Cover.

We consider it one of our main contributions to identify the graph parameter bridge-depth as the right way to capture the kernelization complexity of Vertex Cover parameterizations.

Techniques. To describe our techniques, we introduce some terminology. Let $\alpha(G)$ denote the independence number of graph G, i.e., the maximum size of a set of pairwise nonadjacent vertices. A *blocking set* in a graph G is a vertex set $Y \subseteq V(G)$ such that $\alpha(G \setminus Y) < \alpha(G)$. Hence if Y is a blocking set, then every maximum independent set in G contains a vertex from Y. Earlier kernelizations for Vertex Cover parameterized by distance to a graph class $\mathcal{F}$, starting with the work of Jansen and Bodlaender [22], all rely, either implicitly or explicitly, on having upper-bounds on the size of (inclusion-)minimal blocking sets for graphs in $\mathcal{F}$ [5, 17, 18, 22, 27]. For example, it is known that minimal blocking sets in a bipartite graph have size at most two [18, Cor. 11], while minimal blocking sets in graphs of tree-depth c have size at most 2^c [5, Lemma 1]. Similarly, all the existing superpolynomial kernelization lower bounds for parameterizations by distance to $\mathcal{F}$, rely on $\mathcal{F}$ having minimal blocking sets of arbitrarily large size. Indeed, if $\mathcal{F}$ is closed under disjoint union and has arbitrarily large blocking sets, it is easy to prove a superpolynomial lower bound (cf. [19, Thm. 1]).

Since all positive cases for kernelization are when minimal blocking sets of graphs in $\mathcal{F}$ have bounded size, while one easily obtains lower bounds when the size of minimal blocking sets of graphs in $\mathcal{F}$ is unbounded, the question rises whether a bound on the size of minimal blocking sets is a necessary and sufficient condition for the existence of polynomial kernels. To our initial surprise, we show that for minor-closed families $\mathcal{F}$, this is indeed the case: the purely structural property of having bounded-size minimal blocking sets can always be leveraged into preprocessing algorithms.

For an insight into our techniques, consider an instance (G,k) of Vertex Cover, together with a vertex set $X \subseteq V(G)$ such that $G \setminus X \in \mathcal{F}$ for some minor-closed family $\mathcal{F}$ that has bounded-size minimal blocking sets. The goal of the kernelization is then to reduce to an equivalent instance of size $|X|^{\mathcal{O}(1)}$ in polynomial time. Using ideas of the previous kernelizations [5,22], it is quite simple to reduce the *number of connected components* of $G \setminus X$ to size $|X|^{\mathcal{O}(1)}$. To obtain a polynomial kernel, the challenge is therefore to bound the size of each such component C of $G \setminus X$ to $|X|^{\mathcal{O}(1)}$, so that the overall instance size becomes polynomial in $|X|$. However, the non-existence of large minimal blocking sets does not seem to offer any handle for reducing the size of individual components of $G \setminus X$. The route to the kernelization therefore goes via the detour of bridge-depth. We prove the following relation between the sizes of minimal blocking sets and bridge-depth.

▶ **Theorem 1.2.** *Let $\mathcal{F}$ be a minor-closed family of graphs. Then $\mathcal{F}$ has bounded bridge-depth if and only if the size of minimal blocking sets of graphs in $\mathcal{F}$ is bounded.*

Using this equivalence, we can exploit the fact that all minimal blocking sets of $\mathcal{F}$ are of bounded size, through the fact that the bridge-depth of $G \setminus X \in \mathcal{F}$ is small. This means that there is a bounded-depth elimination process to reduce $G \setminus X$ to the empty graph. We use this bounded-depth process in a technical kernelization algorithm following a recursive scheme, inspired by the earlier kernelization for the parameterization by distance to bounded tree-depth [5].

Let us now discuss the ideas behind the equivalence of Theorem 1.2. We prove that the bridge-depth of graphs in a minor-closed family $\mathcal{F}$ is upper-bounded in terms of the maximum size of minimal blocking sets for graphs in $\mathcal{F}$, by exploiting the Erdős-Pósa property in an interesting way. We analyze an elementary graph structure called *necklace of length t*, which is essentially the multigraph formed by a path of t double-edges. If a simple graph $G \in \mathcal{F}$ contains a necklace of length t as a minor, then there is a minor G' of G (which therefore also belongs to $\mathcal{F}$) that has a minimal blocking set of size $\Omega(t)$. Hence to show that bridge-depth is upper-bounded in terms of the size of minimal blocking sets of graphs in $\mathcal{F}$, it suffices to show that bridge-depth is upper-bounded by the maximum length of a necklace minor of graphs in $\mathcal{F}$. Since the definition of bridge-depth allows for the contraction of all bridges in a single step, it suffices to consider bridgeless graphs. Then we argue that in a bridgeless graph, any pair of maximum-length necklace minor models intersects at a vertex (cf. Lemma 4.6). By the Erdős-Pósa property, this implies that there is a constant-size vertex set that hits all maximum-length necklace minor models, and whose removal therefore strictly decreases the maximum length of a necklace minor. If the length of necklace minor models is bounded, then after a bounded number of steps of this process (interleaved with contracting all bridges) we reduce the maximum length of necklace minor models to zero, which is equivalent to breaking all cycles of the graph. At that point, the bridge-depth is one by definition, and we have obtained the desired upper-bound on the bridge-depth in terms of the length of the longest necklace minor, and therefore blocking set size.

For the other direction of Theorem 1.2, we prove (cf. Theorem 5.4) the tight bound that a minimal blocking set in a graph G has size at most $2^{\mathsf{bd}(G)}$. We use induction to prove this statement, together with an analysis of the structure of a tree of bridges whose removal decreases the bridge-depth. The fact that bipartite graphs have minimal blocking sets of size at most two, allows for an elegant induction step.

Related work. In a recent paper, Hols, Kratsch, and Pieterse [19] also analyze the role of blocking sets in the existence of polynomial kernels for structural parameterizations of Vertex Cover. Note that our paper is independent from, and orthogonal to [19]: we consider the setting of *deterministic* kernelization algorithms for parameterizations to *minor-closed* families $\mathcal{F}$, and obtain an exact characterization of which $\mathcal{F}$ allow for a polynomial kernelization. Hols et al. [19] consider hereditary families $\mathcal{F}$ and give kernelizations for several such parameterizations, without arriving at a complete characterization. Some of the randomized kernelizations they provide do not fit into our framework, but all the deterministic kernelizations they present are captured by Theorem 1.1. Another contribution of [19] is to prove that there is a class $\mathcal{F}$ with minimal blocking sets of size one where Vertex Cover cannot be solved in polynomial time. In particular, there is no polynomial kernel parameterized by the distance to this family $\mathcal{F}$, and thus bounded minimal blocking set size is not sufficient to get a polynomial kernel. This implies that our minor-closed assumption of Theorem 1.1 cannot be dropped.

We refer to the survey by Fellows et al. [13] for an overview of classic results and new research lines concerning kernelization for Vertex Cover. Additional relevant work includes the work by Kratsch [24] on a randomized polynomial kernel for a parameterization related to the difference between twice the cost of the linear-programming relaxation of Vertex Cover and the size of a maximum matching.

Organization. Preliminaries on graphs and complexity are presented in Section 2. Section 3 introduces bridge-depth and its properties. In Section 4 we prove one direction of Theorem 1.2, showing that large bridge-depth implies the existence of large minimal blocking sets. In Section 5 we handle the other direction, proving a tight upper-bound on the size of minimal blocking sets in terms of the bridge-depth. We discuss the kernelization algorithm exploiting bridge-depth in Section 6, with the technical content being available in the full version of the article [4] due to space limitations. We conclude the article in Section 7. The proofs of the results marked with "($\star$)" have been also deferred to the full version [4].

2 Preliminaries

Graphs. We use standard graph-theoretic notation, and we refer the reader to Diestel [11] for any undefined terms. All graphs we consider are finite and undirected. Graphs are simple, unless specifically stated otherwise. A graph G has vertex set $V(G)$ and edge set $E(G)$. Given a graph G and a subset $S \subseteq V(G)$, we say that S is *connected* if $G[S]$ is connected, and we use the shorthand $G \setminus S$ to denote $G[V(G) \setminus S]$. For a single vertex $v \in V(G)$, we use $G \setminus v$ as a shorthand for $G \setminus \{v\}$. Similarly, for a set of edges $T \subseteq E(G)$ we denote by $G \setminus T$ the graph on vertex set $V(G)$ with edge set $E(G) \setminus T$. A cycle on three vertices is called a *triangle*. For a positive integer i, we denote by $[i]$ the set of all integers j such that $1 \leq j \leq i$. Given $v \in V(G)$, we denote $N_G(v) = \{u \mid \{u, v\} \in E(G)\}$, $d_G(v) = |N_G(v)|$ and given $X \subseteq V(G)$, we denote $N_G(X) = \bigcup_{v \in X} N_G(v) \setminus X$. Given $X, Y \subseteq V(G)$, we denote by $N_G^Y(X) = N_G(X) \cap Y$. We may omit the subscript G when it is clear from the context. For distinct vertices u and v of a graph G, the graph G' obtained by *identifying* u and v is defined by removing vertices u and v from G, adding a new vertex uv with $N_{G'}(uv) = N_G(\{u, v\})$, and keeping the other vertices and edges unchanged. Given two adjacent vertices u and v, we define the *contraction* of the edge $\{u, v\}$ as the identification of u and v.

Given a graph G, we denote by $\alpha(G)$ the size of a maximum independent set in G, by $\#\textsc{cc}(G)$ the number of connected components of G, by $\mathsf{diam}(G)$ the diameter of G, and by $\Delta(G)$ the maximum degree of G. Given a graph G and a set $S \subseteq V(G)$, we say that S is a *blocking set* in G if $\alpha(G \setminus S) < \alpha(G)$. The maximum size of an inclusion-wise minimal blocking set of a graph G is denoted by $\mathsf{mbs}(G)$.

A graph H is a minor of graph G if H can be obtained from G by a sequence of edge deletions, edge contractions, and removals of isolated vertices. Let us also recall the definition of minor in the context of multigraphs. Let H be a loopless multigraph. An *H-model M* in a simple graph G is a collection $\{S_x^M \mid x \in V(H)\}$ of pairwise disjoint subsets of $V(G)$ such that $G[S_x^M]$ is connected for every $x \in V(H)$, and such that for every pair of distinct vertices x, y of H, the quantity $|\{\{u,v\} \in E(G) \mid u \in S_x^M, v \in S_y^M\}|$ is at least the number of edges in H between x and y. The vertex set $V(M)$ of M is the union of the vertex sets of the subgraphs in the collection. We say that a graph G contains a loopless multigraph H as a *minor* if G has an H-model.

For the following definitions, we refer the reader to [29] for more details and we only recall here some basic notations and facts. The *tree-depth* of a graph G, denoted by $\mathsf{td}(G)$, is defined recursively. The empty graph without vertices has tree-depth zero. The tree-depth of a disconnected graph is the maximum tree-depth of its connected components. Finally, if G is a nonempty connected graph then $\mathsf{td}(G) = 1 + \min_{v \in V(G)} \mathsf{td}(G \setminus v)$. Equivalent definitions exist in terms of the minimum height of a rooted forest whose closure is a supergraph of G. The tree-width of G is denoted $\mathsf{tw}(G)$ (cf. [2]).

Given a graph family $\mathcal{F}$, an *$\mathcal{F}$-modulator* in a graph G is a subset of vertices $X \subseteq V(G)$ such that $G \setminus X \in \mathcal{F}$. We denote by $\mathsf{dist\text{-}to\text{-}}\mathcal{F}(G)$ the size of a smallest $\mathcal{F}$-modulator in G. For a graph measure f that associates an integer with each graph, and an integer c, a *c-f-modulator* is a modulator to $\mathcal{F}_c^{\mathsf{f}} := \{G \mid \mathsf{f}(G) \leq c\}$. We denote by $c\text{-}\mathsf{f}\text{-}\mathsf{mod}(G) := \mathsf{dist\text{-}to\text{-}}\mathcal{F}_c^{\mathsf{f}}(G)$, that is, the size of a smallest c-f-modulator of G. Typical measures f that we consider here are tree-width, tree-depth, and bridge-depth. Notice that $0\text{-}\mathsf{tw}\text{-}\mathsf{mod}(G)$ corresponds to the minimum size of a vertex cover of G, and $1\text{-}\mathsf{tw}\text{-}\mathsf{mod}(G)$ corresponds to the minimum size of a feedback vertex set of G. Finally, IS (resp. VC) denotes the Maximum Independent Set (resp. Minimum Vertex Cover) problem.

Parameterized complexity. A *parameterized problem* is a language $L \subseteq \Sigma^* \times \mathbb{N}$, for some finite alphabet Σ. For an instance $(x,k) \in \Sigma^* \times \mathbb{N}$, the value k is called the *parameter*. For a computable function $g \colon \mathbb{N} \to \mathbb{N}$, a *kernelization algorithm* (or simply a *kernel*) for a parameterized problem L of *size* g is an algorithm A that given any instance (x,k) of L, runs in polynomial time and returns an instance (x',k') such that $(x,k) \in L \Leftrightarrow (x',k') \in L$ with $|x'|, k' \leq g(k)$. Consult [8,12,14,16,31] for background on parameterized complexity.

3 An introduction to bridge-depth

Let G be a graph. An edge $e \in E(G)$ is a *bridge* if its removal increases the number of connected components of G. We define $G_{\textsc{cb}}$ as the simple graph obtained from G by contracting all bridges of G (the order does not matter.) Observe that, as contracting an edge cannot create a new bridge, $G_{\textsc{cb}}$ has no bridges, implying that $(G_{\textsc{cb}})_{\textsc{cb}} = G_{\textsc{cb}}$. Given a subgraph T of a graph G, we say that T is a *tree of bridges* if T is a tree and, for every $e \in E(T)$, e is a bridge in G. Note that a single vertex is, by definition, a tree of bridges. Note also that with any vertex $v \in V(G_{\textsc{cb}})$ we can associate, in a bijective way, an inclusion-wise maximal tree of bridges T_v of G. The set $\{T_v \mid v \in V(G_{\textsc{cb}})\}$ is a minor model of $G_{\textsc{cb}}$ in G (a

G_{CB}-model, from now on). For any $u, v \in V(G_{\text{CB}})$ such that $\{u, v\} \in E(G_{\text{CB}})$, there is exactly one edge $\{u', v'\} \in E(G)$ with $u' \in T_u$ and $v' \in T_v$. The latter claim can be easily verified by supposing that there are two such edges, implying that some edge in T_u or T_v is involved in a cycle, which contradicts the fact that all the edges in T_u and T_v are bridges.

▶ **Definition 3.1.** *The* bridge-depth $\mathsf{bd}(G)$ *of a graph* G *is recursively defined as follows:*

- *If G is the empty graph without any vertices, then $\mathsf{bd}(G) = 0$.*
- *If G has $\ell > 1$ connected components $\{G_i \mid i \in [\ell]\}$, then $\mathsf{bd}(G) = \max_{i\in[\ell]} \mathsf{bd}(G_i)$.*
- *If G is connected, then $\mathsf{bd}(G) = 1 + \min_{v\in V(G_{\text{CB}})} \mathsf{bd}(G_{\text{CB}} \setminus v)$.*

Informally, bd behaves like tree-depth except that at each step of the recursive definition we are allowed to delete trees of bridges instead of just single vertices, as proved in Item 4 of the following proposition. The following properties of bridge-depth follow from the definitions in an elementary way, often exploiting the fact that if e is a bridge in G, then e is also a bridge in any minor of G that still contains e.

▶ **Proposition 3.2.** *For any graph G the following claims hold:*

1. *$\mathsf{bd}(G) = 1$ if and only if G is a forest with at least one vertex.*
2. *$\mathsf{bd}(G_{\text{CB}}) = \mathsf{bd}(G)$.*
3. *The parameter bd is minor-closed: if G' is a minor of G then $\mathsf{bd}(G') \le \mathsf{bd}(G)$.*
4. *If G is connected, then $\mathsf{bd}(G) = 1 + \min_T \mathsf{bd}(G \setminus V(T))$, where the minimum is taken over all trees of bridges T of G.*
5. *For any $X \subseteq V(G)$, we have $\mathsf{bd}(G) \le |X| + \mathsf{bd}(G \setminus X)$.*
6. *$\mathsf{tw}(G) \le \mathsf{bd}(G)$.*

Proof. The first item follows easily from the definition, while the second one uses that $(G_{\text{CB}})_{\text{CB}} = G_{\text{CB}}$.

Proof of 3: We prove the claim by induction on $|V(G)| + |E(G)|$. Suppose that G has multiple connected components $\{G_i \mid i \in [\#\text{CC}(G)]\}$, and let $\{G'_i \mid i \in [\#\text{CC}(G')]\}$ be the connected components of the minor G' of G. Then each connected component G'_j is a minor of some component G_i of G on fewer than $|V(G)|$ vertices, which gives $\mathsf{bd}(G'_j) \le \mathsf{bd}(G_i)$ by induction. Hence we have $\mathsf{bd}(G') = \max_{j\in[\#\text{CC}(G')]} \mathsf{bd}(G'_j) \le \max_{i\in[\#\text{CC}(G)]} \mathsf{bd}(G_i) = \mathsf{bd}(G)$.

We now deal with the case that G is connected. In general, if some graph G^* is a minor of G, then G^* is a minor of a graph G' obtained from G by removing an edge, contracting an edge, or removing an isolated vertex. Since G is assumed to be connected, the third case cannot occur here. Then by induction, we have $\mathsf{bd}(G^*) \le \mathsf{bd}(G')$, so it suffices to prove that $\mathsf{bd}(G') \le \mathsf{bd}(G)$ for any graph G' obtained by removing or contracting an edge. Let us first prove that if G'_{CB} is a minor of G_{CB}, then $\mathsf{bd}(G') \le \mathsf{bd}(G)$. Indeed, let $v^* \in V(G_{\text{CB}})$ such that $\mathsf{bd}(G) = 1 + \mathsf{bd}(G_{\text{CB}} \setminus v^*)$, and consider an arbitrary component G'_i of G'. Note that $(G'_i)_{\text{CB}}$ is a component of $(G')_{\text{CB}}$, and therefore a minor of G_{CB} by hypothesis. If $(G'_i)_{\text{CB}}$ is a minor of the graph $G_{\text{CB}} \setminus v^*$, then by induction and Item 2 we have $\mathsf{bd}(G'_i) = \mathsf{bd}((G'_i)_{\text{CB}}) \le \mathsf{bd}(G_{\text{CB}} \setminus v^*) < \mathsf{bd}(G)$. Otherwise, any minor model $\{S_x \mid x \in V((G'_i)_{\text{CB}})\}$ of $(G'_i)_{\text{CB}}$ in G_{CB} contains a branch set S_{x^*} with $v^* \in S_{x^*}$. But then $\mathsf{bd}((G'_i)_{\text{CB}}) \le 1 + \mathsf{bd}((G'_i)_{\text{CB}} \setminus x^*)$ by definition, and $(G'_i)_{\text{CB}} \setminus x^*$ is a minor of $G_{\text{CB}} \setminus v^*$, and therefore has bridge-depth at most $\mathsf{bd}(G) - 1$, so that $\mathsf{bd}((G'_i)_{\text{CB}}) \le \mathsf{bd}(G)$. Hence for each component G'_i of G' we have $\mathsf{bd}(G'_i) = \mathsf{bd}((G'_i)_{\text{CB}}) \le \mathsf{bd}(G)$, implying $\mathsf{bd}(G') \le \mathsf{bd}(G)$.

Thus, it only remains to prove that G'_{CB} is a minor of G_{CB}. Let us first assume that G' is obtained from G by removing an edge e. Let $\{T_v \mid v \in V(G_{\text{CB}})\}$ be the G_{CB}-model in G given by the trees of bridges. If e is not a bridge, then e is an edge between T_u and T_v for some vertices $u, v \in V(G_{\text{CB}})$. To obtain G'_{CB} as a minor, we start from G_{CB}, remove edge $\{u, v\}$, and for any edge e' between $T_{u'}$ and $T_{v'}$ (for any $u', v' \in V(G_{\text{CB}})$) that has become a bridge in G' because of the removal of e, we contract $\{u', v'\}$. This implies that G'_{CB} is a minor of G_{CB}. Otherwise, if e is a bridge, then there exists $u \in V(G_{\text{CB}})$ such that $e \in E(T_u)$, and G' has two connected components G'_1 and G'_2. To obtain $(G'_i)_{\text{CB}}$ as a minor, for $i \in [2]$, we start from G_{CB} and remove any vertex v such that $T_v \cap V(G'_i) = \emptyset$ (notice that u appears both in $(G'_1)_{\text{CB}}$ and $(G'_2)_{\text{CB}}$). Thus, both $(G'_1)_{\text{CB}}$ and $(G'_2)_{\text{CB}}$ are minors of G_{CB}, hence G'_{CB} as well. The case where G' is obtained from G by contracting an edge e can be proved using similar but simpler arguments. Indeed, if e is a bridge in G, then we have that $G'_{\text{CB}} = G_{\text{CB}}$, and if it is not, if suffices to contract in G_{CB} the edge $\{u, v\}$ with $u, v \in V(G_{\text{CB}})$ such that e is an edge between T_u and T_v.

Proof of 4: Let $v \in V(G_{\text{CB}})$ and T_v be its associated tree of bridges in G. Observe first that we may have $(G \setminus V(T_v))_{\text{CB}} \neq G_{\text{CB}} \setminus v$. Indeed, if for example we consider G composed of two vertex-disjoint triangles $\{a, b, c\}$, $\{a', b', c'\}$ and an edge $e = \{a, a'\}$, and if we consider $T_v = \{e\}$, then $G_{\text{CB}} \setminus v$ is composed of two disjoint edges, whereas $(G - V(T_v))_{\text{CB}}$ is composed of two isolated vertices. However, it is easy to verify that $(G \setminus V(T_v))_{\text{CB}} = (G_{\text{CB}} \setminus v)_{\text{CB}}$. Let us now prove that $\min_T \mathsf{bd}(G \setminus V(T)) \leq \min_{v \in V(G_{\text{CB}})} \mathsf{bd}(G_{\text{CB}} \setminus v)$. Let v^* be a vertex minimizing $\mathsf{bd}(G_{\text{CB}} \setminus v)$. We have $\min_T \mathsf{bd}(G \setminus V(T)) \leq \mathsf{bd}(G \setminus V(T_{v^*})) = \mathsf{bd}((G \setminus V(T_{v^*})_{\text{CB}})$ using Item 2 in the last equality, and $\mathsf{bd}((G \setminus V(T_{v^*})_{\text{CB}}) = \mathsf{bd}((G_{\text{CB}} \setminus v^*)_{\text{CB}}) = \mathsf{bd}(G_{\text{CB}} \setminus v^*)$ using again Item 2.

For the other inequality, let T^0 be a tree of bridges that minimizes $\mathsf{bd}(G \setminus V(T))$. If T^0 is not inclusion-wise maximal, let T^* be any inclusion-wise maximal tree of bridges containing T^0. Note that as $G \setminus V(T^*)$ is a subgraph of $G \setminus V(T^0)$, by Item 3 we get that $\mathsf{bd}(G \setminus V(T^*)) \leq \mathsf{bd}(G \setminus V(T^0))$, implying that T^* also minimizes $\mathsf{bd}(G \setminus V(T))$. Let $v^* \in V(G_{\text{CB}})$ such that $T_{v^*} = T^*$. We have $\min_{v \in V(G_{\text{CB}})} \mathsf{bd}(G_{\text{CB}} \setminus v) \leq \mathsf{bd}(G_{\text{CB}} \setminus v^*) = \mathsf{bd}((G \setminus T_{v^*})_{\text{CB}}) = \mathsf{bd}(G \setminus T_{v^*})$.

Proof of 5: We use induction on $|X|$, the base case $X = \emptyset$ being trivial. For the induction step, pick an arbitrary $v \in X$, let $X' := X \setminus \{v\}$, and $G' := G \setminus X'$. By induction we have $\mathsf{bd}(G) \leq |X'| + \mathsf{bd}(G')$. Let G'_i be the connected component of G' containing v. Using v as a singleton tree of bridges in G'_i, Item 4 shows that $\mathsf{bd}(G'_i) \leq 1 + \mathsf{bd}(G'_i \setminus v) \leq 1 + \mathsf{bd}(G' \setminus v)$. Since all other components G'_j of G' also occur as components of $G' \setminus v$, it follows that $\mathsf{bd}(G'_j) \leq \mathsf{bd}(G' \setminus v)$, implying $\mathsf{bd}(G') \leq 1 + \mathsf{bd}(G' \setminus v) = 1 + \mathsf{bd}(G \setminus X)$ since $G' \setminus v = G \setminus X$. Hence $\mathsf{bd}(G) \leq |X'| + 1 + \mathsf{bd}(G \setminus X)$.

Proof of 6: We use induction on $|V(G)|$; the base case follows directly from the definitions. It is well-known (cf. [2, Lemma 6]) that the tree-width of G is the maximum tree-width of its biconnected components. Hence it suffices to prove that for an arbitrary biconnected component G' of G, we have $\mathsf{tw}(G') \leq \mathsf{bd}(G')$. If G' consists of a single edge, then $\mathsf{tw}(G') = \mathsf{bd}(G') = 1$. Otherwise, G' is a connected bridgeless graph. This implies $(G')_{\text{CB}} = G'$, so by Definition 3.1 there is a vertex $v \in V(G')$ such that $\mathsf{bd}(G') = 1 + \mathsf{bd}(G' \setminus v)$. Since G' is a minor of G, we have $\mathsf{bd}(G') \leq \mathsf{bd}(G)$ by Item 3. By induction, the tree-width of $G' \setminus v$ is at most $\mathsf{bd}(G' \setminus v) \leq \mathsf{bd}(G) - 1$. Adding vertex v to all bags of a tree decomposition of this width, gives a valid tree decomposition of G' of width at most $\mathsf{bd}(G' \setminus v) + 1 \leq \mathsf{bd}(G')$. Hence $\mathsf{tw}(G') \leq \mathsf{bd}(G')$ for all biconnected components of G. ◀

A $(c+1) \times (c+1)$-grid is a planar graph of tree-width exactly $c+1$ [2, Cor. 89], which implies by Item 6 of Proposition 3.2 that its bridge-depth is larger than c. This gives the following consequence of Proposition 3.2, which will be useful when invoking algorithmic meta-theorems.

▶ **Observation 3.3.** *For each $c \in \mathbb{N}$, the graphs of bridge-depth at most c form a minor-closed family that excludes a planar graph. By the Graph Minor Theorem [35], there is a finite set of forbidden minors $\mathcal{H}_c$ such that $\mathsf{bd}(G) \leq c$ if and only if G excludes all graphs of $\mathcal{H}_c$ as a minor. The set $\mathcal{H}_c$ contains a planar graph, since some planar graphs have bridge-depth $> c$.*

Observation 3.3, together with known results on minor testing, imply the following.

▶ **Proposition 3.4** (Follows from [1, Thm. 7.1])**.** *For each constant $c \in \mathbb{N}$, there is a linear-time algorithm to test whether the bridge-depth of a given graph G is at most c.*

Fomin et al. [15, Thm. 1.3] gave a generic approximation algorithm for finding a small vertex set that hits forbidden minors from a finite forbidden set containing a planar graph. By Observation 3.3, deleting vertices to obtain a graph of bounded bridge-depth fits into their framework.

▶ **Proposition 3.5** (Follows from [15, Thm. 1.3])**.** *For each fixed $c \in \mathbb{N}$ there is a polynomial-time algorithm that, given a graph G, outputs a set $X \subseteq V(G)$ such that $\mathsf{bd}(G \setminus X) \leq c$ and $|X| \leq \mathcal{O}(|X_{\text{opt}}| \log^{2/3} |X_{\text{opt}}|)$, where $|X_{\text{opt}}|$ is the minimum size of such a set.*

The following concept will be crucial to facilitate a recursive approach for reducing graphs of bounded bridge-depth.

▶ **Definition 3.6.** *A* lowering tree *T of a graph G is a tree of bridges (possibly consisting of a single vertex and no bridges) such that $\mathsf{bd}(G \setminus V(T)) = \mathsf{bd}(G) - 1$.*

Item 4 of Proposition 3.2 implies that any connected graph G has a lowering tree.

▶ **Proposition 3.7.** *For each fixed $c \in \mathbb{N}$ there is an algorithm that, given a connected graph G on n vertices of bridge-depth c, computes a lowering tree in $\mathcal{O}(n^2)$ time.*

Proof. Given G, we compute its decomposition into biconnected components, which can be done in linear time taking into account that having bounded bridge-depth implies a linear number of edges [20]. From this decomposition, it is straightforward to identify the inclusion-maximal trees of bridges in G. For each tree of bridges T in G, we can test whether $\mathsf{bd}(G \setminus V(T)) < c = \mathsf{bd}(G)$ in linear time using Proposition 3.4, and we output T if this is the case. By Proposition 3.2, such a tree T exists. Since G is decomposed into at most n trees of bridges, and we need a linear-time computation for each T, this results in an $\mathcal{O}(n^2)$-time algorithm. ◀

4 Bounded minimal blocking sets imply bounded bridge-depth

The goal of this section is to prove one direction of Theorem 1.2, showing that if $\mathcal{F}$ has bounded-size minimal blocking sets, then $\mathcal{F}$ has bounded bridge-depth. As explained in Section 1, we prove this via the intermediate structure of *necklace minors* and show that the bridge-depth of a graph G can be upper-bounded in terms of the longest necklace contained in it as a minor.

This result can be seen as an analog to the fact that the tree-depth of a graph can be bounded in terms of the length of the longest simple path it contains (as a subgraph or as a minor, which is equivalent for paths). A classical proof of this fact (see [29]) is to consider a depth-first search tree of G, bounding the tree-depth of G by the depth of this tree. However, it does not seem immediate to find a similar bound for bridge-depth.

We therefore follow another approach, inspired by the following alternative proof that the tree-depth is upper-bounded by the length of the longest path (which gives a worse bound). Observe that in a connected graph G, any two longest paths intersect at a vertex. (If they did not, one could combine them to make an even longer path.) Given a connected graph G whose longest path has t vertices, we can bound its tree-depth by $f(t) := \sum_{i=1}^{t} i$ as follows. Let P be a longest path in G. Then the longest path in $G \setminus V(P)$ has strictly fewer than t vertices, and by induction the tree-depth of $G \setminus V(P)$ is at most $f(t-1)$. From the definition of tree-depth, it follows that the tree-depth of G is at most $|V(P)| = t$ larger than that of $G \setminus V(P)$, so the tree-depth of G is at most $f(t)$.

In the case of bridge-depth, where paths are replaced with necklaces contained as minors, we cannot afford to remove the entire set of vertices of the corresponding model of a longest necklace, as the size of this set cannot be bounded in terms of the length t of the necklace. To overcome this problem, we will prove in Lemma 4.6, similarly to the case of paths, that there cannot be two vertex-disjoint longest necklaces. Then we resort to the Erdős-Pósa property, which gives us a set of vertices of size $f(t)$ whose removal decreases the maximum length of a longest necklace. We now formalize these ideas.

▶ **Definition 4.1.** *For $t \in \mathbb{N}$, the* necklace of length t*, denoted by N_t, is the multigraph having $t+1$ vertices $\{v_i \mid i \in [t+1]\}$ and two parallel edges between v_i and v_{i+1} for $i \in [t]$.*

▶ **Observation 4.2.** *A simple graph G contains N_t as a minor if and only if G contains $t+1$ vertex-disjoint sets $S_i \subseteq V(G)$ such that each S_i is connected and, for $i \in [t]$, there are at least two edges between S_i and S_{i+1}.*

▶ **Definition 4.3.** *The* necklace-minor length *of a graph G, denoted by $\mathsf{nm}(G)$, is the largest length of a necklace contained in G as a minor, or zero if G contains no such minor.*

We need to introduce the Erdős-Pósa property for packing and covering minor models. Let $\mathcal{F}$ be a finite collection of simple graphs. An *$\mathcal{F}$-model* is an H-model for some $H \in \mathcal{F}$. Two $\mathcal{F}$-models M_1 and M_2 are *disjoint* if $V(M_1) \cap V(M_2) = \emptyset$. Let $\nu_{\mathcal{F}}(G)$ be the maximum cardinality of a packing of pairwise disjoint $\mathcal{F}$-models in G, and let $\tau_{\mathcal{F}}(G)$ be the minimum size of a subset $X \subseteq V(G)$ such that $G \setminus X$ has no $\mathcal{F}$-model. Clearly, $\nu_{\mathcal{F}}(G) \leq \tau_{\mathcal{F}}(G)$. We say that the *Erdős-Pósa property* holds for $\mathcal{F}$-models if there exists a bounding function $f\colon \mathbb{N} \to \mathbb{N}$ such that, for every graph G, $\tau_{\mathcal{F}}(G) \leq f(\nu_{\mathcal{F}}(G))$.

In the case where $\mathcal{F} = \{H\}$ contains a single connected graph H, Robertson and Seymour [34] proved the following result.

▶ **Theorem 4.4** (Robertson and Seymour [34])**.** *Let H be a connected graph. The Erdős-Pósa property holds for H-models if and only if H is planar.*

It is worth mentioning that a tight bounding function when H is planar has been recently obtained by van Batenburg et al. [36]. Theorem 4.4 easily implies the following corollary.

▶ **Corollary 4.5.** *For every $t \geq 1$, the Erdős-Pósa property holds for N_t-models.*

Proof. For $t \geq 1$, let $\mathcal{F}_t$ be the set containing all minor-minimal *simple* graphs that contain the necklace N_t as a minor. By definition, a simple graph G contains an N_t-model if and only if it contains an $\mathcal{F}_t$-model. Clearly, all the graphs in $\mathcal{F}_t$ are connected and planar, and it is easy to see that $|\mathcal{F}_t|$ is bounded by a function of t. For each $F \in \mathcal{F}_t$, by Theorem 4.4 there is a function f_F such that if G does not contain k vertex-disjoint models of F, then all the F-models of G can be hit by at most $f_F(k)$ vertices. This implies that if G does not contain k models of any graph in $\mathcal{F}_t$, then the union of all hitting sets has size bounded by $\sum_{F \in \mathcal{F}_t} f_F(k)$, and since $\mathcal{F}_t$ is finite this is a valid bounding function for N_t-models. ◀

We denote by f_{N_t} the bounding function for N_t-models given by Corollary 4.5. In a connected bridgeless graph, each pair of maximum-length necklace models intersect at a vertex:

▶ **Lemma 4.6.** *If G is a connected bridgeless simple graph with $\mathsf{nm}(G) = t$, then $\nu_{N_t}(G) = 1$.*

Proof. Suppose for contradiction that G contains two disjoint models M^1 and M^2 of N_t. For $i \in [t+1]$ and $\ell \in [2]$, let S_i^ℓ be the vertex set of M^ℓ given by Observation 4.2. Note that these $2t+2$ subsets of vertices of G are pairwise disjoint, and that for any $i \in [t]$, there are at least two edges between S_i^ℓ and S_{i+1}^ℓ. Since G is bridgeless and connected, it is 2-edge-connected and by Menger's theorem [11, § 3.3] G contains two edge-disjoint paths between any pair of vertices. Pick two arbitrary vertices $x_1 \in M^1, x_2 \in M^2$, and let P^1, P^2 be two edge-disjoint paths between them. Consider the subpath Q^ℓ of P^ℓ between the last vertex of M^1 that is visited, until the first vertex of M^2. Let $Q^\ell = (v_1^\ell, \dots, v_{q_\ell}^\ell)$ where $v_1^\ell \in M^1$ and $v_{q_\ell}^\ell \in M^2$. Let a_ℓ such that $v_1^\ell \in S_{a_\ell}^1$ and b_ℓ such that $v_{q_\ell}^\ell \in S_{b_\ell}^2$.

Let us first show that if t is odd, then we can use Q^1 to find an $N_{t'}$-model M' for some $t' > t$ by "gluing" M^1 and M^2, leading to a contradiction. Let $S = S_{a_1}^1 \cup V(Q^1) \cup S_{b_1}^2$. If $a_1 > \frac{t+1}{2}$ define $A = \{S_1^1, \dots, S_{a_1-1}^1\}$, and otherwise define $A = \{S_{a_1+1}^1, \dots, S_{t+1}^1\}$. Similarly, if $b_1 > \frac{t+1}{2}$ define $B = \{S_1^2, \dots, S_{b_1-1}^2\}$, and otherwise define $B = \{S_{b_1+1}^2, \dots, S_{t+1}^2\}$. Note that the sets A, S, B are pairwise disjoint. Since t is odd, it can be easily checked that $M' = A \cup \{S\} \cup B$ is an $N_{t'}$-model in G for some $t' > t$; see Figure 1(a) for an illustration.

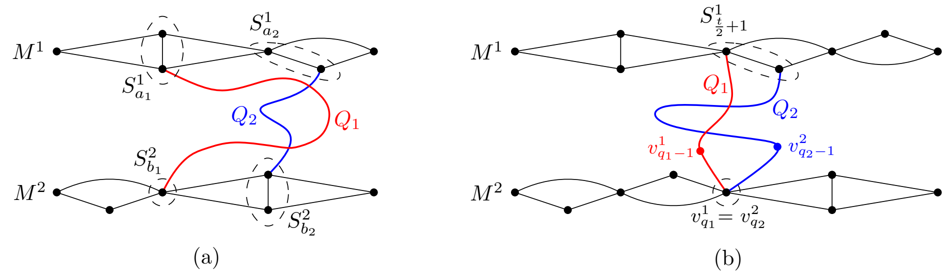


Figure 1 (a) Example with $t = 3$ and $a_1 = b_1 = 2$. (b) Example with $t = 4$.

Let us now consider the case where t is even. Note first that if there exists $\ell \in [2]$ such that $a_\ell \neq \frac{t}{2} + 1$ or $b_\ell \neq \frac{t}{2} + 1$, then we can use Q^ℓ to find an $N_{t'}$-model for some $t' > t$ as in the previous case. Hence, it only remains to consider the case where $a_1 = b_1 = a_2 = b_2 = \frac{t}{2} + 1$, meaning that Q^1 and Q^2 are two edge-disjoint paths, both between $S^1_{\frac{t}{2}+1}$ and $S^2_{\frac{t}{2}+1}$. Let $A = \{S_1^1, \dots, S^1_{\frac{t}{2}}\}$, $B = \{S_1^2, \dots, S^2_{\frac{t}{2}}\}$, and $S = S^1_{a_1} \cup (V(Q^1) \setminus \{v^1_{q_1}\}) \cup (V(Q^2) \setminus \{v^2_{q_2}\})$. We claim that $M' = A \cup \{S, S^2_{\frac{t}{2}+1}\} \cup B$ is an N_{t+1}-model. Indeed, note in particular there are two edges between S and $S^2_{\frac{t}{2}+1}$ as we cannot have $v^1_{q_1-1} = v^2_{q_2-1}$ and $v^1_{q_1} = v^2_{q_2}$ because Q^1 and Q^2 are edge-disjoint and G is a simple graph; see Figure 1(b) for an illustration. ◀

By combining Corollary 4.5 with Lemma 4.6 we easily get the following corollary.

▶ **Corollary 4.7.** *Let G be a connected bridgeless graph and $t = \mathsf{nm}(G)$. Then G contains a set of vertices X with $|X| \leq f_{N_t}(1)$ such that $\mathsf{nm}(G \setminus X) < \mathsf{nm}(G)$, where $f_{N_t} : \mathbb{N} \to \mathbb{N}$ is the bounding function given by Corollary 4.5.*

Proof. By Lemma 4.6, it follows that $\nu_{N_t}(G) = 1$, and therefore by Corollary 4.5

$$\tau_{N_t}(G) \;\le\; f_{N_t}(\nu_{N_t}(G)) \;=\; f_{N_t}(1).$$

Thus, there exists a set $X \subseteq V(G)$ with $|X| \le f_{N_t}(1)$ such that $G \setminus X$ has no N_t-model, implying that $\mathsf{nm}(G \setminus X) < t$. ◀

We are finally in position to prove the following theorem.

▶ **Theorem 4.8.** *There is a function $f\colon \mathbb{N} \to \mathbb{N}$ such that $\mathsf{bd}(G) \le f(\mathsf{nm}(G))$ for all graphs G.*

Proof. We prove the statement by induction on $\mathsf{nm}(G)$, for the function f defined by $f(t) := 1 + \sum_{i=1}^{t} f_{N_i}(1)$. If $\mathsf{nm}(G) = 0$, then G is a forest, and by definition of bridge-depth we get $\mathsf{bd}(G) = 1 = f(0)$. Suppose now that $\mathsf{nm}(G) = t$ with $t > 0$.

Consider the case that G is connected. Then G_{CB} is also connected and has no bridge, and thus we can apply Corollary 4.7 and get a set $X \subseteq V(G_{\text{CB}})$ with $|X| \le f_{N_t}(1)$ such that $\mathsf{nm}(G') < t$, where $G' = G_{\text{CB}} \setminus X$. By Item 5 of Proposition 3.2, we get that $\mathsf{bd}(G) = \mathsf{bd}(G_{\text{CB}}) \le |X| + \mathsf{bd}(G')$. Let $G'_1, \ldots, G'_\ell$ be the connected components of G'. As $\mathsf{nm}(G') < t$, we get that $\mathsf{nm}(G'_i) < t$ for every $i \in [\ell]$. Then, by induction hypothesis it follows that, for every $i \in [\ell]$, $\mathsf{bd}(G'_i) \le f(t-1) = 1 + \sum_{i=1}^{t-1} f_{N_i}(1)$. Thus, as $\mathsf{bd}(G') = \max_{i\in[\ell]} \mathsf{bd}(G'_i) \le 1 + \sum_{i=1}^{t-1} f_{N_i}(1)$, we get that

$$\mathsf{bd}(G) \;\le\; |X| + \mathsf{bd}(G') \;\le\; f_{N_t}(1) + 1 + \sum_{i=1}^{t-1} f_{N_i}(1) \;=\; 1 + \sum_{i=1}^{t} f_{N_i}(1) \;=\; f(\mathsf{nm}(G)).$$

Finally, if G is disconnected, let $G_1, \ldots, G_\ell$ be its connected components, and note that $\mathsf{bd}(G) = \max_{i\in[\ell]} \mathsf{bd}(G_i)$. Since for every $i \in [\ell]$ it holds that $\mathsf{nm}(G_i) \le \mathsf{nm}(G)$, and since the function f is non-decreasing, by applying the above case to each connected component of G we get that

$$\mathsf{bd}(G) \;=\; \max_{i\in[\ell]} \mathsf{bd}(G_i) \;\le\; \max_{i\in[\ell]} f(\mathsf{nm}(G_i)) \;\le\; \max_{i\in[\ell]} f(\mathsf{nm}(G)) \;=\; f(\mathsf{nm}(G)).$$ ◀

Now that we established a relation between bridge-depth and necklace minors, our next step is to relate necklace minors to blocking sets. For this purpose, we use the known triangle-path gadget.

▶ **Definition 4.9.** *A* triangle-path of length t *is the graph consisting of t vertex-disjoint triangles, with vertex sets $\{\{a_i, b_i, c_i\} \mid i \in [t]\}$, together with the $t-1$ edges $\{\{b_i, a_{i+1}\} \mid i \in [t-1]\}$. The* triangle-path-minor length *of a graph G, denoted by $\mathsf{tpm}(G)$, is the largest length of a triangle-path contained in G as a minor, or zero if no such minor exists.*

A slight variation of this gadget was used by Fomin and Strømme [17, Def. 6]. We observe the following (cf. [17, Obs. 3–5]).

▶ **Observation 4.10.** *Let G be a triangle-path of length $t \ge 2$. Then $\mathsf{mbs}(G) \ge t+2$, as $\{a_1, c_1\} \cup \{b_t, c_t\} \cup \{c_i \mid i \in [2, t-1]\}$ is a minimal blocking set.*

▶ **Lemma 4.11.** *For any graph G, $\mathsf{tpm}(G) \ge \lfloor \frac{\mathsf{nm}(G)+1}{2} \rfloor$.*

Proof. Let $t = \mathsf{nm}(G)$, and let $\{S_i \mid i \in [t+1]\}$ be an N_t-model in G. Let $i \in [\lfloor \frac{t+1}{2} \rfloor]$ and let $e_1 = \{u_1, v_1\}$ and $e_2 = \{u_2, v_2\}$ be the two edges between S_{2i-1} and S_{2i}, with $u_\ell \in S_{2i-1}$ and $v_\ell \in S_{2i}$. If $u_1 \neq u_2$ then there is a partition A_1, A_2 of S_{2i-1} such that $u_i \in A_i$ and A_i is connected for $i \in [2]$, and we define $L_i = \{A_1, A_2\}$, $R_i = \{S_{2i}\}$. Otherwise, if $u_1 = u_2$, then necessarily $v_1 \neq v_2$, and we define symmetrically $L_i = \{S_{2i-1}\}$ and $R_i = \{A_1, A_2\}$. In both cases we get that $L_i \cup R_i$ is a model of a triangle, and moreover there is an edge between a vertex in R_i and a vertex in L_{i+1} for every $i \in [\lfloor \frac{t+1}{2} \rfloor - 1]$. This implies that $\bigcup_{i \in [\lfloor \frac{t+1}{2} \rfloor]} (L_i \cup R_i)$ is a model of a triangle-path of length $\lfloor \frac{t+1}{2} \rfloor$ in G. ◀

▶ **Corollary 4.12.** *There is a function $g \colon \mathbb{N} \to \mathbb{N}$ such that $\mathsf{bd}(G) \leq g(\mathsf{tpm}(G))$ for all graphs G.*

Proof. By Lemma 4.11, we have that $\mathsf{tpm}(G) \geq \mathsf{nm}(G)/2$. By letting $g(t) := f(2t)$, where f is the function given by Theorem 4.8, we get the desired result. ◀

▶ **Corollary 4.13.** *Let $\mathcal{F}$ be a minor-closed family of graphs. If $\mathcal{F}$ has unbounded bridge-depth then it contains the family $\mathcal{F}^{\mathsf{tp}}$ of all triangle-paths.*

Using this corollary, we can prove one direction of Theorem 1.2.

▶ **Theorem 4.14.** *Let $\mathcal{F}$ be a minor-closed family of graphs of unbounded bridge-depth. Then there are graphs in $\mathcal{F}$ that have arbitrarily large minimal blocking sets.*

Proof. By Corollary 4.13, $\mathcal{F}$ contains all triangle-paths. Since a triangle-path of length t contains a minimal blocking set of size $t + 2$ by Observation 4.10, the theorem follows. ◀

Theorem 4.14 is phrased for graph families, rather than individual graphs. There is no function h such that $\mathsf{bd}(G) \leq h(\mathsf{mbs}(G))$ for all G: a bipartite grid graph can have arbitrarily large tree-width and therefore bridge-depth, but its minimal blocking sets have size at most two (cf. Lemma 5.2).

5 Bounded bridge-depth implies bounded-size blocking sets

In this section we prove the other direction of Theorem 1.2: minimal blocking sets in a graph G have size at most $2^{\mathsf{bd}(G)}$. We need the following consequence of Kőnig's theorem.

▶ **Lemma 5.1.** *Let G be a bipartite graph and let M be a maximum matching in G. Every maximum independent set of G contains all vertices that are not saturated by M, and exactly one endpoint of each edge in M.*

Proof. Consider a maximum independent set S in G. Then $\overline{S} := V(G) \setminus S$ is a minimum vertex cover of G. By Kőnig's theorem (cf. [11, Thm. 2.1.1]) we have $|\overline{S}| = |M|$. Since $\overline{S}$ is a vertex cover it contains at least one endpoint of each edge of M; since $|\overline{S}| = |M|$ it contains exactly one endpoint of each edge of M, and no other vertices of G. So the complement S contains all vertices that are not saturated by M, and exactly one endpoint of each edge in M. ◀

The next lemma shows that minimal blocking sets in a bipartite graph have at most two vertices. This was known before, see for example [18, Thm. 14]. Our self-contained proof highlights an additional property of such minimal blocking sets: the two vertices of minimal blocking sets of size two belong to opposite partite sets. This will be crucial later on.

▶ **Lemma 5.2.** *Let G be a bipartite graph with partite sets A and B. If $Y \subseteq V(G)$ is a blocking set in G, then there is a blocking set $Y' \subseteq Y$ in G such that one of the following holds:*

- *$|Y'| = 1$, or*
- *$Y' = \{a, b\}$ for some $a \in A$ and $b \in B$.*

Proof. Let M be a maximum matching in G, let $V(M)$ be the saturated vertices, and let $U := V(G) \setminus V(M)$ be the unsaturated vertices. Let $R_{A \cap U}$ be the vertices that can be reached by an M-alternating path from $A \cap U$ (which necessarily starts with a non-matching edge). Let $R_{B \cap Y}$ be the vertices that can be reached by an M-alternating path that starts with a *matching edge* from a vertex of $B \cap Y$. Note that both types of alternating paths move from A to B over non-matching edges, and move from B to A over matching edges.

We first deal with some cases in which we easily obtain a blocking set Y' as desired.

Case 1: $A \cap Y \cap R_{A \cap U} \neq \emptyset$. Let $a \in A \cap Y \cap R_{A \cap U}$. Then $a \in A$ can be reached by an M-alternating path P that starts in an unsaturated vertex in the same partite set, implying that P has even length and ends with a matching edge into a. Hence $M' := M \oplus E(P)$, where $\oplus$ denotes the symmetric difference, is a new maximum matching, and it does not saturate $a \in A \cap Y$. Lemma 5.1 applied to M' implies that all maximum independent sets of G contain a, showing that $Y' := \{a\}$ is a blocking set of size one.

Case 2: $B \cap U \cap R_{B \cap Y} \neq \emptyset$. By definition, some $u \in B \cap U$ can be reached by an M-alternating path P that starts in some vertex $b \in B \cap Y$ that belongs to the same partite set. Similarly as in the previous case, $M' := M \oplus E(P)$ is a new maximum matching that does not saturate b, so by Lemma 5.1 applied to M' we conclude that $Y' := \{b\}$ is a blocking set of size one.

Case 3: $A \cap Y \cap R_{B \cap Y} \neq \emptyset$. By definition, some $a \in A \cap Y$ is reachable by an M-alternating path P from some $b \in B \cap Y$, and P starts with a matching edge. Since it ends in the other partite set, it ends with a matching edge as well; hence both a and b are saturated. We claim that $Y' := \{a, b\}$ is a blocking set in G, as desired. Let $a = a_1, b_1, \ldots, a_k, b_k = b$ be the vertices on P, so that $\{a_i, b_i\} \in M$ for all $i \in [k]$ and $\{b_i, a_{i+1}\} \in E(G) \setminus M$ for $i \in [k-1]$. By Lemma 5.1, a maximum independent set in G contains one endpoint of each of the edges $\{a_i, b_i\} \in M$. A maximum independent set avoiding a_1 therefore has to contain b_1, preventing it from containing a_2, forcing it to contain b_2, and so on. Hence a maximum independent set avoiding a_1 contains b_k, proving that $Y' := \{a, b\} = \{a_1, b_k\}$ is a blocking set in G.

Case 4: $B \cap U \cap R_{A \cap U} \neq \emptyset$. Then some unsaturated vertex of A can reach an unsaturated vertex of B by an M-alternating path P. But then M is not a maximum matching since $M \oplus E(P)$ is larger; a contradiction. Hence this case cannot occur.

Assume now that none of the cases above hold. We will conclude the proof of the lemma by deriving a contradiction. Let $R := R_{A \cap U} \cup R_{B \cap Y}$. The following will be useful.

▷ Claim 5.3. If $a \in A \cap R$ and $\{a, b\} \in E(G)$, then $b \in B \cap R$.

Proof. By definition, $a \in A \cap R$ implies a is reachable by some M-alternating path P that moves to A over matching edges and moves to B over non-matching edges, such that P starts in a vertex $v \in (A \cup U) \cup (B \cap Y)$. But then b is also reachable by such an alternating path from v: if $\{a, b\} \in M$ then, since P ends at a, edge $\{a, b\}$ must be the last edge of P, so a prefix of P is an M-alternating path reaching b; if $\{a, b\} \notin M$ then appending $\{a, b\}$ to P yields such an M-alternating path. Hence $b \in R$, and $b \in B$ follows since G is bipartite. ◁

Now consider the following set: $S := (A \cap R) \cup (B \setminus R)$.

We will prove that S is a maximum independent set of G disjoint from Y, contradicting the assumption that Y is a blocking set. To see that S is indeed an independent set, consider any vertex from $A \cap S$, which belongs to $A \cap R$. By Claim 5.3 all neighbors of a belong to $B \cap R$, and are therefore not contained in S. Hence S is indeed an independent set. To see that it is maximum, by Lemma 5.1 it suffices to argue it contains all of U and one endpoint of each edge in M.

To see that S contains all vertices of $A \cap U$, note that all such vertices are trivially in $R_{A \cap U}$ and therefore in R, implying their presence in $A \cap R$ and therefore in S. To see that S contains all vertices of $B \cap U$, it suffices to show that $B \cap U \cap R = \emptyset$, which follows from the fact that neither Case 2 nor Case 4 is applicable. Hence S contains all vertices of U.

To see that S contains an endpoint of each edge of M, let $\{a, b\} \in M$ be arbitrary with $a \in A$ and $b \in B$. If $b \notin R$ then clearly $b \in S$, as desired. If $b \in R$, then this is witnessed by an alternating path P that reaches b and ends with a non-matching edge. Extending P with the edge $\{a, b\} \in M$ then shows that $a \in R$, so that $a \in A \cap R$ is an endpoint of the edge contained in S.

Hence S is a maximum independent set in G. Since Case 1 and Case 3 do not apply, it follows that $A \cap R \cap Y = \emptyset$, so that $S \cap A$ contains no vertex from Y. Since all vertices of $B \cap Y$ are trivially in $R_{B \cap Y}$ and therefore in R, it follows that $B \setminus R$ contains no vertex from Y. Hence S is a maximum independent set in G disjoint from Y, contradicting the assumption that Y is a blocking set. ◀

We will use Lemma 5.2 to power the induction step in the proof of the next theorem, which gives the desired upper-bound on the size of minimal blocking sets in terms of bridge-depth. The main idea in the induction step is as follows. For a connected graph G, we consider a tree of bridges T for which $\mathsf{bd}(G \setminus V(T)) < \mathsf{bd}(G)$. We can summarize the relevant ways in which a maximum independent set in G can be composed out of maximum independent sets for the connected components of $G \setminus E(T)$, into a weighted tree T' that is obtained from T by adding a pendant leaf to each vertex. In turn, maximum-weight independent sets in T' correspond to maximum independent sets a bipartite graph obtained from T' by replacing each vertex by a set of false twins. Applying Lemma 5.2 to this bipartite graph points to two vertices that form a blocking set. We can translate this back into two components of $G \setminus E(T)$ that are sufficient for constructing a blocking set in G, and apply induction using the fact that $\mathsf{bd}(G \setminus V(T)) < \mathsf{bd}(G)$.

▶ **Theorem 5.4** ($\star$). *Let G be a graph and $Y_G \subseteq V(G)$ a blocking set in G. There is a blocking set $Y'_G \subseteq Y_G$ in G of size at most $2^{\mathsf{bd}(G)}$.*

Note that Theorem 5.4 and Theorem 4.14 together prove Theorem 1.2. We finish the section by showing that the upper-bound of $2^{\mathsf{bd}(G)}$ on the size of minimal blocking sets is tight.

▶ **Theorem 5.5.** *For every $c \in \mathbb{N}$, there is a graph G with $\mathsf{bd}(G) \leq c$ that contains a minimal blocking set of size 2^c.*

Proof. Recall the notion of triangle-path from Definition 4.9. For $t \geq 2$, let a *truncated triangle-path of length t* be the graph U_t obtained from a triangle-path of length t by removing vertices a_1 and b_t; see Figure 2. Analogously to Observation 4.10, we show that $Y_t := \{c_i \mid i \in [t]\}$ is a minimal blocking set in U_t. Since Y_t is an independent set of size t, while (the remainders of) the triangles in U_t partition the vertices of U_t into t cliques, it follows that $\alpha(U_t) = t$. The set Y_t is a blocking set, since $U_t \setminus Y_t$ is a path on $2(t-1)$ vertices, whose independence number is only $t - 1$. Finally, it is easy to see that for any $y \in Y_t$, there is a size-t independent set in $U_t \setminus (Y_t \setminus y)$ that consists of the vertex y and, for every (remainder of a) triangle in U_t, the vertex closest to y.

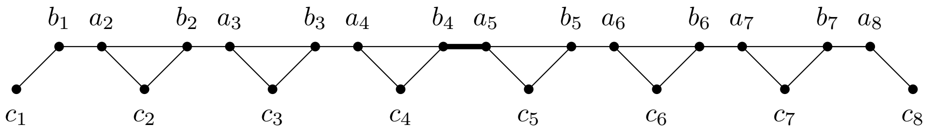


Figure 2 Truncated triangle path U_8 of length 8, illustrating Theorem 5.5. Removing the fat middle bridge and its incident vertices, leaves two connected components isomorphic to U_4.

Hence U_t has a minimal blocking set of size t, for all $t \geq 2$. To prove the theorem, it therefore suffices to show that $\mathsf{bd}(U_{2^c}) \leq c$ for all $c \in \mathbb{N}$. We prove this by induction on c. For $c = 1$, note that the graph U_2 is just the four-vertex path. Hence it is a forest, implying $\mathsf{bd}(U_2) = 1$ by Proposition 3.2. For $c > 1$, consider the graph U_{2^c}. By construction, the middle edge $e = \{b_{2^{c-1}}, a_{2^{c-1}+1}\}$ is a bridge in U_{2^c}. Let T be the tree in U_{2^c} consisting of the single bridge e. Note that removing $V(T)$ splits U_{2^c} evenly, into two connected components that are both isomorphic to $U_{2^{c-1}}$. By induction, $\mathsf{bd}(U_{2^{c-1}}) \leq c - 1$. Then Proposition 3.2 shows that $\mathsf{bd}(U_{2^c}) \leq 1 + \mathsf{bd}(U_t \setminus V(T)) = 1 + (c - 1) = c$. ◀

6 Kernelization for modulators to bounded bridge-depth

To establish the positive direction of Theorem 1.1, we develop a polynomial kernel for Vertex Cover parameterized by the size of a modulator X whose removal leaves a graph of constant bridge-depth; an approximately optimal such set X can be computed using Proposition 3.5. As the kernelization is technical and consists of many different reduction rules, with a nontrivial size analysis, the material is deferred to the full version [4]. In this limited space, we present the high-level idea behind the kernelization and the role of bridge-depth.

Consider an instance (G, k) of Vertex Cover with a modulator X such that $\mathsf{bd}(G \setminus X) \in \mathcal{O}(1)$. As explained in the introduction, using the fact that minimal blocking sets for the components C of $G \setminus X$ have bounded size, the number of such components can easily be bounded by $|X|^{\mathcal{O}(1)}$. To bound the size of individual components, the definition of bridge-depth ensures that in each connected component C of $G \setminus X$ there is a tree of bridges $T \subseteq E(C)$ (called a *lowering tree*) such that removing the vertex set $V(T)$ from C decreases the bridge-depth of C. By designing new problem-specific reduction rules, we shrink the tree of bridges to size polynomial in the parameter. This is where the main technical work of the kernelization step lies. It properly subsumes the earlier kernelization for the parameterization by distance to a forest, which is imported as a black box in all previous works [5, 17–19, 27]. Having bounded the number of components of $G \setminus X$, together with the size of a lowering tree of bridges in each component, we now proceed as follows: in each component C of $G \setminus X$ we move the vertices from a lowering tree of bridges into the set X. This blows up $|X|$ by a polynomial factor, but strictly decreases the bridge-depth of the graph $G \setminus X$. We then recursively kernelize the resulting instance. When the bridge-depth of $G \setminus X$ reaches zero, the graph $G \setminus X$ is empty and the kernelization is completed. Full details can be found in the full version [4].

The negative direction of Theorem 1.1 is much easier to establish. Using the fact that a minor-closed family $\mathcal{F}$ of unbounded bridge-depth contains all triangle paths, a kernelization lower bound for modulators to such $\mathcal{F}$ follows easily using known gadgets.

7 Conclusion

In this paper we introduced the graph parameter bridge-depth and used it to characterize the minor-closed graph classes $\mathcal{F}$ for which Vertex Cover parameterized by $\mathcal{F}$-modulator has a polynomial kernel. It would be interesting to see whether the characterization can be extended to subgraph-closed or even hereditary graph classes. If a characterization exists of the hereditary graph classes whose modulators lead to a polynomial kernel, it will likely not be as clean as Theorem 1.1: it will have to deal with the fact that bipartite graphs can be arbitrarily complex in terms of width parameters, while bipartite modulators allow for a polynomial kernel. Hence such a characterization has to capture parity conditions of $\mathcal{F}$.

A natural attempt to generalize our approach to deal with bipartite graphs is to consider the following parameter, which we call *bipartite-contraction-depth*: we mimic the definition of bridge-depth (cf. Definition 3.1), except that we redefine the graph G_{CB} to be the graph obtained from G by *simultaneously* contracting all edges that do *not* lie on an odd cycle. Note that bipartite-contraction-depth generalizes bridge-depth, in the sense that bridges do not lie on an odd cycle, and that the bipartite-contraction-depth of a graph with an odd cycle transversal of size k is at most $k+1$. Having defined this parameter, we would need, in order to obtain a statement similar to Theorem 4.14, that large bipartite-contraction-depth implies the existence of structures that allow to obtain kernel lower bounds, similarly to the fact that large bridge-depth implies the existence of large triangle-paths (cf. Corollary 4.13). The appropriate structure here seems to be an *odd-cycle-path of length t*, defined as a set of t vertex-disjoint odd cycles $C_1, \ldots, C_t$, and a set of $t-1$ vertex-disjoint paths (of any length) connecting C_i to C_{i+1} for $i \in [t-1]$, in such a way that for every $i \in \{2, \ldots, t-1\}$, the two attachment vertices in C_i are distinct. Now the expected property would be that large bipartite-contraction-depth forces long odd-cycle-paths. Unfortunately, this is not true. Indeed, consider the *Escher wall of size h* depicted in [33, Fig. 3]. It is proved in [33] that this graph does not contain two vertex-disjoint odd cycles, but a smallest hitting set for odd cycles has size h. Since there are no two vertex-disjoint cycles, a longest odd-cycle-path has length one. On the other hand, it can be easily verified that an Escher wall of size h has bipartite-contraction-depth $\Omega(h)$. Informally, this can be seen by noting that, initially, all edges lie on an odd cycle, hence a vertex removal is required, and that each such removal cascades in a constant number of contractions until all edges lie again on an odd cycle. Since a smallest hitting set for odd cycles of an Escher wall of size h has size h, the claimed bound follows. Therefore, summarizing this discussion, if one aims at a result similar to Theorem 1.1 that also applies to families $\mathcal{F}$ containing bipartite graphs, it seems that significant new ideas are required.

Another open research direction consists of a further algorithmic exploration of the merits of bridge-depth. We expect that several polynomial-space fixed-parameter tractable algorithms that work for graphs of bounded tree-depth [7, 32] can be extended to work with bridge-depth instead. Which other ways to enrich the recursive definition of tree-depth lead to novel algorithmic insights? As for kernelization purposes, it is plausible that bridge-depth also characterizes the existence of polynomial kernels for other problems other than Vertex Cover, parameterized by the vertex-deletion distance of the input graph to a minor-closed graph class.

References

1 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996. doi:10.1137/S0097539793251219.

2 Hans L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theor. Comput. Sci.*, 209(1-2):1–45, 1998. doi:10.1016/S0304-3975(97)00228-4.

3 Hans L. Bodlaender. Kernelization, exponential lower bounds. In *Encyclopedia of Algorithms*, pages 1013–1017. Springer, 2016. doi:10.1007/978-1-4939-2864-4_521.

4 Marin Bougeret, Bart M. P. Jansen, and Ignasi Sau. Bridge-Depth Characterizes which Structural Parameterizations of Vertex Cover Admit a Polynomial Kernel. *CoRR*, abs/2004.12865, 2020. arXiv:2004.12865.

5 Marin Bougeret and Ignasi Sau. How much does a treedepth modulator help to obtain polynomial kernels beyond sparse graphs? *Algorithmica*, 81(10):4043–4068, 2019. doi:10.1007/s00453-018-0468-8.

6 Jannis Bulian and Anuj Dawar. Graph isomorphism parameterized by elimination distance to bounded degree. *Algorithmica*, 75(2):363–382, 2016. doi:10.1007/s00453-015-0045-3.

7 Li-Hsuan Chen, Felix Reidl, Peter Rossmanith, and Fernando Sánchez Villaamil. Width, depth, and space: Tradeoffs between branching and dynamic programming. *Algorithms*, 11(7):98, 2018. doi:10.3390/a11070098.

8 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms.* Springer, 2015. doi:10.1007/978-3-319-21275-3.

9 Marek Cygan, Daniel Lokshtanov, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. On the hardness of losing width. *Theory Comput. Syst.*, 54(1):73–82, 2014. doi:10.1007/s00224-013-9480-1.

10 Holger Dell and Dieter van Melkebeek. Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. *J. ACM*, 61(4):23:1–23:27, 2014. doi:10.1145/2629620.

11 Reinhard Diestel. *Graph Theory.* Springer-Verlag, Heidelberg, 5th edition, 2016.

12 Rod G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity.* Texts in Computer Science. Springer, 2013.

13 Michael R. Fellows, Lars Jaffke, Aliz Izabella Király, Frances A. Rosamond, and Mathias Weller. What is known about vertex cover kernelization? In Hans-Joachim Böckenhauer, Dennis Komm, and Walter Unger, editors, *Adventures Between Lower Bounds and Higher Altitudes - Essays Dedicated to Juraj Hromkovič on the Occasion of His 60th Birthday*, volume 11011 of *Lecture Notes in Computer Science*, pages 330–356. Springer, 2018. doi:10.1007/978-3-319-98355-4_19.

14 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory.* Springer-Verlag, 2006. doi:10.1007/3-540-29953-X.

15 Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, Geevarghese Philip, and Saket Saurabh. Hitting forbidden minors: Approximation and kernelization. *SIAM J. Discrete Math.*, 30(1):383–410, 2016. doi:10.1137/140997889.

16 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: Theory of Parameterized Preprocessing.* Cambridge University Press, 2019. doi:10.1017/9781107415157.

17 Fedor V. Fomin and Torstein J. F. Strømme. Vertex cover structural parameterization revisited. In *Proc. 42nd WG*, volume 9941 of *LNCS*, pages 171–182, 2016. doi:10.1007/978-3-662-53536-3_15.

18 Eva-Maria C. Hols and Stefan Kratsch. Smaller parameters for vertex cover kernelization. In *Proc. 12th IPEC*, volume 89 of *LIPIcs*, pages 20:1–20:12, 2017. doi:10.4230/LIPIcs.IPEC.2017.20.

19 Eva-Maria C. Hols, Stefan Kratsch, and Astrid Pieterse. Elimination distances, blocking sets, and kernels for vertex cover. In *Proc. 37nd STACS*, volume 154 of *LIPIcs*, pages 36:1–36:14, 2020. doi:10.4230/LIPIcs.STACS.2020.36.

20 John E. Hopcroft and Robert Endre Tarjan. Efficient algorithms for graph manipulation (algorithm 447). *Commun. ACM*, 16(6):372–378, 1973. doi:10.1145/362248.362272.

21 Bart M. P. Jansen and Hans L. Bodlaender. Vertex cover kernelization revisited: Upper and lower bounds for a refined parameter. In *Proc. 28th STACS*, volume 9 of *LIPIcs*, pages 177–188, 2011. doi:10.4230/LIPIcs.STACS.2011.177.

22 Bart M. P. Jansen and Hans L. Bodlaender. Vertex cover kernelization revisited - upper and lower bounds for a refined parameter. *Theory Comput. Syst.*, 53(2):263–299, 2013. doi:10.1007/s00224-012-9393-4.

23 Bart M. P. Jansen and Astrid Pieterse. Polynomial kernels for hitting forbidden minors under structural parameterizations. In *Proc. 26th ESA*, volume 112 of *LIPIcs*, pages 48:1–48:15, 2018. doi:10.4230/LIPIcs.ESA.2018.48.

24 Stefan Kratsch. A randomized polynomial kernelization for vertex cover with a smaller parameter. *SIAM J. Discrete Math.*, 32(3):1806–1839, 2018. doi:10.1137/16M1104585.

25 Stefan Kratsch and Magnus Wahlström. Representative sets and irrelevant vertices: New tools for kernelization. In *Proc. 53rd FOCS*, pages 450–459, 2012. doi:10.1109/FOCS.2012.46.

26 Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Kernelization - Preprocessing with a guarantee. In *The Multivariate Algorithmic Revolution and Beyond*, pages 129–161, 2012. doi:10.1007/978-3-642-30891-8_10.

27 Diptapriyo Majumdar, Venkatesh Raman, and Saket Saurabh. Polynomial kernels for vertex cover parameterized by small degree modulators. *Theory Comput. Syst.*, 62(8):1910–1951, 2018. doi:10.1007/s00224-018-9858-1.

28 George L. Nemhauser and Leslie E. Trotter Jr. Vertex packings: structural properties and algorithms. *Math. Program.*, 8:232–248, 1975. doi:10.1007/BF01580444.

29 Jaroslav Nesetril and Patrice Ossona de Mendez. *Sparsity - Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and combinatorics*. Springer, 2012. doi:10.1007/978-3-642-27875-4.

30 Jaroslav Nesetril and Patrice Ossona de Mendez. Tree-depth, subgraph coloring and homomorphism bounds. *Eur. J. Comb.*, 27(6):1022–1041, 2006. doi:10.1016/j.ejc.2005.01.010.

31 Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006. doi:10.1093/acprof:oso/9780198566076.001.0001.

32 Michal Pilipczuk and Marcin Wrochna. On space efficiency of algorithms working on structural decompositions of graphs. *ACM TOCT*, 9(4):18:1–18:36, 2018. doi:10.1145/3154856.

33 Dieter Rautenbach and Bruce A. Reed. The Erdös-Pósa Property for Odd Cycles in Highly Connected Graphs. *Combinatorica*, 21(2):267–278, 2001. doi:10.1007/s004930100024.

34 Neil Robertson and Paul D. Seymour. Graph minors. V. Excluding a planar graph. *J. Comb. Theory, Ser. B*, 41(1):92–114, 1986. doi:10.1016/0095-8956(86)90030-4.

35 Neil Robertson and Paul D. Seymour. Graph minors. XX. Wagner's conjecture. *J. Comb. Theory, Ser. B*, 92(2):325–357, 2004. doi:10.1016/j.jctb.2004.08.001.

36 Wouter Cames van Batenburg, Tony Huynh, Gwenaël Joret, and Jean-Florent Raymond. A tight Erdös-Pósa function for planar minors. In *Proc. 30th SODA*, pages 1485–1500. SIAM, 2019. doi:10.1137/1.9781611975482.

The Complexity of Promise SAT on Non-Boolean Domains

Alex Brandts
Department of Computer Science, University of Oxford, UK
alex.brandts@cs.ox.ac.uk

Marcin Wrochna
Department of Computer Science, University of Oxford, UK
marcin.wrochna@cs.ox.ac.uk

Stanislav Živný
Department of Computer Science, University of Oxford, UK
standa.zivny@cs.ox.ac.uk

Abstract

While 3-SAT is NP-hard, 2-SAT is solvable in polynomial time. Austrin, Guruswami, and Håstad [FOCS'14/SICOMP'17] proved a result known as "$(2+\varepsilon)$-SAT is NP-hard". They showed that the problem of distinguishing k-CNF formulas that are g-satisfiable (i.e. some assignment satisfies at least g literals in every clause) from those that are not even 1-satisfiable is NP-hard if $\frac{g}{k} < \frac{1}{2}$ and is in P otherwise. We study a generalisation of SAT on arbitrary finite domains, with clauses that are disjunctions of unary constraints, and establish analogous behaviour. Thus we give a dichotomy for a natural fragment of promise constraint satisfaction problems (PCSPs) on arbitrary finite domains.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness; Theory of computation → Constraint and logic programming

Keywords and phrases promise constraint satisfaction, PCSP, polymorphisms, algebraic approach, label cover

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.17

Category Track A: Algorithms, Complexity and Games

Related Version A full version of the paper is available [9] at `https://arxiv.org/abs/1911.09065`.

Funding This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 714532). The paper reflects only the authors' views and not the views of the ERC or the European Commission. The European Union is not liable for any use that may be made of the information contained therein.
Alex Brandts: Royal Society Enhancement Award and NSERC PGS Doctoral Award
Stanislav Živný: Royal Society University Research Fellowship

Acknowledgements We would like to thank the anonymous reviewers for their comments.

1 Introduction

It is a classic result that while 3-SAT is NP-hard [12, 22], 2-SAT can be solved in polynomial-time [21]. Austrin, Guruswami, and Håstad [2] considered the promise problem $(1, g, k)$-SAT (for integers $1 \le g \le k$): given a k-CNF formula with the promise that there is an assignment that satisfies at least g literals in each clause, find an assignment that satisfies at least one literal in each clause. They showed that the problem is NP-hard if $\frac{g}{k} < \frac{1}{2}$ and in P otherwise. Viewing k-SAT as $(1, 1, k)$-SAT, this shows that, in a natural sense, the transition from tractability to hardness occurs just after 2 and not just before 3.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 17; pp. 17:1–17:13
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The *set-satisfiability* (SetSAT) problem generalises the Boolean satisfiability problem to larger domains and we prove that it exhibits an analogous hardness transition. As in (a, g, k)-SAT, for integer constants $1 \leq a \leq g \leq k$ and $1 \leq s < d$, an instance of the (a, g, k)-SetSAT problem is a conjunction of clauses, where each clause is a disjunction of k literals. However, variables $x_1, \ldots, x_n$ can take values in a larger domain $[d] = \{1, \ldots, d\}$, while literals take the form "$x_i \in S$", where S is any subset of the domain $[d]$ of size s. As in the Boolean case, an assignment $\sigma\colon \{x_1, \ldots, x_n\} \to [d]$ is g-satisfying if it satisfies at least g literals in every clause. In (a, g, k)-SetSAT with set size s and domain size d, given an instance promised to be g-satisfiable, we are to find an a-satisfying assignment. When $s = 1$ and $d = 2$ we recover Boolean promise SAT, whereas when $a = g = 1$ we recover the non-promise version of SetSAT.

The most natural case of SetSAT is when we allow *all* nontrivial unary constraints (sets) as literals, i.e., the case $s = d - 1$. (While we defined sets defining literals to have size exactly s, one can simulate sets of size at most s by replacing them with all possible supersets of size s; see the proof of [9, Proposition A.5]). More generally one could consider the problem restricted to any family of literals. Our work deals with the "folded" case: if a set S is available as a literal, then for all permutations of the domain π, $\pi(S)$ is also available as a literal. In this case only the cardinality of S matters, and in fact only the maximum available cardinality matters, so all such problems are equivalent to (a, g, k)-SetSAT, for some constants a, g, k, s, d.

1.1 Related work

Our main motivation to study SetSAT as a promise problem is the fact that it constitutes a natural fragment of so-called promise constraint satisfaction problems (PCSPs), which are problems defined by homomorphisms between relational structures (see Section 2 for more details). PCSPs were studied as early as in the classic work of Garey and Johnson [16] on approximate graph colouring, but a systematic study originated in the paper of Austrin et al. [2]. In a series of papers [5, 6, 7], Brakensiek and Guruswami linked PCSPs to the universal-algebraic methods developed for the study of non-uniform CSPs [4]. In particular, the notion of (weak) polymorphisms, formulated in [2], allowed some ideas developed for CSPs to be used in the context of PCSPs. The algebraic theory of PCSPs was then lifted to an abstract level by Barto, Bulín, Krokhin, and Opršal in [10, 3]. Consequently, this theory was used by Ficak, Kozik, Olšák, and Stankiewicz to obtain a dichotomy for symmetric Boolean PCSPs [15], thus improving on an earlier result from [6], which gave a dichotomy for symmetric Boolean PCSPs with folding. Further resent results on PCSPs include the work of Krokhin and Opršal [20], Brakensiek and Guruswami [8], and Austrin, Bhangale, and Potukuchi [1].

Variants of the Boolean satisfiability problem over larger domains have been defined using CNFs by Gil, Hermann, Salzer, and Zanuttini [17] and DNFs by Chen and Grohe [11] but, as far as we are aware, have not been studied as promise problems before.

1.2 Results

We completely resolve the complexity of (a, g, k)-SetSAT. As our main result, we show that the complexity of $(1, g, k)$-SetSAT depends only on the ratio $\frac{g}{k}$.

▶ **Theorem 1.** *$(1, g, k)$-SetSAT with set size s and domain size $s+1$ is solvable in polynomial time if $\frac{g}{k} \geq \frac{s}{s+1}$ and is NP-hard otherwise.*

Our result generalises the case of $(1, g, k)$-SAT, where $s = 1$ and the hardness threshold is $\frac{1}{2}$. The general case, when $a \neq 1$ or $d > s + 1$, follows by simple reductions (cf. [9, Corollary A.4]). The positive side of the theorem is proved by a simple randomised algorithm based on classical work of Papadimitriou [23], just as in the Boolean case. The main difficulty is in proving NP-hardness when the ratio $\frac{g}{k}$ is close to, but below $\frac{s}{s+1}$.

Following [2] and the more abstract algebraic framework of [3], the hardness proof relies on understanding polymorphisms, i.e., high-arity functions $f\colon [d]^n \to [d]$ which describe the symmetries of our computational problem. In the Boolean case, the proof of [2] relies on showing that every polymorphism depends on only a few variables (in other words, is a junta), and that this condition suffices for a reduction from the gap label cover problem. In our case, this condition does not hold, and neither do the various generalisations of it used in later work on PCSPs [15, 3, 20]. In fact, we show in [9, Section 6] that SetSAT has significantly richer, more robust polymorphisms, which makes the application of many such conditions impossible. Our main technical contribution is a new condition that guarantees an NP-hardness reduction from a multilayered variant of the gap label cover problem.

As in previous work, the combinatorial core of our NP-hardness results for SetSAT relies on identifying, in every polymorphism $f\colon [d]^n \to [d]$, a small set of distinguished coordinates. The rough idea is that a polymorphism encodes a 1-in-n choice analogously to the long code, and the reduction relies on being able to decode f with small ambiguity.

The set of distinguished coordinates could be, in the simplest case, those on which f depends (called *essential coordinates*) and, as shown in [2], a small set of essential coordinates is sufficient for hardness of $(1, g, k)$-SAT if $\frac{g}{k} < \frac{1}{2}$. More generally, the distinguished set S could be such that some partial assignment to S makes f constant (as a function of its remaining coordinates), or restricts the range of f (called *fixing* [2, 15] and *avoiding* [3] sets, respectively). As shown in [9, Section 6], the polymorphisms of SetSAT on non-Boolean domains do not have small sets of coordinates that are essential, fixing, or avoiding. Instead, in this paper we introduce the notion of a *smug set* of f. We say that a set $S \subseteq [n]$ is smug if for some input $(a_1, \dots, a_n)$ to f, the coordinates i whose values a_i agree with the output $f(a_1, \dots, a_n)$ are exactly those in S. We show that every polymorphism of SetSAT has a smug set of constant size (independent of n) and cannot have many disjoint smug sets.

In previous work, it was crucial that essential coordinates respect minors. We say that (an m-ary function) g is a *minor* of (an n-ary function) f if $g(x_1, \dots, x_m) \approx f(x_{\pi(1)}, \dots, x_{\pi(n)})$ for some $\pi\colon [n] \to [m]$ (that is, g is obtained from f by identifying or permuting coordinates of f, or introducing inessential coordinates). In that case, if S contains all essential coordinates of f, then $\pi(S)$ contains all essential coordinates of g. This does not hold for smug sets; instead, if S is a smug set of g, then its pre-image $\pi^{-1}(S)$ is a smug set of f. The pre-image may however be much larger. Still, these properties of smug sets are enough to guarantee that, in any sufficiently long chain of minors, if one chooses a random coordinate in a small smug set from each function in the chain, then for some two functions in the chain the choices will agree, respecting the minor relation between them with constant probability. We show that this condition is sufficient to obtain NP-hardness from a layered gap label cover problem. See Section 4 for details.

We note that several other properties of label cover variants were used before in the context of polymorphisms. Guruswami and Sandeep [18] use "smoothness" of NP-hard label cover instances (introduced by Khot [19]) so that a minor relation π needs to be respected only if it is injective on a small set S. This allows them to use sets S which are *weakly fixing*, i.e. the partial assignment to S which makes f constant does not necessarily have to assign the same value to all coordinates in S. Layered label cover was introduced by Dinur, Guruswami, Khot, and Regev [13] to tighten the approximation hardness for hypergraph vertex cover. In the proof of hardness of hypergraph colouring by Dinur, Regev, and Smyth [14], as reinterpreted in [3], layered label cover is used to partition polymorphisms into an arbitrary constant number L of parts, so that only minors within one part need to be respected. This implies that in any chain of minors with $L + 1$ functions, some two functions will be in the same part and hence the minor between them will be respected; our approach is hence similar, though apparently more general, in this aspect. Another feature used in [14, 3] is that the bound on the size of a set of special coordinates or on the number of disjoint such sets may be any subpolynomial function in n, not necessarily a constant. These different features of NP-hard label cover instances can be combined; however, this is not necessary for our result.

2 Preliminaries

Let $[n] = \{1, 2, \dots, n\}$. For a set A, we call $R \subseteq A^k$ a *relation* of arity $\mathrm{ar}(R) = k$ and $f\colon A^k \to B$ a function of arity $\mathrm{ar}(f) = k$.

We take the domain of the variables in SetSAT to be $[d]$ and for a fixed $s < d$ we identify each literal with the indicator function of some $S \subseteq [d], |S| = s$: $S(x) = \mathbb{1}[x \in S]$. For a SetSAT instance (or *formula*) with n variables $x_1, \dots, x_n$, an assignment to the variables is a function $\sigma : \{x_1, \dots, x_n\} \to [d]$. An assignment σ is called a g-satisfying assignment for an instance ϕ if σ satisfies at least g literals in every clause of ϕ. A 1-satisfying assignment is usually simply called a satisfying assignment. A formula is called g-satisfiable if there exists a g-satisfying assignment to its variables, and satisfiable if there exists a 1-satisfying assignment.

The SAT problem corresponds to the SetSAT problem with $d = 2$ and $s = 1$, so SetSAT does indeed generalise SAT. Note that every SetSAT instance is trivially unsatisfiable when $s = 0$ and satisfiable when $s = d$, so we exclude these cases in our analysis. We now give the formal definition of (a, g, k)-SetSAT.

▶ **Definition 2.** *Let $1 \leq s < d$ and $1 \leq a \leq g \leq k$. The (a, g, k)-SetSAT problem is the following promise problem. In the decision version, given a SetSAT instance where each clause has k literals, accept the instance if it is g-satisfiable and reject it if it is not a-satisfiable. In the search version, given a g-satisfiable SetSAT instance, find an a-satisfying assignment.*

We will prove hardness only for the decision version of (a, g, k)-SetSAT and tractability only for the search version. This suffices since the decision version of (a, g, k)-SetSAT is polynomial-time reducible to the corresponding search problem. This is discussed in [9, Appendix A], where it is also shown how to obtain simple hardness results for SetSAT. In particular, [9, Propositions A.1 and A.3] show that we can focus on the case of $(1, g, k)$-SetSAT with $d = s + 1$.

Promise CSPs

We describe how the SetSAT problem fits into the general framework of promise CSPs (PCSPs). For a more in-depth algebraic study of PCSPs, we refer the reader to [3].

A *relational structure* $\mathbf{A}$ is a tuple $(A; R_1, \dots, R_m)$ where each R_i is a relation on A. We say that two relational structures are *similar* if their relations have the same sequence of arities. A *homomorphism* between similar relational structures $\mathbf{A} = (A; R_1^{\mathbf{A}}, \dots, R_m^{\mathbf{A}})$ and $\mathbf{B} = (B; R_1^{\mathbf{B}}, \dots, R_m^{\mathbf{B}})$ is a function $h : A \to B$ such that $(a_1, \dots, a_{\mathrm{ar}(R_i^{\mathbf{A}})}) \in R_i^{\mathbf{A}}$ implies $(h(a_1), \dots, h(a_{\mathrm{ar}(R_i^{\mathbf{A}})})) \in R_i^{\mathbf{B}}$ for all i. We denote this by $\mathbf{A} \to \mathbf{B}$.

▶ **Definition 3.** *Let* $(\mathbf{A}, \mathbf{B})$ *be a pair of similar relational structures such that there is a homomorphism* $\mathbf{A} \to \mathbf{B}$*. The pair* $(\mathbf{A}, \mathbf{B})$ *is called the* template *of the* promise constraint satisfaction problem $\mathrm{PCSP}(\mathbf{A}, \mathbf{B})$*. The decision version of* $\mathrm{PCSP}(\mathbf{A}, \mathbf{B})$ *is as follows: given as input a relational structure* $\mathbf{C}$ *similar to* $\mathbf{A}$ *and* $\mathbf{B}$*, decide whether* $\mathbf{C}$ *admits a homomorphism to* $\mathbf{A}$*, or does not even admit a homomorphism to* $\mathbf{B}$*. The* promise *is that it is never the case that* $\mathbf{C} \to \mathbf{B}$ *but* $\mathbf{C} \not\to \mathbf{A}$*. The search problem asks to find a homomorphism* $\mathbf{C} \to \mathbf{B}$*, given that there exists a homomorphism* $\mathbf{C} \to \mathbf{A}$*.*

Since (a, g, k)-SetSAT is a PCSP where all relations have fixed arity k, it is possible to transform SetSAT instances from their CNF representation into the PCSP representation of Definition 3. Let f be a bijection between $[d^k]$ and the set of clauses containing k literals (ignoring the variables they contain). We can represent each SetSAT instance Ψ as a relational structure $\mathbf{C} = (C; R_1^{\mathbf{C}}, \ldots, R_{d^k}^{\mathbf{C}})$, where $C = \{x_1, \ldots, x_n\}$ is the set of variables appearing in Ψ and $R_i^{\mathbf{C}}$ is a k-ary relation corresponding to the clause $f(i)$. For each clause $(S_1(x_1) \vee \ldots \vee S_k(x_k))$ of type $f(i)$ in Ψ, we add the tuple $(x_1, \ldots, x_k)$ to $R_i^{\mathbf{C}}$, so that each $R_i^{\mathbf{C}}$ collects the tuples of variables appearing in clauses of type $f(i)$.

Now define $R_i^{\mathbf{A}}$ (respectively $R_i^{\mathbf{B}}$) to be the k-ary relation over $[d]$ containing $(a_1, \ldots, a_k)$ if and only if $(a_1, \ldots, a_k)$ g-satisfies (respectively a-satisfies) the clause $f(i)$ when the variable of the j-th literal of $f(i)$ is set to a_j, for $1 \leq j \leq k$. Let $\mathbf{A} = ([d], R_1^{\mathbf{A}}, \ldots, R_{d^k}^{\mathbf{A}})$ and $\mathbf{B} = ([d], R_1^{\mathbf{B}}, \ldots, R_{d^k}^{\mathbf{B}})$. Then (a, g, k)-SetSAT is precisely $\mathrm{PCSP}(\mathbf{A}, \mathbf{B})$: the identity function is a homomorphism from $\mathbf{A}$ to $\mathbf{B}$, a homomorphism $\mathbf{C} \to \mathbf{A}$ represents a g-satisfying assignment to Ψ, and a homomorphism $\mathbf{C} \to \mathbf{B}$ represents an a-satisfying assignment to Ψ.

Polymorphisms

The following concept from the algebraic study of PCSPs is central to our hardness result.

Let $f : A^m \to B$ be a function. We say that f is a *polymorphism* of the template $(\mathbf{A}, \mathbf{B})$ if, for $\bar{a}^1, \ldots, \bar{a}^m \in R_i^{\mathbf{A}}$, we have that $f(\bar{a}^1, \ldots, \bar{a}^m) \in R_i^{\mathbf{B}}$; here f is applied componentwise. We will denote by $\mathrm{Pol}(\mathbf{A}, \mathbf{B})$ the set of all polymorphisms of the template $(\mathbf{A}, \mathbf{B})$. A simple example of a polymorphism of every template with $A = B$ is a *projection*, which is a function $p_i^{(m)} : A^m \to B$ of the form $p_i^{(m)}(x_1, \ldots, x_m) = x_i$. More generally, for every template, trivial polymorphisms are given by *dictators*, which are functions p of the form $p(x_1, \ldots, x_m) = f(x_i)$, where f is a homomorphism from A to B.

In particular, $f : [d]^m \to [d]$ is a polymorphism of (a, g, k)-SetSAT if for every SetSAT clause C of width k and for every tuple $\bar{v}^1, \ldots, \bar{v}^m \in [d]^k$ of g-satisfying assignments to C, we have that $f(\bar{v}^1, \ldots, \bar{v}^m)$ is an a-satisfying assignment to C.

3 Tractability

How big must one make the fraction of satisfied literals in order for the SetSAT problem to become tractable? The following proposition shows that $\frac{s}{s+1}$ is sufficient.

▶ **Proposition 4.** *For* $1 \leq s < d$ *and* $\frac{g}{k} \geq \frac{s}{s+1}$*,* $(1, g, k)$*-SetSAT is solvable in expected polynomial time.*

Proof. Algorithm 1 finds a satisfying assignment to a g-satisfiable formula in expected polynomial time. The algorithm and its analysis are based on [2, Proposition 6.1], which in turn is based on Papadimitriou's randomised algorithm for 2-SAT [23].

Algorithm 1 Randomised algorithm for $(1, g, k)$-SetSAT with $\frac{g}{k} \ge \frac{s}{s+1}$.

```
1: $x \leftarrow$ arbitrary assignment
2: while $x$ does not satisfy input formula $\phi$ do
3:     Arbitrarily pick a falsified clause $C$
4:     Randomly choose from $C$ a literal $S(x_i)$
5:     Randomly choose a value for $x_i$ so that $S(x_i)$ is satisfied
   return $x$
```

Suppose that ϕ has a g-satisfying assignment x^*. Let x^t be the assignment obtained in iteration t of the algorithm, and let $D_t = \text{dist}(x^t, x^*)$, where $\text{dist}(x, y)$ is the Hamming distance between x and y. Since $D_t - D_{t-1} \in \{-1, 0, 1\}$ for every t, we have

$$\begin{aligned}\mathbb{E}(D_t - D_{t-1}) &= \mathbb{P}(D_t - D_{t-1} = 1) - \mathbb{P}(D_t - D_{t-1} = -1)\\ &\le \frac{k-g}{k} - \frac{g}{k}\frac{1}{s} \le 0 \quad \text{if and only if} \quad \frac{g}{k} \ge \frac{s}{s+1}.\end{aligned}$$

The sequence $D_0, D_1, \ldots$ is a random walk starting between 0 and n with steps either unbiased or biased toward 0. With constant probability, such a walk hits 0 within n^2 steps and so the probability that the algorithm fails to find a satisfying assignment within crn^2 steps is at most 2^{-r} for some constant c. ◀

▶ Remark 5. The proof of Proposition 4 can be modified to show that Algorithm 1 also finds a satisfying assignment when each literal corresponds to a set of size *at most* s. This makes sense intuitively, as smaller literals give the algorithm a better chance of setting x_i equal to x_i^* in Step 5.

We show that if $\frac{g}{k} \ge \frac{s}{s+1}$ then $(1, g, k)$-SetSAT has a specific family of polymorphisms that leads to a *deterministic* algorithm based on linear programming.

A function $f : A^m \to B$ is *symmetric* if $f(a_1, \ldots, a_m) = f(a_{\pi(1)}, \ldots, a_{\pi(m)})$ for all $a_1, \ldots, a_m \in A$ and all permutations π on $[m]$.

▶ **Definition 6.** *A symmetric function $f : [d]^m \to [d]$ is a* plurality *if*

$$f(x_1, \ldots, x_m) = \mathit{argmax}_{a \in [d]}\{\#\ \mathit{of\ occurrences\ of}\ a\ \mathit{in}\ (x_1, \ldots, x_m)\},$$

with ties broken in such a way that f is symmetric.

We will also use the fact that all polymorphisms of SetSAT are *conservative*; i.e., they always return one of their input values, as the following proposition shows.

▶ **Proposition 7.** *All polymorphisms of $(1, g, k)$-SetSAT are conservative.*

Proof. Let $f : [d]^m \to [d]$ be such that $f(a_1, \ldots, a_m) = b$ and $b \notin \{a_1, \ldots, a_m\}$. If S is a literal not containing b, then the clause $(S(x_1) \vee \ldots \vee S(x_k))$ is g-satisfied (even k-satisfied) by setting all x_i equal to any one of the a_j. Thus taking the m assignments $(x_1 = \cdots = x_k = a_j)_{1 \le j \le m}$ and applying f to each component, we get the assignment $x_1 = \cdots = x_k = b$ which clearly does not 1-satisfy the clause, and so f cannot be a polymorphism. ◀

▶ **Proposition 8.** *Let $s \ge 1$. If $\frac{g}{k} > \frac{s}{s+1}$ then every plurality function is a polymorphism of $(1, g, k)$-SetSAT. If $\frac{g}{k} = \frac{s}{s+1}$ then every plurality function of arity $m \not\equiv 0 \mod s+1$ is a polymorphism of $(1, g, k)$-SetSAT, and no symmetric function of arity $m \equiv 0 \mod s+1$ is a polymorphism of $(1, g, k)$-SetSAT.*

Proof. Let f be a plurality function of arity m. Given m g-satisfying assignments to a clause of width k, we are guaranteed to have at least mg satisfying values among the mk total values. Therefore there is a coordinate i, $1 \leq i \leq k$, containing at least $\lceil \frac{mg}{k} \rceil$ satisfying values, that is, at least $\lceil \frac{mg}{k} \rceil$ values not equal to the value b forbidden by the i-th literal of the clause. In order for f to be a polymorphism it suffices that $\lceil \frac{mg}{k} \rceil > \frac{s}{s+1}m$, since then b will appear fewer than $\frac{m}{s+1}$ times and will never be returned using the plurality rule. But $\frac{g}{k} > \frac{s}{s+1}$ is equivalent to $\frac{mg}{k} > \frac{s}{s+1}m$, and the latter implies that $\lceil \frac{mg}{k} \rceil > \frac{s}{s+1}m$, so f is a polymorphism.

In the case that $\frac{g}{k} = \frac{s}{s+1}$, the same argument works so long as $\frac{mg}{k}$ is not an integer, since by taking the ceiling we obtain a value strictly greater than $\frac{s}{s+1}m$. Since $\frac{g}{k} = \frac{s}{s+1}$, we have $\frac{g}{k}m = \frac{s}{s+1}m$ and this is an integer only if m is a multiple of $s+1$.

To show that there are no symmetric polymorphisms when $\frac{g}{k} = \frac{s}{s+1}$ and m is a multiple of $s+1$, note that this equality implies that k is divisible by $s+1$. Let M be the $(s+1) \times (s+1)$ matrix whose first row is $12 \cdots s+1$ and whose i-th row for $2 \leq i \leq s+1$ is obtained from the $(i-1)$-st row by shifting it cyclically to the left by one coordinate. We stack $\frac{k}{s+1}$ copies of M on top of each other and take $\frac{m}{s+1}$ copies of this stack side-by-side to form the $k \times m$ matrix M'. If f is symmetric, it returns the same value b when applied to each row of M'. Every column of M' satisfies exactly an $\frac{s}{s+1}$-fraction of the literals in a clause whose k literals all forbid b. On the other hand, the assignment produced by applying f to each row of M' does not even 1-satisfy this clause, so f is not a polymorphism. ◀

Proposition 8 has interesting consequences for solvability of $(1, g, k)$-SetSAT via linear programming relaxations. By [3, Theorem 7.9], $(1, g, k)$-SetSAT is solvable by the basic linear programming relaxation if $\frac{g}{k} > \frac{s}{s+1}$ (since there exist symmetric polymorphisms of all arities) but not solvable by the basic linear programming relaxation if $\frac{g}{k} = \frac{s}{s+1}$ (since there do not exist symmetric polymorphisms of all arities). By [8, Theorem 3.1], $(1, g, k)$-SetSAT is solvable by the combined basic linear programming and affine relaxation if $\frac{g}{k} \geq \frac{s}{s+1}$ (since there exist symmetric polymorphisms of infinitely many arities). We note that iterative rounding of the basic linear relaxation could also be used to get a deterministic algorithm as in [2].

4 Layered label cover and smug sets

An *ℓ-layered label cover* instance is a sequence of $\ell + 1$ sets $X_0, \ldots, X_\ell$ (called *layers*) of variables with range $[m]$, for some *domain size* $m \in \mathbb{N}$, and a set of constraints Φ. Each constraint is a function (often called a projection constraint) from a variable $x \in X_i$ to a variable in a further layer $y \in X_j$, $i < j$: that is, a function denoted $\phi_{x \to y}$ which is satisfied by an assignment $\sigma\colon X_0 \cup \cdots \cup X_\ell \to [m]$ if $\sigma(y) = \phi_{x \to y}(\sigma(x))$. A *chain* is a sequence of variables $x_i \in X_i$ for $i = 0, \ldots, \ell$ such that there are constraints $\phi_{x_i \to x_j}$ between them, for $i < j$. A chain is *weakly satisfied* if at least one of these constraints is satisfied.

The basis for our hardness result is the hardness of distinguishing fully satisfiable instances from those where no constant fraction of chains can be weakly satisfied. This follows by a simple adaptation of a reduction from the work of Dinur, Guruswami, Khot, and Regev [13], which we defer to the full version [9, Appendix B].

▶ **Theorem 9.** *For every $\ell \in \mathbb{N}$ and $\varepsilon > 0$, there is an $m \in \mathbb{N}$ such that it is NP-hard to distinguish ℓ-layered label cover instances with domain size m that are fully satisfiable from those where not even an ε-fraction of all chains is weakly satisfied.*

In order to use Theorem 9 to derive hardness for PCSPs, we use the algebraic approach: every PCSP is equivalent to a promise problem about satisfying minor conditions with polymorphisms. We give definitions first, following [3], to where we refer the reader for a more detailed exposition.

For $f\colon A^n \to B$, $g\colon A^m \to B$ and $\pi\colon [n] \to [m]$, we say that g is the *minor* of f obtained from π if

$$g(x_1, \dots, x_m) \approx f(x_{\pi(1)}, \dots, x_{\pi(n)}), \tag{1}$$

where $g \approx f$ means that the values of g and f agree on every input in A^m. We write $f \xrightarrow{\pi} g$ as a shorthand for (1). For $\pi\colon [n] \to [m]$, the expression $f \xrightarrow{\pi} g$ is called a *minor identity.*

A *minion* on a pair of sets (A, B) is a non-empty set of functions from A^n to B (for $n \in \mathbb{N}$) that is closed under taking minors.

A *bipartite minor condition* is a finite set Σ of minor identities where the sets of function symbols used on the left- and right-hand sides are disjoint. More precisely, Σ is a pair of disjoint sets U and V of function symbols of arity n and m, respectively, and a set of minor identities of the form $f \xrightarrow{\pi} g$, where $g \in U$, $f \in V$ and $\pi : [n] \to [m]$. A bipartite minor condition Σ is *satisfied* in a minion $\mathcal{M}$ if there is an assignment $\xi : U \cup V \to \mathcal{M}$ that assigns to each function symbol a function from $\mathcal{M}$ of the corresponding arity so that for every identity $f \xrightarrow{\pi} g$ in Σ, we have $\xi(f) \xrightarrow{\pi} \xi(g)$ in $\mathcal{M}$. A bipartite minor condition is called *trivial* if it is satisfied in every minion, or equivalently, in the minion consisting of all projections on $\{0, 1\}$. Since choosing a projection of arity n is the same as choosing an element of $[n]$, deciding whether a bipartite minor condition is trivial is the same as standard label cover.

We can now define the *promise satisfaction of a minor condition* problem. For a minion $\mathcal{M}$ and an integer m, $\mathrm{PMC}_{\mathcal{M}}(m)$ is the following promise problem: given a bipartite minor condition Σ that involves only symbols of arity at most m, the answer should be YES if Σ is trivial and NO if Σ is not satisfiable in $\mathcal{M}$ (the promise is that either of those two cases holds, i.e. an algorithm can behave arbitrarily otherwise). Barto et al. [3] show that $\mathrm{PCSP}(\mathbf{A}, \mathbf{B})$ is log-space equivalent to $\mathrm{PMC}_{\mathcal{M}}(m)$, for $\mathcal{M} = \mathrm{Pol}(\mathbf{A}, \mathbf{B})$ and m a constant depending on $\mathbf{A}$ only.

A final piece of notation before we prove a corollary of Theorem 9. A *chain of minors* is a sequence of the form $f_0 \xrightarrow{\pi_{0,1}} f_1 \xrightarrow{\pi_{1,2}} \dots \xrightarrow{\pi_{\ell-1,\ell}} f_\ell$. We shall then write $\pi_{i,j}\colon [\mathrm{ar}(f_i)] \to [\mathrm{ar}(f_j)]$ for the composition of $\pi_{i,i+1}, \dots, \pi_{j-1,j}$, for any $0 \le i < j \le \ell$. Note that $f_i \xrightarrow{\pi_{i,j}} f_j$.

▶ **Corollary 10** (of Theorem 9). *Let $\mathcal{M}$ be a minion. Suppose there are constants $k, \ell \in \mathbb{N}$ and an assignment of a set of at most k coordinates $\mathrm{sel}(f) \subseteq [\mathrm{ar}(f)]$ to every $f \in \mathcal{M}$ such that for every chain of minors $f_0 \xrightarrow{\pi_{0,1}} f_1 \xrightarrow{\pi_{1,2}} \dots \xrightarrow{\pi_{\ell-1,\ell}} f_\ell$, there are $0 \le i < j \le \ell$ such that $\pi_{i,j}(\mathrm{sel}(f_i)) \cap \mathrm{sel}(f_j) \neq \emptyset$. Then $\mathrm{PMC}_{\mathcal{M}}(m)$ is NP-hard, for m large enough. In particular, if $\mathcal{M} = \mathrm{Pol}(\mathbf{A}, \mathbf{B})$, then $\mathrm{PCSP}(\mathbf{A}, \mathbf{B})$ is NP-hard.*

Proof. For ℓ, k as in the assumption, let $\varepsilon := \frac{1}{k}$ and let m be as given by Theorem 9. We reduce an ℓ-layered label cover instance by replacing each variable x with a symbol f_x of arity m and each constraint $\phi_{x\to y}\colon [m] \to [m]$ by the minor condition $f_x \xrightarrow{\phi_{x\to y}} f_y$. If the original instance was fully satisfiable, the new instance is trivial (i.e., fully satisfiable by projections).

If the constructed instance is satisfied by functions in the minion $\mathcal{M}$, we define an assignment to the original instance by selecting, for each variable x, a random coordinate from $\mathrm{sel}(f_x) \subseteq [m]$ (uniformly, independently). The assumption guarantees a set of constraints $\phi_{x\to y}$ such that (1) each chain contains at least one and (2) for each such constraint $\phi_{x\to y}$, we have $\phi_{x\to y}(\mathrm{sel}(f_x)) \cap \mathrm{sel}(f_y) \neq \emptyset$. The random choice then satisfies each of these constraints, and hence weakly satisfies each chain, with probability at least $\frac{1}{k} = \varepsilon$. The expected fraction of weakly satisfied chains is thus at least ε and a standard maximisation-of-expectation procedure deterministically finds an assignment which certifies this. ◀

The following definition is crucial to our results.

▶ **Definition 11.** *For a function $f\colon A^{\mathrm{ar}(f)} \to B$ we say that a set of coordinates $S \subseteq [\mathrm{ar}(f)]$ is a* smug *set if there is an input vector $\bar{v} \in A^{\mathrm{ar}(f)}$ such that $S = \{i \mid v_i = f(\bar{v})\}$.*

We will use the following to prove hardness of $(1, g, k)$-SetSAT.

▶ **Corollary 12.** *Let $\mathcal{M}$ be a minion. Suppose there are constants $k, \ell \in \mathbb{N}$ such that the following holds, for every $f \in \mathcal{M}$:*

- *f has a smug set of at most k coordinates,*
- *f has no family of more than ℓ (pairwise) disjoint smug sets,*
- *if $f \xrightarrow{\pi} g$ and S is a smug set of g, then $\pi^{-1}(S)$ is a smug set of f.*

Then $\mathrm{PMC}_{\mathcal{M}}(m)$ is NP-hard, for m large enough. In particular, if $\mathcal{M} = \mathrm{Pol}(\mathbf{A}, \mathbf{B})$, then $\mathrm{PCSP}(\mathbf{A}, \mathbf{B})$ is NP-hard.

Proof. For each $f \in \mathcal{M}$, we define $\mathrm{sel}(f)$ as a smug set of at most k coordinates, arbitrarily chosen (some such set exists by the first condition). Consider a chain $f_0 \xrightarrow{\pi_{0,1}} f_1 \xrightarrow{\pi_{1,2}} \dots \xrightarrow{\pi_{\ell-1,\ell}} f_\ell$. Suppose to the contrary that for each $0 \le i < j \le \ell$, $\pi_{i,j}(\mathrm{sel}(f_i))$ is disjoint from $\mathrm{sel}(f_j)$, or equivalently, that $\mathrm{sel}(f_i)$ is disjoint from $\pi_{i,j}^{-1}(\mathrm{sel}(f_j))$. This implies that $\pi_{0,i}^{-1}(\mathrm{sel}(f_i))$ is disjoint from $\pi_{0,i}^{-1}(\pi_{i,j}^{-1}(\mathrm{sel}(f_j))) = \pi_{0,j}^{-1}(\mathrm{sel}(f_j))$. That is, the sets $\pi_{0,i}^{-1}(\mathrm{sel}(f_i))$ for $i = 0 \dots \ell$ are pairwise disjoint. By the third condition they are smug sets of f_0. But by the second condition this is impossible. ◀

We note that in the proof of Corollary 12, the exact definition of "smug" is irrelevant, as long as it satisfies the above three conditions.

It is easy to check that the definition of "smug" satisfies the third condition for any functions $f \xrightarrow{\pi} g$, not necessarily polymorphisms. Indeed, if an input $\bar{v} \in A^{\mathrm{ar}(g)}$ to g gives a smug set $S = \{j \mid v_j = g(\bar{v})\}$, then the corresponding input $\bar{u} \in A^{\mathrm{ar}(f)}$ to f defined as $u_i := v_{\pi(i)}$ satisfies $f(\bar{u}) = g(\bar{v})$ and hence gives a smug set $\{i \mid u_i = f(\bar{u})\} = \{i \mid v_{\pi(i)} = g(\bar{v})\} = \{i \mid \pi(i) \in S\} = \pi^{-1}(S)$.

The definition of "smug" is particularly well-suited to our problem, because whether f is a polymorphisms or not depends only on its family of smug sets.

▶ **Lemma 13.** *Let $1 \le s$ and $1 \le g < k$. A function $f\colon [s+1]^m \to [s+1]$ is a polymorphism of $(1, g, k)$-SetSAT if and only if there is no multiset $S_1, \dots, S_k$ of smug sets of f, such that each coordinate $\ell \in [m]$ is contained in at most $k - g$ of them.*

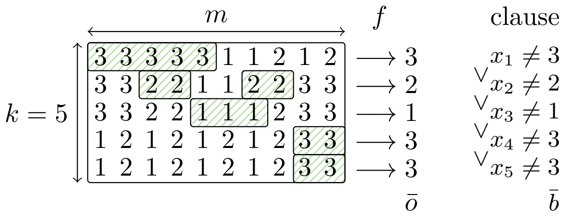

m
f
clause
$k = 5$
3 3 3 3 3 1 1 2 1 2 ⟶ 3 $x_1 \neq 3$
3 3 2 2 1 1 2 2 3 3 ⟶ 2 $\vee x_2 \neq 2$
3 3 2 2 1 1 1 2 3 3 ⟶ 1 $\vee x_3 \neq 1$
1 2 1 2 1 2 1 2 3 3 ⟶ 3 $\vee x_4 \neq 3$
1 2 1 2 1 2 1 2 3 3 ⟶ 3 $\vee x_5 \neq 3$
$\bar{o}$
$\bar{b}$

Figure 1 Illustration of Lemma 13. Smug sets $S \subseteq [m]$ are highlighted in each row.

Proof. A function $f\colon [s+1]^m \to [s+1]$ is not a polymorphism if and only if there is a clause of the form $x_1 \neq b_1 \vee \dots \vee x_k \neq b_k$ (for some column vector $\bar{b} \in [s+1]^k$) and a sequence of m column vectors $\bar{v}^1, \dots, \bar{v}^m \in [s+1]^k$ each of which g-satisfies the clause, but for which the vector $\bar{o} = f(\bar{v}^1, \dots, \bar{v}^m)$ (with f applied coordinatewise) does not even 1-satisfy the clause. The latter is equivalent to saying that $o_i = b_i$ for $i \in [k]$, that is, applying f to the i-th row gives $f(v_i^1, \dots, v_i^m) = b_i$. The former is equivalent to saying that for each column $\bar{v}$ in $\bar{v}^1, \dots, \bar{v}^m$, the condition $v_i \neq b_i$ holds for at least g indices $i \in [k]$ of that column. The two are hence equivalent to saying that for each column $\bar{v}^\ell$, $\ell \in [m]$, the condition $v_i^\ell = f(v_i^1, \dots, v_i^m)$ holds for at most $k - g$ indices $i \in [k]$ in that column. In other words, the k row vectors $(v_i^1, \dots, v_i^m)$ for $i \in [k]$ have smug sets such that ℓ is contained in at most $k - g$ of these sets, for each coordinate $\ell \in [m]$. ◀

Checking the second condition for polymorphisms of our SetSAT problem is easy.

▶ **Lemma 14.** *For every polymorphism f of $(1, g, k)$-SetSAT with domain size $s + 1$, if $S_1, \dots, S_n$ are disjoint smug sets of f, then $n < \frac{k}{k-g}$.*

Proof. Suppose to the contrary that $n \geq \frac{k}{k-g}$. Then we can build a multiset containing each S_i up to $k - g$ times until we have exactly k in total. We thus obtain a multiset of k smug sets such that every coordinate is contained in at most $k - g$ of them. ◀

5 Finding small smug sets

It is easy to show NP-hardness when $\frac{g}{k} \leq \frac{1}{2}$ (cf. [9, Proposition A.8]). We now show a general reduction by finding a small smug set for $(1, g, k)$-SetSAT whenever $\frac{g}{k} < \frac{s}{s+1}$.

Consider a polymorphism $f \colon [s + 1]^m \to [s + 1]$ of $(1, g, k)$-SetSAT (with set size s and domain size $s + 1$).

▶ **Lemma 15.** *There exists a smug set of size at most $s - 1$, or a family of s disjoint minimal smug sets $S_1, \dots, S_s$.*

Proof. Suppose that every smug set has size at least s. We show by induction on t that there is a family of t disjoint minimal smug sets $S_1, \dots, S_t$. Suppose we found $S_1, \dots, S_t$ for some $0 \leq t < s$ and we want to find S_{t+1}. Let T be a set containing one arbitrary coordinate from each S_i, $i = 1 \dots t$. Let $\bar{v} \in [s + 1]^m$ be the input vector with values $t + 2$ on T, i on $S_i \setminus T$ (for $i = 1 \dots t$) and $t + 1$ on the remaining coordinates $R := [m] \setminus (S_1 \cup \dots \cup S_t)$. Since $|T| \leq t < s$, T is not smug, so $f(\bar{v}) \neq t + 2$. By minimality, $S_i \setminus T$ are not smug for $i = 1 \dots t$, so $f(\bar{v}) \neq i$. Therefore, by conservativity of f (Proposition 7), the only remaining option is $f(\bar{v}) = t + 1$. Thus R is smug and disjoint from S_i. Taking S_{t+1} to be a minimal smug set contained in R proves the induction step. ◀

Together with Lemma 14, Lemma 15 already establishes (via Corollary 12) NP-hardness when $s \geq \frac{k}{k-g} = \frac{g}{k-g} + 1$ (equivalently, $\frac{g}{k} \leq \frac{s-1}{s}$): since there cannot be s disjoint smug sets, every polymorphism has a smug set of size at most $s - 1$. The proof in the general case, when $\frac{g}{k} < \frac{s}{s+1}$, extends this approach by first finding (assuming there are no small smug sets) disjoint minimal smug sets $S_1, \dots, S_s$, then exploiting the fact that each has a special coordinate whose removal makes it not smug, and using these coordinates to find further variants of each S_i with new special coordinates.

▶ **Lemma 16.** *Let $\frac{g}{k} < \frac{s}{s+1}$ (equivalently, $s > \frac{g}{k-g}$). Every polymorphism of $(1, g, k)$-SetSAT on s has a smug set of size at most g.*

Proof. Consider a polymorphism $f \colon [s + 1]^m \to [s + 1]$ of $(1, g, k)$-SetSAT. We prove by induction on t that there is a smug set of size at most $t - 1$, or there is a sequence of smug sets $S_1, \dots, S_t$ and a set T such that (see Figure 2):

(i) $|T| = t$ and $|T \cap S_i| = 1$ for $i = 1 \dots t$ (hence $S_i \cap T \neq S_{i'} \cap T$ for $i \neq i'$);
(ii) $S_i \setminus T$ is not smug for $i = 1 \dots t$;
(iii) $S_i \cap S_{i'} = \emptyset$ if $i \not\equiv i' \mod s$;
(iv) $S_i \supseteq S_{i-s} \setminus T$ for $i > s$.

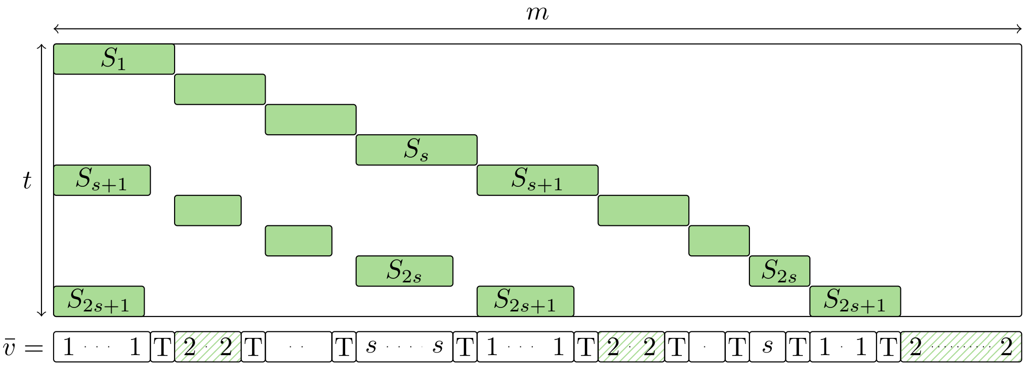

m
t
S_1
S_s
S_{s+1}
S_{s+1}
S_{2s}
S_{2s}
S_{2s+1}
S_{2s+1}
S_{2s+1}
$\bar{v} =$ 1 · · · 1 T 2 · 2 T · · T s · · · s T 1 · · · 1 T 2 · 2 T · T s T 1 · 1 T 2 · · · · · · · · 2

Figure 2 Illustration of smug sets obtained in the proof of Lemma 16. Each row represents one of the sets in the sequence $S_1, \dots, S_t$. The set T is formed by coordinates with a T and get values $s+1$. The vector $\bar{v}$ is used to find the next row S_{t+1}.

By Lemma 15 we can start with $t = s$ (by taking any T containing one coordinate from each S_i). Suppose the above is true for $t \geq s$ and let us prove the same for $t+1$. If there is a smug set of size at most t then we are done, so assume that T is not smug. Let $\bar{v} \in [s+1]^m$ be the input vector with value $s+1$ on T and different values from $\{1, \dots, s\}$ on $S_{t-i} \setminus T$ for $i = 0 \dots s-2$ and on the set of remaining coordinates $R := [m] \setminus (S_t \cup \dots \cup S_{t-s+2} \cup T)$. Then by (ii), R is smug.

Observe that R contains $S_{t-s+1} \setminus T$, because $S_t, \dots, S_{t-s+2}, T$ are disjoint from that set by (iii). We define S_{t+1} to be a minimal subset of R among smug sets containing $S_{t-s+1} \setminus T$. By (ii) $S_{t-s+1} \setminus T$ itself is not smug, so there exists some coordinate ℓ in $S_{t+1} \setminus S_{t-s+1}$. We choose it arbitrarily and set $T' := T \cup \{\ell\}$.

We claim that the sequence of smug sets $S_1, \dots, S_{t+1}$ and the set T' satisfy the above conditions. By minimality $S_{t+1} \setminus T'$ is not smug, so it satisfies (ii) and by definition it satisfies (iv). The set S_{t+1} is disjoint from $S_t, \dots, S_{t-s+2}, T$, because R was. It is also disjoint from S_i for $i \not\equiv t+1 \mod s$, because for every such i, $S_i \setminus T$ is contained in one of $S_t, \dots, S_{t-s+2}$; this proves (iii). In particular ℓ is not contained in any of these sets, and since it is not contained in S_{t-s+1}, it is in fact not contained in any S_i with $i < t+1$. Hence $|T'| = t$ and $|T' \cap S_i| = |T \cap S_i| = 1$ for $i < t+1$. Clearly also $|T' \cap S_{t+1}| = |\{\ell\}| = 1$. Therefore, (i) is satisfied, concluding the inductive proof.

Let us now consider sets as guaranteed above for $t = g+1$ (assuming there is no smug set of size at most g). Let $\bar{v} \in [s+1]^m$ be the input vector with value $i+1$ on $S_{t-i} \setminus T$ for $i = 0 \dots s-1$, and value $s+1$ on the remaining coordinates $R := ([m] \setminus (S_t \cup \dots \cup S_{t-s+1})) \cup T$. By (ii) the sets $S_{t-i} \setminus T$ are not smug, so R is smug. We claim that the multiset obtained from $\{S_1, \dots, S_t\}$ by adding $(k-g-1)$ copies of the set R contradicts Lemma 13: that is, each coordinate in $[m]$ is covered at most $k-g$ times by this multiset.

Consider first the coordinates contained in R. By definition of R, they are disjoint from $S_{t-i} \setminus T$ for $i = 0 \dots s-1$. By (iv), they are also disjoint from all sets $S_i \setminus T$ for $i = 0 \dots t$, because every such set is contained in one of the former. Hence if a coordinate in R is also contained in one of $S_1, \dots, S_t$, then it is contained in T and therefore in at most one of $S_1, \dots, S_t$, by (i). In total, it is thus covered at most $(k-g-1)+1 = k-g$ times.

Consider now coordinates outside of R. By (iii), they can be covered only by sets S_i with congruent indices $i \mod s$. Since $s > \frac{g}{k-g}$, we have $s(k-g) > g$, so there are $t = g+1 \leq s(k-g)$ distinct indices in total in $\{1, \ldots, t\}$. Hence at most $k-g$ of them can be pairwise congruent to each other $\mod s$. Thus coordinates outside of R are also covered at most $k-g$ times. ◀

This concludes the proof that smug sets satisfy the first condition of Corollary 12 for polymorphisms of $(1, g, k)$-SetSAT with set size s and domain size $s+1$, assuming $\frac{g}{k} < \frac{s}{s+1}$. Therefore, the problem is NP-hard. The full dichotomy then follows by simple reductions, see [9, Corollary A.4].

References

1 Per Austrin, Amey Bhangale, and Aditya Potukuchi. Improved inapproximability of rainbow coloring. In *Proceedings of the 31st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'20)*, pages 1479–1495. SIAM, 2020. doi:10.1137/1.9781611975994.90.

2 Per Austrin, Venkatesan Guruswami, and Johan Håstad. (2+ϵ)-SAT Is NP-hard. *SIAM Journal on Computing*, 46(5):1554–1573, 2017. doi:10.1137/15M1006507.

3 Libor Barto, Jakub Bulín, Andrei A. Krokhin, and Jakub Opršal. Algebraic approach to promise constraint satisfaction. *arXiv:1811.00970*, 2019. Version 3, 21 June 2019. arXiv:1811.00970.

4 Libor Barto, Andrei Krokhin, and Ross Willard. Polymorphisms, and how to use them. In Andrei Krokhin and Stanislav Živný, editors, *Complexity and approximability of Constraint Satisfaction Problems*, volume 7 of *Dagstuhl Follow-Ups*, pages 1–44. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/DFU.Vol7.15301.i.

5 Joshua Brakensiek and Venkatesan Guruswami. New Hardness Results for Graph and Hypergraph Colorings. In *Proceedings of the 31st Conference on Computational Complexity (CCC'16)*, volume 50 of *LIPIcs*, pages 14:1–14:27. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.CCC.2016.14.

6 Joshua Brakensiek and Venkatesan Guruswami. Promise Constraint Satisfaction: Structure Theory and a Symmetric Boolean Dichotomy. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'18)*, pages 1782–1801. SIAM, 2018. doi:10.1137/1.9781611975031.117.

7 Joshua Brakensiek and Venkatesan Guruswami. An Algorithmic Blend of LPs and Ring Equations for Promise CSPs. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'19)*, pages 436–455. SIAM, 2019. doi:10.1137/1.9781611975482.28.

8 Joshua Brakensiek and Venkatesan Guruswami. Symmetric polymorphisms and efficient decidability of PCSPs. In *Proceedings of the 31st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'20)*, pages 297–304. SIAM, 2020. doi:10.1137/1.9781611975994.18.

9 Alex Brandts, Marcin Wrochna, and Stanislav Živný. The complexity of promise SAT on non-Boolean domains. *CoRR*, abs/1911.09065, 2019. arXiv:1911.09065.

10 Jakub Bulín, Andrei A. Krokhin, and Jakub Opršal. Algebraic approach to promise constraint satisfaction. In *Proceedings of the 51st Annual ACM SIGACT Symposium on the Theory of Computing (STOC'19)*, pages 602–613. ACM, 2019. doi:10.1145/3313276.3316300.

11 Hubie Chen and Martin Grohe. Constraint satisfaction with succinctly specified relations. *J. Comput. Syst. Sci.*, 76(8):847–860, 2010. doi:10.1016/j.jcss.2010.04.003.

12 Stephen Cook. The complexity of theorem proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing (STOC'71)*, pages 151–158, 1971. doi:10.1145/800157.805047.

13 Irit Dinur, Venkatesan Guruswami, Subhash Khot, and Oded Regev. A new multilayered PCP and the hardness of hypergraph vertex cover. *SIAM J. Comput.*, 34(5):1129–1146, 2005. doi:10.1137/S0097539704443057.

14 Irit Dinur, Oded Regev, and Clifford D. Smyth. The hardness of 3-uniform hypergraph coloring. *Combinatorica*, 25(5):519–535, 2005. doi:10.1007/s00493-005-0032-4.

15 Miron Ficak, Marcin Kozik, Miroslav Olšák, and Szymon Stankiewicz. Dichotomy for Symmetric Boolean PCSPs. In *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming (ICALP'19)*, volume 132, pages 57:1–57:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.57.

16 M. R. Garey and David S. Johnson. The complexity of near-optimal graph coloring. *J. ACM*, 23(1):43–49, 1976. doi:10.1145/321921.321926.

17 Àngel J. Gil, Miki Hermann, Gernot Salzer, and Bruno Zanuttini. Efficient algorithms for description problems over finite totally ordered domains. *SIAM J. Comput.*, 38(3):922–945, 2008. doi:10.1137/050635900.

18 Venkatesan Guruswami and Sai Sandeep. Rainbow coloring hardness via low sensitivity polymorphisms. *SIAM J. Discrete Math.*, 34(1):520–537, 2020. doi:10.1137/19M127731X.

19 Subhash Khot. Hardness results for coloring 3-colorable 3-uniform hypergraphs. In *Proc. 43rd Symposium on Foundations of Computer Science (FOCS 2002)*, pages 23–32. IEEE Computer Society, 2002. doi:10.1109/SFCS.2002.1181879.

20 Andrei Krokhin and Jakub Opršal. The complexity of 3-colouring H-colourable graphs. In *Proceedings of the 60th Annual IEEE Symposium on Foundations of Computer Science (FOCS'19)*, pages 1227–1239. IEEE, 2019. doi:10.1109/FOCS.2019.00076.

21 Melven Krom. The decision problem for a class of first-order formulas in which all disjunctions are binary. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 13:15–20, 1967. doi:10.1002/malq.19670130104.

22 Leonid Levin. Universal search problems (in Russian). *Problems of Information Transmission (in Russian)*, 9(3):115–116, 1973. URL: http://www.mathnet.ru/links/84e4c96b64cc22a33a4bdae3d4815887/ppi914.pdf.

23 Christos H. Papadimitriou. On selecting a satisfying truth assignment. In *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science (FOCS'91)*, pages 163–169. IEEE Computer Society, 1991. doi:10.1109/SFCS.1991.185365.

A Simple Dynamization of Trapezoidal Point Location in Planar Subdivisions

Milutin Brankovic
University of Sydney, Australia
mbra7655@uni.sydney.edu.au

Nikola Grujic
University of Sydney, Australia
ngru0072@uni.sydney.edu.au

André van Renssen
University of Sydney, Australia
andre.vanrenssen@sydney.edu.au

Martin P. Seybold
University of Sydney, Australia
mpseybold@gmail.com

Abstract

We study how to dynamize the Trapezoidal Search Tree (TST) – a well known randomized point location structure for planar subdivisions of kinetic line segments.

Our approach naturally extends incremental leaf-level insertions to recursive methods and allows adaptation for the online setting. The dynamization carries over to the Trapezoidal Search DAG (TSD), which has linear size and logarithmic point location costs with high probability. On a set S of non-crossing segments, each TST update performs expected $\mathcal{O}(\log^2 |S|)$ operations and each TSD update performs expected $\mathcal{O}(\log |S|)$ operations.

We demonstrate the practicality of our method with an open-source implementation, based on the Computational Geometry Algorithms Library, and experiments on the update performance.

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

Keywords and phrases Dynamization, Trapezoidal Search Tree, Trapezoidal Search DAG, Backward Analysis, Point Location, Planar Subdivision, Treap, Order-maintenance

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.18

Category Track A: Algorithms, Complexity and Games

Related Version https://arxiv.org/abs/1912.03389

Supplementary Material https://github.com/milutinB/dynamic_trapezoidal_map_impl

Funding *Martin P. Seybold*: Supported under the Australian Research Council Discovery Projects funding scheme (project number DP180102870).

Acknowledgements We want to thank the University of Sydney's undergraduate Research Internship Program, which fostered the pursuit of this topic.

1 Introduction

Our results on vertical ray shooting in dynamic planar subdivisions and dynamic order maintenance originate from the search for a simple and practical data structure, with guarantees, that facilitate building the intersection graph of thin geometric objects, which is a natural precursor in meshing fracture networks [14]. This note studies how to structure 2D kinetic and non-kinetic line segment data in a fully-dynamic online setting that supports

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 18; pp. 18:1–18:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

intersection reporting. Beside sweeping plane applications, such data structures are also of interest for map matching problems [22] and for point location queries in dynamic, constrained Delaunay triangulations [4].

Most works term the "dynamic point location problem" as to quickly find the first edge directly above or below a query point, which is vertical ray shooting queries over a dynamic set of edges. We also use this unfortunate terminology throughout, though there are works (e.g. [18]) that solve the seemingly more complicated problem of deciding if two query points are in the same face of the dynamic subdivision. Planar subdivisions are often categorized by additional geometric properties on the subdivisions' faces. Authors frequently consider cases where all faces are convex, or all faces intersect horizontal lines at most twice (monotone faces). Clearly, monotone faces and faces' boundaries formed by monotone curves (e.g. line segments) are distinct properties.

Trapezoids are a key ingredient to simplify the study of geometric problems concerning sets of line segments S. We use three well known [2, 11, 15, 20] and closely related [13] structures. These are (i) the trapezoidal decomposition $\mathcal{A}(S)$ of a plane induced by S, (ii) the Trapezoidal Search Tree (TST), and (iii) the Trapezoidal Search DAG (TSD) (see Section 2). TST $\mathcal{T}(S,\pi)$ and TSD $\mathcal{D}(S,\pi)$ stem from deterministic algorithms, that incrementally insert the segments in S according to a, typically random, permutation π over S. Decomposition $\mathcal{A}(S)$ has size $\mathcal{O}(|S|+k)$, where k denotes the number of *crossing pairs* of segments in S. Since segment boundaries of a general planar subdivision are non-crossing ($k=0$), the refined decomposition $\mathcal{A}(S)$ of the boundary segments S is within a constant of the subdivision's size, regardless of the face shape.

Chiang and Tamassia's 1992 survey [9, Chapter 6] reviews several dynamizations for planar subdivisions. In [10], the two authors describe a dynamization for the special case of a trapezoidal decomposition $\mathcal{A}$ of monotone subdivisions. Using several tree structures, they achieve $\mathcal{O}(\log|S|)$ point location query time within an $\mathcal{O}(|S|\log|S|)$ size structure, which allows fully-dynamic updates in amortized $\mathcal{O}(\log^2|S|)$ operations. Also for monotone subdivisions, Goodrich and Tamassia show with [12] how to maintain interlaced, monotone spanning trees over the edges of the planar graph and its dual graph. This leads to an $\mathcal{O}(|S|)$ sized structure with an update time of $O(\log|S|+m)$, for insertion of a monotone chain of length m, but point location queries take $\mathcal{O}(\log^2|S|)$ operations. Cheng and Janardan [8] show how to achieve equal bounds for more general, connected subdivisions.

Most recent approaches that achieve *online, fully-dynamic updates* within $\mathcal{O}(|S|)$ size and $\mathcal{O}(\log|S|)$ query time are extensions of the works of Baumgarten et al. [5] that uses dynamic fractional cascading to enable vertical ray shooting queries. Arge et al. [3] show a trade-off improvement, i.e. raising the fan-out of tree nodes to $\Theta(\log^{\varepsilon}|S|)$ allows to lower the point location query time bound to $\mathcal{O}(\log|S|)$. However, insertions take $\mathcal{O}(\log^{1+\varepsilon}|S|)$ amortized time and deletions take $\mathcal{O}(\log^{2+\varepsilon}|S|)$ amortized time per update. Chan and Nekrich [7] added Multi-Colored Segment Trees to the approach and applied de-randomization and de-amortization techniques to finally derive update bounds of $\mathcal{O}(\log^{1+\varepsilon}|S|)$ for both, insertions and deletions. To our knowledge, these are currently the best bounds on the pointer machine (see Table 1 in [7] for an overview). Refining this approach, Munro and Nekrich [17] recently analyzed the I/O counts in an external memory model with block sizes of $\Omega(\log^8 n)$.

The *Randomized Incremental Construction (RIC)* of TSDs provides simple, and thereby practical, point location structures of expected $\mathcal{O}(|S|)$ size whose longest search path is with high probability also in $\mathcal{O}(\log|S|)$. Their analysis is a celebrated result of Mulmuley [15], who argues on a random experiment that draws geometric objects, and Seidel [20], who uses a backward-analysis to bound the expected costs. Vastly unmentioned is Mulmuley's book [16], which extends TSDs to a specialized (offline) dynamic setting of a random insert/delete string for a fixed set of segments S, in which each element has a probability of $1/|S|$ to appear in each update request. The approach thrives on a randomly chosen, fixed order on S, rotations in the structure, and regular leaf-level insertions and deletions. Several distributed book chapters finally show that executing such a (+/-)-sequence of m updates has a total expected cost of $\mathcal{O}(m \log m + k \log m)$, where k is the number of intersections in the fixed set S. Schwarzkopf [19] independently describes a similar update model that also leads to a dynamization. His analysis provides an expected $\mathcal{O}(\log^2 n)$ time bound for update operations from the sequence, where n denotes the current number of non-crossing segments, but the expected point location query bound is w.h.p. in $\mathcal{O}(\log^2 n)$.

In [2], Agarwal et al. describe a TST over a static set of segments, that move over time. Their approach is based on a RIC of TSTs with expected $\mathcal{O}(|S| \log |S| + k)$ size whose depth is w.h.p in $\mathcal{O}(\log |S|)$. The authors mention a bound on the expected construction time of $\mathcal{O}(|S| \log^2 |S| + k \log |S|)$ and show that randomization is crucial to resolve a structural change, due to kinetic (adversarial) movement of the line segments, in expected $\mathcal{O}(\log |S|)$ operations. One may, however, form constant velocity trajectories for (point-like) segments that lead to $\Omega(n\sqrt{n})$ many topological updates for any binary space partition method [1].

Insightful work of Hemmer et al. [13] recently revealed a certain bijection between the search paths of $\mathcal{T}(S, \pi)$ and the "un-folded" search paths of $\mathcal{D}(S, \pi)$ by means of a structural induction argument (along fixed π) on the two incremental algorithms. The authors show Las Vegas type point location guarantees for the RIC of TSDs by bounding computation time of the length of a longest search path in a TSD by means of *the size* of the related TST.

1.1 Paper Organization and Section Contributions

Our surprisingly simple, thereby practical (c.f. Section 6), approach uses only one search structure and performs insertions and deletions directly on higher levels of it. Since we maintain randomness of the priority orders on the segment sets, the expected size and expected point location query bounds are retained for each state of the structures. Section 2 reviews the basic RIC of TSTs and TSDs in a unified notation and we introduce our natural, recursive extensions of those basic primitives in Section 3.

Our algorithms therein enable fully-dynamic updates in both data structures in the offline setting, i.e. the complete ground set of segments S is known in advance. Our extension to the online setting provides a simpler, randomized solution to the dynamic order-maintenance problem with expected update cost of $\mathcal{O}(\log |S|)$ based on an arithmetic coding scheme on Treaps (c.f. Section 5). Known solutions to this problem provide $\mathcal{O}(1)$ amortized update costs [24] or deterministic worst case $\mathcal{O}(\log^2 n)$ update cost based on de-amortization techniques [6]. Our algorithms are suitable for pointer machines with arithmetic since they only compare the spatial location of the input points, intersection points, and priority values. Our need of constant time arithmetic operations on fixed-point numbers with $\mathcal{O}(\log |S|)$-bits is solely due to the online order-maintanance problem. In particular, the method does not require indirect addressing of Random Access Memory.

To our knowledge, we provide the first analysis for the update costs of an *online fully-dynamic* data structure that originates from the classic RIC method. Though our Section 4 argues for non-crossing segments only, our geometric interpretation of TST nodes provides a 4-ply covering property that enables new proof methods. We show a new form of Backward-Analysis on the permutation space and a bound on simple "segment searches" for affected search nodes in the structure (avoiding maintenance of so-called conflict lists or $\mathcal{A}$). For TSTs, we obtain expected update costs of $\mathcal{O}(\log^2 |S|)$ for insertion and deletion. Which we

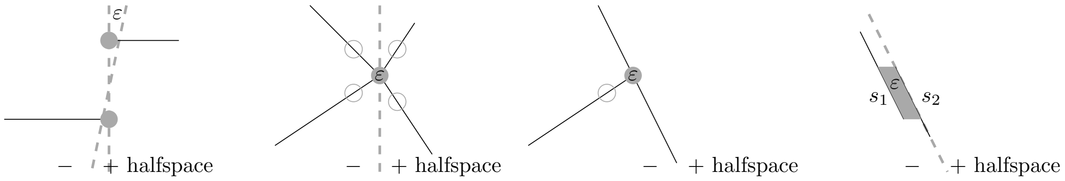


Figure 1 Standard shear transformation (left) and three refined tie-breaking rules for point/point meets, point/segment meets (geometric), and segment/segment overlaps (lexicographic).

can improve further for TSDs to an expected update cost of $\mathcal{O}(\log|S|)$ for insertion and deletion. Hence randomization allows improvements on the amortized deletion bound of $\mathcal{O}(\log^{2+\varepsilon}|S|)$ in [3] and on the $\mathcal{O}(\log^{1+\varepsilon}|S|)$ deterministic update bounds in [7], without losing the size or the point location query bounds.

2 Basic Definitions, Algorithms and Properties

A segment $s = \overline{pq}$ between two distinct points $p, q \in \mathbb{R}^2$ is the set $\{p + \alpha(q - p) : \alpha \in [0, 1]\}$ of points in the Euclidean plane. Two segments are called disjoint if $s \cap s' = \emptyset$ and *overlapping* if $s \cap s'$ contains more than one point. For the case of one common point, we say that two segments *meet* if it is an endpoint that is contained in the other segment and otherwise they are called *intersecting* (or *crossing*). The segment boundaries of a general planar subdivision, for example, have no intersections, but may well contain points in which arbitrarily many segments meet. A line, parallel to the y-axis, through a point is called a *vertical-cut* (e.g. through an end or intersection point) and the line through the endpoints of a segment is called an *edge-cut*. To avoid ambiguity, we denote the open halfspaces of a cut with "$-$" and "$+$". For vertical-cuts $-$ denotes the one with lower x-coordinates and for edge-cuts $-$ denotes the one with the lower values of the y-axis.

As in [11, Chapter 6], our presentation assumes that no two distinct points (end or intersection) have exactly the same x-coordinate, unless they are a common endpoint in which the segments meet. As usual, this assumption is resolved by an implicit, infinitesimal shear transformation which translates to a tie-breaking rule in the geometric orientation predicate that resolves to comparing the y-coordinates. A simple extension of the halfspace partitioning for the remaining cases (of segments and cuts due to a segment) conceptually moves the points infinitesimally away from the degeneracy along their segment for a consistent decision. To also assign co-linear cases consistently we finally resolve to lexicographic comparison (c.f. Figure 1). We also assume that the whole domain is bounded by a very large rectangle, which is also a trapezoid. Extensions to unbounded problems are commonly achieved by careful case treatment of unbounded faces of arrangements (e.g. in [23]).

Our combinatorial analysis (c.f. Section 4) of the proposed dynamization requires more precise terms to establish a geometric correspondence, which we introduce now in a unified review of the well known incremental algorithms. We frequently identify a permutation π from the set of permutations $\mathbf{P}(S)$ over S as bijection $\pi : S \to \{1, \ldots, |S|\}$ and call $\pi(s)$ the *priority* of a segment $s \in S$.

2.1 Trapezoidal Search Trees $\mathcal{T}(S, \pi)$

A TST over a trapezoidal domain $\Delta \subseteq \mathbb{R}^2$ is a certain hierarchical Binary Space Partition with vertical cuts and edge cuts induced by the segments in S. These rooted trees are defined by inductively applying the deterministic insertion algorithm for the segments of S in *ascending priority* order. Hence, the structure is uniquely determined by (S, π).

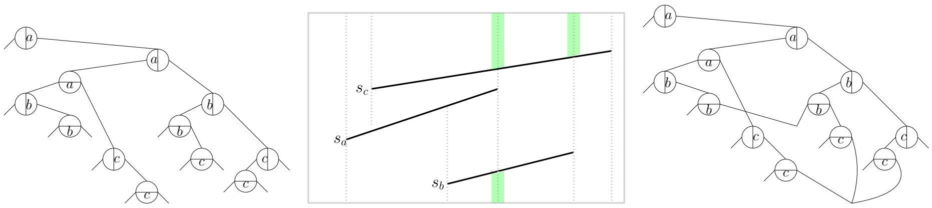


Figure 2 Example of TST Leaf-Insert (left) and TSD Leaf-Insert (right) for segments with priority order $s_a < s_b < s_c$ (middle). The vertical merges of Algorithm 2 are indicated in green.

Every node v of the binary tree is associated with a trapezoidal region $\Delta(v) \subseteq \Delta$ and, on the empty set of segments, $\mathcal{T}(\{\}, ())$ contains only the root r with $\Delta(r) = \Delta$. In TSTs, every non-leaf node v has two child nodes v^-, v^+ and stores a cut that signifies the halfspace partition of $\Delta(v)$ in $\Delta(v^-)$ and $\Delta(v^+)$, which enables point-location search descents. We say that this cut *destroys* the trapezoidal region $\Delta(v)$ and use this to extend the priority assignment to also comprise tree nodes.

We denote the priority of node v with $p(v)$. More precisely, leaves have priority $+\infty$ and non-leaf nodes have the priority of the segment whose cut insertion destroys the node. E.g. the cuts that constitute the (3 or 4) boundaries of $\Delta(v)$ have smaller priorities than $p(v)$.

Algorithm 1 Leaf-Insert($\mathcal{T}, s$).

1. Search for the leaf nodes $L = \{u_1, \dots, u_l\} \subseteq \mathcal{T}$ with $\Delta(u_i) \cap s \neq \emptyset$.
2. Create refined slabs of these regions by vertically partitioning each $\Delta(u_i)$ with cuts due to endpoints of s or intersection points of s with edge-cuts bounding $\Delta(u_i)$.
3. Partition the intersected slab regions further with the edge cut through s.

See Algorithm 1 for the steps to insert the next segment s in a TST $\mathcal{T}$. Note that nodes' priorities are monotonically increasing on paths from the root. Procedure Leaf-Insert inserts $3, 2$ or 1 cuts on the region of a leaf and the edge-cut is always the last cut that is performed. We refer to these patterns as vertical-vertical-edge (VVE), vertical-edge (VE), and edge (E) destruction. To remove unnecessary ambiguity in the tree structure, we use the additional convention that in a VVE-destruction, the left of the two vertical cuts is inserted first (e.g. as parent of the right vertical cut). See Figure 2 for an example.

▶ **Theorem 2.1** (TST size and depth [2]). *Let S be a set of segments and k_S the number of intersecting pairs among them. The expected size of a TST over S is bounded by*

$$\mathop{\mathbb{E}}_{\pi \in \mathbf{P}(S)} |\mathcal{T}(S, \pi)| = \mathcal{O}(|S| \log |S| + k_S) \ .$$

The expected leaf depth is $\mathcal{O}(\log |S|)$ and the maximum leaf depth of $\mathcal{T}$ is w.h.p. $\mathcal{O}(\log |S|)$.

Clearly, any TST over a set of segments S contains at least $\Omega(|S| + k_S)$ nodes and there are certain instances that may have size $\Omega(|S|^3)$. Agarwal et al. [2] mention an upper bound of $\mathcal{O}(|S| \log^2 |S| + k_S \log |S|)$ expected time for the purely incremental construction.

2.2 Trapezoidal Decomposition $\mathcal{A}(S)$ and Search DAG $\mathcal{D}(S, \pi)$

One may save space in such a search structure by considering a certain planar subdivision $\mathcal{A}(S)$ that is induced from a set of segments S. Given aforementioned discussion of degeneracies, this subdivision is defined by emitting two vertical rays (in negative and positive y-direction) from each end or intersection point until the ray meets the first segment or the bounding rectangle. Each face of $\mathcal{A}(S)$ is a trapezoid with 3 or 4 boundary edges and standard double counting establishes a size of $\mathcal{O}(|S| + k_S)$ for these planar decompositions. Note that $\mathcal{A}(S)$ is independent of segment priorities.

Mulmuley [15] and Seidel [20] consider the incremental process of inserting an additional segment s and obtaining $\mathcal{A}(S \cup \{s\})$ from $\mathcal{A}(S)$. Algorithm 2 shows the additional merging phase, to "contract" pre-existing vertical-cuts (between the regions of the nodes in L) towards their emission point until they meet s, to maintain the property that the leaf regions coincide with $\mathcal{A}(S)$. Hence, the planar subdivision of the leaves of a TST, i.e. $\{\Delta(v) \subseteq \mathbb{R}^2 : v \in \mathcal{T}(S, \pi), v \text{ is leaf}\}$, is a *refinement* of $\mathcal{A}(S)$, regardless of π.

Algorithm 2 Leaf-Insert($\mathcal{D}, s$).

```
1. Search for the leaf nodes $L = \{u_1, \dots, u_l\} \subseteq \mathcal{D}$ with $\Delta(u_i) \cap s \neq \emptyset$.
2. Create refined slabs of these regions by vertically partitioning each $\Delta(u_i)$ with cuts due
   to endpoints of $s$ or the intersection points of $s$ with edge-cuts bounding $\Delta(u_i)$.
3. Partition the intersected slab regions further with the edge cut through $s$.
4. Scan over the vertical cuts between nodes in $L$ that cross $s$.
   Merge the vertical cuts of the refined regions whose emission point is now blocked by $s$.
```

Though the resulting search graph is a DAG (also known as History DAG), the basic data stored with a node is still identical with the binary TST tree nodes (e.g. Figure 2). One key ingredient for the well known, expected construction time of $\mathcal{O}(|S| \log |S| + k_S)$, is to perform the search for affected leaves (c.f. Step 1) quickly. A way to achieve this, without prior knowledge of all future segment insertions, is to also maintain an explicit graph representation of the planar subdivision $\mathcal{A}(S)$ of the segments S that are currently contained in the structure. This trick allows to perform only one point location query for, e.g., the left endpoint of s, followed by a walk along s through $\mathcal{A}(S)$. Our Section 3.1 solves this differently. We summarize the well known aspects of this incremental algorithm in the following statement.

▶ **Theorem 2.2** (TSD size and depth [11, 15, 20]). *Let S be a set of segments and k_S the number of intersecting pairs among them. The expected size of a TSD over S is bounded by*

$$\mathop{\mathbb{E}}_{\pi \in \mathbf{P}(S)} |\mathcal{D}(S, \pi)| = \mathcal{O}(|S| + k_S) \ .$$

The expected search path length is $\mathcal{O}(\log |S|)$ and the maximum search path length of $\mathcal{D}$ is w.h.p. $\mathcal{O}(\log |S|)$.

The TST and TSD structure also allow simple deletions of segments in a certain decremental setting, that deletes all segments in *descending* priority order. That is, the modifications that the structure undergoes in the steps of Algorithm 2 are simply undone in exactly the reverse order (steps 4 to 2), causing the same amount of work. This is what Seidel's classic backward-analysis, that analyses dropping the last element instead of appending it, exploits.

The insightful work of [13], recently revealed a certain bijection between the search paths of $\mathcal{T}(S, \pi)$ and the (valid) search paths in $\mathcal{D}(S, \pi)$ by means of a structural induction argument (along fixed π) on the two incremental algorithms. This allows them to bound the runtime of computing the length of a longest search path in a TSD $\mathcal{D}(S, \pi)$ by means of the size of the related TST $\mathcal{T}(S, \pi)$. Based on this technique, our analysis of the update time in TSTs (c.f. Section 4) carries over to an upper bound in TSDs as well.

The main geometric difference between the structures is that in TSTs the region of a node is always a subset of its parents' region, whereas in TSDs the region of a node may extend further to the left and right, due to vertical merges, but not across the top and bottom boundaries. We now describe our recursive algorithms for vertical partitions, vertical merges, edge partitions, and edge merges that operate on intermediary priority levels.

3 Recursive Primitives for Dynamic Updates

Characteristic for our approach is that insertions and deletions are performed directly on higher levels of the search structures rather than solely on leaves. We first describe the recursive primitives for inserting a new segment s in the structure of $\mathcal{T}(S, \pi)$.

We choose a random position $p \in \{1, \dots, |S| + 1\}$ uniformly in which we emplace the element s, between $(p-1)$ and p, in the sequence π, calling the resulting priority order π'. Our recursive algorithms then update the structure $\mathcal{T}(S, \pi)$ exactly to $\mathcal{T}(S \cup \{s\}, \pi')$. More precisely, we restrict the search (c.f. Step 1 of Algorithms 1 and 2) to those nodes with priority smaller than p, which leads to a set L of affected subtree roots whose priorities are larger than p in π'. We conceptually "hang out" these nodes by creating a copy L' of them and reverting those in L to leaves, temporarily. The insertion then proceeds in the same sequence as on regular leaves, but every binary space partition on nodes of L is accompanied by a matching call with the recursive primitive on the respective subtree of the node's copy in L'. After this process, we have a matching root in L' for each leaf that was created below L, which we then "hang in" instead of the simple leaf. Given the close relation of the two structures, it turns out that our recursive primitives, with few changes, already provide the necessities for dynamic updates of TSDs.

This and the following section assume that priority value comparisons, i.e. $\pi'(s') > \pi'(s)$, are decidable in constant time. Section 5 shows a new and simple solution, based on Treaps, to solve the dynamic order maintenance problem sufficiently fast for the online setting. Given the discussion above on the possible E-, VE- and VVE-destruction patterns, we simplify notation in this section by denoting with Descend(v) the tuple of $0, 2, 3$, or 4 descendants with the next higher priority value than $p(v)$. E.g. Descend$(v) = ()$ if v is a leaf, Descend$(v) = (v_a, v_b)$ if v underwent an E-destruction, Descend$(v) = (v_l, v_a, v_b)$ or Descend$(v) = (v_a, v_b, v_r)$ if v underwent a VE-destruction, and Descend$(v) = (v_l, v_a, v_b, v_r)$ if v underwent a VVE-destruction. With v_l, v_a, v_b and v_r we denote the respective left, above, below and right child of the node's destruction (e.g. Figure 3).

3.1 Priority Restricted Searches

We use the following top-down refinement process to derive the node set L, such that its nodes u_i are sorted by the sequence in which s stabs $\Delta(u_i)$ from left-to-right: First locate the search node u, with maximal $p(u) < p(s)$, that fully contains s in $\Delta(u)$. Place u in an initially empty list. Successively replace the leftmost node u in the list with $p(u) < p(s)$ with the nodes of Descend(u), whose region intersects s, until all node priorities exceed $p(s)$.

Note that nodes with priority larger than $p(s)$ are not refined and the spatial location of the regions of the nodes in Descend$(\cdot)$ allows us to easily keep the set L sorted by left-to-right stabbing sequence of s.

3.2 Recursive Vertical Partitions and Merges

Our methods are inspired by sorting algorithms on lists of integers, though they move cuts and child relations among nodes based on the nodes' priorities. We first introduce the recursive primitive V-Partition(u, q, v^-, v^+), where $q \in \mathbb{R}^2$ denotes the point that induces the vertical cut $c(q)$, u is an affected tree node, and v^-, v^+ the roots of the respective, initially empty, result trees. As outlined above, the primitives update the search structure to the state that regular leaf insertion (in ascending priority order) would have created under the presence of vertical cut $c(q)$ splitting $\Delta(u)$ in $\Delta^- \cup \Delta^+$. (A lengthy, rigorous proof fixes $\Delta(u)$ and performs a structural induction argument over the sequence of successive cuts that destroy the region.) See Figure 3 for the recursive cases by destruction patterns and Algorithm 3.

The recursive node visits are similar to a search for points on $c(q)$ and V-Partition performs at most two recursive calls per node in TSTs. Reverting these steps in exactly the reverse order provides the inverse operation V-Merge(u^-, u^+, q, v) on two trees with adjacent search regions (c.f. Algorithm 4). Note that the TSD primitives actually perform *fewer* recursive calls, since vertical cuts are contracted in this structure.

3.3 Recursive Edge Partitions and Merges

Based on the recursive primitives for vertical cuts, we now introduce the recursive edge partition Partition(u, c, v^-, v^+) on regions $\Delta(u)$ that are fully crossed by the edge cut c (c.f. Algorithm 5). As above, v^- and v^+ denote the initially empty result trees for the respective partition of $\Delta(u)$ in $\Delta^- \cup \Delta^+$.

If the edge cut c does not intersect the edge cut that destroys u, the recursive results on nodes of Descend(u) only need V-Merge calls on one side of c to produce the result. E.g. the left case in the figure of Algorithm 5: First a V-Merge call on (v_a^+, v_r^+) and then with v_l^+. For new intersections with c, let i denote the intersection point. We first use V-Partition(u_a, i, v_{al}, v_{ar}) and V-Partition(u_b, i, v_{bl}, v_{br}) to create trees v_{al}, v_{bl} and v_{ar}, v_{br} for the respective left and right sides of i, prior to the edge partition with c. Finally, we use V-Merge to combine those results properly. E.g. the two gray areas in right case in the figure of Algorithm 5.

Note that for segments from a planar subdivision, the treatment of intersections is not necessary. The inverse primitive for edge cut merges (of two adjacent regions) can be derived analogously, by executing the inverse primitives in exactly the reverse order.

4 Counting Search Nodes in Affected Regions

To improve readability in this section, we denote for singleton elements the set union $S \cup \{s\}$ with $S + s$ and the set difference $S \setminus \{s\}$ with $S - s$. With $\mathbf{P}(S)$ we denote the set of bijective mappings $\pi : S \to \{1, \ldots, |S|\}$. For $S' \subseteq S$ we use the predicate "$\pi(S') \leq |S'|$" to abbreviate that $\pi(s) \leq |S'|$ holds for each $s \in S'$, i.e. the permutation begins with the elements of S'.

The following definitions concern sets of segments and their (deterministic) induced trapezoidal subdivision $\mathcal{A}$ of the plane. For $s \in S \setminus B$, we define $F_B(s) \subseteq \mathbb{R}^2$ to denote the region of the faces in $\mathcal{A}(B)$ that are intersected by s, that is $F_B(s) = \bigcup_{\{f \in \mathcal{A}(B): f \cap s \neq \emptyset\}} f$. Moreover, for $s \in B$ we define the neighborhood zone $N_B(s) \subseteq \mathbb{R}^2$ to be the union of all faces of $\mathcal{A}(B)$ that are adjacent to s (e.g. s contributes as edge, vertex or intersection cut). Note that every point in $\mathbb{R}^2$ is in at most 4 neighborhood zones and $N_{B+s}(s) = F_B(s)$.

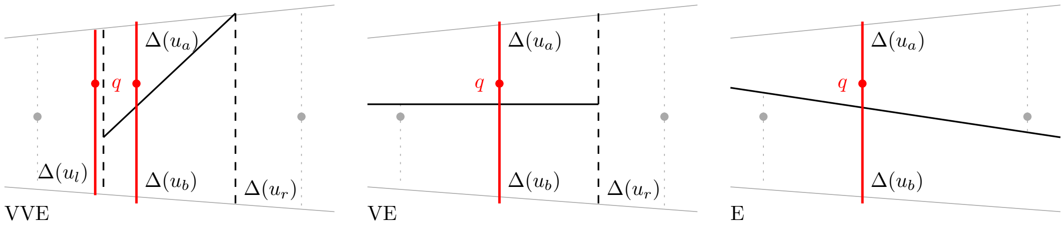


Figure 3 Cases during recursions of V-Partition(u, q, v^-, v^+) and V-Merge(u^-, u^+, q, v).

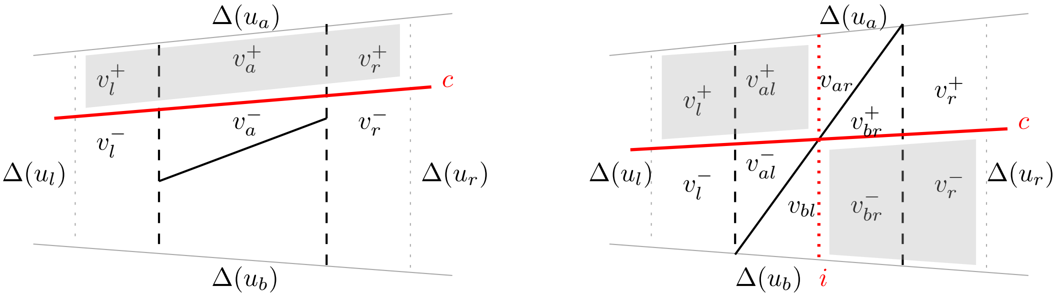
Algorithm 3 V-Partition(u, q, v^-, v^+): Assertion: $c(q)$ intersects $\Delta(u)$.

```
      Stop if $u$ is a leaf; Let $(u_l, u_a, u_b, u_r) := \text{Descend}(u)$.
TST:  IF $c(q)$ intersects $\Delta(u_a)$ and $\Delta(u_b)$
TSD:  IF $q \in \Delta(u_a)$                                    "$q \in \Delta(u_b)$" analogue
        If present, move $u_l$ and its parent's vertical cut in $v^-$.
        If present, move $u_r$ and its parent's vertical cut in $v^+$.
        Cut both unoccupied leaves, that is one child of $v^-$ and one of $v^+$, with the edge
        cut. Let $v_a^-, v_b^-, v_a^+, v_b^+$ denote these leaves.
        V-Partition$(u_a, q, v_a^-, v_a^+)$
  TST:  V-Partition$(u_b, q, v_b^-, v_b^+)$
  TSD:  Set both below pointers on $u_b$ instead of $v_b^-$ and $v_b^+$.
      ELSE IF $c(q)$ intersects $\Delta(u_l)$           "$c(q)$ intersects $\Delta(u_r)$" symmetrically
        Move $u_r$ (if present) , $u_a, u_b$ and their parents' cuts in $v^+$.
        Let $v_l^+$ denote the unoccupied leaf.
        V-Partition$(u_l, q, v^-, v_l^+)$
```

Algorithm 4 V-Merge(u^-, u^+, q, v): Assertion: $c(q)$ bounds $\Delta(u^-)$ and $\Delta(u^+)$.

```
      IF $u^-$ is leaf THEN Move contents of $u^+$ in $v$ and stop.      "$u^+$ leaf" analogue
      Let $(u_l^-, u_a^-, u_b^-, u_r^-) := \text{Descend}(u^-)$ and $(u_l^+, u_a^+, u_b^+, u_r^+) := \text{Descend}(u^+)$.
TST:  IF $p(u^-) = p(u^+)$
TSD:  IF $p(u^-) = p(u^+)$ and $u_b^- = u_b^+$                           "$u_a^- = u_a^+$" analogue
        If present, move $u_l^-$ and its parent's vertical cut in $v$.
        If present, move $u_r^+$ and its parent's vertical cut in the unoccupied leaf of $v$.
        Cut the new unoccupied leaf of $v$ with the edge cut. Let $v_a, v_b$ denote these
        leaves.
        V-Merge$(u_a^-, u_a^+, q, v_a)$
  TST:  V-Merge$(u_b^-, u_b^+, q, v_b)$
  TSD:  Set the below pointer on $u_b^-$ instead of $v_b$.
      ELSE IF $p(u^-) < p(u^+)$                                          "$>$" symmetrically
        Move $u_l^-$ (if present) , $u_a^-, u_b^-$ and their parents' cuts in $v$.
        Let $v_r$ denote the unoccupied leaf.
        V-Merge$(u_r^-, u^+, q, v_r)$
```

Algorithm 5 Partition(u, c, v^-, v^+): Assertion: c crosses $\Delta(u)$ entirely.



```
Stop if $u$ is a leaf.
Let $(u_l, u_a, u_b, u_r) :=$ Descend$(u)$ and $l_1, l_2$ denote the vertical cuts destroying $\Delta(u)$.
IF $c$ does not intersect $\Delta(u_b)$ properly                     "not $\Delta(u_a)$" analogue
    Allocate new nodes $v_l^-, v_l^+, v_a^-, v_a^+, v_r^-, v_r^+, v_{ar}^+$.
    Partition$(u_l, c, v_l^-, v_l^+)$; Partition$(u_a, c, v_a^-, v_a^+)$; Partition$(u_r, c, v_r^-, v_r^+)$
    V-Merge$(v_a^+, v_r^+, l_2, v_{ar}^+)$
    V-Merge$(v_l^+, v_{ar}^+, l_1, v^+)$
    Place $v_l^-, v_a^-, u_b, v_r^-$ below respective cuts under $v^-$.
ELSE IF edge cut of $u$ is steeper than $c$                  "less steep" symmetrically
    Let $i$ denote the vertical cut (induced by the intersection of the edge-cut and $c$).
    Allocate new nodes $v_l^-, v_l^+, v_r^-, v_r^+$ as well as $v_{ar}, v_{al}, v_{al}^-, v_{al}^+$ and $v_{bl}, v_{br}, v_{br}^-, v_{br}^+$.
    Partition$(u_l, c, v_l^+, v_l^-)$; Partition$(u_r, c, v_r^+, v_r^-)$
    V-Partition$(u_a, i, v_{al}, v_{ar})$; V-Partition$(u_b, i, v_{bl}, v_{br})$
    Partition$(v_{al}, c, v_{al}^-, v_{al}^+)$; Partition$(v_{br}, c, v_{br}^-, v_{br}^+)$

    Place V-Merge$(v_l^+, v_{al}^+, l_1, \cdot)$ as left child under cut $i$ below $v^+$.
    Place $v_r^+$ and its parent's cut in the unoccupied leaf of $v^+$.
    Cut the unoccupied leaf of $v^+$ with the edge cut of $u$.
    Place $v_{ar}$ and $v_{br}^+$ in these leaves.

    Place $v_l^-$ and its parent's cut in $v^-$.
    Place V-Merge$(b_{br}^-, v_r^-, l_2, \cdot)$ as right child under cut $i$ in $v^-$.
    Cut the unoccupied leaf of $v^-$ with the edge cut of $u$.
    Place $v_{al}^-$ and $v_{bl}$ in these leaves.
```

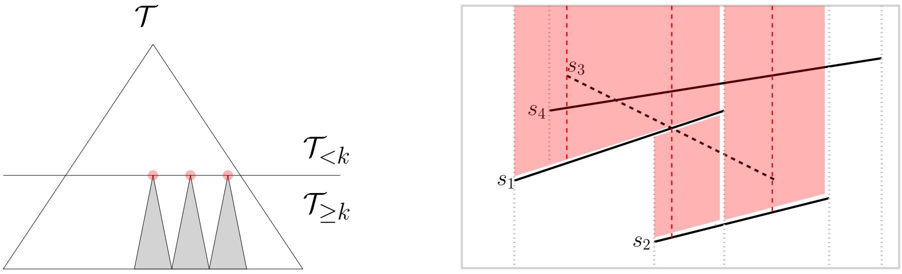
Given a TST $\mathcal{T}(S, \pi)$ and some priority value $r \in \{1, \ldots, |S|\}$, we consider partitions of its nodes in classes $\mathcal{T}_{<r}, \mathcal{T}_{=r}$, and $\mathcal{T}_{>r}$, having priority less, equal or larger than r. E.g. $\mathcal{T}_{<1} = \emptyset$ and $\mathcal{T}_{>r}$ always contains the leaf nodes. As introduced in Section 2.1, we identify every node v in $\mathcal{T}$ with its associated search region $\Delta(v) \subseteq \mathbb{R}^2$.

From the refinement property (c.f. Section 2.2), we have that the leaves' partition $\{\Delta(v) \subseteq \mathbb{R}^2 : v \in \mathcal{T}(S, \pi),\ p(v) = \infty\}$ of the domain is a (set theoretic) refinement of $\mathcal{A}(S)$, for any S and $\pi \in \mathbf{P}(S)$. This applies in particular for a set of segments $S_{\leq r} = \{s \in S : \pi(s) \leq r\}$. Since additional leaf-insertions only refine further, we have that the region $\Delta(v)$ of a node $v \in \mathcal{T}_{>r}$ is either contained in or disjoint from neighborhood zone $N_{S_{\leq r}}(s)$ for any $s \in S_{\leq r}$.

This provides the following 4-ply covering property. For any $B \subseteq S$ with a $\pi \in \mathbf{P}(S)$ such that $\pi(B) \leq |B|$, we have that

$$\sum_{s \in B} \Big| \{ v \in \mathcal{T}_{=l}(S, \pi) \ : \ \Delta(v) \subseteq N_B(s) \} \Big| \ \leq \ 4 \Big| \mathcal{T}_{=l}(S, \pi) \Big| \ , \tag{1}$$

for each $l > |B|$.



(a) Partition of nodes in $\mathcal{T}$ in $\mathcal{T}_{<k}$ and $\mathcal{T}_{\geq k}$ due to node priority. Affected nodes are in red and their subtrees in gray.

(b) The neighborhood zone of s_3 (shaded in red) is $F_{\{s_1,s_2\}}(s_3) = N_{\{s_1,s_2,s_3\}}(s_3)$.

Figure 4 Illustration of affected subtrees (left) and neighborhood zone refinement (right).

▶ **Definition 4.1** (Affected Nodes). *Given a segment $s \in S$ of designated priority rank $r \in \{1, \dots, |S|\}$, we call a node v of $\mathcal{T}_{\geq r}$ over $S - s$ affected if and only if $\Delta(v) \cap s \neq \emptyset$ and it is topmost in $\mathcal{T}$. That is, v has no parent u in $\mathcal{T}_{\geq r}$ with $p(u) \leq p(v)$.*

Clearly, for every $v \in \mathcal{T}_{\geq r}$ whose region $\Delta(v)$ intersects s, we have $\Delta(v) \subseteq F_{S_{<r}}(s)$ as well. In other words, the affected nodes correspond precisely to the leaves of $\mathcal{T}(S_{<r}, \pi)$, the tree over the first $r-1$ segments, that are intersected by s. See Figure 4a for an illustration and Figure 4b for an example. In this example, decomposition $\mathcal{A}(\{s_1, s_2\})$ has 7 faces of which s_3 intersects 3. The neighborhood region of s_3 is $F_{\{s_1,s_2\}}(s_3) = N_{\{s_1,s_2,s_3\}}(s_3)$ and shaded in red. The TST for $\left(\{s_1, s_2, s_4\}, \left(\begin{smallmatrix} s_1 & s_2 & s_4 \\ 1 & 2 & 3 \end{smallmatrix}\right)\right)$ has 13 leaf regions and the TSD has 10 leaves. Note that for TST nodes v with priority $p(v) \geq 3$, the region $\Delta(v)$ is either fully contained or outside the red zone.

Given a $\pi \in \mathbf{P}(S-s)$ and a value $p \in \{1, \dots, |S|\}$, we call the region $F_{\{s' \in (S-s)\ :\ \pi(s') < p\}}(s)$ the *p-neighborhood of s*.

▶ **Lemma 4.2** (Zone Covering). *Let S be a set of non-crossing segments, $p \in \{1, \dots, |S|\}$ a fixed value, and $s \in S$. For any $l \geq p$, the expected number of priority l nodes of a random TST over $S - s$ that are in the p-neighborhood of s is upper bounded with*

$$\mathop{\mathbb{E}}_{\pi \in \mathbf{P}(S-s)} \left| \left\{ v \in \mathcal{T}_{=l}(S-s, \pi) \ : \Delta(v) \subseteq F \right\} \right| \ \leq \ \frac{4}{p} \mathop{\mathbb{E}}_{\pi \in \mathbf{P}(S-s)} \left| \mathcal{T}_{=l}(S-s, \pi) \right| ,$$

where F denotes the p-neighborhood of s.

On a sequence of elements, we call the process to place a new element between the elements of positions $p-1$ and p *emplacing* at p and *dropping* is the reverse operation. E.g. old elements of index less than p remain and those of at least p are moved one position to the right. We perform a backward analysis and consider dropping the element at position p in sequences of $\mathbf{P}(S)$ instead of emplacing a $s \in S$ at position p in sequences of $\mathbf{P}(S-s)$.

Proof. The main idea is to group the sequences in $\mathbf{P}(S)$ by those having the same cardinality p subsets $B \in \binom{S}{p}$ in the first p entries to use that $\bigcup_{s \in B} N_B(s) = \mathbb{R}^2$, as in Eq. 1.

For $\pi \in \mathbf{P}(S)$, we denote with $\pi^{-1}(p) \in S$ the element that π maps to position p and with $\pi - s$ we denote the sequence of $\mathbf{P}(S-s)$ that results from dropping element s. To shorten notation, we define a random variable γ that counts the relevant nodes of a tree within a certain region $N \subseteq \mathbb{R}^2$, that is $\gamma(S, \pi, N) = |\{v \in \mathcal{T}_{=l}(S, \pi) \ : \ \Delta(v) \subseteq N\}|$.

We count the expected number of nodes within $\mathcal{T}(S - \pi^{-1}(p), \pi - \pi^{-1}(p))$, that fall into the region $N_{\{s \in S\ :\ \pi(s) \leq p\}}\left(\pi^{-1}(p)\right)$, which is the neighborhood zone of the element at position p among those with priority at most p.

$$
\begin{aligned}
&\mathop{\mathbb{E}}_{\pi \in \mathbf{P}(S)} \gamma\left(S - \pi^{-1}(p), \pi - \pi^{-1}(p), N_{\{s \in S\ :\ \pi(s) \leq p\}}\left(\pi^{-1}(p)\right)\right) && (2)\\
=&\frac{1}{|S|!} \sum_{B \in \binom{S}{p}} \sum_{s \in B} \sum_{\substack{\pi \in \mathbf{P}(S)\ :\\ \pi(B) \leq p,\ \pi(s) = p}} \gamma(S - s, \pi - s, N_B(s)) && (3)\\
\leq&\frac{1}{|S|!} \sum_{B \in \binom{S}{p}} \sum_{s \in B} \frac{4}{p} \sum_{\substack{\pi \in \mathbf{P}(S)\ :\\ \pi(B) \leq p,\ \pi(s) = p}} \gamma(S - s, \pi - s, \mathbb{R}^2) && (4)\\
=&\frac{4}{|S|!\ p} \sum_{s \in S} \sum_{B \in \binom{S-s}{p-1}} \sum_{\substack{\pi \in \mathbf{P}(S-s)\ :\\ \pi(B) < p}} \gamma(S - s, \pi, \mathbb{R}^2) && (5)\\
=&\frac{1}{|S|} \sum_{s \in S} \frac{4}{p} \frac{1}{|S - s|!} \sum_{\pi \in \mathbf{P}(S-s)} \gamma(S - s, \pi, \mathbb{R}^2) && (6)\\
=&\frac{1}{|S|} \sum_{s \in S} \ \frac{4}{p} \mathop{\mathbb{E}}_{\pi \in \mathbf{P}(S-s)} \gamma(S - s, \pi, \mathbb{R}^2) && (7)
\end{aligned}
$$

Equation (3) is due to regrouping the summation terms by the sequences that have the same sets of elements B in the first p positions, (4) due to the ordinary[1] "camel trick" of the form $\sum_{s \in B} f(s) = \sum_{s \in B} \frac{1}{|B|} \sum_{s \in B} f(s)$ and the 4-ply covering. Equation (5) uses the combinatorial identity $\binom{n}{k}\binom{k}{1} = \binom{n}{1}\binom{n-1}{k-1}$ to first choose $s \in S$ for position p. Since the terms in (7) do not depend on the spatial location of s, the bound holds for any $s \in S$. ◀

We now bound the expected total size of subtrees below affected nodes (c.f. Figure 4a).

▶ **Lemma 4.3** (Subtree Sizes). *Let S be a set of non-crossing segments, $p \in \{1, \ldots, |S|\}$ uniformly at random, and $s \in S$. The expected total size of affected subtrees in a random TST over $(S - s)$ is $\mathcal{O}\left(\log^2 |S|\right)$.*

Proof. We use Lemma 4.2 to bound the number of nodes in $\mathcal{T}(S - s, \pi)$ that have a priority of at least p and have a region that is in the p-neighborhood of s.

$$
\begin{aligned}
&\frac{1}{|S|} \sum_{p=1}^{|S|} \mathop{\mathbb{E}}_{\pi \in \mathbf{P}(S-s)} \left|\left\{ v \in \mathcal{T}_{\geq p}(S - s, \pi)\ :\ \Delta(v) \subseteq F_{\{s' \in (S-s)\ :\ \pi(s') < p\}}(s) \right\}\right| && (8)\\
\leq\ & \frac{1}{|S|} \sum_{p=1}^{|S|} \frac{4}{p} \mathop{\mathbb{E}}_{\pi \in \mathbf{P}(S-s)} \left|\mathcal{T}_{\geq p}(S - s, \pi)\right| && (9)\\
\leq\ & \frac{1}{|S|} \sum_{p=1}^{|S|} \frac{4}{p} \mathop{\mathbb{E}}_{\pi \in \mathbf{P}(S-s)} \left|\mathcal{T}(S - s, \pi)\right| && (10)\\
\leq\ & \frac{1}{|S|} \sum_{p=1}^{|S|} \frac{4}{p} \mathcal{O}\left(|S - s| \log |S - s|\right) \leq \mathcal{O}\left(\log^2 |S|\right) && (11)
\end{aligned}
$$

Bound (11) is due to the expected size of the whole tree (c.f. Theorem 2.1). ◀

[1] E.g. proof of Theorem 3.1.4 in [16, p. 90] or proof of Theorem 6.3 in [11, p. 136].

With a slightly more advanced application of our arguments presented thus far, we now bound the cost for simple iterated ray-shooting search to find the affected subtree roots for a query segment that has a designated random priority rank (c.f. Section 3.1).

▶ **Lemma 4.4** (Segment Search). *Let S be a set of non-crossing segments, $p \in \{1, \dots, |S|\}$ uniformly at random, and $s \in S$. The expected time to find the affected nodes within a random TST over $(S - s)$ is $\mathcal{O}\left(\log^2 |S|\right)$.*

Proof. The search visits only nodes v with regions $\Delta(v) \cap s \neq \emptyset$. Since Lemma 4.3 bounds the size of the result set, we only need to bound the number of nodes with priority smaller p. We do so by invoking Lemma 4.2 $\lceil \log_2 p \rceil$ times.

$$\frac{1}{|S|}\sum_{p=1}^{|S|} \mathop{\mathbb{E}}_{\pi\in\mathbf{P}(S-s)} \left|\left\{v \in \mathcal{T}_{<p}(S-s,\pi) \ : \ \Delta(v)\cap s \neq \emptyset\right\}\right| \tag{12}$$

$$\leq \frac{1}{|S|}\sum_{p=1}^{|S|}\sum_{j=1}^{\lceil \log_2 p\rceil} \mathop{\mathbb{E}}_{\pi\in\mathbf{P}(S-s)} \left|\left\{v \in \mathcal{T}(S-s,\pi) : 2^{j-1} < p(v) \leq 2^j, \ \Delta(v)\cap s \neq \emptyset\right\}\right| \tag{13}$$

$$= \frac{1}{|S|}\sum_{p=1}^{|S|}\sum_{j=1}^{\lceil \log_2 p\rceil} \frac{4}{2^{j-1}} \mathop{\mathbb{E}}_{\pi\in\mathbf{P}(S-s)} \left|\left\{v \in \mathcal{T}(S-s,\pi) \ : \ 2^{j-1} < p(v) \leq 2^j\right\}\right| \tag{14}$$

$$\leq \frac{1}{|S|}\sum_{p=1}^{|S|}\sum_{j=1}^{\lceil \log_2 p\rceil} \frac{4}{2^{j-1}} \mathcal{O}\left(2^j \log(2^j)\right) \tag{15}$$

$$= \frac{1}{|S|}\sum_{p=1}^{|S|}\sum_{j=1}^{\lceil \log_2 p\rceil} \mathcal{O}\left(j\right) \leq \frac{1}{|S|}\sum_{p=1}^{|S|} \mathcal{O}(\log^2 p) \leq \mathcal{O}\left(\log^2 |S|\right) \tag{16}$$

In (13), we use "linearity of expectation" to count search nodes of $\mathcal{T}_{<p}$ in batches of priority intervals of the form $(2^{j-1}, 2^j]$. Bound (14) is due to invoking Lemma 4.2 for s according to the priority interval. To bound a batch j in (15) we use Theorem 2.1 on the whole TST size over the first 2^j segments. ◀

Now we have all necessary arguments for our bound on dynamic update costs in TSTs.

▶ **Theorem 4.5.** *Let S be a set of non-crossing segments and $s \in S$. The expected cost of inserting s in a random TST over $(S - s)$ is $\mathcal{O}(\log^2 |S|)$.*

Proof. Let $1 \leq m \leq |\mathcal{T}|$ denote the total number of nodes within affected subtrees of $\mathcal{T}$, prior to the insertion call. We argue that the total number of nodes visited by the insertion procedure is $\mathcal{O}(m)$, which establishes the result based on Lemma 4.3 and 4.4 for non-crossing segments.

For each root of an affected subtree, we have at most 2 calls of V-Partition to slab the subtree. After the slabbing stage of the insertion, we have at most $3m$ nodes in total.

The edge Partition calls on one slab are independent from those of another slab in $\mathcal{T}$. Neglecting node removals due to V-Merge briefly, the total number of temporarily created nodes of Partition increases at most by a factor of 2. Since we cannot remove more than what was created, the total number of nodes that V-Merge may visit is bounded by $6m$. ◀

Given the duality of the update procedures, deletion of a $s \in S$ having priority $p(s)$ visits exactly the same nodes as insertion of s among $(S - s)$ with priority value $p(s)$. Hence the expected node visits are equal as well. Since our bound on expected update operations counts TST search nodes, we may employ the method of Hemmer et al. [13], on the bijection between TST and TSD search paths, to obtain equal asymptotic bounds for the expected update costs in TSDs. A more detailed consideration, however, allows an improvement.

4.1 Improved Update Bounds For TSDs

The region of each TSD node may be represented by its top, bottom and vertical boundary cuts. To use "linearity of expectation", we decompose the random variable $|\mathcal{D}(S,\pi)|$ in a sum of $\mathcal{O}(|S|^5)$ binary indicator variables $\mathcal{N}_{i,i',i'',i''',j}$ that are 1 if and only if $\mathcal{D}(S,\pi)$ contains a node whose boundary is constituted by cuts with the priorities i, i', i'', i''' and gets destroyed by the segment with priority j. That is $|\mathcal{D}_{=l}| = \sum_{i,i',i'',i'''} \mathcal{N}_{i,i',i'',i''',l}$. The number of these indicator variables only depends on $|S|$, but not on π or the spatial location of the segments.

▶ **Lemma 4.6** (Overlap into Zones). *Let S be a set of non-crossing segments, $p \in \{1, \ldots, |S|\}$ a fixed value, and $s \in S$. For any $l \geq p$, the expected number of priority l nodes of a random TSD over $S - s$ that overlap the p-neighborhood of s is upper bounded with*

$$\mathop{\mathbb{E}}_{\pi \in \mathbf{P}(S-s)} \left| \left\{ v \in \mathcal{D}_{=l}(S-s,\pi) \ : \Delta(v) \cap F \neq \emptyset \right\} \right| \ \leq \ \frac{\mathcal{O}(1)}{p} \mathop{\mathbb{E}}_{\pi \in \mathbf{P}(S-s)} \left| \mathcal{D}_{=l}(S-s,\pi) \right| \ ,$$

where F denotes the p-neighborhood of s.

For the proof recall that, for any subset $C \subseteq S$ and its maximum priority element $m \in C$, classic Backward Analysis (e.g. Chapter 6 in [11]) shows that the number d_C, denoting the number of faces in $\mathcal{A}(C - m)$ that segment m intersects (a.k.a destroys), is expected $\mathcal{O}(1)$. Since the neighborhood zones are a union of faces of $\mathcal{A}(C - m)$, element m also intersects no more than $4 \cdot d_C$ of the neighborhood zones in N_{C-m}.

Proof. The proof is analogue to Lemma 4.2, except from Eq. 4 which uses that every TST node is in at most 4 neighborhood zones. Given the refined decomposition of $|\mathcal{D}_{=l}|$, showing that *each* of the priority l nodes intersects at most $\mathcal{O}(1)$ zones of N_B gives the statement due to linearity of expectation.

A priority $l < \infty$ node, i.e. a non-leaf node, is either fully crossed by the segment $\pi^{-1}(l)$ or contains an endpoint of it. In the first case, the number of N_B zones that the node region intersects is bounded by $4 \cdot d_C$, where $C = B + \pi^{-1}(l)$. For the endpoint case, let $|B| < l', l'' < l$ denote the priorities of the top and bottom segment that bound the node's trapezoidal region. Hence, the node does not intersect more than $4(d_{C'} + d_{C''})$ neighborhood zones where $C' = B + \pi^{-1}(l')$ and $C'' = B + \pi^{-1}(l'')$. The argument for a leaf node, i.e. $l = \infty$, is analogue to the endpoint case. ◀

▶ **Corollary 4.7.** *Let S be a set of non-crossing segments and $s \in S$. The expected cost of inserting s in a random TSD over $(S - s)$ is $\mathcal{O}(\log |S|)$.*

Proof. Using Lemma 4.6, Equation 11 improves to $\frac{1}{|S|} \sum_{p=1}^{|S|} \mathcal{O}(\frac{1}{p}|S - s|)$, which evaluates to an expected number of $\mathcal{O}(\log |S|)$ TSD nodes to overlap into the affected zone. Moreover, Equation 15 improves to $\frac{1}{|S|} \sum_{p=1}^{|S|} \sum_{j=1}^{\lceil \log_2 p \rceil} \mathcal{O}\left(\frac{1}{2^{j-1}} 2^j\right)$, which evaluates to an expected number of $\mathcal{O}(\log |S|)$ TSD nodes that are visited in the segment search. Since our recursive update primitives visit a TSD node at most once, using the same arguments as in the proof of Theorem 4.5 on TSDs provides the expected update cost as stated. ◀

5 Offline vs Online – Maintaining Small Codes For Dynamic Orders

To support online emplacement and dropping of additional elements in a sequence $\pi \in \mathbf{P}(S)$, we use a more flexible representation than standard bijections $\pi : S \to \{1, \dots, |S|\}$. We represent sequences as injective mappings $\tau : S \to (0,1) \subseteq \mathbb{R}$. If the values of τ are stored with the elements of S, we can evaluate the required $\pi(s) < \pi(s')$ predicates with comparison on τ in constant time. However, dynamically extending τ for a new element $s \notin S$ such that $\tau(s)$ falls with equal probability in the intervals between $\{\tau(s') : s' \in S\}$ is challenging. E.g. simple random sampling of $\tau(s)$ uniformly out of $(0,1)$ does not provide this. Since single registers of the pointer machine model are confined to numbers with only $\mathcal{O}(\log |S|)$ bits, simple interval halving strategies may well produce inefficiently long codes for τ as well.

We solve this problem using one additional randomized data structure to maintain orders among n elements of a totally ordered set of "keys'. Treaps are a conceptual fusion of Binary Search Trees (BSTs), over the nodes' key values, and (Min-)Heaps, over the nodes' priority values. Insertions and deletions are performed similarly to BSTs and additionally standard tree rotations are used to maintain the heap property on the randomly chosen priority values. Seidel and Aragon [21] show not only that Treaps achieve $\mathcal{O}(\log n)$ depth with high probability but also that, even if the cost of a rotation is proportional to the whole subtree size, insertions and deletions still perform expected $\mathcal{O}(\log n)$ operations.

For our simple in-order numbering scheme in a fixed BST, we consider an injective mapping from root-to-node paths, which we read as smaller/larger strings over the alphabet $\{s, l\}$, to binary fractional number from $(0,1) \subseteq \mathbb{Q}$ coded as "1"-terminated strings over the alphabet $\{0, 1\}$. More precisely, for the empty path string (the root node r) the associated value is $\gamma(r) = 0.1_{(2)}$, s-edges are coded as 0 and l-edges are coded as 1. E.g. the left child node v_s of the root has code $\gamma(v_s) = 0.01_{(2)}$, the right child node v_l of the root has code $\gamma(v_l) = 0.11_{(2)}$, and we have $\gamma(v_s) < \gamma(r) < \gamma(v_l)$. On paths of a BST, the search tree property extends to the order of these γ values of the nodes. This allows us to check if u is before v in the in-order sequence by comparing the values of $\gamma(u)$ and $\gamma(v)$. Moreover, this recursive code definition can be assigned in a top-down fashion for all nodes in a subtree under v, once $\gamma(v)$ is assigned.

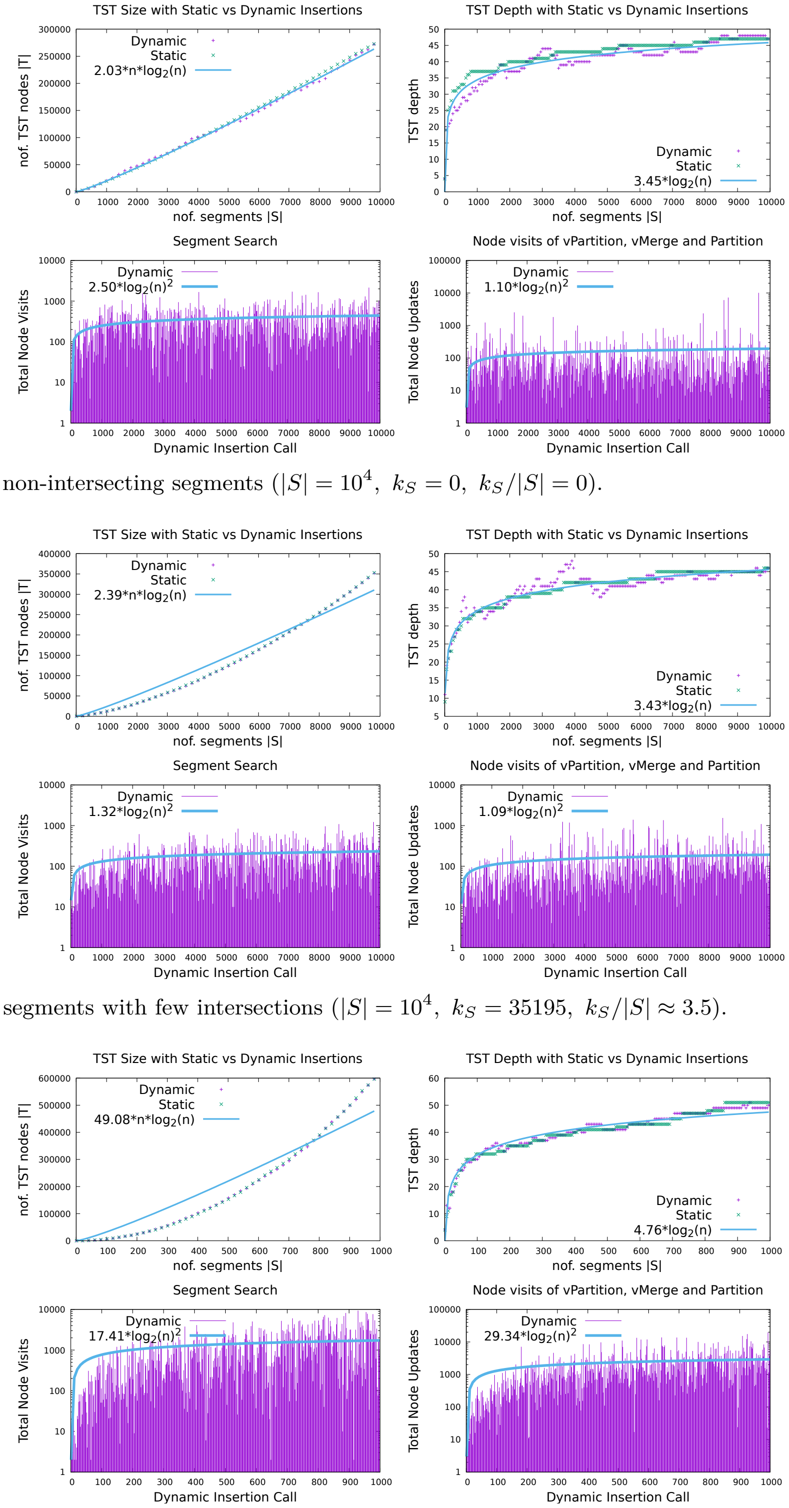
In order to maintain the proper values τ for a set S, we augment the Treap nodes to also store the total number of nodes in their subtree. To emplace a new element, we first choose an integer $1 \leq p \leq |S| + 1$ uniformly at random. Next we use the standard Treap insertion to add a new node resembling the p-smallest key-value as a new leaf. After the bottom-up re-balancing rotations are finished, we simply re-visit the whole rotated subtree in a top-down fashion to overwrite the key value of each node v with the new value $\gamma(v)$. The deletion of an element is handled analogously. To evaluate an order predicate between two elements, we simply compare the current key values due to the Treap, which are stored with the elements of S. We summarize this with the following statement.

▶ **Observation 5.1.** *Representation of sequences $\pi \in \mathbf{P}(S)$ requires no more than $\mathcal{O}(|S|)$ space and dynamic updates take expected $\mathcal{O}(\log |S|)$ operations. After successful updates, evaluation of a $\pi(s) < \pi(s')$ predicate on $s, s' \in S$ takes $\mathcal{O}(1)$ operations.*

6 Implementation and Experiments

We implemented[2] our fully-dynamic approach for TSTs based on the exact predicates and exact construction for line segments of the Computational Geometry Algorithms Library [23].

[2] `https://github.com/milutinB/dynamic_trapezoidal_map_impl`



(a) Random non-intersecting segments ($|S| = 10^4,\ k_S = 0,\ k_S/|S| = 0$).

(b) Random segments with few intersections ($|S| = 10^4,\ k_S = 35195,\ k_S/|S| \approx 3.5$).

(c) Random segments with many intersections ($|S| = 10^3,\ k_S = 120730,\ k_S/|S| \approx 120.7$).

Figure 5 Experimental results on TST size (top left), TST depth (top right), and node visits of the segment search (bottom left) and updates (bottom right) for RIC TSTs (green) and Dynamic TSTs (purple). The blue curve indicates the respective function fit on the data of Dynamic TSTs.

Our experiments focus on evaluating the practicality of the approach and the tightness of the analysis. We use the standard RIC of TSTs as verification and baseline comparison. To measure the performance of our segment search and dynamic updates, we count the total number of node visits during recursions of the update and the search. To capture the asymptotic behavior in the experiments, we perform many insertion calls, with either ascending segment priority values (static TST) or a random shuffle of them (dynamic TST).

We created random sets of segments with varying numbers of intersections based on the 64-bit Mersenne Twister random number generator of the C++ Standard Library. The experiment without intersections is comprised of horizontal segments, for which we first chose a y-coordinate uniformly (from the domain range) and then two x-coordinates uniformly (see Figure 5a). The experiment with few intersections is comprised of short segments, for which we chose a point, a direction, and a length uniformly at random (on average 3% of the domain boundary length; see Figure 5b). The experiment on many intersections are generated by choosing the coordinates of both endpoints uniformly at random (see Figure 5c).

Figure 5 shows the results on the segment search and recursive node visits of updates. For ease of comparison, we also plot a function fit[3] (thick blue lines) of the well known size and depth bounds (top rows) and our new segment search and update bounds (bottom rows) as overlay on the experimental data. Note that only on the non-intersecting data set a comparison of function fit and bounds is meaningful. We do however also provide it for the other experiments as a "trend line" that indicates the additional costs due to segment intersections. To allow a better visual perception of the average cost per insertion, Figures 5a and 5b show every 20th data point and Figure 5c shows every 2nd data point as impulse.

The experiment on non-intersecting segments matches with our analysis of segment search and update operations on TSTs. As anticipated, TST updates on segments with many intersections are more expensive.

References

1 Pankaj K. Agarwal, Julien Basch, Mark de Berg, Leonidas J. Guibas, and John Hershberger. Lower bounds for kinetic planar subdivisions. *Discrete & Computational Geometry*, 24(4):721–733, 2000. doi:10.1007/s004540010060.

2 Pankaj K. Agarwal, Jeff Erickson, and Leonidas J. Guibas. Kinetic binary space partitions for intersecting segments and disjoint triangles. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 107–116, 1998. URL: http://dl.acm.org/citation.cfm?id=314613.

3 Lars Arge, Gerth Stølting Brodal, and Loukas Georgiadis. Improved dynamic planar point location. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 305–314, 2006. doi:10.1109/FOCS.2006.40.

4 Daniel Bahrdt and Martin P. Seybold. Rational points on the unit sphere: Approximation complexity and practical constructions. In *Proceedings of the 2017 ACM on International Symposium on Symbolic and Algebraic Computation (ISSAC)*, pages 29–36, 2017. doi:10.1145/3087604.3087639.

5 Hanna Baumgarten, Hermann Jung, and Kurt Mehlhorn. Dynamic point location in general subdivisions. *Journal of Algorithms*, 17(3):342–380, 1994. doi:10.1006/jagm.1994.1040.

6 Michael A. Bender, Jeremy T. Fineman, Seth Gilbert, Tsvi Kopelowitz, and Pablo Montes. File maintenance: When in doubt, change the layout! In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1503–1522, 2017. doi:10.1137/1.9781611974782.98.

[3] We use the standard least-squares Marquardt-Levenberg algorithm of GNUPlot 5.2.

7 Timothy M. Chan and Yakov Nekrich. Towards an optimal method for dynamic planar point location. In *Proceedings of the 56th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 390–409, 2015. doi:10.1109/FOCS.2015.31.

8 Siu-Wing Cheng and Ravi Janardan. New results on dynamic planar point location. *SIAM Journal on Computing*, 21(5):972–999, 1992. doi:10.1137/0221057.

9 Y. Chiang and R. Tamassia. Dynamic algorithms in computational geometry. *Proceedings of the IEEE*, 80(9):1412–1434, 1992. doi:10.1109/5.163409.

10 Yi-Jen Chiang and Roberto Tamassia. Dynamization of the trapezoid method for planar point location in monotone subdivisions. *International Journal of Computational Geometry & Applications*, 2(3):311–333, 1992. doi:10.1142/S0218195992000184.

11 Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational Geometry: Algorithms and Applications, 3rd Edition.* Springer, 2008. doi:10.1007/978-3-540-77974-2.

12 Michael T. Goodrich and Roberto Tamassia. Dynamic trees and dynamic point location. *SIAM Journal on Computing*, 28(2):612–636, 1998. doi:10.1137/S0097539793254376.

13 Michael Hemmer, Michal Kleinbort, and Dan Halperin. Optimal randomized incremental construction for guaranteed logarithmic planar point location. *Computational Geometry: Theory and Applications*, 58:110–123, 2016. doi:10.1016/j.comgeo.2016.07.006.

14 Alex Hobé, Daniel Vogler, Martin P. Seybold, Anozie Ebigbo, Randolph R. Settgast, and Martin O. Saar. Estimating fluid flow rates through fracture networks using combinatorial optimization. *Advances in Water Resources*, 122:85–97, 2018. doi:10.1016/j.advwatres.2018.10.002.

15 Ketan Mulmuley. A fast planar partition algorithm, I. *Journal of Symbolic Computation*, 10(3-4):253–280, 1990. doi:10.1016/S0747-7171(08)80064-8.

16 Ketan Mulmuley. *Computational Geometry: An Introduction Through Randomized Algorithms.* Prentice Hall, 1994.

17 J. Ian Munro and Yakov Nekrich. Dynamic planar point location in external memory. In *Proceedings of the 35th International Symposium on Computational Geometry (SoCG)*, pages 52:1–52:15, 2019. doi:10.4230/LIPIcs.SoCG.2019.52.

18 Eunjin Oh and Hee-Kap Ahn. Point location in dynamic planar subdivisions. In *Proceedings of the 34th International Symposium on Computational Geometry (SoCG)*, pages 63:1–63:14, 2018. doi:10.4230/LIPIcs.SoCG.2018.63.

19 Otfried Schwarzkopf. Dynamic maintenance of geometric structures made easy. In *Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 197–206, 1991. doi:10.1109/SFCS.1991.185369.

20 Raimund Seidel. A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. *Computational Geometry: Theory and Applications*, 1:51–64, 1991. doi:10.1016/0925-7721(91)90012-4.

21 Raimund Seidel and Cecilia R. Aragon. Randomized search trees. *Algorithmica*, 16(4-5):464–497, 1996. doi:10.1007/BF01940876.

22 Martin P. Seybold. Robust map matching for heterogeneous data via dominance decompositions. In *Proceedings of the 2017 SIAM International Conference on Data Mining (SDM)*, pages 813–821, 2017. doi:10.1137/1.9781611974973.91.

23 The CGAL Project. *CGAL User and Reference Manual.* CGAL Editorial Board, 4.14.1 edition, 2019. URL: https://doc.cgal.org/4.14.1/Manual/packages.html.

24 Athanasios K. Tsakalidis. Maintaining order in a generalized linked list. *Acta Informatica*, 21:101–112, 1984. doi:10.1007/BF00289142.

Faster Minimization of Tardy Processing Time on a Single Machine

Karl Bringmann
Saarland University, Saarland Informatics Campus (SIC), Saarbrücken, Germany
Max Planck Institute for Informatics, Saarland Informatics Campus (SIC), Saarbrücken, Germany
bringmann@cs.uni-saarland.de

Nick Fischer
Saarland University, Saarland Informatics Campus (SIC), Saarbrücken, Germany
Max Planck Institute for Informatics, Saarland Informatics Campus (SIC), Saarbrücken, Germany
nfischer@mpi-inf.mpg.de

Danny Hermelin
Department of Industrial Engineering and Management,
Ben-Gurion University of the Negev, Beersheba, Israel
hermelin@bgu.ac.il

Dvir Shabtay
Department of Industrial Engineering and Management,
Ben-Gurion University of the Negev, Beersheba, Israel
dvirs@bgu.ac.il

Philip Wellnitz
Max Planck Institute for Informatics, Saarland Informatics Campus (SIC), Saarbrücken, Germany
wellnitz@mpi-inf.mpg.de

Abstract

This paper is concerned with the $1||\sum p_j U_j$ problem, the problem of minimizing the total processing time of tardy jobs on a single machine. This is not only a fundamental scheduling problem, but also a very important problem from a theoretical point of view as it generalizes the Subset Sum problem and is closely related to the 0/1-Knapsack problem. The problem is well-known to be NP-hard, but only in a weak sense, meaning it admits pseudo-polynomial time algorithms. The fastest known pseudo-polynomial time algorithm for the problem is the famous Lawler and Moore algorithm which runs in $O(P \cdot n)$ time, where P is the total processing time of all n jobs in the input. This algorithm has been developed in the late 60s, and has yet to be improved to date.

In this paper we develop two new algorithms for $1||\sum p_j U_j$, each improving on Lawler and Moore's algorithm in a different scenario:

- Our first algorithm runs in $\tilde{O}(P^{7/4})$ time[1], and outperforms Lawler and Moore's algorithm in instances where $n = \tilde{\omega}(P^{3/4})$.
- Our second algorithm runs in $\tilde{O}(\min\{P \cdot D_\#, P + D\})$ time, where $D_\#$ is the number of *different* due dates in the instance, and D is the sum of all different due dates. This algorithm improves on Lawler and Moore's algorithm when $n = \tilde{\omega}(D_\#)$ or $n = \tilde{\omega}(D/P)$. Further, it extends the known $\tilde{O}(P)$ algorithm for the single due date special case of $1||\sum p_j U_j$ in a natural way.

Both algorithms rely on basic primitive operations between sets of integers and vectors of integers for the speedup in their running times. The second algorithm relies on fast polynomial multiplication as its main engine, while for the first algorithm we define a new "skewed" version of $(\max, \min)$-convolution which is interesting in its own right.

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

Keywords and phrases Weighted number of tardy jobs, sumsets, convolutions

[1] Throughout the paper we use $\tilde{O}(\cdot)$ to suppress logarithmic factors.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 19; pp. 19:1–19:12
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.19

Category Track A: Algorithms, Complexity and Games

Related Version A full version of the paper is available at `https://arxiv.org/abs/2003.07104`.

Funding *Karl Bringmann* and *Nick Fischer*: This work is part of the project TIPEA that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 850979).

1 Introduction

In this paper we consider the problem of minimizing the total processing times of tardy jobs on a single machine. In this problem we are given a set of n jobs $J = \{1, \ldots, n\}$, where each job j has a *processing time* $p_j \in \mathbb{N}$ and a *due date* $d_j \in \mathbb{N}$. A *schedule* σ for J is a permutation $\sigma : \{1, \ldots, n\} \to \{1, \ldots, n\}$. In a given schedule σ, the *completion time* C_j of a job j under σ is given by $C_j = \sum_{\sigma(i) \leq \sigma(j)} p_i$, that is, the total processing time of jobs preceding j in σ (including j itself). Job j is *tardy* in σ if $C_j > d_j$, and *early* otherwise. Our goal is find a schedule with minimum total processing time of tardy jobs. If we assign a binary indicator variable U_j to each job j, where $U_j = 1$ if j is tardy and otherwise $U_j = 0$, our objective function can be written as $\sum p_j U_j$. In the standard three field notation for scheduling problems of Graham [5], this problem is denoted as the $1||\sum p_j U_j$ problem (the 1 in the first field indicates a single machine model, and the empty second field indicates there are no additional constraints).

The $1||\sum p_j U_j$ problem is a very natural and fundamental scheduling problem, which models a very basic scheduling scenario. As it includes Subset Sum as a special case (see below), the $1||\sum p_j U_j$ problem is NP-hard. However, it is only hard in the weak sense, meaning it admits pseudo-polynomial time algorithms. The focus of this paper is on developing fast pseudo-polynomial time algorithms for $1||\sum p_j U_j$, improving in several settings on the best previously known solution from the late 60s. Before we describe our results, we discuss the previously known state of the art of the problem, and describe how our results fit into this line of research.

1.1 State of the Art

A famous generalization of the $1||\sum p_j U_j$ problem is the $1||\sum w_j U_j$ problem. Here, each job j also has a weight w_j in addition to its processing time p_j and due date d_j, and the goal is to minimize the total weight (as opposed to total processing times) of tardy jobs. This problem has already been studied in the 60s, and even appeared in Karp's fundamental paper from 1972 [6]. The classical algorithm of Lawler and Moore [10] for the problem is one of the earliest and most prominent examples of pseudo-polynomial algorithms, and it is to date the fastest known algorithm even for the special case of $1||\sum p_j U_j$. Letting $P = \sum_{j \in J} p_j$, their result can be stated as follows:

▶ **Theorem 1.1** ([10]). *$1||\sum w_j U_j$ and $1||\sum p_j U_j$ can both be solved in $O(P \cdot n)$ time.*

Note that as we assume that all processing times are integers, we have $n \leq P$, and so the running time of the algorithm in Theorem 1.1 can be bounded by $O(P^2)$. In fact, it makes perfect sense to analyze the time complexity of a pseudo-polynomial time algorithm for either problems in terms of P, as P directly corresponds to the total input length when integers are encoded in unary. Observe that while the case of $n = P$ (all jobs have unit processing times) essentially reduces to sorting, there are several non-trivial cases where n is smaller than P yet still quite significant in the $O(P \cdot n)$ term of Theorem 1.1. The fundamental question this paper addresses is:

> *"Can we obtain algorithms with running times $O(P^{2-\varepsilon})$, for any fixed $\varepsilon > 0$, for either $1||\sum w_j U_j$ or $1||\sum p_j U_j$?"*

For $1||\sum w_j U_j$ there is some evidence that the answer to this question should be no. Karp [6] observed that the special case of the $1||\sum w_j U_j$ problem where all jobs have the same due date d, the $1|d_j = d|\sum w_j U_j$ problem, is essentially equivalent to the classical 0/1-Knapsack problem. Cygan *et al.* [4] and Künnemann *et al.* [9] studied the $(\min, +)$-Convolution problem (see Section 2), and conjectured that the $(\min, +)$-convolution between two vectors of length n cannot be computed in $\tilde{O}(n^{2-\varepsilon})$ time, for any $\varepsilon > 0$. Under this $(\min, +)$-Convolution Conjecture, they obtained lower bounds for several Knapsack related problems. In our terms, their result can be stated as follows:

▶ **Theorem 1.2** ([4, 9])**.** *There is no $\tilde{O}(P^{2-\varepsilon})$ time algorithm for the $1|d_j = d|\sum w_j U_j$ problem, for any $\varepsilon > 0$, unless the $(\min, +)$-Convolution Conjecture is false. In particular, $1||\sum w_j U_j$ has no such algorithm under this conjecture.*

Analogous to the situation with $1||\sum w_j U_j$, the special case of $1||\sum p_j U_j$ where all jobs have the same due date d (the $1|d_j = d|\sum p_j U_j$ problem) is equivalent to the classical Subset Sum problem. Recently, there has been significant improvements for Subset Sum resulting in algorithms with $\tilde{O}(T + n)$ running times [2, 7], where n is number of integers in the instance and T is the target. Due to the equivalence between the two problems, this yields the following result for the $1|d_j = d|\sum p_j U_j$ problem:

▶ **Theorem 1.3** ([2, 7])**.** *$1|d_j = d|\sum p_j U_j$ can be solved in $\tilde{O}(P)$ time.*

On the other hand, due to equivalence of $1|d_j = d|\sum p_j U_j$ and Subset Sum, we also know that Theorem 1.3 above cannot be significantly improved unless the Strong Exponential Time Hypothesis (SETH) fails. Specifically, combining a recent reduction from k-SAT to Subset Sum [1] with the equivalence of Subset Sum and $1|d_j = d|\sum p_j U_j$, yields the following:

▶ **Theorem 1.4** ([1])**.** *There is no $\tilde{O}(P^{1-\varepsilon})$ time algorithm for the $1|d_j = d|\sum p_j U_j$ problem, for any $\varepsilon > 0$, unless SETH fails.*

Nevertheless, Theorem 1.4 still leaves quite a big gap for the true time complexity of $1||\sum p_j U_j$, as it can potentially be anywhere between the $\tilde{O}(P)$ time known already for the special case of $1|d_j = d|\sum p_j U_j$ (Theorem 1.3), and the $O(Pn) = O(P^2)$ time of Lawler and Moore's algorithm (Theorem 1.1). This is the starting point of our paper.

1.2 Our Results

The main contribution of this paper is two new pseudo-polynomial time algorithms for $1||\sum p_j U_j$, each improving on Lawler and Moore's algorithm in a different sense. Our algorithms take a different approach to that of Lawler and Moore in that they rely on fast operators between sets and vectors of numbers.

Our first algorithm improves Theorem 1.1 in case there are sufficiently many jobs in the instance compared to the total processing time. More precisely, our algorithm has a running time of $\tilde{O}(P^{7/4})$, and so it is faster than Lawler and Moore's algorithm in case $n = \tilde{\omega}(P^{3/4})$.

▶ **Theorem 1.5.** *$1||\sum p_j U_j$ can be solved in $\tilde{O}(P^{7/4})$ time.*

The algorithm in Theorem 1.5 uses a new kind of convolution which we coined "Skewed Convolution" and is interesting in its own right. In fact, one of the main technical contributions of this paper is a fast algorithm for the $(\max, \min)$-Skewed-Convolution problem (see definition in Section 2).

Our second algorithm for $1||\sum p_j U_j$ improves Theorem 1.1 in case there are not too many different due dates in the problem instance; that is, $D_\# = |\{d_j : j \in J\}|$ is relatively small when compared to n. This is actually a very natural assumption, for instance in cases where delivery costs are high and products are batched to only few shipments. Let D denote the sum of the different due dates in our instance. Then our second result can be stated as follows:

▶ **Theorem 1.6.** *$1||\sum p_j U_j$ can all be solved in $\tilde{O}(\min\{P \cdot D_\#, P + D\})$ time.*

The algorithm in Theorem 1.6 uses basic operations between sets of numbers, such as the sumset operation (see Section 2) as basic primitives for its computation, and ultimately relies on fast polynomial multiplication for its speedup. It should be noted that Theorem 1.6 includes the $\tilde{O}(P)$ result of Theorem 1.3 for $1|d_j = d|\sum p_j U_j$ as a special case where $D_\# = 1$ or $D = d$.

1.3 Roadmap

The paper is organized as follows. In Section 2 we discuss all the basic primitives that are used by our algorithms, including some basic properties that are essential for the algorithms. We then present our second algorithm in Section 3, followed by our first algorithm in Section 4. Section 5 describes our fast algorithm for the skewed version of $(\max, \min)$-convolution, and is the main technical part of the paper. Finally, we conclude with some remarks and open problems in Section 6.

2 Preliminaries

In the following we discuss the basic primitives and binary operators between sets/vectors of integers that will be used in our algorithms. In general, we will use the letters X and Y to denote sets of non-negative integers (where order is irrelevant), and the letters A and B to denote vectors of non-negative integers.

Sumsets

The most basic operation used in our algorithms is computing the sumset of two sets of non-negative integers:

▶ **Definition 2.1** (Sumsets). *Given two sets of non-negative integers X_1 and X_2, the* sumset *of X_1 and X_2, denoted $X_1 \oplus X_2$, is defined by*

$$X_1 \oplus X_2 = \{x_1 + x_2 : x_1 \in X_1, x_2 \in X_2\}.$$

Clearly, the sumset $X_1 \oplus X_2$ can be computed in $O(|X_1| \cdot |X_2|)$ time. However, in certain cases we can do better using fast polynomial multiplication. Consider the two polynomials $p_1[\alpha] = \sum_{x \in X_1} \alpha^x$ and $p_2[\beta] = \sum_{x \in X_2} \beta^x$. Then the exponents of all terms in $p_1 \cdot p_2$ with non-zero coefficients correspond to elements in the sumset $X_1 \oplus X_2$. Since multiplying two polynomials of maximum degree d can be done in $O(d \log d)$ time [3], we have the following:

▶ **Lemma 2.2.** *Given two sets of non-negative integers $X_1, X_2 \subseteq \{0, \ldots, P\}$, one can compute the sumset $X_1 \oplus X_2$ in $O(P \log P)$ time.*

Set of all Subset Sums

Given set of non-negative integers X, we will frequently be using the set of all sums generated by subsets of X:

▶ **Definition 2.3** (Subset Sums). *For a given set of non-negative integers X, define the set of all subset sums $\mathcal{S}(X)$ as the set of integers given by*

$$\mathcal{S}(X) = \Big\{ \sum_{x \in Y} x : Y \subseteq X \Big\}.$$

Here, we always assume that $0 \in \mathcal{S}(X)$ (as it is the sum of the empty set).

We can use Lemma 2.2 above to compute $\mathcal{S}(X)$ from X rather efficiently: First, split X into two sets X_1 and X_2 of roughly equal size. Then recursively compute $\mathcal{S}(X_1)$ and $\mathcal{S}(X_2)$. Finally, compute $\mathcal{S}(X) = \mathcal{S}(X_1) \oplus \mathcal{S}(X_2)$ via Lemma 2.2. The entire algorithm runs in $\tilde{O}(\sum_{x \in X} x)$ time.

▶ **Lemma 2.4** ([7]). *Given a set of non-negative integers X, with $P = \sum_{x \in X} x$, one can compute $\mathcal{S}(X)$ in $\tilde{O}(P)$ time.*

Convolutions

Given two vectors $A = (A[i])_{i=0}^{n}$, $B = (B[j])_{j=0}^{n}$, the $(\circ, \bullet)$-Convolution problem for binary operators $\circ$ and $\bullet$ is to compute a vector $C = (C[k])_{k=0}^{2n}$ with

$$C[k] = \bigcirc_{i+j=k} A[i] \bullet B[j].$$

A prominent example of a convolution problem is $(\min, +)$-Convolution discussed above; another similarly prominent example is $(\max, \min)$-Convolution which can be solved in $\tilde{O}(n^{3/2})$ time [8]. For our purposes, it is convenient to look at a *skewed* variant of this problem:

▶ **Definition 2.5** (Skewed Convolution). *Given two vectors $A = (A[i])_{i=0}^{n}$, $B = (B[j])_{j=0}^{n}$, we define the $(\max, \min)$-Skewed-Convolution problem to be the problem of computing the vector $C = (C[k])_{k=0}^{2n}$ where the kth entry in C equals*

$$C[k] = \max_{i+j=k} \min\{A[i], B[j] + k\}$$

for each $k \in \{0, \ldots, 2n\}$.

The main technical result of this paper is an algorithm for $(\max, \min)$-Skewed-Convolution that is significantly faster than the naive $O(n^2)$ time algorithm.

▶ **Theorem 2.6.** *The $(\max, \min)$-Skewed-Convolution problem for vectors of length n can be solved in $\tilde{O}(n^{7/4})$ time.*

3 Algorithm via Sumsets and Subset Sums

In the following section, we provide a proof of Theorem 1.6 by presenting an algorithm for $1||\sum p_j U_j$ running in $\tilde{O}(\min\{P \cdot D_\#, P + D\})$ time. Recall that $J = \{1, \ldots, n\}$ denotes our input set of jobs, and p_j and d_j respectively denote the processing time and due date of job $j \in \{1, \ldots, n\}$. Our goal is to determine the minimum total processing time of tardy jobs in any schedule for J. Throughout the section we let $d^{(1)} < \cdots < d^{(D_\#)}$ denote the $D_\# \leq n$ *different* due dates of the jobs in J.

A key observation for the $1||\sum p_jU_j$ problem, used already by Lawler and Moore, is that any instance of the problem always has an optimal schedule of a specific type, namely an Earliest Due Date schedule. An Earliest Due Date (EDD) schedule is a schedule $\pi : J \to \{1, \ldots, n\}$ such that

- any early job precedes all late jobs in π, and
- any early job precedes all early jobs with later due dates.

In other words, in an EDD schedule all early jobs are scheduled before all tardy jobs, and all early jobs are scheduled in non-decreasing order of due dates.

▶ **Lemma 3.1** ([10]). *Any $1||\sum p_jU_j$ instance has an optimal schedule which is EDD.*

The $D_\#$-many due dates in our instance partition the input set of job J in a natural manner: Define $J_i = \{j : d_j = d^{(i)}\}$ for each $i \in \{1, \ldots, D_\#\}$. Furthermore, let $X_i = \{p_j : j \in J_i\}$ the processing-times of job in J_i. According to Lemma 3.1 above, we can restrict our attention to EDD schedules. Constructing such a schedule corresponds to choosing a subset $E_i \subseteq J_i$ for each due date $d^{(i)}$ such that $\sum_{j \in E_\ell, \ell \leq i} p_j \leq d^{(i)}$ holds for each $i \in \{1, \ldots, D_\#\}$. Moreover, the optimal EDD schedule maximizes the total sum of processing times in all selected E_i's.

Our algorithm is given in Algorithm 1. It successively computes sets $S_1, \ldots, S_{D_\#}$, where set S_i corresponds to the set of jobs $J_1 \cup \cdots \cup J_i$. In particular, S_i includes the total processing-time of any possible set-family of early jobs $\{E_1, \ldots, E_i\}$. Thus, each $x \in S_i$ corresponds to the total processing time of early jobs in a subset of $J_1 \cup \cdots \cup J_i$. The maximum value $x \in S_{D_\#}$ therefore corresponds to the maximum total processing time of early jobs in any schedule for J. Thus, the algorithm terminates by returning the optimal total weight of tardy jobs $P - x$.

Algorithm 1 SumsetScheduler(J).

```
Let $d^{(1)} < \ldots < d^{(D_\#)}$ denote the different due dates of jobs in $J$.
Compute $X_i = \{p_j : d_j = d^{(i)}\}$ for each $i \in \{1, \ldots, D_\#\}$.
Compute $\mathcal{S}(X_1), \ldots, \mathcal{S}(X_{D_\#})$.
Let $S_0 = \emptyset$.
for $i = 1, \ldots, D_\#$ do
    – Compute $S_i = S_{i-1} \oplus \mathcal{S}(X_i)$.
    – Remove any $x \in S_i$ with $x > d^{(i)}$.
Return $P - x$, where $x$ is the maximum value in $S_{D_\#}$.
```

Correctness of our algorithm follows immediately from the definitions of sumsets and subset sums, and from the fact that we prune out elements $x \in S_i$ with $x > d^{(i)}$ at each step of the algorithm. This is stated more formally in the lemma below.

▶ **Lemma 3.2.** *Let $i \in \{1, \ldots, D_\#\}$, and let S_i be the set of integers at the end of the second step of $5(i)$. Then $x \in S_i$ if and only if there are sets of jobs $E_1 \subseteq J_1, \ldots, E_i \subseteq J_i$ such that*

- $\sum_{j \in \bigcup_{\ell=1}^{i} E_\ell} p_j = x$, *and*
- $\sum_{j \in E_\ell, \ell \leq i_0} p_j \leq d^{(i_0)}$ *holds for each $i_0 \in \{1, \ldots, i\}$.*

Proof. The proof is by induction on i. For $i = 1$, note that $S_1 = \mathcal{S}(X_1) \setminus \{x : x > d^{(1)}\}$ at the end of step $5(1)$. Since $\mathcal{S}(X_1)$ includes the total processing time of any subset of jobs in J_1, the first condition of the lemma holds. Since $\{x : x > d^{(1)}\}$ includes all integers violating the second condition of the lemma, the second condition holds.

Let $i > 1$, and assume the lemma holds for $i - 1$. Consider some $x \in S_i$ at the end of the second step of $5(i)$. Then by Definition 2.1, we have $x = x_1 + x_2$ for some $x_1 \in S_{i-1}$ and $x_2 \in \mathcal{S}(X_i)$ due the first step of $5(i)$. By definition of $\mathcal{S}(X_i)$, there is some $E_i \subseteq J_i$ with total processing time x_2. By our inductive hypothesis there is $E_1 \subseteq J_1, \ldots, E_{i-1} \subseteq J_{i-1}$ such that $\sum_{j \in \bigcup_{\ell=1}^{i} E_\ell} p_j = x_1$, and $\sum_{j \in E_\ell, \ell \le i_0} p_j \le d^{(i_0)}$ holds for each $i_0 \in \{1, \ldots, i-1\}$. Furthermore, by the second step of $5(i)$, we know that $\sum_{j \in E_\ell, \ell \le i} p_j = x \le d^{(i)}$. Thus, $E_1, \ldots, E_i$ satisfy both conditions of the lemma. ◀

Let us next analyze the time complexity of the SumsetScheduler algorithm. Steps 1 and 2 can be both performed in $\tilde{O}(n) = \tilde{O}(P)$ time. Next observe that step 3 can be done in total $\tilde{O}(P)$ time using Lemma 2.4, as $X_2, \ldots, X_{D_\#}$ is a partition of the set of all processing times of J, and these all sum up to P. Next, according to Lemma 2.2, each sumset operation at step 5 can be done in time proportional to the largest element in the two sets, which is always at most P. Thus, since we perform at most $D_\#$ sumset operations, the merging step requires $\tilde{O}(D_\# \cdot P)$ time, which gives us the total running time of the algorithm above.

Another way to analyze the running time of SumsetScheduler is to observe that the maximum element participating in the ith sumset is bounded by $d^{(i+1)}$. It follows that we can write the running time of the merging step as $\tilde{O}(D)$, where $D = \sum_{i=1}^{D_\#} d^{(i)}$. Thus, we have just shown that $1||\sum p_j U_j$ can be solved in $\tilde{O}(\min\{D_\# \cdot P, D + P\})$ time, completing the proof of Theorem 1.6.

4 Algorithm via Fast Skewed Convolutions

We next present our $\tilde{O}(P^{7/4})$ time algorithm for $1||\sum p_j U_j$, providing a proof of Theorem 1.5. As in the previous section, we let $d^{(1)} < \cdots < d^{(D_\#)}$ denote the $D_\# \le n$ different due dates of the input jobs J, and $J_i = \{j : d_j = d^{(i)}\}$ and $X_i = \{p_j : j \in J_i\}$ as in Section 3 for each $i \in \{1, \ldots, D_\#\}$.

For a consecutive subset of indices $I = \{i_0, i_0 + 1, \ldots, i_1\}$, with $i_0, \ldots, i_1 \in \{1, \ldots, D_\#\}$, we define a vector $M(I)$, where $M(I)[x]$ equals the latest (that is, maximum) time point x_0 for which there is a subset of the jobs in $\bigcup_{i \in I} J_i$ with total processing time equal to x that can all be scheduled early in an EDD schedule starting at x_0. If no such subset of jobs exists, we define $M(I)[x] = +\infty$.

For a singleton set $I = \{i\}$, the vector $M(I)$ is easy to compute once we have computed the set $\mathcal{S}(X_i)$:

$$M(\{i\})[x] = \begin{cases} d^{(i)} - x & \text{if } x \in \mathcal{S}(X_i) \text{ and } x \le d^{(i)}, \\ +\infty & \text{otherwise.} \end{cases} \tag{1}$$

For larger sets of indices, we have the following lemma.

▶ **Lemma 4.1.** *Let $I_1 = \{i_0, i_0 + 1, \ldots, i_1\}$ and $I_2 = \{i_1 + 1, i_1 + 2, \ldots, i_2\}$ be any two sets of consecutive indices with $i_0, \ldots, i_1, \ldots, i_2 \in \{1, \ldots, D_\#\}$. Then for any value x we have:*

$$M(I_1 \cup I_2)[x] = \max_{x_1 + x_2 = x} \min\{M(I_1)[x_1], M(I_2)[x_2] - x_1\}.$$

Proof. Let $I = I_1 \cup I_2$. Then $M(I)[x]$ is the latest time point after which a subset of jobs $J^* \subseteq \bigcup_{i \in I} J_i$ of total processing time x can be scheduled early in an EDD schedule. Let x_1 and x_2 be the total processing times of jobs in $J_1^* = J^* \cap \left(\bigcup_{i \in I_1} J_i\right)$ and $J_2^* = J^* \cap \left(\bigcup_{i \in I_2} J_i\right)$, respectively. Then $x = x_1 + x_2$. Clearly, $M(I)[x] \le M(I_1)[x_1]$, since we have to start scheduling the jobs in J_1^* at time $M(I_1)[x_1]$ by latest. Similarly, it holds that

$M(I)[x] \leq M(I_2)[x_2] - x_1$ since the jobs in J_2^* are scheduled at latest at $M(I_2)[x_2]$ and the jobs in J_1^* have to be processed before that time point in an EDD schedule. In combination, we have shown that LHS $\leq$ RHS in the equation of the lemma.

To prove that LHS $\geq$ RHS, we construct a feasible schedule for jobs in $\bigcup_{i \in I} J_i$ starting at RHS. Let x_1 and x_2 be the two values with $x_1 + x_2 = x$ that maximize RHS. Then there is a schedule which schedules some jobs $J_1^* \subseteq \bigcup_{i \in I_1} J_i$ of total processing time x_1 beginning at time $\min\{M(I_1)[x_1], M(I_2)[x_2] - x_1\} \leq M(I_1)[x_1]$, followed by a another subset of jobs $J_2^* \subseteq \bigcup_{i \in I_2} J_i$ of total processing time x_2 starting at time $\min\{M(I_1)[x_1], M(I_2)[x_2] - x_1\} + x_1 \leq M(I_2)[x_2]$. This is a feasible schedule starting at time RHS for a subset of jobs in $\bigcup_{i \in I} J_i$ which has total processing time x. ◀

Note that the equation given in Lemma 4.1 is close but not precisely the equation defined in Definition 2.5 for the $(\min, \max)$-Skewed-Convolution problem. Nevertheless, the next lemma shows that we can easily translate between these two concepts.

▶ **Lemma 4.2.** *Let A and B be two integer vectors of P entries each. Given an algorithm for computing the* $(\max, \min)$*-Skewed-Convolution of A and B in $T(P)$ time, we can compute in $T(P) + O(P)$ time the vector $C = A \otimes B$ defined by*

$$C[x] = \max_{x_1 + x_2 = x} \min\{A[x_1], B[x_2] - x_1\}.$$

Proof. Given A and B, construct two auxiliary vectors A_0 and B_0 defined by $A_0[x] = B[x] + x$ and $B_0[x] = A[x]$ for each entry x. Compute the $(\max, \min)$-Skewed-Convolution of A_0 and B_0, and let C_0 denote the resulting vector. We claim that the vector C defined by $C[x] = C_0[x] - x$ equals $A \otimes B$. Indeed, we have

$$\begin{aligned} C_0[x] - x &= \max_{x_1 + x_2 = x} \min\{A_0[x_1], B_0[x_2] + x\} - x \\ &= \max_{x_1 + x_2 = x} \min\{A_0[x_1] - x, B_0[x_2]\} \\ &= \max_{x_1 + x_2 = x} \min\{B[x_1] + x_1 - x, A[x_2]\} \\ &= \max_{x_1 + x_2 = x} \min\{B[x_1] - x_2, A[x_2]\} \\ &= \max_{x_1 + x_2 = x} \min\{A[x_1], B[x_2] - x_1\}, \end{aligned}$$

where in the third step we expanded the definition of A_0 and B_0 and in the last step we used the symmetry of x_1 and x_2. ◀

We are now in position to describe our algorithm called ConvScheduler which is depicted in Algorithm 2. The algorithm first computes the subset sums $\mathcal{S}(X_1), \ldots, \mathcal{S}(X_{D_\#})$, and the set of vectors $\mathcal{M} = \{M_1, \ldots, M_{D_\#}\}$. Following this, it iteratively combines every two consecutive vectors in $\mathcal{M}$ by using the $\otimes$ operation. The algorithm terminates when $\mathcal{M} = \{M_1\}$, where at this stage M_1 corresponds to the entire set of input jobs J. It then returns $P - x$, where x is the maximum value with $M_1[x] < \infty$; by definition, this corresponds to a schedule for J with $P - x$ total processing time of tardy jobs. For convenience of presentation, we assume that $D_\#$ is a power of 2.

Correctness of this algorithm follows directly from Lemma 4.1. To analyze its time complexity, observe that steps 1–4 can be done in $\tilde{O}(P)$ time (using Lemma 2.4). Step 5 is performed $O(\log D_\#) = O(\log P)$ times, and each step requires a total of $\tilde{O}(P^{7/4})$ time according to Theorem 2.6, as the total sizes of all vectors at each step is $O(P)$. Finally, step 6 requires $O(P)$ time. Summing up, this gives us a total running time of $\tilde{O}(P^{7/4})$, and completes the proof of Theorem 1.5 (apart from the proof of Theorem 2.6).

Algorithm 2 ConvScheduler(J).

```
Let $d^{(1)} < \ldots < d^{(D_\#)}$ denote the different due dates of jobs in $J$.
Compute $X_i = \{p_j : d_j = d^{(i)}\}$ for each $i \in \{1, \ldots, D_\#\}$.
Compute $\mathcal{S}(X_1), \ldots, \mathcal{S}(X_{D_\#})$.
Compute $\mathcal{M} = \{M_1 = M(1), \ldots, M_{D_\#} = M(D_\#)\}$.
while $|\mathcal{M}| > 1$ do
    – Compute $M_i = M_{2i-1} \otimes M_{2i}$ for each $i \in \{1, \ldots, |\mathcal{M}|/2\}$.
Return $P - x$, where $x$ is the maximum value with $M_1[x] < \infty$.
```

5 Fast Skewed Convolutions

In the following section we present our algorithm for $(\max, \min)$-Skewed-Convolution, and provide a proof for Theorem 2.6. Let $A = (A[i])_{i=0}^n$ and $B = (B[j])_{j=0}^n$ denote the input vectors for the problem throughout the section.

We begin by first defining the problem slightly more generally, in order to facilitate our recursive strategy later on. For this, for each integer $\ell \in \{0, \ldots, \log n\}$, let $A^\ell = \lfloor A/2^\ell \rfloor$ and $B^\ell = \lfloor B/2^\ell \rfloor$, where rounding is done component-wise. We will compute vectors $C^\ell = (C^\ell[k])_{k=0}^{2n}$ defined by:

$$C^\ell[k] = \max_{i+j=k} \min\{A^\ell[i], B^\ell[j] + \lfloor k/2^\ell \rfloor\}.$$

Observe that a solution for $\ell = 0$ yields a solution to the original $(\max, \min)$-Skewed-Convolution problem, and for $\ell \geq \log 2n$ the problem degenerates to $(\max, \min)$-Convolution.

We next define a particular kind of additive approximation of vectors C^ℓ. We say that a vector D^ℓ is a *good approximation* of C^ℓ if $C^\ell[k] - 2 \leq D^\ell[k] \leq C^\ell[k]$ for each $k \in \{0, \ldots, 2n\}$. Now, the main technical part of our algorithm is encapsulated in the following lemma.

▶ **Lemma 5.1.** *There is an algorithm that computes C^ℓ in $\tilde{O}(n^{7/4})$ time, given A^ℓ, B^ℓ, and a good approximation D^ℓ of C^ℓ.*

We postpone the proof of Lemma 5.1 for now, and instead show that it directly yields our desired algorithm for $(\max, \min)$-Skewed-Convolution:

Proof of Theorem 2.6. In order to compute $C = C^0$, we perform an (inverse) induction on ℓ: As mentioned before, if $\ell \geq \log 2n$, then we can neglect the "$+ \lfloor k/2^\ell \rfloor$" term and compute C^ℓ in $\tilde{O}(n^{3/2}) = \tilde{O}(n^{7/4})$ time using a single $(\max, \min)$-Convolution computation [8].

For the inductive step, let $\ell < \log 2n$ and assume that we have already computed $C^{\ell+1}$. We construct the vector $D^\ell = 2C^{\ell+1}$, and argue that it is a good approximation of C^ℓ. Indeed, for each entry k, on the one hand, we have:

$$\begin{aligned} D^\ell[k] &= 2C^{\ell+1}[k] = 2 \cdot \max_{i+j=k} \min\{\lfloor A^\ell[i]/2 \rfloor, \lfloor B^\ell[j]/2 \rfloor + \lfloor k/2^{\ell+1} \rfloor\} \\ &\leq \max_{i+j=k} \min\{A^\ell[i], B^\ell[j] + \lfloor k/2^\ell \rfloor\} = C^\ell[k]; \end{aligned}$$

and on the other hand, we have:

$$\begin{aligned} D^\ell[k] &= 2C^{\ell+1}[k] = 2 \cdot \max_{i+j=k} \min\{\lfloor A^\ell[i]/2 \rfloor, \lfloor B^\ell[j]/2 \rfloor + \lfloor k/2^{\ell+1} \rfloor\} \\ &\geq \max_{i+j=k} \min\{A^\ell[i] - 1, B^\ell[j] + \lfloor k/2^\ell \rfloor - 2\} \geq C^\ell[k] - 2. \end{aligned}$$

Thus, using D^ℓ we can apply Lemma 5.1 above to obtain C^ℓ in $\tilde{O}(n^{7/4})$ time. Since there are $O(\log n)$ inductive steps overall, this is also the overall time complexity of the algorithm. ◀

It remains to prove Lemma 5.1. Recall that we are given A^ℓ, B^ℓ, and D^ℓ, and our goal is to compute the vector C^ℓ in $\tilde{O}(n^{7/4})$ time. We construct two vectors L^ℓ and R^ℓ with $2n$ entries each, defined by

$$L^\ell[k] = \max\left\{A^\ell[i_0] : \begin{array}{c} A^\ell[i_0] \le B^\ell[k-i_0] + \lfloor k/2^\ell \rfloor \text{ and} \\ D^\ell[k] \le A^\ell[i_0] \le D^\ell[k]+2 \end{array}\right\},$$

and

$$R^\ell[k] = \max\left\{B^\ell[j_0] + \lfloor k/2^\ell \rfloor : \begin{array}{c} B^\ell[j_0] + \lfloor k/2^\ell \rfloor \le A^\ell[k-j_0] \text{ and} \\ D^\ell[k] \le B^\ell[j_0] + \lfloor k/2^\ell \rfloor \le D^\ell[k]+2 \end{array}\right\}$$

for $k \in \{0, \dots, 2n\}$. That is, $L^\ell[k]$ and $R^\ell[k]$ respectively capture the largest value attained as the left-hand side or right-hand side of the inner min-operation in $C^\ell[k]$, as long as that value lies in the feasible region approximated by $D^\ell[k]$. Since D^ℓ is a good approximation, the following lemma is immediate from the definitions:

▶ **Lemma 5.2.** *$C^\ell[k] = \max\{L^\ell[k], R^\ell[k]\}$ for each $k \in \{0, \dots, 2n\}$.*

According to Lemma 5.2, it suffices to compute L^ℓ and R^ℓ. We focus on computing L^ℓ as the algorithm for computing R^ℓ follows after applying minor modifications.

Let $0 < \delta < 1$ be a fixed constant to be determined later. We say that an index $k \in \{0, \dots, n\}$ is *light* if

$$|\{i : D^\ell[k] \le A^\ell[i] \le D^\ell[k]+2\}| \le n^\delta.$$

Informally, k is light if the number of candidate entries $A^\ell[i]$ which can equal $C^\ell[k]$ is relatively small (recall that $D^\ell[k] \le C^\ell[k] \le D^\ell[k]+2$, as D^ℓ is a good approximation of C^ℓ). If k is not light then we say that it is *heavy.*

Our algorithm for computing L^ℓ proceeds in three main steps: In the first step it handles all light indices, in the second step it sparsifies the input vector, and in the third step it handles all heavy indices:

- *Light indices:* We begin by iterating over all light indices $k \in \{0, \dots, 2n\}$. For each light index k, we iterate over all entries $A^\ell[i]$ satisfying $D^\ell[k] \le A^\ell[i] \le D^\ell[k]+2$, and set $L^\ell[k]$ to be the maximum $A^\ell[i]$ among those entries with $A^\ell[i] \le B^\ell[k-i] + \lfloor k/2^\ell \rfloor$. Note that after this step, we have

  $$L^\ell[k] = \max\{A^\ell[i_0] : A^\ell[i_0] \le B^\ell[k-i_0] + \lfloor k/2^\ell \rfloor \text{ and } D^\ell[k] \le A^\ell[i_0] \le D^\ell[k]+2\}$$

  for each light index k.

- *Sparsification step:* After dealing with the light indices, several entries of A^ℓ become redundant. Consider an entry $A^\ell[i]$ for which $|\{i_0 : A^\ell[i]-2 \le A^\ell[i_0] \le A^\ell[i]+2\}| \le n^\delta$. Then all indices k for which $L^\ell[k]$ might equal $A^\ell[i]$ must be light, and are therefore already dealt with in the previous step. Consequently, it is safe to replace $A^\ell[i]$ by $-\infty$ so that $A^\ell[i]$ no longer plays a role in the remaining computation.

- *Heavy indices:* After the sparsification step A^ℓ contains few distinct values. Thus, our approach is to fix any such value v and detect whether $L^\ell[k] \ge v$. To that end, we translate the problem into an instance of $(\max, \min)$-Convolution: Let $(A^\ell_v[i])_{i=0}^n$ be an be an indicator-like vector defined by $A^\ell_v[i] = +\infty$ if $A^\ell[i] = v$, and otherwise $A^\ell_v[i] = -\infty$. We next compute the vector L^ℓ_v defined by $L^\ell_v[k] = \lfloor k/2^\ell \rfloor + \max_{i+j=k} \min\{A^\ell_v[i], B^\ell[j]\}$ using a single computation of $(\max, \min)$-Convolution.

We choose

$$L^\ell[k] = \max\{v : L^\ell_v[k] \geq v \text{ and } D^\ell[k] \leq v \leq D^\ell[k] + 2\}$$

for any heavy index k and claim that $L^\ell[k]$ equals $\max\{A^\ell[i_0] : A^\ell[i_0] \leq B^\ell[k-i_0]+\lfloor k/2^\ell \rfloor\}$. On the one hand, if $L^\ell_v[k] \geq v$ then there are indices i and j with $i + j = k$ for which $A^\ell[i] = v$ and $B^\ell[j] + \lfloor k/2^\ell \rfloor \geq A^\ell[i] = v$. Thus, the computed value $L^\ell[k]$ is not greater than

$$L^\ell[k] \leq \max\{A^\ell[i_0] : A^\ell[i_0] \leq B^\ell[k - i_0] + \lfloor k/2^\ell \rfloor \text{ and } D^\ell[k] \leq A^\ell[i_0] \leq D^\ell[k] + 2\}.$$

On the other hand, for all values v for which $A^\ell[i] = v$ for some $i \in \{0, \dots, n\}$, we have if $v = A^\ell[i] \leq B^\ell[k - i] + \lfloor k/2^\ell \rfloor$ then $A^\ell_v[i] = -\infty$, which in turn implies that $A^\ell_v[i] \geq B^\ell[k - i] + \lfloor k/2^\ell \rfloor \geq A^\ell[i] = v$. Thus, our selection of $L^\ell[k]$ is also at least as large as

$$L^\ell[k] \geq \max\{A^\ell[i_0] : A^\ell[i_0] \leq B^\ell[k - i_0] + \lfloor k/2^\ell \rfloor \text{ and } D^\ell[k] \leq A^\ell[i_0] \leq D^\ell[k] + 2\},$$

and hence, these two values must be equal.

This completes the description of our algorithm. As we argued its correctness above, what remains is to analyze its time complexity. Note that we can determine in $O(\log n)$ time whether an index k is light or heavy, by first sorting the values in A^ℓ. For each light index k, determining $L^\ell[k]$ can be done in $O(n^\delta)$ time (on the sorted A^ℓ), giving us a total of $\tilde{O}(n^{1+\delta})$ time for the first step. For the second step, we can determine whether a given entry $A^\ell[i]$ can be replaced with $-\infty$ in $O(\log n)$ time, giving us a total of $\tilde{O}(n)$ time for this step.

Consider then the final step of the algorithm. Observe that after exhausting the sparsification step, A^ℓ contains at most $O(n^{1-\delta})$ many distinct values: For any surviving value v, there is another (perhaps different) value v' of difference at most 2 from v that occurs at least n^δ times in A^ℓ, and so there can only be at most $O(n^{1-\delta})$ such distinct values. Thus, the running time of this step is dominated by the running time of $O(n^{1-\delta})$ (max, min)-Convolution computations, each requiring $\tilde{O}(n^{3/2})$ time using the algorithm of [8], giving us a total of $\tilde{O}(n^{5/2-\delta})$ time for this step.

Thus, the running time of our algorithm is dominated by the $\tilde{O}(n^{1+\delta})$ running time of its first step, and the $\tilde{O}(n^{5/2-\delta})$ running time of its last step. Choosing $\delta = 3/4$ gives us $\tilde{O}(n^{7/4})$ time for both steps, which is the time promised by Lemma 5.1. Thus, Lemma 5.1 holds.

6 Discussion and Open Problems

In this paper we presented two algorithms for the $1||\sum p_j U_j$ problem; the first running in $\tilde{O}(P^{7/4})$ time, and the second running in $\tilde{O}(\min\{P \cdot D_\#, P + D\})$ time. Both algorithms provide the first improvements over the classical Lawler and Moore algorithm in 50 years, and use more sophisticated tools such as polynomial multiplication and fast convolutions. Moreover, both algorithms are very easy to implement given a standard ready made FFT implementation for fast polynomial multiplication. Nevertheless, there are still a few ways which our results can be improved or extended:

- *Multiple machines:* A natural extension of the $1||\sum p_j U_j$ problem is to the setting of multiple parallel machines, the $Pm||\sum p_j U_j$. Lawler and Moore's algorithm can be used to solve $Pm||\sum p_j U_j$ in $O(P^m \cdot n)$ time, where m is the number of machines. A priori, there is no reason to believe that this cannot be improved to $\tilde{O}(P^m)$, or even better. It is not hard to extend the algorithm in Theorem 1.6 to an algorithm with running time $\tilde{O}(P^m \cdot D_\#)$ for the m parallel machine setting, by using m-variate polynomials for implementing sumsets in Lemma 2.2. However, a similar extension for the algorithm in Theorem 1.5 is far less direct.

- *Even faster skewed convolutions:* We have no indication that our algorithm for $(\max, \min)$-Skewed-Convolution is the fastest possible. It would interesting to see whether one can improve its time complexity, say to $\tilde{O}(P^{3/2})$. Naturally, any such improvement would directly improve Theorem 1.5.

  Conversely, one could try to obtain some sort of lower bound for the problem, possibly in the same vein as Theorem 1.2. Improving the time complexity beyond $\tilde{O}(P^{3/2})$ seems difficult as this would directly imply an improvement to the $(\max, \min)$-Convolution problem. Indeed, let A, B be a given $(\max, \min)$-Convolution instance and construct vectors A_0, B_0 with $A_0[i] = N \cdot A[i]$ and $B_0[j] = N \cdot B[j]$ for $N = 2n + 1$. If C_0 is the $(\max, \min)$-Skewed-Convolution of A_0 and B_0 (that is, $C_0[k] = \max_{i+j=k} \min\{A_0[i], B_0[j] + k\}$), then the vector C with $C[k] = \lfloor C_0[k]/N \rfloor$ is the $(\max, \min)$-Convolution of A and B.

- *Other scheduling problems:* Finally, it will be interesting to see other scheduling problems where the techniques used in this paper can be applied. A good first place to start might be to look at other problems which directly generalize Subset Sum.

References

1 Amir Abboud, Karl Bringmann, Danny Hermelin, and Dvir Shabtay. SETH-based lower bounds for subset sum and bicriteria path. *Proc. of of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 41–57, 2019.

2 Karl Bringmann. A near-linear pseudopolynomial time algorithm for subset sum. In *Proc. of of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1073–1084, 2017.

3 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition.* The MIT Press, 3rd edition, 2009.

4 Marek Cygan, Marcin Mucha, Karol Wegrzycki, and Michal Wlodarczyk. On problems equivalent to (min, +)-convolution. *ACM Trans. Algorithms*, 15(1):14:1–14:25, 2019.

5 Ronald L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429, 1969.

6 Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

7 Konstantinos Koiliaris and Chao Xu. A faster pseudopolynomial time algorithm for subset sum. In *Proc. of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1062–1072, 2017.

8 Sambasiva R. Kosaraju. Efficient tree pattern matching. In *Proc. of the 30th annual symposium on Foundations Of Computer Science (FOCS)*, pages 178–183, 1989.

9 Marvin Künnemann, Ramamohan Paturi, and Stefan Schneider. On the fine-grained complexity of one-dimensional dynamic programming. In *Proc. of the 44th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 21:1–21:15, 2017.

10 Eugene L. Lawler and James M. Moore. A functional equation and its application to resource allocation and sequencing problems. *Management Science*, 16(1):77–84, 1969.

Fréchet Distance for Uncertain Curves

Kevin Buchin
Department of Mathematics and Computer Science, TU Eindhoven, Netherlands
k.a.buchin@tue.nl

Chenglin Fan
Department of Computer Science, University of Texas at Dallas, Richardson, TX, USA
cxf160130@utdallas.edu

Maarten Löffler
Department of Information and Computing Sciences, Utrecht University, Netherlands
m.loffler@uu.nl

Aleksandr Popov
Department of Mathematics and Computer Science, TU Eindhoven, Netherlands
a.popov@tue.nl

Benjamin Raichel
Department of Computer Science, University of Texas at Dallas, Richardson, TX, USA
benjamin.raichel@utdallas.edu

Marcel Roeloffzen
Department of Mathematics and Computer Science, TU Eindhoven, Netherlands
m.j.m.roeloffzen@tue.nl

Abstract

In this paper we study a wide range of variants for computing the (discrete and continuous) Fréchet distance between uncertain curves. We define an uncertain curve as a sequence of *uncertainty regions*, where each region is a disk, a line segment, or a set of points. A *realisation* of a curve is a polyline connecting one point from each region. Given an uncertain curve and a second (certain or uncertain) curve, we seek to compute the lower and upper bound Fréchet distance, which are the minimum and maximum Fréchet distance for any realisations of the curves.

We prove that both problems are NP-hard for the continuous Fréchet distance, and the upper bound problem remains hard for the discrete Fréchet distance. In contrast, the lower bound discrete Fréchet distance can be computed in polynomial time using dynamic programming. Furthermore, we show that computing the expected discrete or continuous Fréchet distance is #P-hard when the uncertainty regions are modelled as point sets or line segments.

On the positive side, we argue that in any constant dimension there is a FPTAS for the lower bound problem when Δ/δ is polynomially bounded, where δ is the Fréchet distance and Δ bounds the diameter of the regions. We then argue there is a near-linear-time 3-approximation for the decision problem when the regions are convex and roughly δ-separated. Finally, we study the setting with Sakoe–Chiba bands, restricting the alignment of the two curves, and give polynomial-time algorithms for upper bound and expected (discrete) Fréchet distance for point-set-modelled uncertainty regions.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Curves, Uncertainty, Fréchet Distance, Hardness

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.20

Category Track A: Algorithms, Complexity and Games

Related Version The full version is available at `https://arxiv.org/abs/2004.11862` [13].

Funding *Chenglin Fan*: NSF CRII Award 1566137 and CAREER Award 1750780
Aleksandr Popov: Netherlands Organisation for Scientific Research (NWO), project no. 612.001.801
Benjamin Raichel: NSF CRII Award 1566137 and CAREER Award 1750780
Marcel Roeloffzen: Netherlands Organisation for Scientific Research (NWO), project no. 628.011.005

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 20; pp. 20:1–20:20
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

In this paper we investigate the well-studied topic of curve similarity in the context of the burgeoning area of geometric computing under uncertainty. While classical algorithms in computational geometry typically assume the input point locations are known exactly, in recent years there has been a concentrated effort to adapt these algorithms to uncertain inputs, which can more faithfully model real-world inputs. The need to model such uncertain inputs is perhaps no more clear than for the location data of a moving object obtained from physical devices, which is inherently imprecise due to issues such as measurement error, sampling error, and network latency [42, 43]. Moreover, to ensure location privacy, one may purposely add uncertainty to the data by adding noise or reporting positions as geometric regions rather than points. (See the survey by Krumm [35] and the references therein.)

Here we consider both the continuous and discrete Fréchet distance for uncertain curves. Given the applications above, our uncertain input is given as a sequence of compact regions, from which a polygonal curve is *realised* by selecting one point from each region. Our goal is to find, for a given pair of uncertain curves, the upper bound, lower bound, and expected Fréchet distance, where the upper (resp. lower) bound Fréchet distance is the maximum (resp. minimum) distance over any realisation. For the expected Fréchet distance we assume a probability distribution is provided that describes how each vertex on a curve is chosen from the compact region. Previously, Ahn et al. [5] considered the lower bound problem for the discrete Fréchet distance, giving a polynomial-time algorithm for points in constant dimension. The authors also gave efficient approximation algorithms for the discrete upper bound Fréchet distance for uncertain inputs, where the approximation factor depends on the spread of the region diameters or how well-separated they are. Subsequently, Fan and Zhu showed that the discrete upper bound Fréchet distance is NP-hard for uncertain inputs modelled as thin rectangles [25]. To our knowledge, we are the first to consider either variant for the continuous Fréchet case, and the first to consider the expected Fréchet distance.

1.1 Previous Work

Geometric computing under uncertainty. The two most common models of geometric uncertainty are the locational model [36] and the existential model [46, 48]. In the existential model the location of an uncertain point is known, but the point may not be present; in the locational model we know that each uncertain point exists, but not its exact location.

In this paper we consider the locational model. Each uncertain point is a set of potential locations. We call an uncertain point *indecisive* if the set of potential locations is finite, or *imprecise* if the set is not finite but is a convex region. A *realisation* of a set of uncertain points is a selection of one point from each uncertain point. The goal is typically to compute the realisation of a set of uncertain points that minimises or maximises some quantity (e.g. area, distance, perimeter) of some underlying geometric structure (e.g. convex hull, MST) [1, 7, 14, 17, 21, 23, 28, 34, 37, 38, 39, 45, 47]. By assigning a probability distribution to uncertain points, one can also consider the expectation or distribution of various measures [2, 4, 32, 41].

Fréchet distance. Computing the Fréchet distance between two precise curves can be done in near-quadratic time [3, 6, 12], and assuming the Strong Exponential Time Hypothesis (SETH) it cannot be computed or even approximated well in strongly subquadratic time [9, 15]. However, for several restricted versions the Fréchet distance can be calculated more quickly, for example for c-packed curves [20], when the edges are long [29], or when the alignment of

Table 1 Hardness results for the decision problems in this paper. Ahn et al. [5] solve the lower bound problem for disks, but their algorithm extends to the indecisive curves as well as line segment imprecision.

		indecisive	imprecise	
			disks	line segments
discrete Fréchet distance	LB	Polynomial [5]	Polynomial [5]	Polynomial [5]
	UB	NP-complete	NP-complete	NP-complete
	Exp	#P-hard	—	#P-hard
Fréchet distance	LB	Polynomial	—	NP-complete
	UB	NP-complete	NP-complete	NP-complete
	Exp	#P-hard	—	—

curves is restricted [11, 40]. Many variants of the problem have been considered: Fréchet distance with shortcuts [16, 19], weak Fréchet distance [6], discrete Fréchet distance [3, 22], Fréchet gap distance [24], Fréchet distance under translations [10, 26], and more.

1.2 Our Contributions

In this paper we present an extensive study of the Fréchet distance for uncertain curves. We provide a wide range of hardness results and present several approximations and polynomial-time solutions to restricted versions. We are the first to consider the continuous Fréchet distance in the uncertain setting, as well as the first to consider the expected Fréchet distance.

On the negative side, we present a plethora of hardness results (Table 1; details follow in Section 2). The hardness of the lower bound case is curious: while the variants discrete Fréchet distance on imprecise inputs [5] and, as we prove, continuous Fréchet distance on indecisive inputs both permit a simple dynamic programming solution, the variant continuous Fréchet distance on imprecise input has just enough (literal) wiggle room to show NP-hardness by reduction from SubsetSum.

We complement the lower bound hardness result by two approximation algorithms (Section 3). The first is a FPTAS for general uncertain curves in constant dimension when the ratio between the diameter of the uncertain points and the lower bound Fréchet distance is polynomially bounded. The second is a 3-approximation for separated imprecise curves, but uses a simpler greedy approach that runs in near-linear time.

The NP-hardness of the upper bound by a reduction from CNF-SAT is less surprising, but requires a careful set-up and analysis of the geometry to then extend it to a reduction from #CNF-SAT to the expected (discrete or continuous) Fréchet distance. However, by adding the common constraint that the alignment between the curves needs to stay within a Sakoe–Chiba [44] band of constant width (see Section 4 for definition and results), we can solve these problems in polynomial time for indecisive curves. Sakoe–Chiba bands are frequently used for time-series data [8, 33, 44] and trajectories [11, 18], when the alignment should (or is expected to) not vary too much from a certain "natural" alignment.

1.3 Preliminaries

Curves. Denote $[n] \equiv \{1, 2, \dots, n\}$. Consider a *sequence* of d-dimensional points $\pi = \langle p_1, p_2, \dots, p_n \rangle$. A *polygonal curve* π is defined by these points by linearly interpolating between the successive points and can be seen as a continuous function: $\pi(i + \alpha) = (1 - \alpha)p_i + \alpha p_{i+1}$ for $i \in [n-1]$ and $\alpha \in [0, 1]$. The *length* of such a curve is the number of

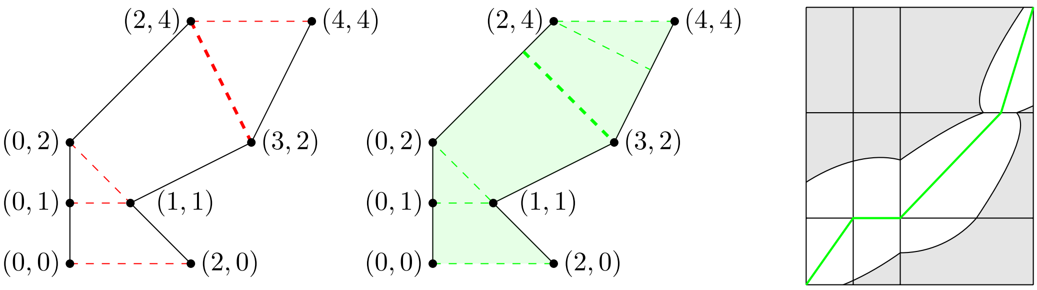


Figure 1 Left: Discrete Fréchet distance, where an optimal coupling is shown in dashed red lines. Middle: Fréchet distance, dashed green lines indicate specific values for δ for optimal functions ϕ_1, ϕ_2. Right: Free-space diagram for threshold $\delta = 2.15$. One can draw a monotonous path from the lower left corner to the upper right corner of the diagram, so the Fréchet distance between trajectories is below the threshold.

its vertices, $|\pi| = n$. Where we deem important to distinguish between points that are a part of the curve and other points, we denote the polygonal curve by $\pi = \langle \pi_1, \pi_2, \ldots, \pi_n \rangle$. We denote the *concatenation* of two polygonal curves π and σ of lengths n and m by $\pi \parallel \sigma$; the new curve follows π, then has a segment between $\pi(n)$ and $\sigma(1)$, and then follows σ. Similarly, $p \parallel q$ (or simply pq) denotes the line segment between points p and q. We can generalise this notation:

$$\Big\|_{i\in[n]} p_i \equiv p_1 \parallel p_2 \parallel \cdots \parallel p_n \equiv \pi \,.$$

We denote a *subcurve* from vertex i to j of curve π as $\pi[i : j] \equiv p_i \parallel p_{i+1} \parallel \cdots \parallel p_j$.

Metrics definitions. Given two points $x, y \in \mathbb{R}^d$, denote their Euclidean distance by $\|x - y\|$. For two compact sets $X, Y \subset \mathbb{R}^d$, denote their distance by $\|X - Y\| = \min_{x\in X, y\in Y} \|x - y\|$. Throughout we treat the dimension d as a small constant.

Let Φ_n denote the set of all *reparametrisations* of length n, defined as continuous non-decreasing functions $\phi : [0, 1] \to [1, n]$ where $\phi(0) = 1$ and $\phi(1) = n$. Given a pair of curves π and σ of lengths n and m, respectively, and corresponding reparametrisations $\phi_1 \in \Phi_n$ and $\phi_2 \in \Phi_m$, define $\text{width}_{\phi_1,\phi_2}(\pi, \sigma) = \max_{t\in[0,1]} \|\pi(\phi_1(t)) - \sigma(\phi_2(t))\|$.

The width represents the maximum distance between two points traversing the curves from start to end according to ϕ_1 and ϕ_2 (which allow varying speed, but no backtracking). The *Fréchet distance* $d_{\mathrm{F}}(\pi, \sigma)$ is defined as the minimum possible width over all such traversals:

$$d_{\mathrm{F}}(\pi, \sigma) = \inf_{\phi_1\in\Phi_n, \phi_2\in\Phi_m} \text{width}_{\phi_1,\phi_2}(\pi, \sigma) = \inf_{\phi_1\in\Phi_n, \phi_2\in\Phi_m} \max_{t\in[0,1]} \|\pi(\phi_1(t)) - \sigma(\phi_2(t))\| \,.$$

The *discrete Fréchet distance* $d_{\mathrm{dF}}(\pi, \sigma)$ is defined similarly, except that we do not traverse edges of the curves, but must jump from one vertex to the next on either or both curves. We define a valid *coupling* as a sequence $c = \langle (p_1, q_1), \ldots, (p_r, q_r) \rangle$ of pairs from $[n] \times [m]$ where $(p_1, q_1) = (1, 1)$, $(p_r, q_r) = (n, m)$, and, for any $i \in [r - 1]$ we have $(p_{i+1}, q_{i+1}) \in \{(p_i + 1, q_i), (p_i, q_i + 1), (p_i + 1, q_i + 1)\}$. Let $\mathcal{C}$ be the set of all valid couplings on curves of lengths n and m, then

$$d_{\mathrm{dF}}(\pi, \sigma) = \inf_{c\in\mathcal{C}} \max_{s\in[|c|]} \|\pi(p_s) - \sigma(q_s)\| \,,$$

where $c_s = (p_s, q_s)$ for all $s \in [|c|]$. Both distances, as well as a common approach to computing Fréchet distance, are illustrated in Figure 1.

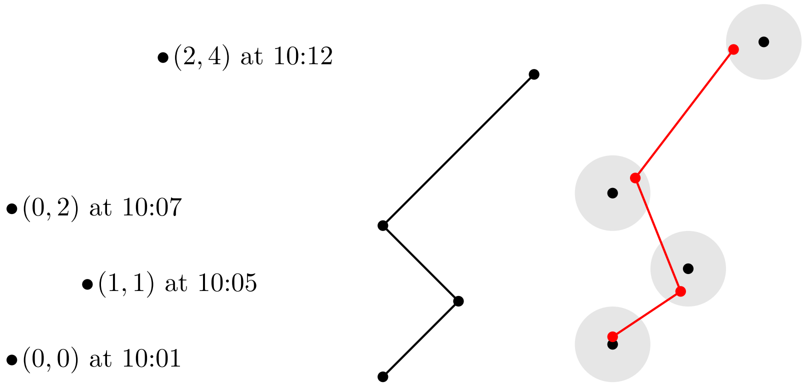


Figure 2 Left: Trajectory data. Middle: Polygonal curve on the data. Right: Imprecise curve with disks as imprecision regions and real curve.

Uncertainty model. An *uncertain* point is commonly represented as a compact region $U \subset \mathbb{R}^d$. Usually, it is a finite set of points, a disk, a rectangle, or a line segment. The intuition is that only one point from this region represents the true location of the point; however, we do not know which one. A *realisation* p of such a point is one of the points from the region U. When needed we assume the realisations are drawn from U according to a known probability distribution $\mathbb{P}$. We denote the diameter of any compact set (e.g. an uncertain point) $U \subset \mathbb{R}^d$ by $\operatorname{diam}(U) = \max_{p,q \in U} \|p - q\|$. An *indecisive* point is a special case of an uncertain point: it is a set of points $U = \{p_1, \dots, p_k\}$, where each point $p_i \in \mathbb{R}^d$ for $i \in [k]$. Similarly, an *imprecise* point is a compact convex region $U \subset \mathbb{R}^d$. We will often use disks or line segments as such regions. Note that a precise point is a special case of an indecisive point (set of size one) and an imprecise point (disk of radius zero).

Uncertain curves and distances. Define an *uncertain curve* as a sequence of uncertain points $\mathcal{U} = \langle U_1, \dots, U_n \rangle$. A *realisation* $\pi \Subset \mathcal{U}$ of an uncertain curve is a polygonal curve $\pi = \langle p_1, \dots, p_n \rangle$, where each p_i is a realisation of the corresponding uncertain point U_i. We denote the set of all realisations of an uncertain curve $\mathcal{U}$ by $\operatorname{Real}(\mathcal{U})$ (see Figure 2). In a probabilistic setting, we write $\pi \Subset_{\mathbb{P}} \mathcal{U}$ to denote that each point of π gets drawn from the corresponding uncertainty region independently according to distribution $\mathbb{P}$.

For uncertain curves $\mathcal{U}$ and $\mathcal{V}$, define the upper bound, lower bound, and expected discrete Fréchet distance (and extend to continuous Fréchet distance $d_{\mathrm{F}}^{\max}$, $d_{\mathrm{F}}^{\min}$, $d_{\mathrm{F}}^{\mathbb{E}(\mathbb{P})}$ using d_{F}) as:

$$d_{\mathrm{dF}}^{\max}(\mathcal{U}, \mathcal{V}) = \max_{\pi \Subset \mathcal{U}, \sigma \Subset \mathcal{V}} d_{\mathrm{dF}}(\pi, \sigma)\,, \qquad d_{\mathrm{dF}}^{\min}(\mathcal{U}, \mathcal{V}) = \min_{\pi \Subset \mathcal{U}, \sigma \Subset \mathcal{V}} d_{\mathrm{dF}}(\pi, \sigma)\,,$$

$$d_{\mathrm{dF}}^{\mathbb{E}(\mathbb{P})}(\mathcal{U}, \mathcal{V}) = \mathbb{E}_{\pi \Subset_{\mathbb{P}} \mathcal{U}, \sigma \Subset_{\mathbb{P}} \mathcal{V}}[d_{\mathrm{dF}}(\pi, \sigma)]\,.$$

If the distribution is clear from the context, we write $d_{\mathrm{F}}^{\mathbb{E}}$ and $d_{\mathrm{dF}}^{\mathbb{E}}$. The definitions above also apply if one of the curves is precise, as a precise curve is a special case of an uncertain curve.

2 Hardness Results

In this section, we first discuss the hardness results for the upper bound and expected value of the continuous and discrete Fréchet distance for indecisive and imprecise curves. We then show hardness of finding the lower bound continuous Fréchet distance on imprecise curves.

2.1 Upper Bound and Expected Fréchet Distance

We present proofs of NP-hardness and #P-hardness for the upper bound and expected Fréchet distance for both indecisive and imprecise curves by showing polynomial-time reductions from CNF-SAT and #CNF-SAT (counting version). We consider the upper bound problem for indecisive curves and then illustrate how the construction can be used to show #P-hardness for the expected Fréchet distance (both discrete and continuous). We then illustrate how the construction can be adapted to show hardness for imprecise curves. All our constructions are in two dimensions. The missing proofs can be found in the full version [13].

2.1.1 Upper Bound Fréchet Distance on Indecisive Curves

Define the following problem and its continuous counterpart, using $d_{\mathrm{F}}^{\max}$ instead:

▶ **Problem 1.** *Upper Bound Discrete Fréchet: Given two uncertain curves $\mathcal{U}$ and $\mathcal{V}$ and a threshold $\delta \in \mathbb{R}^+$, decide if $d_{\mathrm{dF}}^{\max}(\mathcal{U}, \mathcal{V}) > \delta$.*

Suppose we are given a CNF-SAT formula C with n clauses, C_1 to C_n, on m boolean variables, x_1 to x_m. We pick some value $0.12 \le \varepsilon < 0.25$.[1] Construct a *variable* curve, where each variable corresponds to an indecisive point with locations $(0, 0.5 + \varepsilon)$ and $(0, -0.5 - \varepsilon)$; the locations are interpreted as assigning the variable True and False. Any realisation of the curve corresponds to a variable assignment. Each indecisive point is followed by a precise point that is far away, to force synchronisation with the other curve:

$$\mathrm{VG}_j = \{(0, 0.5 + \varepsilon), (0, -0.5 - \varepsilon)\} \parallel (2, 0)\,.$$

Consider a specific clause C_i of the formula. We define an *assignment gadget* $\mathrm{AG}_{i,j}$ for each variable x_j and clause C_i depending on how the variable occurs in the clause.

$$\mathrm{AG}_{i,j} = \begin{cases} (0, -0.5) \parallel (1, 0) & \text{if } x_j \text{ is a literal of } C_i, \\ (0, 0.5) \parallel (1, 0) & \text{if } \neg x_j \text{ is a literal of } C_i, \\ (0, 0) \parallel (1, 0) & \text{otherwise.} \end{cases}$$

Note that if assignment $x_j = \text{True}$ makes a clause C_i true, then the first precise point of the corresponding assignment gadget appears at distance $1 + \varepsilon$ from the realisation corresponding to setting $x_j = \text{True}$ of the indecisive point in VG_j. We can repeat the construction, yielding a *variable clause gadget* and an *assignment clause gadget:*

$$\mathrm{VCG} = (-2, 0) \parallel \mathop{\Big\|}_{j \in [m]} \mathrm{VG}_j\,, \qquad \mathrm{ACG}_i = (-1, 0) \parallel \mathop{\Big\|}_{j \in [m]} \mathrm{AG}_{i,j}\,.$$

Consider the Fréchet distance between the two gadgets. Observe that matching a synchronisation point from one gadget with a non-synchronisation point in the other yields a distance more than $1 + \varepsilon$, whereas matching synchronisation points pairwise and non-synchronisation points pairwise will yield the distance at most $1 + \varepsilon$. So we only consider one-to-one couplings, i.e. we match point i on one curve to point i on the other curve, for all i.

Now, if a realisation corresponds to a satisfying assignment, then for some x_j we have picked the realisation that is opposite from the coupled point on the clause curve, yielding the bottleneck distance of $1 + \varepsilon$. If the realisation corresponds to a non-satisfying assignment, then the synchronisation points establish the bottleneck, yielding the distance 1. So, we can clearly distinguish between a satisfying and a non-satisfying assignment for a clause.

[1] This range is determined by the relative distances in the construction.

Next, we define the *variable curve* and the *clause curve* as follows:

$$\mathrm{VC} = (0,0) \parallel \mathrm{VCG} \parallel (0,0)\,, \qquad \mathrm{CC} = \mathop{\Big\Vert}_{i\in[n]} \mathrm{ACG}_i\,.$$

Observe that the synchronisation points at $(-2,0)$ and $(-1,0)$ ensure that for any optimal coupling we match up VCG with some ACG_i as described before. Also note that all the points on CC are within distance 1 from $(0,0)$. Therefore, we can always pick any one of n clauses to align with VCG, and couple the remaining points to $(0,0)$; the bottleneck distance will then be determined by the distance between VCG and the chosen ACG_i.

Now consider a specific realisation of VCG. If the corresponding assignment does not satisfy C, then we can synchronise VCG with a clause that is false to obtain a distance of 1. If the assignment corresponding to the realisation satisfies all clauses, we must synchronise VCG with a satisfied clause, which yields a distance of $1+\varepsilon$. The construction is shown in Figures 3 and 4.

We can use similar reasoning to arrive at the same conclusion if we compute the Fréchet distance instead. The necessary adaptations are presented in the full version [13].

▶ **Theorem 2.** *The problems* Upper Bound Discrete Fréchet *and* Upper Bound Continuous Fréchet *for indecisive curves are NP-hard.*

2.1.2 Expected Fréchet Distance on Indecisive Curves

We show that finding expected discrete Fréchet distance is #P-hard by providing a polynomial-time reduction from #CNF-SAT, i.e. the problem of finding the number of satisfying assignments to a CNF-SAT formula. Missing details can be found in the full version [13]. Define the following problem and its continuous counterpart:

▶ **Problem 3.** Expected Discrete Fréchet: *Find* $d_{\mathrm{dF}}^{\mathbb{E}(\mathbb{U})}(\mathcal{U},\mathcal{V})$ *for uncertain curves* $\mathcal{U},\mathcal{V}$.

The main idea is to derive an expression for the number of satisfying assignments in terms of $d_{\mathrm{dF}}^{\mathbb{E}(\mathbb{U})}(\mathrm{VC},\mathrm{CC})$. This works, since there is a one-to-one correspondence between boolean variable assignment and a choice of realisation of VC, so counting the number of satisfying assignments corresponds to finding the proportion of realisations yielding large Fréchet distance. We can establish the result for Expected Continuous Fréchet similarly.

▶ **Theorem 4.** *The problems* Expected Discrete Fréchet *and* Expected Continuous Fréchet *for indecisive curves are #P-hard.*

2.1.3 Imprecise Curves

We have so far considered indecisive points; instead, we can look at imprecise points, namely, line segments or disks. We can show similar hardness results in that setting. We alter the construction – instead of the point $\{(0,0.5+\varepsilon),(0,-0.5-\varepsilon)\}$, we either have the disk centred at $(0,0)$ with radius $0.5+\varepsilon$ or the line segment connecting $(0,-0.5-\varepsilon)$ and $(0,0.5+\varepsilon)$. Observe that the locations of the indecisive point are still on the disk or the line segment. We can show that the upper bound decision problem is NP-hard by showing that we can always consider only the extreme locations on the imprecise points that coincide with the locations of the indecisive points.

▶ **Theorem 5.** *The problems* Upper Bound Discrete Fréchet *and* Upper Bound Continuous Fréchet *for imprecise curves modelled as line segments or disks are NP-hard.*

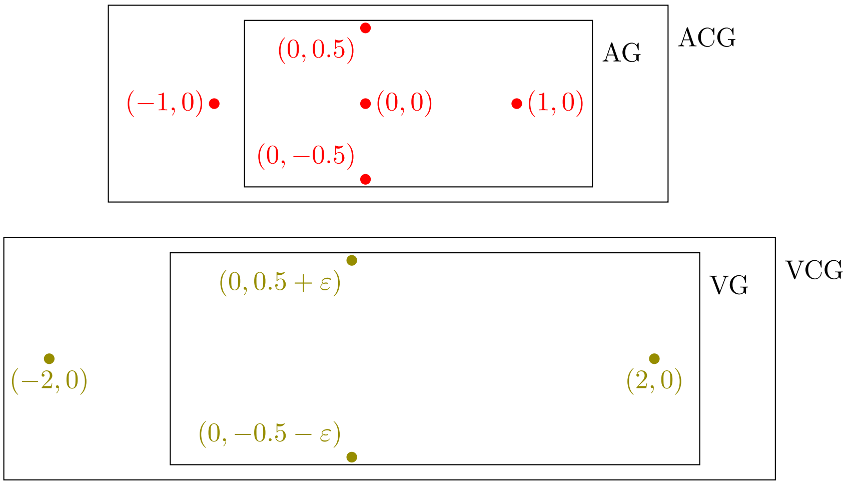


Figure 3 Illustration of gadgets used in the basic construction.

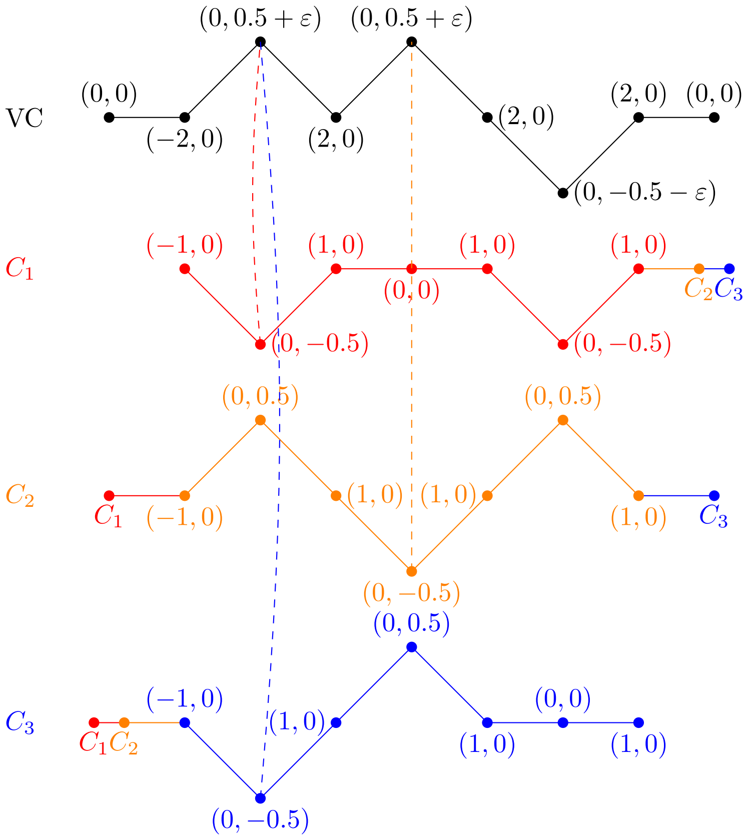


Figure 4 Realisation of VC for assignment $x_1 = \text{TRUE}$, $x_2 = \text{TRUE}$, $x_3 = \text{FALSE}$ and the CC for formula $C = (x_1 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2)$. Note that $C = \text{TRUE}$ with the given variable assignment. Also note that we can choose any of C_1, C_2, C_3 to align with VC; we always get the bottleneck distance of $1 + \varepsilon$, as all three are satisfied, so here $d_{\text{dF}}(\text{VC}, \text{CC}) = 1 + \varepsilon$.

We can also consider the value of expected Fréchet distance on imprecise points. We show the result only for points modelled as line segments; in principle, we believe that for disks a similar result holds, but the specifics of our reduction do not allow for clean computations.

We cannot immediately use our construction: we treat subsegments at the ends of the imprecision segments as TRUE and FALSE, but we have no interpretation for points in the centre part of a segment. So, we want to separate the realisations that pick any such invalid points. To that aim, we introduce extra gadgets to the clause curve that act as clauses, but catch these invalid realisations, so each of them yields the distance of 1. Now we have three distinct cases: realisation is satisfying, non-satisfying, or invalid. We can derive the expression connecting $d_{\mathrm{dF}}^{\mathbb{E}(\mathbb{U})}$ and the number of satisfying assignments.

▶ **Theorem 6.** *The problem* EXPECTED DISCRETE FRÉCHET *for imprecise curves modelled as line segments is #P-hard.*

2.2 Lower Bound Fréchet Distance

In this section, we prove that computing the lower bound Fréchet distance is NP-hard. The missing proofs can be found in the full version [13]. Unlike the upper bound proofs, this reduction uses the NP-hard problem SUBSET-SUM. Consider the following problems.

▶ **Problem 7.** LOWER BOUND CONTINUOUS FRÉCHET: *Given a polygonal curve π with n vertices, an uncertain curve $\mathcal{U}$ with m vertices, and a threshold $\delta > 0$, decide if $d_{\mathrm{F}}^{\min}(\pi, \mathcal{U}) \leq \delta$.*

▶ **Problem 8.** SUBSET-SUM: *Given a set $S = \{s_1, \ldots, s_n\}$ of n positive integers and a target integer τ, decide if there exists an index set I such that $\sum_{i \in I} s_i = \tau$.*

2.2.1 An Intermediate Problem

We start by reducing SUBSET-SUM to a more geometric intermediate curve-based problem.

▶ **Definition 9.** *Let $\alpha > 0$ be some value, and let $\sigma = \langle \sigma_1, \ldots, \sigma_{2n+1} \rangle$ be a polygonal curve. Call σ an* α-regular curve *if for all $1 \leq i \leq 2n+1$, the x-coordinate of σ_i is $i \cdot \alpha$. Let $Y = \{y_1, \ldots, y_n\}$ be a set of n positive integers. Call σ a* Y-respecting curve *if:*

1. *For all $1 \leq i \leq n$, σ passes through the point $((2i + 1/2)\alpha, 0)$.*
2. *For all $1 \leq i \leq n$, σ either passes through the point $((2i - 1/2)\alpha, 0)$ or $((2i - 1/2)\alpha, -y_i)$.*

Intuitively, the above definition requires σ to pass through $((2i + 1/2)\alpha, 0)$ as it reflects the y-coordinate about the line $y = 0$ (see Figure 5). Thus, if the curve also passes through $((2i - 1/2)\alpha, 0)$, the two reflections cancel each other. If it passes through $((2i - 1/2)\alpha, -y_i)$, the lemma below argues that y_i shows up in the final vertex height.

▶ **Lemma 10.** *Let σ be a Y-respecting α-regular curve, and let I be the subset of indices i such that σ passes through $((2i - 1/2)\alpha, -y_i)$. If $\sigma_1 = (\alpha, 0)$, then $\sigma_{2n+1} = ((2n+1)\alpha, 2\sum_{i \in I} y_i)$.*

The following is needed in the next section, and follows from the proof of the above.

▶ **Corollary 11.** *For a set $Y = \{y_1, \ldots, y_n\}$, let $M = \sum_{i=1}^{n} y_i$. For any vertex σ_i of a Y-respecting α-regular curve, its y-coordinate is at most $2M$ and at least $-2M$.*

▶ **Problem 12.** RR-CURVE: *Given a set $Y = \{y_1, \ldots, y_n\}$ of n positive integers, a value $\alpha = \alpha(Y) > 0$, and an integer τ, decide if there is a Y-respecting α-regular curve $\sigma = \langle \sigma_1, \ldots, \sigma_{2n+1} \rangle$ such that $\sigma_1 = (\alpha, 0)$ and $\sigma_{2n+1} = ((2n+1)\alpha, 2\tau)$.*

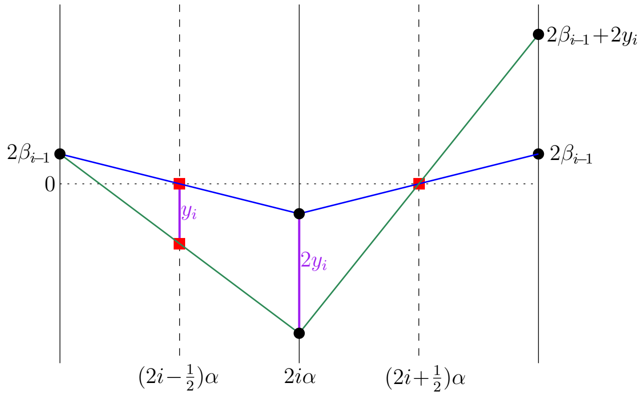


Figure 5 Passing through $((2i-1/2)\alpha,0)$ does not change the height, and passing through $((2i-1/2)\alpha,-y_i)$ adds $2y_i$.

By Lemma 10, Subset-Sum immediately reduces to the above problem by setting $Y=S$. Note that for this reduction it suffices to use any positive constant for α; however, we allow α to depend on Y, as this will ultimately be needed in our reduction to Problem 7.

▶ **Theorem 13.** *For any $\alpha(Y)>0$, RR-Curve is NP-hard.*

2.2.2 Reduction to Lower Bound Fréchet Distance

Let α, τ, $Y=\{y_1,\dots,y_n\}$ be an instance of RR-Curve. In this section, we show how to reduce it to an instance δ, π, $\mathcal{U}$ of Problem 7, where the uncertain regions in $\mathcal{U}$ are vertical line segments. The main idea is to use $\mathcal{U}$ to define an α-regular curve, and use π to enforce that it is Y-respecting. Specifically, let $M=\sum_{i=1}^{n} y_i$. Then $\mathcal{U}=\langle v_1,\dots,v_{2n+1}\rangle$, where each v_i is a vertical segment whose horizontal coordinate is $i\alpha$ and whose vertical extent is given by the interval $[-2M,2M]$. By Corollary 11, we have the following simple observation.

▶ **Observation 14.** *The set of all Y-respecting α-regular curves is a subset of* $\mathrm{Real}(\mathcal{U})$.

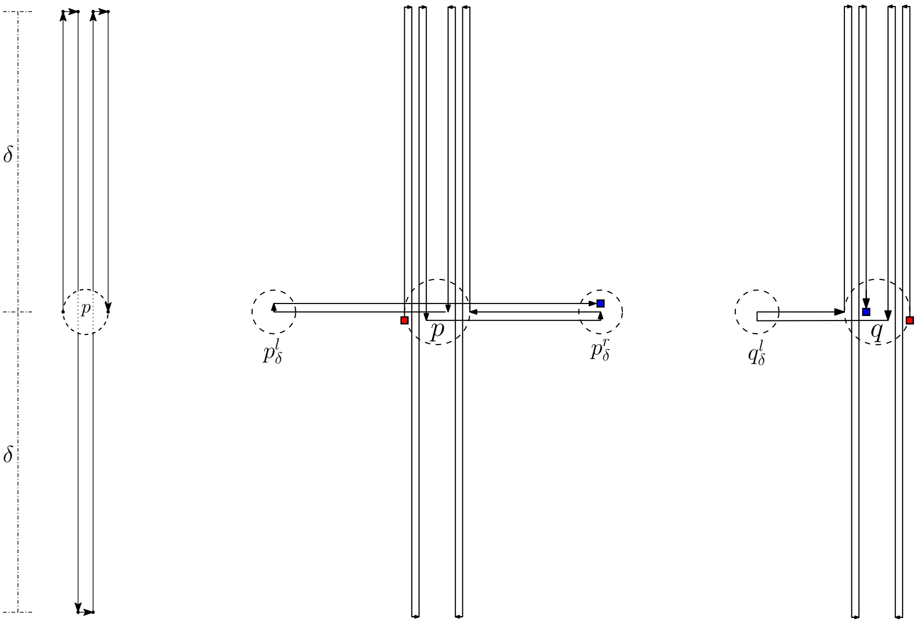
Thus, the main challenge is to define π to enforce that the realisation is Y-respecting. To that end, we first describe a gadget forcing the realisation to pass through a specified point.

▶ **Definition 15.** *For any point $p=(x,y)\in\mathbb{R}^2$ and value $\delta>0$, let the δ* gadget *at p, denoted $\mathrm{g}_\delta(p)$, be the curve: $(x,y)\parallel(x,y+\delta)\parallel(x,y-\delta)\parallel(x,y+\delta)\parallel(x,y)$. See Figure 6a.*

▶ **Lemma 16.** *Let $p=(x,y)\in\mathbb{R}^2$ be a point, and let ℓ be any line segment. Then if $d_\mathrm{F}(\ell,\mathrm{g}_\delta(p))\le\delta$, then ℓ must pass through p.*

For our uncertain curve to be Y-respecting, it must pass through all points of the form $((2i+1/2)\alpha,0)$. This condition is satisfied by the lemma above by placing a δ gadget at each such point. The second condition of a Y-respecting curve is that it passes through $((2i-1/2)\alpha,0)$ or $((2i-1/2)\alpha,-y_i)$. This condition is much harder to encode, and requires putting several δ gadgets together to create a composite gadget, which we now describe.

▶ **Definition 17.** *For any point $p=(x,y)\in\mathbb{R}^2$ and value $\delta>0$, let $p^l_\delta=(x-\delta/2,y)$ and $p^r_\delta=(x+\delta/2,y)$. Define the δ* lower composite gadget *at p, denoted $\mathrm{lcg}_\delta(p)$, to be the curve $\mathrm{g}_\delta(p)\parallel p^r_\delta\parallel\mathrm{g}_\delta(p)\parallel p^l_\delta\parallel p^r_\delta$. See Figure 6b. Define the δ* upper composite gadget *at q, denoted $\mathrm{ucg}_\delta(q)$, to be the curve $\mathrm{g}_\delta(q)\parallel q^l_\delta\parallel\mathrm{g}_\delta(q)$. See Figure 6c. Define the δ* composite gadget *of p and q, denoted $\mathrm{cg}_\delta(p,q)$, to be the curve $\mathrm{lcg}_\delta(p)\parallel\mathrm{ucg}_\delta(q)$.*



(a) $\mathrm{g}_\delta(p)$. **(b)** $\mathrm{lcg}_\delta(p)$. **(c)** $\mathrm{ucg}_\delta(q)$.

Figure 6 Depiction of gadgets $\mathrm{g}_\delta(p)$, $\mathrm{lcg}_\delta(p)$, and $\mathrm{ucg}_\delta(p)$. Circles represent zero-area points. For the right two figures, the red / blue square represents the starting / ending point.

To use this composite gadget, we centre the lower gadget at height $-y_i$ and the upper gadget directly above it at height zero. As the two gadgets are on top of each other, ultimately we require our uncertain curve to go back and forth once between consecutive vertical line segments, for which we have the following key property.

▶ **Lemma 18.** *Let $p = (x_p, -y_p)$ and $q = (x_p, 0)$ be points in $\mathbb{R}^2$. Let $\sigma = \langle a, b, c, d\rangle$ be a three-segment curve such that $b_x > x_p + \delta$ and $c_x < x_p - \delta$. If $d_\mathrm{F}(\sigma, \mathrm{cg}_\delta(p, q)) \le \delta$, then:*

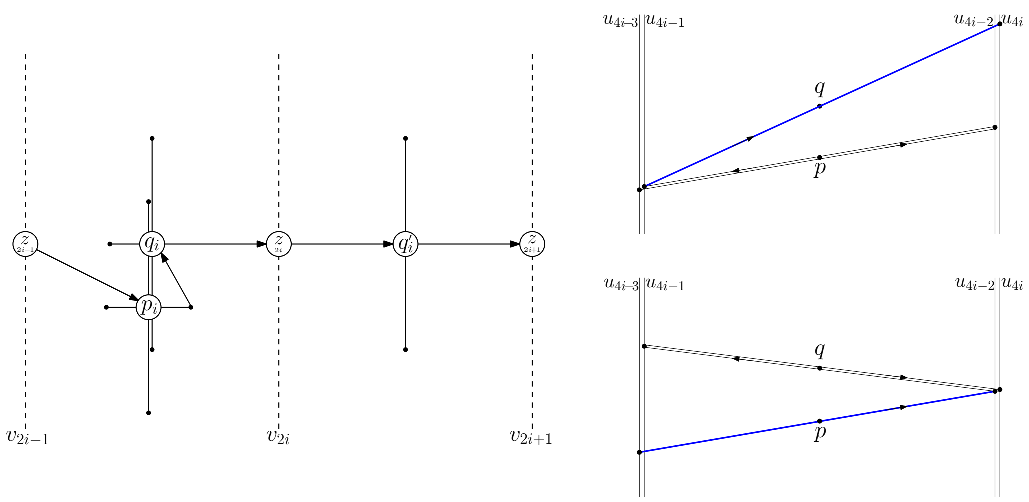
(i) *the segment ab must pass through p,*
(ii) *the segment cd must pass through q, and*
(iii) *the segment bc must either pass through p or through q.*

In particular, either ab and bc are on the same line, or cd and bc are on the same line.

Let v_l, v_r be vertical segments lying to the left and right of $\mathrm{cg}_\delta(p, q)$ further than δ away, and let z_l, z_r be the points on v_l and v_r at the same height as q. Consider the uncertain curve $\mathcal{U} = \langle U_1, U_2, U_3, U_4\rangle$, where $U_1 = U_3 = v_l$ and $U_2 = U_4 = v_r$. By Lemma 18, if there is a curve $\sigma \Subset \mathcal{U}$ such that $d_\mathrm{F}(\sigma, z_l \parallel \mathrm{cg}_\delta(p, q) \parallel z_r) \le \delta$, then implicitly it defines a single edge from v_l to v_r either passing through p or passing through q (see Figure 7b, whose notation is defined below). The following lemma acts as a rough converse of Lemma 18.

▶ **Lemma 19.** *Let $p = (x_p, -y_p)$ and $q = (x_p, 0)$ be points in $\mathbb{R}^2$, with $y_p \le \delta/4$. Let $\sigma = \langle p, b, c, q\rangle$ be a curve such that $x_p + \delta < b_x \le x_p + 1.1\delta$, $x_p - 1.1\delta \le c_x < x_p - \delta$, and $-\delta/2 \le b_y, c_y \le \delta/2$. If bc passes through either p or q, then $d_\mathrm{F}(\sigma, \mathrm{cg}_\delta(p, q)) \le \delta$.*

We now give the reduction from RR-Curve to Problem 7, whose correctness follows from the lemmas and discussion above. (See the full version [13] for details.) Let $\alpha(Y)$, τ, $Y = \{y_1, \dots, y_n\}$ be an instance of RR-Curve. For the reduction to Problem 7, we set $\delta = 4M$, where $M = \sum_{i=1}^n y_i$. Theorem 13 allows us to choose how to set $\alpha(Y)$, and we



(a) Pictorial representation of λ_i.

(b) The two solutions.

Figure 7 On the left, λ_i. On the right, the two possible solutions with Fréchet distance at most δ. The top (resp. bottom) corresponds to an α-regular curve passing through q (resp. p).

set $\alpha = 2.1\delta = 8.4M$. Let $V = \{v_1, \dots, v_{2n+1}\}$ be a set of vertical line segments where all upper (resp. lower) endpoints of the segments have height $2M$ (resp. $-2M$), and for all i, the x-coordinate of v_i is $i\alpha$. Let $\mathcal{U} = \langle U_1, \dots, U_{4n+1}\rangle$ be the uncertain curve such that $U_{4n+1} = v_{2n+1}$, and for all $1 \le i \le n$, $U_{4i-3} = v_{2i-1}$, $U_{4i-2} = v_{2i}$, $U_{4i-1} = v_{2i-1}$, and $U_{4i} = v_{2i}$. For $1 \le i \le 2n+1$, define the points $z_i = (i\alpha, 0)$, and for $1 \le i \le n$, define $q_i = ((2i - 1/2)\alpha, 0)$, $q'_i = ((2i + 1/2)\alpha, 0)$, and $p_i = ((2i - 1/2)\alpha, -y_i)$. For a given value $1 \le i \le n$, consider the curve $\lambda_i = z_{2i-1} \parallel \mathrm{cg}_\delta(p_i, q_i) \parallel z_{2i} \parallel \mathrm{g}_\delta(q'_i)$ (see Figure 7a). Let $s = (\alpha, 0)$ and $t = ((2n+1)\alpha, 2\tau)$. Then $\pi = \mathrm{g}_\delta(s) \parallel \lambda_1 \parallel \lambda_2 \parallel \cdots \parallel \lambda_{n-1} \parallel \lambda_n \parallel \mathrm{g}_\delta(t)$.

▶ **Theorem 20.** *Lower Bound Continuous Fréchet (Problem 7) is NP-hard, even when the uncertain regions are all equal-length vertical segments with the same height and the same horizontal distance (to the left or right) between adjacent uncertain regions.*

3 Algorithms for Lower Bound Fréchet Distance

In the previous section, we have shown that the decision problem for $d_F^{\min}$ is hard, given a polygonal curve and an uncertain curve with line-segment-based imprecision model. Interestingly, the same problem is solvable in polynomial time for indecisive curves. The key idea is that we can use a dynamic programming approach similar to that for computing Fréchet distance [6] and only keep track of realisations of the last indecisive point considered so far. (Note that one can also reduce the problem to Fréchet distance between paths in *DAG complexes,* studied by Har-Peled and Raichel [31], but this yields a slower running time.)

Consider the setting with an indecisive curve $\mathcal{V} = \langle V_1, \dots, V_n\rangle$ of n points and a precise curve $\pi = \langle p_1, \dots, p_m\rangle$ with m points; each indecisive point has k possible realisations, $V_i = \{q_i^1, \dots, q_i^k\}$. We can propagate reachability column by column. Define $\mathrm{Feas}(i, \ell)$ to be the *feasibility column* for realisation q_i^ℓ of U_i. This is a set of intervals on the vertical cell boundary line in the free-space diagram (see Figure 1), corresponding to the subintervals of one curve within distance δ from a point on the other curve. It is computed exactly the same way as for the precise Fréchet distance – it depends on the distance between a point and a line segment and gives a single interval on each vertical cell boundary.

Represent the standard dynamic program for computing Fréchet distance so that it operates column by column, grouping propagation of reachable intervals between vertically aligned cells. Call that procedure $\mathrm{Prop}(R)$, where R is the *reachability column* for point i and the result is the reachability column for point $i+1$ on one of the curves. The reachability column is a set of intervals on a vertical line, indicating the points in the free-space diagram that are reachable from the lower left corner with a monotone path.

Define $\mathrm{Reach}(i,s)$ to be the reachability column induced by q_i^s, where a point is in a reachability interval if it can be reached by a monotone path for some realisation of the previous points. Then we iterate over all the realisations of the previous column, getting precise cells, and propagate the reachable intervals as in the precise Fréchet distance algorithm:

$$\mathrm{Reach}(i+1,\ell) = \mathrm{Feas}(i+1,\ell) \cap \bigcup_{\ell' \in [k]} \mathrm{Prop}(\mathrm{Reach}(i,\ell'))\,.$$

For the column corresponding to U_1, we set one reachable interval of a single point at the bottom for all realisations p_1^s for which $\|q_1^s - p_1\| \le \delta$.

▶ **Theorem 21.** *Given an indecisive curve $\mathcal{V} = \langle V_1, \dots, V_n \rangle$ with k options per point, a precise curve $\pi = \langle p_1, \dots, p_m \rangle$, and a threshold $\delta > 0$, we can decide if $d_{\mathrm{F}}^{\min}(\pi, \mathcal{V}) \le \delta$ in time $\Theta(mnk^2)$ in the worst case, using $\Theta(mk)$ space. We can also report the realisation of $\mathcal{V}$ realising Fréchet distance at most δ, using $\Theta(mnk)$ space instead. Call the algorithm that solves the problem and reports a fitting realisation* $\textsc{Decider}(\delta, \pi, \mathcal{V})$.

We can extend this result to two indecisive curves. This result highlights a distinction between $d_{\mathrm{F}}^{\min}$ and $d_{\mathrm{F}}^{\max}$ and between different uncertainty models. To tackle $d_{\mathrm{F}}^{\min}$ with general uncertain curves, we develop approximation algorithms.

3.1 Approximation by Grids

Given a polygonal curve π and a general uncertain curve $\mathcal{U}$, in this section we show how to find a curve $\sigma \Subset \mathcal{U}$ such that $d_{\mathrm{F}}(\pi, \sigma) \le (1+\varepsilon) d_{\mathrm{F}}^{\min}(\pi, \mathcal{U})$. This is accomplished by carefully discretising the regions, in effect approximately reducing the problem to the indecisive case, for which we then can use Theorem 21. Missing proofs can be found in the full version [13].

For simplicity we assume the uncertain regions have constant complexity. Throughout, we assume $d_{\mathrm{F}}^{\min}(\pi, \mathcal{U}) > 0$, as justified by the following lemma.

▶ **Lemma 22.** *Let π be a polygonal curve with n vertices, and $\mathcal{U}$ an uncertain curve with m vertices. Then one can determine whether $d_{\mathrm{F}}^{\min}(\pi, \mathcal{U}) = 0$ in $O(mn)$ time.*

We call an algorithm a $(1+\varepsilon)$-*decider* for Problem 7, if when $d_{\mathrm{F}}^{\min}(\pi, \mathcal{U}) \le \delta$, it returns a curve $\sigma \Subset \mathcal{U}$ such that $d_{\mathrm{F}}(\pi, \sigma) \le (1+\varepsilon)\delta$, and when $d_{\mathrm{F}}^{\min}(\pi, \mathcal{U}) > (1+\varepsilon)\delta$, it returns $\textsc{False}$ (in between either answer is allowed). In this section, we present a $(1+\varepsilon)$-*decider* for Problem 7. We make use of the following standard observation.

▶ **Observation 23.** *Given a curve $\pi = \langle \pi_1, \dots, \pi_n \rangle$, call a curve $\sigma = \langle \sigma_1, \dots, \sigma_n \rangle$ an r-perturbation of π if $\|\pi_i - \sigma_i\| \le r$ for all i. Since $\|\pi_i - \sigma_i\|, \|\pi_{i+1} - \sigma_{i+1}\| \le r$, all points of the segment $\sigma_i\sigma_{i+1}$ are within distance r of $\pi_i\pi_{i+1}$. For segments this implies that $d_{\mathrm{F}}(\pi_i\pi_{i+1}, \sigma_i\sigma_{i+1}) \le r$, which implies that $d_{\mathrm{F}}(\pi, \sigma) \le r$ by composing the mappings for all i.*

The high-level idea is to replace $\mathcal{U}$ with the set of grid points it intersects, however, as our uncertain regions may avoid the grid points, we need to include a slightly larger set of points.

▶ **Definition 24.** *Let U be a compact subset of $\mathbb{R}^d$. We now define the set of points* $\mathrm{EG}_r(U)$ *which we call the* expanded r-grid points *of U.*

Let $B(\sqrt{d}r)$ denote the ball of radius $\sqrt{d}r$, centred at the origin. Let $\mathrm{Thick}(U, r) = U \oplus B(\sqrt{d}r)$*, where $\oplus$ denotes Minkowski sum. Let G_r be the regular grid of side length r, and let* $\mathrm{GT}_r(U)$ *be the subset of grid vertices from G_r that fall in* $\mathrm{Thick}(U, r)$*. Finally, define*

$$\mathrm{EG}_r(U) = \{p \mid p = \arg\min_{q \in U} \|q - x\| \textit{ for } x \in \mathrm{GT}_r(U)\} .$$

The following lemma argues that one can build a decider by using grids as hinted above. Using this decider, we can solve the corresponding optimisation problem.

▶ **Lemma 25.** *There is a $(1+\varepsilon)$-decider for Problem 7 with running time $O(mn \cdot (1 + (\Delta/(\varepsilon\delta))^{2d}))$, for $1 \geq \varepsilon > 0$, where $\Delta = \max_i \mathrm{diam}(U_i)$ is the maximum diameter of an uncertain region.*

▶ **Theorem 26.** *Let π be a polygonal curve with n vertices, $\mathcal{U}$ an uncertain curve with m vertices, and $\delta = d_{\mathrm{F}}^{\min}(\pi, \mathcal{U})$. Then for any $1 \geq \varepsilon > 0$, there is an algorithm which returns a curve $\sigma \Subset \mathcal{U}$ such that $d_{\mathrm{F}}(\pi, \sigma) \leq (1+\varepsilon)\delta$, whose running time is $O(mn(\log(mn) + (\Delta/(\varepsilon\delta))^{2d}))$, where $\Delta = \max_i \mathrm{diam}(U_i)$ is the maximum diameter of an uncertain region.*

If the polygonal curve π is replaced with an uncertain curve $\mathcal{W}$, it easy to argue that this approach extends to approximating $d_{\mathrm{F}}^{\min}(\mathcal{W}, \mathcal{U})$.

3.2 Greedy Algorithm

Here we argue that there is a simple 3-decider for Problem 7, running in near-linear time in the plane. The idea is to greedily and iteratively pick $\sigma_i \in U_i$ so as to allow us to get as far as possible along π. Without any assumptions on $\mathcal{U}$, this greedy procedure may walk too far ahead and get stuck. Thus, here we assume that consecutive U_i are separated to ensure that optimal solutions do not lag too far behind. Here we also assume the U_i are convex, i.e. imprecise, and have constant complexity, as it simplifies certain definitions. In this section, let $\pi = \langle \pi_1, \ldots, \pi_n \rangle$ be a polygonal curve and let $\mathcal{U} = \langle U_1, \ldots, U_m \rangle$ be an imprecise curve.

▶ **Definition 27.** *Call $\mathcal{U}$* γ-separated *if for all $1 \leq i < m$, $\|U_i - U_{i+1}\| > \gamma$ and each U_i is convex. Define an* r-visit *of U_i to be any maximal-length contiguous portion of $\pi \cap (U_i \oplus B(2r))$ which intersects $U_i \oplus B(r)$, where $\oplus$ denotes Minkowski sum. If $\mathcal{U}$ is γ-separated for $\gamma \geq 4r$, then any r-visit of U_i is disjoint from any r-visit of U_j for $i \neq j$, in which case define the* true r-visit *of U_i to be the first visit of U_i which occurs after the true r-visit of U_{i-1}. (For U_1 it is the first r-visit.)*

▶ **Lemma 28.** *If $\mathcal{U}$ is γ-separated for $\gamma \geq 4r$, then for any curve $\sigma \Subset \mathcal{U}$ and any reparametrisations f and g such that $\mathrm{width}_{f,g}(\pi, \sigma) \leq r$, σ_i must map to a point on the true r-visit of U_i for all i.*

For two points α and β on π, let $\alpha \leq \beta$ denote that α occurs before β, and for any points $\alpha \leq \beta$ let $\pi(\alpha, \beta)$ denote the subcurve between α and β.

▶ **Definition 29.** *The* δ-greedy sequence *of π with respect to $\mathcal{U}$, denoted* $\mathrm{gs}(\pi, \mathcal{U}, \delta)$*, is the longest possible sequence $\alpha = \langle \alpha_1, \ldots, \alpha_k \rangle$ of points on π, where $\alpha_1 = \pi_1$, and for any $i > 1$, α_i is the point furthest along π such that $\|\alpha_i - U_i\| \leq \delta$ and $d_{\mathrm{F}}(\alpha_{i-1}\alpha_i, \pi(\alpha_{i-1}, \alpha_i)) \leq 2\delta$.*

▶ **Observation 30.** *For any $i \leq k$, let $\alpha^i = \langle \alpha_1, \ldots, \alpha_i \rangle$ be the ith prefix of* $\mathrm{gs}(\pi, \mathcal{U}, \delta)$*. Then $d_{\mathrm{F}}(\alpha^i, \pi(\alpha_1, \alpha_i)) \leq 2\delta$, and $\alpha^i \Subset \mathcal{U}_i \oplus B(\delta)$, where $\mathcal{U}_i \oplus B(\delta) = \langle U_1 \oplus B(\delta), \ldots, U_i \oplus B(\delta) \rangle$.*

The following is the main lemma used to argue the correctness of our greedy approach, and it makes use of helper Lemma 28.

▶ **Lemma 31.** *If $\mathcal{U}$ is 10δ-separated and $d_{\mathrm{F}}^{\min}(\pi,\mathcal{U}) \leq \delta$, then $\mathrm{gs}(\pi,\mathcal{U},\delta)$ has length m and $\alpha_m = \pi_n$.*

The following lemma is the only place where we require the points to be in $\mathbb{R}^2$. The proof is interesting and uses a result from Guibas et al. [30].

▶ **Lemma 32.** *For π and $\mathcal{U}$ in $\mathbb{R}^2$, where $\mathcal{U}$ is 10δ-separated, $\mathrm{gs}(\pi,\mathcal{U},\delta)$ is computable in $O(m + n\log n)$ time.*

▶ **Theorem 33.** *Let $\mathcal{U}$ be $10r$-separated for some $r > 0$. There is a 3-decider for Problem 7 with running time $O(m + n\log n)$ in the plane that works for any query value $0 < \delta \leq r$.*

Proof. Compute $\mathrm{gs}(\pi,\mathcal{U},\delta)$. If it has length m, then let $\sigma = \langle \sigma_1, \dots, \sigma_m\rangle$ be any curve in $\mathrm{Real}(\mathcal{U})$ such that $\|\sigma_i - \alpha_i\| \leq \delta$ for all i. If this occurs and if $\alpha_m = \pi_n$, we output σ as our solution, and otherwise we output FALSE. Thus, the running time follows from Lemma 32.

Observe that if we output a curve σ, then $d_{\mathrm{F}}(\sigma,\pi) \leq 3\delta$, using the triangle inequality:

$$d_{\mathrm{F}}(\sigma,\pi) \leq d_{\mathrm{F}}(\sigma,\alpha) + d_{\mathrm{F}}(\alpha,\pi) \leq \delta + 2\delta = 3\delta\,.$$

Thus, we only need to argue that when $d_{\mathrm{F}}^{\min}(\pi,\mathcal{U}) \leq \delta$, a curve is produced, which is immediate from Lemma 31. ◀

4 Algorithms for Upper Bound and Expected Fréchet Distance

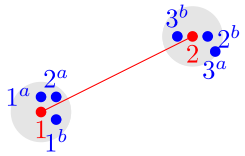
As shown in Section 2.1, finding the upper bound and expected discrete and continuous Fréchet distance is hard even for simple uncertainty models. However, restricting the possible couplings between the curves makes the problem solvable in polynomial time. In this section, we use *indecisive* curves. Define a Sakoe–Chiba time band [44] in terms of reparametrisations of the curves: for a band of width w and all $t \in [0,1]$, if $\phi_1(t) = x$, then $\phi_2(t) \in [x - w, x + w]$. In the discrete case we only couple point i on one curve to points $i \pm w$ on the other curve.

4.1 Upper Bound Discrete Fréchet Distance

First of all, let us discuss a simple setting. Suppose we are given a curve $\sigma = \langle q_1, \dots, q_n\rangle$ of n precise points and $\mathcal{U} = \langle U_1, \dots, U_n\rangle$ of n indecisive points, each of them having ℓ options, so for all $i \in [n]$ we have $U_i = \{p_i^1, \dots, p_i^\ell\}$. We would like to answer the following decision problem: *"If we restrict the couplings to a Sakoe–Chiba band of width w, is it true that $d_{\mathrm{dF}}^{\max}(\mathcal{U},\sigma) \leq \delta$ for some given threshold $\delta > 0$?"* So, we want to solve the decision problem for the upper bound discrete Fréchet distance between a precise and an indecisive curve.

In a fully precise setting the discrete Fréchet distance can be computed using dynamic programming [22]. We create a table where the rows correspond to vertices of one curve, say σ, and columns correspond to vertices of the other curve, say π. Each table entry (i,j) then contains a TRUE or FALSE value indicating if there is a coupling between $\sigma[1:j]$ and $\pi[1:i]$ with maximum distance at most δ. We use a similar approach.

Suppose we position $\mathcal{U}$ to go horizontally along the table, and σ to go vertically. Consider an arbitrary column in the table and suppose that we fix the realisation of $\mathcal{U}$ up to the previous column. Then we can simply consider the new column ℓ times, each time picking a different realisation for the new point on $\mathcal{U}$, and compute the resulting reachability. As we do this for the entire column at once, we can ensure consistency of our choice of realisation.



F F	F T	T T
T T	T F	F F

$$\begin{array}{|c|}\hline F\\\hline T\\\hline\end{array} \to \begin{array}{|c|c|}\hline F & T\\\hline T & F\\\hline\end{array} \to \begin{array}{|c|}\hline T\\\hline F\\\hline\end{array}$$

Figure 8 Left: An indecisive and a precise curve. Middle: Distance matrix. "T T" in the bottom left cell means $\|1 - 1^a\| \leq \delta$ and $\|1 - 1^b\| \leq \delta$. Right: Computing reachability matrix, column by column. Note two reachability vectors for the second column.

This procedure will give us a set of binary reachability vectors for the new column, each vector corresponding to a realisation. The *reachability vector* is a boolean vector that, for the cell (i, j) of the table, states whether for a particular realisation π of $\mathcal{U}[1 : i]$ the discrete Fréchet distance between π and $\sigma[1 : j]$ is below some threshold δ.

An important observation is that we do not need to distinguish between the realisations that give the same reachability vector: once we start filling out the next column, all we care about is the existence of some realisation leading to that particular reachability vector. So, we can keep a *set* of binary vectors corresponding to reachability in the column.

This procedure was suggested for a specific realisation. However, we can also repeat this for each previous reachability vector, only keeping the unique results. As all the realisation choices happen along $\mathcal{U}$, by treating the table column-by-column we ensure that we do not have issues with inconsistent choices. Therefore, repeating this procedure n times, we fill out the last column of the table. At that point, if any vector has False in the top right cell, then there is some realisation $\pi \Subset \mathcal{U}$ such that $d_{\mathrm{dF}}(\pi, \sigma) > \delta$, and hence $d_{\mathrm{dF}}^{\max}(\mathcal{U}, \sigma) > \delta$.

In more detail, we use two tables, distance matrix D and reachability matrix R. First of all, we initialise the distance matrix D and the reachability of the first column for all possible locations of U_1. Then we fill out R column-by-column. We take the reachability of the previous column and note that any cell can be reached either with the horizontal step or with the diagonal step. We need to consider various extensions of the curve $\mathcal{U}$ with one of the ℓ realisations of the current point: the distance matrix should allow the specific coupling. Assume we find that a certain cell is reachable; if allowed by the distance matrix, we can then go upwards, marking cells above the current cell reachable, even if they are not directly reachable with a horizontal or diagonal step. Then we remember the newly computed vector; we only add distinct vectors. The computation is illustrated in Figure 8; missing details can be found in the full version [13]. We use the following loop invariant to show correctness.

▶ **Lemma 34.** *Consider column i. Every reachability vector of this column corresponds to at least one realisation of $\mathcal{U}[1 : i]$ and the discrete Fréchet distance between that realisation and $\sigma[1 : \min(n, i + w)]$; and every realisation corresponds to some reachability vector.*

▶ **Theorem 35.** *Problem Upper Bound Discrete Fréchet restricted to a Sakoe–Chiba time band of width w on a precise curve and an uncertain curve on indecisive points with ℓ options, both of length n, can be solved in time $\Theta(4^w \ell n \sqrt{w})$ in the worst case.*

Now we extend our previous result to the setting where both curves are indecisive, so instead of σ we have $\mathcal{V} = \langle V_1, \ldots, V_n \rangle$, with, for each $j \in [n]$, $V_j = \{q_j^1, \ldots, q_j^\ell\}$. Suppose we pick a realisation for curve $\mathcal{V}$. Then we can apply the algorithm we just described. We cannot run it separately for every realisation; instead, note that the part of the realisation that matters for column i is the points from $i - w$ to $i + w$, since any previous or further points are outside the time band. So, we can fix these $2w + 1$ points and compute the column. We do so for each possible combination on these $2w + 1$ points.

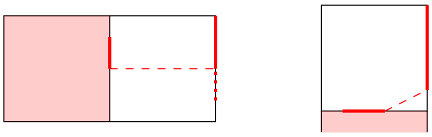


Figure 9 Reachability adjustments. Left: Although the dotted interval is free according to the distance matrix, only the solid interval is reachable from the cell on the left with a monotone path, assuming the cell on the left is free. Right: The full interval that is marked as free is reachable.

▶ **Theorem 36.** *Suppose we are given two indecisive curves of length n with ℓ options per indecisive point. Then we can compute the upper bound discrete Fréchet distance restricted to a Sakoe–Chiba band of width w in time $\Theta(4^w \ell^{2w+1} n\sqrt{w})$.*

4.2 Expected Discrete Fréchet Distance

To compute the expected discrete Fréchet distance with time bands, we need two observations:

1. For any two precise curves, there is a single threshold δ where the answer to the decision problem changes – a *critical value*; it is the distance between two points on the curves.
2. We can modify our algorithm to store associated counts with each reachability vector, obtaining the fraction of realisations that yield the answer TRUE for a given threshold δ.

We can execute our algorithm for each critical value and get the cumulative distribution function $\mathbb{P}(d_{\text{dF}}(\pi, \sigma) > \delta)$ for $\pi, \sigma \Subset_{\mathbb{U}} \mathcal{U}, \mathcal{V}$. Using the fact that the cumulative distribution function is a step function, we compute $d_{\text{dF}}^{\mathbb{E}}$.

▶ **Theorem 37.** *Suppose we are given two indecisive curves of length n with ℓ options per indecisive point. Then we can compute the expected discrete Fréchet distance when constrained to a Sakoe–Chiba band of width w in time $\Theta(4^w \ell^{2w+3} n^2 w^2)$ in the worst case.*

4.3 Continuous Fréchet Distance

We can adapt our time band algorithms to handle continuous Fréchet distance. Instead of the boolean reachability vectors, we use vectors of *free space* cells, introduced by Alt and Godau [6, 27]. We need to now store reachability intervals on cell borders (see Figure 9). The number of these intervals is limited: for any cell, the upper value of the interval is defined by the distance matrix, so yielding at most ℓ^2 values; the lower value of the interval is defined by the distance matrix or by one of the cells from the same row, yielding exponential dependency on w. However, the algorithm is still polynomial-time in n.

We can also store the associated counts. We then find critical values, in line with those arising in precise curve Fréchet distance [6]. This way we adapt our algorithm for computing expected distance to continuous case, and it runs in time polynomial in n for fixed w and ℓ, as desired. Further details are provided in the full version [13].

▶ **Theorem 38.** *Suppose we are given two indecisive curves of length n with ℓ options per indecisive point. Then we can compute upper bound Fréchet distance and expected Fréchet distance restricted to a Sakoe–Chiba band of fixed width w in time polynomial in n.*

References

1 Manuel Abellanas, Ferran Hurtado, Christian Icking, Rolf Klein, Elmar Langetepe, Lihong Ma, Belén Palop, and Vera Sacristán. Smallest color-spanning objects. In *Algorithms – ESA 2001*, volume 2161 of *Lecture Notes in Computer Science*, pages 278–289, Berlin, Germany, 2001. Springer Berlin Heidelberg. doi:10.1007/3-540-44676-1_23.

2 Pankaj K. Agarwal, Boris Aronov, Sariel Har-Peled, Jeff M. Phillips, Ke Yi, and Wuzhou Zhang. Nearest-neighbor searching under uncertainty II. *ACM Transactions on Algorithms (TALG)*, 13(1):3:1–3:25, December 2016. doi:10.1145/2955098.

3 Pankaj K. Agarwal, Rinat Ben Avraham, Haim Kaplan, and Micha Sharir. Computing the discrete Fréchet distance in subquadratic time. *SIAM Journal on Computing*, 43(2):429–449, 2014. doi:10.1137/130920526.

4 Pankaj K. Agarwal, Alon Efrat, Swaminathan Sankararaman, and Wuzhou Zhang. Nearest-neighbor searching under uncertainty I. *Discrete & Computational Geometry*, 58(3):705–745, July 2017. doi:10.1007/s00454-017-9903-x.

5 Hee-Kap Ahn, Christian Knauer, Marc Scherfenberg, Lena Schlipf, and Antoine Vigneron. Computing the discrete Fréchet distance with imprecise input. *International Journal of Computational Geometry & Applications*, 22(01):27–44, 2012. doi:10.1142/S0218195912600023.

6 Helmut Alt and Michael Godau. Computing the Fréchet distance between two polygonal curves. *International Journal of Computational Geometry and Applications*, 5(1):75–91, 1995. doi:10.1142/S0218195995000064.

7 Esther M. Arkin, Aritra Banik, Paz Carmi, Gui Citovsky, Matthew J. Katz, Joseph S.B. Mitchell, and Marina Simakov. Selecting and covering colored points. *Discrete Applied Mathematics*, 250:75–86, December 2018. doi:10.1016/j.dam.2018.05.011.

8 Donald J. Berndt and James Clifford. Using dynamic time warping to find patterns in time series. In *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*, pages 359–370, Palo Alto, CA, USA, 1994. AAAI Press. doi:10.5555/3000850.3000887.

9 Karl Bringmann. Why walking the dog takes time: Fréchet distance has no strongly subquadratic algorithms unless SETH fails. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 661–670, Piscataway, NJ, USA, August 2014. IEEE. doi:10.1109/FOCS.2014.76.

10 Karl Bringmann, Marvin Künnemann, and André Nusser. Fréchet distance under translation: Conditional hardness and an algorithm via offline dynamic grid reachability. In *Proceedings of the Thirtieth Annual ACM–SIAM Symposium on Discrete Algorithms (SODA 2019)*, pages 2902–2921. Society for Industrial and Applied Mathematics, January 2019. doi:10.5555/3310435.3310615.

11 Kevin Buchin, Maike Buchin, and Joachim Gudmundsson. Constrained free space diagrams: A tool for trajectory analysis. *International Journal of Geographical Information Science*, 24(7):1101–1125, July 2010. doi:10.1080/13658810903569598.

12 Kevin Buchin, Maike Buchin, Wouter Meulemans, and Wolfgang Mulzer. Four Soviets walk the dog: Improved bounds for computing the Fréchet distance. *Discrete & Computational Geometry*, 58(1):180–216, 2017. doi:10.1007/s00454-017-9878-7.

13 Kevin Buchin, Chenglin Fan, Maarten Löffler, Aleksandr Popov, Benjamin Raichel, and Marcel Roeloffzen. Fréchet distance for uncertain curves, April 2020. arXiv:2004.11862.

14 Kevin Buchin, Maarten Löffler, Pat Morin, and Wolfgang Mulzer. Preprocessing imprecise points for Delaunay triangulation: Simplified and extended. *Algorithmica*, 61(3):674–693, November 2011. doi:10.1007/s00453-010-9430-0.

15 Kevin Buchin, Tim Ophelders, and Bettina Speckmann. SETH says: Weak Fréchet distance is faster, but only if it is continuous and in one dimension. In *Proceedings of the Thirtieth Annual ACM–SIAM Symposium on Discrete Algorithms (SODA '19)*, pages 2887–2901. Society for Industrial and Applied Mathematics, January 2019. doi:10.5555/3310435.3310614.

16 Maike Buchin, Anne Driemel, and Bettina Speckmann. Computing the Fréchet distance with shortcuts is NP-hard. In *Proceedings of the Thirtieth Annual Symposium on Computational Geometry (SoCG 2014)*, pages 367–376, New York, NY, USA, June 2014. Association for Computing Machinery. doi:10.1145/2582112.2582144.

17 Sandip Das, Partha P. Goswami, and Subhas C. Nandy. Smallest color-spanning object revisited. *International Journal of Computational Geometry & Applications*, 19(5):457–478, October 2009. doi:10.1142/S0218195909003076.

18 Thomas Devogele, Laurent Etienne, Maxence Esnault, and Florian Lardy. Optimized discrete Fréchet distance between trajectories. In *Proc. 6th ACM SIGSPATIAL Workshop on Analytics for Big Geospatial Data*, pages 11–19, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3150919.3150924.

19 Anne Driemel and Sariel Har-Peled. Jaywalking your dog: Computing the Fréchet distance with shortcuts. *SIAM Journal on Computing*, 42(5):1830–1866, October 2018. doi:10.1137/120865112.

20 Anne Driemel, Sariel Har-Peled, and Carola Wenk. Approximating the Fréchet distance for realistic curves in near linear time. *Discrete & Computational Geometry*, 48(1):94–127, July 2012. doi:s00454-012-9402-z.

21 Anne Driemel, Herman Haverkort, Maarten Löffler, and Rodrigo I. Silveira. Flow computations on imprecise terrains. *Journal of Computational Geometry (JoCG)*, 4(1):38–78, 2013. doi:10.20382/jocg.v4i1a3.

22 Thomas Eiter and Heikki Mannila. Computing discrete Fréchet distance. Technical Report CD-TR 94/64, Technishe Universität Wien, April 1994. URL: http://www.kr.tuwien.ac.at/staff/eiter/et-archive/cdtr9464.pdf [cited 2019-04-23].

23 Chenglin Fan, Jun Luo, and Binhai Zhu. Tight approximation bounds for connectivity with a color-spanning set. In *Algorithms and Computation (ISAAC 2013)*, volume 8283 of *Lecture Notes in Computer Science*, pages 590–600, Berlin, Germany, 2013. Springer Berlin Heidelberg. doi:10.1007/978-3-642-45030-3_55.

24 Chenglin Fan and Benjamin Raichel. Computing the Fréchet gap distance. In *33rd International Symposium on Computational Geometry (SoCG 2017)*, volume 77 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 42:1–42:16, Dagstuhl, Germany, 2017. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SoCG.2017.42.

25 Chenglin Fan and Binhai Zhu. Complexity and algorithms for the discrete Fréchet distance upper bound with imprecise input, February 2018. arXiv:1509.02576v2.

26 Omrit Filtser and Matthew J. Katz. Algorithms for the discrete Fréchet distance under translation. In *16th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2018)*, volume 101 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 20:1–20:14, Dagstuhl, Germany, 2018. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SWAT.2018.20.

27 Michael Godau. A natural metric for curves: Computing the distance for polygonal chains and approximation algorithms. In *STACS 91: Proceedings of 8th Annual Symposium on Theoretical Aspects of Computer Science*, volume 480 of *Lecture Notes in Computer Science*, pages 127–136, Berlin, Germany, 1991. Springer Berlin Heidelberg. doi:10.1007/BFb0020793.

28 Chris Gray, Frank Kammer, Maarten Löffler, and Rodrigo I. Silveira. Removing local extrema from imprecise terrains. *Computational Geometry*, 45(7):334–349, 2012. doi:10.1016/j.comgeo.2012.02.002.

29 Joachim Gudmundsson, Majid Mirzanezhad, Ali Mohades, and Carola Wenk. Fast Fréchet distance between curves with long edges. *International Journal of Computational Geometry & Applications*, 29(2):161–187, 2019. doi:10.1142/S0218195919500043.

30 Leonidas J. Guibas, John E. Hershberger, Joseph S. B. Mitchell, and Jack S. Snoeyink. Approximating polygons and subdivisions with minimum-link paths. *International Journal of Computational Geometry & Applications*, 3(4):383–415, 1993. doi:10.1142/S0218195993000257.

31 Sariel Har-Peled and Benjamin Raichel. The Fréchet distance revisited and extended. *ACM Transactions on Algorithms (TALG)*, 10(1):3:1–3:22, January 2014. doi:10.1145/2532646.

32 Allan Jørgensen, Jeff Phillips, and Maarten Löffler. Geometric computations on indecisive points. In *Algorithms and Data Structures (WADS 2011)*, volume 6844 of *Lecture Notes in Computer Science*, pages 536–547, Berlin, Germany, 2011. Springer Berlin Heidelberg. doi:10.1007/978-3-642-22300-6_45.

33 Eamonn Keogh and Chotirat Ann Ratanamahatana. Exact indexing of dynamic time warping. *Knowledge and Information Systems*, 7(3):358–386, 2005. doi:10.1007/s10115-004-0154-9.

34 Christian Knauer, Maarten Löffler, Marc Scherfenberg, and Thomas Wolle. The directed Hausdorff distance between imprecise point sets. *Theoretical Computer Science*, 412(32):4173–4186, 2011. doi:10.1016/j.tcs.2011.01.039.

35 John Krumm. A survey of computational location privacy. *Personal and Ubiquitous Computing*, 13(6):391–399, August 2009. doi:10.1007/s00779-008-0212-5.

36 Maarten Löffler. *Data Imprecision in Computational Geometry.* PhD thesis, Universiteit Utrecht, October 2009. URL: https://dspace.library.uu.nl/bitstream/handle/1874/36022/loffler.pdf [cited 2019-06-15].

37 Maarten Löffler and Wolfgang Mulzer. Unions of onions: Preprocessing imprecise points for fast onion decomposition. *Journal of Computational Geometry (JoCG)*, 5(1):1–13, 2014. doi:10.20382/jocg.v5i1a1.

38 Maarten Löffler and Jack Snoeyink. Delaunay triangulations of imprecise points in linear time after preprocessing. *Computational Geometry: Theory and Applications*, 43(3):234–242, 2010. doi:10.1016/j.comgeo.2008.12.007.

39 Maarten Löffler and Marc van Kreveld. Largest and smallest tours and convex hulls for imprecise points. In *Algorithm Theory – SWAT 2006*, volume 4059 of *Lecture Notes in Computer Science*, pages 375–387, Berlin, Germany, 2006. Springer Berlin Heidelberg. doi:10.1007/11785293_35.

40 Anil Maheshwari, Jörg-Rüdiger Sack, Kaveh Shahbaz, and Hamid Zarrabi-Zadeh. Fréchet distance with speed limits. *Computational Geometry*, 44(2):110–120, 2011. doi:10.1016/j.comgeo.2010.09.008.

41 Jian Pei, Bin Jiang, Xuemin Lin, and Yidong Yuan. Probabilistic skylines on uncertain data. In *Proceedings of the 33rd International Conference on Very Large Data Bases*, pages 15–26. VLDB Endowment, September 2007. doi:10.5555/1325851.1325858.

42 Dieter Pfoser and Christian S. Jensen. Capturing the uncertainty of moving-object representations. In *Advances in Spatial Databases*, volume 1651 of *Lecture Notes in Computer Science*, pages 111–131, Berlin, Germany, June 1999. Springer Berlin Heidelberg. doi:10.1007/3-540-48482-5_9.

43 A. Prasad Sistla, Ouri Wolfson, Sam Chamberlain, and Son Dao. Querying the uncertain position of moving objects. In *Temporal Databases: Research and Practice*, volume 1399 of *Lecture Notes in Computer Science*, pages 310–337. Springer Berlin Heidelberg, Berlin, Germany, 1998. doi:10.1007/BFb0053708.

44 Hiroaki Sakoe and Seibi Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 26(1):43–49, February 1978. doi:10.1109/TASSP.1978.1163055.

45 Jeff Sember and William Evans. Guaranteed Voronoi diagrams of uncertain sites. In *Proceedings of the 20th Canadian Conference on Computational Geometry (CCCG 2008)*, pages 203–206, 2008. URL: http://cccg.ca/proceedings/2008/paper50full.pdf.

46 Subhash Suri, Kevin Verbeek, and Hakan Yıldız. On the most likely convex hull of uncertain points. In *Algorithms – ESA 2013*, volume 8125 of *Lecture Notes in Computer Science*, pages 791–802, Berlin, Germany, 2013. Springer Berlin Heidelberg. doi:10.1007/978-3-642-40450-4_67.

47 Marc van Kreveld, Maarten Löffler, and Joseph S. B. Mitchell. Preprocessing imprecise points and splitting triangulations. *SIAM Journal on Computing*, 39(7):2990–3000, May 2010. doi:10.1137/090753620.

48 Man Lung Yiu, Nikos Mamoulis, Xiangyuan Dai, Yufei Tao, and Michail Vaitis. Efficient evaluation of probabilistic advanced spatial queries on existentially uncertain data. *IEEE Transactions on Knowledge and Data Engineering*, 21(1):108–122, 2009. doi:10.1109/TKDE.2008.135.

Counting Homomorphisms in Plain Exponential Time

Andrei A. Bulatov
School of Computing Science, Simon Fraser University, Burnaby, Canada
https://www.cs.sfu.ca/~abulatov/
abulatov@sfu.ca

Amineh Dadsetan
School of Computing Science, Simon Fraser University, Burnaby, Canada
amineh.dadsetan@gmail.com

Abstract

In the counting Graph Homomorphism problem (#GraphHom) the question is: Given graphs G, H, find the number of homomorphisms from G to H. This problem is generally #P-complete, moreover, Cygan et al. proved that unless the Exponential Time Hypothesis fails there is no algorithm that solves this problem in time $O(|V(H)|^{o(|V(G)|)})$. This, however, does not rule out the possibility that faster algorithms exist for restricted problems of this kind. Wahlström proved that #GraphHom can be solved in plain exponential time, that is, in time $O((2k+1)^{|V(G)|+|V(H)|}\mathrm{poly}(|V(H)|, |V(G)|))$ provided H has clique width k. We generalize this result to a larger class of graphs, and also identify several other graph classes that admit a plain exponential algorithm for #GraphHom.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness

Keywords and phrases graph homomorphisms, plain exponential time, clique width

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.21

Category Track A: Algorithms, Complexity and Games

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

1 Introduction

The Exponential Time Hypothesis (ETH) [16] essentially suggests that the Satisfiability problem does not admit an algorithm that is significantly faster than the straightforward brute force algorithm. The ETH has been widely used to obtain (conditional) lower bounds on the complexity of various problems, see [18] for a fairly recent survey. It however does not rule out nontrivial algorithms for many other hard problems.

One of such problems is the Graph Homomorphism problem (GraphHOM for short). A homomorphism from a graph G to a graph H is a mapping $\varphi\colon V(G) \to V(H)$ such that for any edge $ab \in E(G)$ the pair $\varphi(a)\varphi(b)$ is an edge of H. GraphHOM asks, given graphs G and H, whether or not there exists a homomorphism from G to H [14]. In the counting version of this problem, denoted #GraphHOM, the goal is to find the number of homomorphisms from G to H. These two problems can be solved just by checking all possible mappings from a given graph G to a given graph H, which takes time $O^*(|V(H)|^{|V(G)|})$, where O^* denotes asymptotics up to a polynomial factor. Assuming the ETH Cygan et al. [6] proved that the general GraphHom and therefore #GraphHom cannot be solved in time $O(|V(H)|^{o(|V(G)|)})$. A similar bound for the more general Constraint Satisfaction Problem (CSP) was established in [22], and some related hardness results have also been obtained in [5].

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 21; pp. 21:1–21:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In spite of these results, there are several ways to restrict GraphHom which sometimes result in a problem admitting a faster algorithm. For graph classes $\mathcal{G}, \mathcal{H}$, GraphHom$(\mathcal{G}, \mathcal{H})$ denotes the problem GraphHom in which the input graphs G, H belong to $\mathcal{G}, \mathcal{H}$, respectively. #GraphHOM can be restricted in the same way. Both problems have received much attention in their own right and as a special case of the general CSP, and much is known about their computational complexity. We will use the symbol $-$ to indicate that an input graph is not restricted. In particular, it is known that GraphHom$(-, \mathcal{H})$ is solvable in polynomial time only if every graph in $\mathcal{H}$ contains a loop or is bipartite [15]. It is also known that #GraphHom$(-, \mathcal{H})$ is solvable in polynomial time only if every graph in $\mathcal{H}$ is complete with all the loops present or is complete bipartite [9]. In the remaining cases these problems are shown to be NP- and #P-complete, respectively. Similarly, it is known that GraphHom$(\mathcal{G}, -)$ [13] and #GraphHom$(\mathcal{G}, -)$ [8] are solvable in polynomial time if and only if the class of cores of the graphs from $\mathcal{G}$ in the former case, and the class $\mathcal{G}$ itself in the latter case have bounded tree width, respectively.

Here we focus on such restrictions that give rise to problems solvable still in exponential time but much faster than brute force. Specifically, GraphHom$(\mathcal{G}, \mathcal{H})$ or #GraphHom$(\mathcal{G}, \mathcal{H})$ are said to be solvable in *plain exponential* time if there is a solution algorithm running in time $O^*(c^{|V(G)|+|V(H)|})$, where c is a constant. In this paper we study problems of the form #GraphHom$(-, \mathcal{H})$, however, clearly, all the easiness results for #GraphHom$(-, \mathcal{H})$ also hold for GraphHom$(-, \mathcal{H})$. If the problem #GraphHom$(-, \mathcal{H})$ is solvable in plain exponential time, we call the class $\mathcal{H}$ a *plain exponential class*.

Plain exponential classes of graphs have received substantial attention in the literature. The most well known such class is $\mathcal{K}$, the class of all cliques. Note that #GraphHom$(-, \mathcal{K})$ is equivalent to the #Graph Colouring problem, in which the problem is, given a graph G and a number k, to find the number of k-colourings of G. A fairly straightforward dynamic programming algorithm solves this problem in time $O^*(3^{|V(G)|})$; we outline this algorithm in Example 8. A more sophisticated algorithm [17] solves #GraphHom$(-, \mathcal{K})$ in time $O^*(2^{|V(G)|})$. If $\mathcal{H}$ is a class of graphs of tree width k then #GraphHom$(-, \mathcal{H})$ is solvable in time $O^*((k+3)^{|V(G)|})$, see [11]. For the class $\mathcal{D}_c$ of graphs of degree at most c the problem #GraphHom$(-, \mathcal{D}_c)$ can be solved in time $O^*(c^{|V(G)|})$ by a minor modification of the brute force enumeration algorithm, see Example 7. Finally, Wahlström [23] obtained probably the most general result so far on plain exponential graph classes, proving that if $\mathcal{H}$ only contains graphs of clique width at most k then #GraphHom$(-, \mathcal{H})$ can be solved in time $O^*((2k+1)^{|V(G)|+|V(H)|})$. The algorithm from [23] is also dynamic programming and uses the representation of (labeled) graphs of bounded clique width through a sequence of operations such as disjoint union, connecting vertices with certain labels, and relabeling vertices. Such sequences are called *k-expressions*.

In this paper we aim at a systematic study of plain exponential classes of graphs. As the first step we further expand the class of graphs for which plain exponential counting algorithms are possible by adding one more operation to the construction of graphs of bounded clique width. In a nutshell, the new operation expands a graph H to a graph H' in such a way that H is a retract of H', and the preimages of vertices of H are connected in a regular way. The new class of graphs includes families of graphs of unbounded clique width, for instance, hypercubes, grids, cliques with subdivided edges, and therefore is strictly larger than the class of graphs of bounded clique width. By means of this new set of operations one can define a new graph "width" measure that we call *extended clique width*, only this new measure involves two parameters rather than one. Graphs of extended clique width (k, r) can also be represented by *extended (k, r)-expressions*. Let $\mathcal{X}_{k,r}$ denote the class of graphs whose extended width parameters are at most k, r, respectively. (In this case we will say that such a graph has extended clique width *at most* (k, r).) Although in most cases in this paper the only parameter that matters is $\max\{k, r\}$, we think that further stratification is useful for a number of more precise results.

We then show that given an arbitrary graph G, a graph H of extended clique width (k, r), and an extended (k, r)-expression Φ representing H, the number $\mathsf{hom}(G, H)$ of homomorphisms from G to H can be found in time $O^*((2\max(k, r) + 1)^{2|V(G)|})$. Similar to [23], the algorithm is dynamic programming and iteratively computes numbers $\mathsf{hom}(G', H')$, where G' is an induced subgraph of G and H' is a graph represented by a subexpression of Φ. Clearly, as one cannot assume that an extended (k, r)-expression representing H is known in advance, this algorithm alone does not guarantee that $\mathcal{X}_{k,r}$ is plain exponential. However, we also show that given a graph H of extended clique width at most (k, r), an extended (k, r)-expression representing H can be found in time $O^*((4\max(k, r) + 4)^{|V(H)|})$. Combined with the previous result we thus obtain the following

▶ **Theorem 1.** *For any fixed k, r the class of graphs of extended clique width at most (k, r) is plain exponential.*

Next, we show that the classes of graphs of bounded extended clique width are quite general. Let $\mathsf{Hypercubes}$ denote the class of all hypercubes and let Grids denote the class of all rectangular grids. Also, for a class $\mathcal{H}$ of graphs, $\mathcal{K}(\mathcal{H})$ denotes the class of graphs H obtained as follows. Take $H' \in \mathcal{H}$, a clique on vertices $\{v_1, \dots, v_n\}$, and for every edge v_iv_j of the clique, $i \neq j$, replace this edge with a copy of H', that is, connect v_i, v_j to all vertices of H' and include all the edges of H'. It is known that all three classes have unbounded clique width [20, 3], and $\mathcal{K}(\mathcal{H})$ has unbounded clique width even when $\mathcal{H}$ contains just one graph with one vertex.

▶ **Theorem 2.** $\mathsf{Hypercubes}$ *has extended clique width at most (2,1),* Grids *has extended clique width at most (6,1). For any class $\mathcal{H}$ of extended clique width (k, r), the class $\mathcal{K}(\mathcal{H})$ has extended clique width at most $(k + 5, \max(r, 1))$.*

By Theorem 1 this immediately implies that classes $\mathsf{Hypercubes}$ and Grids are plain exponential. For subdivisions of cliques we prove a stronger result.

▶ **Proposition 3.** *For any plain exponential class $\mathcal{H}$ of graphs (not necessarily of bounded extended clique width), the class $\mathcal{K}(\mathcal{H})$ is also plain exponential.*

It seems that there are two general reasons for a class of graphs to be plain exponential: to have bounded (extended) clique width or to have bounded degree. Classes $\mathsf{Hypercubes}$ and $\mathcal{K}(\mathcal{H})$ witness that bounded extended clique width (and in fact even bounded clique width) does not imply bounded degree. By proving that graphs from $\mathcal{X}_{k,r}$ satisfy certain nontrivial property and showing that a random c-regular graph for $c > 3$ (unsurprisingly) does not satisfy this property with high probability, we show that $\mathcal{D}_c$ does not have bounded extended clique width. The two types of classes can be combined together to obtain new plain exponential classes. Let G, H be graphs. The Cartesian product $G\Box H$ of G and H is defined to be the graph with vertex set $V(G) \times V(H)$ and edges $(u_1, v_1)(u_2, v_2)$ such that either $u_1 = u_2$ and $v_1v_2 \in E(H)$, or $v_1 = v_2$ and $u_1u_2 \in E(G)$.

▶ **Theorem 4.** *Let $\mathcal{G}$ be a plain exponential class of graphs and $\mathcal{H}$ of bounded degree. Then $\mathcal{G}\Box\mathcal{H} = \{G\Box H \mid G \in \mathcal{G}, H \in \mathcal{H}\}$ is plain exponential.*

Note that for another standard graph product, $G \times H$, where edges are given by the rule: $(u_1, v_1)(u_2, v_2)$ is an edge if and only if $u_1u_2 \in E(G)$ and $v_1v_2 \in E(H)$, a similar result is almost trivial, as we observe in Example 9.

There is no doubt that plain exponential classes are much more diverse than what is shown above. For instance, for a class $\mathcal{G}$ of graphs, let $\mathcal{G}^{+d}$ denote the class of graphs G such that it is possible to remove up to d vertices from G to obtain a graph from $\mathcal{G}$. Then as is easily seen, $\mathcal{G}^{+d}$ is plain exponential whenever $\mathcal{G}$ is plain exponential. Also, there are some odd plain exponential class of graphs (odd in the sense we could not fit it into any of the types above). Let Kneser_k denote the well studied class of Kneser graphs, see, e.g. [19]: Kneser_k is the class of graphs, whose vertices are the k-element subsets of a certain set, and two vertices are connected if and only if the corresponding subsets are disjoint. A plain exponential algorithm for Kneser_k (for a fixed k) exists, see [2, 21]. We find an alternative algorithm for this class of graphs.

▶ **Theorem 5.** *For every k the class Kneser_k is plain exponential.*

The class Kneser_k however does not fit in any of the more general classes of plain exponential graphs.

Due to space restrictions not all proofs are included in this paper. For the missing proofs the reader is referred to the full version of the paper [7].

2 Preliminaries: Homomorphisms and Clique width

2.1 Homomorphisms, plain exponential time

By $[n]$ we denote the set $\{1, \dots, n\}$ and by $[[n]]$ the set $\{0, 1, \dots, n\}$. As always we denote the vertex set of a graph G by $V(G)$, and its edge set by $E(G)$. A *homomorphism* of a graph G to a graph H is a mapping $\varphi\colon V(G) \to V(H)$ such that $\varphi(u)\varphi(v) \in E(H)$ for any $uv \in E(G)$. By $\mathsf{hom}(G, H)$ we denote the number of homomorphisms from G to H. The *Counting Graph Homomorphism* problem, #GraphHom, is defined as follows: given graphs G, H, find the number of homomorphisms from G to H. Its decision version – does there exist a homomorphism from G to H? – is denoted by GraphHom. For more on graph homomorphisms see [14]. If H is allowed only from a class $\mathcal{H}$ of graphs, the resulting counting and decision problems are denoted #GraphHom$(-, \mathcal{H})$ and GraphHom$(-, \mathcal{H})$, respectively.

We will be concerned with the complexity and the best running time of algorithms for #GraphHom$(-, \mathcal{H})$. In particular, we say that a class $\mathcal{H}$ of graphs is *plain exponential* if there is an algorithm that solves the problem #GraphHom$(-, \mathcal{H})$ in *plain exponential time*: there exists a constant c such that on input G, H, $H \in \mathcal{H}$, the algorithm runs in time $O^*(c^{|V(G)|+|V(H)|})$, where O^* means asymptotics up to a factor polynomial in $|V(G)|, |V(H)|$. Note that we will always assume that G and H are connected, since otherwise the existence or the number of homomorphisms from G to H can be deduced from those of their connected components.

▶ **Example 6.** (H-Colouring.) If $\mathcal{H}$ consists of just one graph, H, the problems #GraphHom$(-, \mathcal{H})$, GraphHom$(-, \mathcal{H})$ are known as #H-Colouring and H-Colouring, respectively. The #H-Colouring problem is solvable in polynomial time if H is a complete graph with all loops present, or is a complete bipartite graph [9]. The H-Colouring problem is solvable in polynomial time if H contains a loop or is bipartite [15]. Otherwise these problems are #P- and NP-complete, respectively. Since the brute force algorithm for this problems runs in $O(|V(H)|^{|V(G)|})$ time, #H-Colouring and H-Colouring are always solvable in plain exponential time. Also, by inspecting the solution algorithms from [9, 15] these results can be slightly generalized: #GraphHom$(-,\mathcal{H})$ is solvable in polynomial time whenever every graph from $\mathcal{H}$ is a complete graph with all loops, or a complete bipartite graph. Similarly GraphHom$(-,\mathcal{H})$ is polynomial time solvable if every graph from H contains a loop or is bipartite.

▶ **Example 7.** (Graphs of bounded degree.) As is mentioned in the introduction, if the degrees of graphs from $\mathcal{H}$ are bounded by a number c, the (improved) brute force algorithm solves #GraphHom$(-,\mathcal{H})$, GraphHom$(-,\mathcal{H})$ in time $O^*(c^{|V(G)|})$. Let G, H be input graphs, $H \in \mathcal{H}$. Recall that we assume G is connected; otherwise the procedure below has to be performed for each connected component, and the results multiplied. Order the vertices $v_1, \dots, v_n$ of G in such a way that each vertex except for the first one is adjacent to one of the preceding vertices. Then the brute force algorithm is organized as follows: Assign images to $v_1, \dots, v_n$ in turn. There are $|V(H)|$ possibilities to map v_1, but then if v_i is adjacent to v_j, $j < i$, the image of v_j is fixed, and therefore there are at most c possibilities for the image of v_i. Thus, the algorithm runs in $O^*(c^n)$. This approach also allows H to have a bounded number of vertices of high degree.

▶ **Example 8.** (Graphs of bounded clique width.) Let $\mathcal{C}_k$ denote the class of all graphs of clique width at most k (to be defined in Section 2.2). Then #GraphHom$(-,\mathcal{C}_k)$, GraphHom$(-,\mathcal{C}_k)$ can be solved in time $O^*((2k+1)^{|V(G)|+|V(H)|})$, implying that $\mathcal{C}_k$ is plain exponential [23].

Here we briefly describe the simple algorithm solving #GraphHom$(-,\mathcal{K})$, where $\mathcal{K}$ is the class of cliques. Given a graph G and a number s (or, equivalently, the clique K_s) the solution algorithm maintains an array $N(S,\ell)$ for $S \subseteq V$ and $\ell \le s$, which contains the number of homomorphisms from the subgraph of G induced by S to an ℓ-element clique. To compute each $N(S,\ell)$ we go over all subsets $S' \subseteq S$, consider the vertices from S' to be mapped to the ℓ-th vertex of the ℓ-clique. Then there are $N(S-S', \ell-1)$ ways to map the remaining vertices, and $N(S,\ell)$ is the sum of all numbers like this. It is not hard to see that the running time of this algorithm is $O^*(3^{|V(G)|})$. It can be improved to run in time $O^*(2^{|V(G)|})$ [17], and some further improvements are possible in certain cases [10].

▶ **Example 9.** Often plain exponential classes can be combined to obtain a new plain exponential class. For graphs G, H let $G \times H$ denote their product, the graph with vertex set $V(G) \times V(H)$ and edges $(u_1, v_2)(u_2, v_2)$ whenever $u_1u_2 \in E(G)$ and $v_1v_2 \in E(H)$. Also, for graph classes $\mathcal{G}, \mathcal{H}$, let $\mathcal{G} \times \mathcal{H}$ denote the class $\{G \times H \mid G \in \mathcal{G}, H \in \mathcal{H}\}$. If $\mathcal{G}, \mathcal{H}$ are plain exponential, then so is $\mathcal{G} \times \mathcal{H}$. Indeed, let π_1, π_2 denote the projection homomorphisms of $G \times H$ onto G and H, respectively; that is, $\pi_1 : (u,v) \mapsto u$ and $\pi_2 : (u,v) \mapsto v$. For any graph T a mapping $\varphi : V(T) \to V(G) \times V(H)$ is a homomorphism if and only if the mappings $\varphi_1 = \pi_1 \circ \varphi_1 : V(T) \to V(G)$ and $\varphi_2 = \pi_2 \circ \varphi : V(T) \to V(H)$ are homomorphisms. In this case $\varphi(u,v) = (\varphi_1(u), \varphi_2(v))$. This immediately implies that $\mathsf{hom}(T, G \times H) = \mathsf{hom}(T, G) \cdot \mathsf{hom}(T, \mathcal{H})$, and the result follows.

We will often deal with vertex labeled graphs. It will be convenient to represent labels on vertices of a graph G as a *label function* $\pi \colon V(G) \to [k]$, in which case we say that G is *k-labeled.* The graph $G = (V, E)$ equipped with a label function π will be denoted by $\mathbb{G} = (V, E, \pi)$. The k-labeled graph $\mathbb{G}$ is then called a *k-labeling* of G. Let $\mathbb{G}_1 = (V_1, E_1, \pi_1)$ and $\mathbb{G}_2 = (V_2, E_2, \pi_2)$ be k-labeled graph. A mapping $\varphi \colon V_1 \to V_2$ is a *homomorphism* of k-labeled graph $\mathbb{G}_1$ to k-labeled graph $\mathbb{G}_2$ if it is a homomorphism of graph $G_1 = (V_1, E_1)$ to $G_2 = (V_2, E_2)$ respecting the labeling, that is, $\pi_2(\varphi(v)) = \pi_1(v)$ for every $v \in V_1$.

The following notation will also be useful. Let again $\mathbb{G}_1, \mathbb{G}_2$ be k-labeled graphs, such that V_1, V_2 are disjoint. Then $\mathbb{G}_1 \bigoplus \mathbb{G}_2 = (V_1 \uplus V_2, E_1 \uplus E_2, \pi_1 \uplus \pi_2)$, where $\pi_1 \uplus \pi_2(v) = \pi_1(v)$, if $v \in V_1$, and $\pi_1 \uplus \pi_2(v) = \pi_2(v)$, if $v \in V_2$.

Finally, the subgraph of a graph $G = (V, E)$ induced by a set $S \subseteq V$ is denoted by $G[S]$. For a k-labeled graph $\mathbb{G} = (V, E, \pi)$, by $\mathbb{G}[S]$ we denote the k-labeled subgraph induced by $S \subseteq V$. Note that the label function of $\mathbb{G}[S]$ is $\pi|_S$, i.e., the restriction of π on the set S.

2.2 Clique width and k-expressions

The simplest way to introduce clique width of a graph is through k-expressions.

▶ **Definition 10.** *The following operators are defined on k-labeled graphs.*

- $\cdot_i$: *Construct a graph with one vertex, which is labeled $i \in [k]$.*
- $\rho_{i \to j}(\mathbb{G})$: *Relabel all vertices with label $i \in [k]$ of a k-labeled graph $\mathbb{G}$ to label $j \in [k]$.*
- $\eta_{ij}(\mathbb{G})$, *for $i \neq j$: Add an edge from every vertex labeled i to every vertex labeled j in $\mathbb{G}$, i.e. add edges uv for any vertices u, v where u has label i and v has label j.*
- $\mathbb{G}_1 \bigoplus \mathbb{G}_2$: *The disjoint union of k-labeled graphs $\mathbb{G}_1$ and $\mathbb{G}_2$.*

*A k-*expression *is any (properly formed) formula using the above operators.*

Every k-expression represents a k-labeled graph. We say that a graph $G = (V, E)$ is represented by k-expression Φ, if there exists a k-labeling π of the vertices of G such that Φ represents $\mathbb{G} = (V, E, \pi)$. A graph has clique width *k if k is minimal so that the graph is represented by a k-expression. The class of all graphs of clique width at most k is denoted by $\mathcal{C}_k$.*

Wahlström in [23] used k-expressions of graphs to show that $\mathcal{C}_k$ is plain exponential. However, k-expressions suitable for his plain exponential algorithm must satisfy an extra condition. Let Φ be a k-expression representing a k-labeled graph $\mathbb{G}$. Note that any subexpression of Φ represents a subgraph of $\mathbb{G}$. We say that k-expression Φ is *safe* if for every its subexpression $\Phi_1 \bigoplus \Phi_2$ such that Φ_1, Φ_2 represent graphs $\mathbb{G}_1, \mathbb{G}_2$, respectively, the graph $\mathbb{G}_i$ equals $\mathbb{G}[V(\mathbb{G}_i)]$ for $i = 1, 2$. In other words all edges of $\mathbb{G}$ between vertices of $\mathbb{G}_i$, $i = 1, 2$, are already edges of $\mathbb{G}_i$.

▶ **Lemma 11** ([23]).

(1) *Every graph of clique width k can be represented by a safe k-expression.*

(2) *A safe k-expression for a graph of clique width k can be found in plain exponential time.*

A class $\mathcal{G}$ of graphs has bounded clique width if $\mathcal{G} \subseteq \mathcal{C}_k$ for some k. Classes of bounded clique width include cliques, cographs, and distance-hereditary graphs [12, 4]. We will also be interested in nice graph classes that do not have bounded clique width. These include classes Hypercubes of hypercubes, Grids of rectangular grids, and subdivisions of cliques $\mathcal{K}(\mathcal{H})$ (introduced in Section 1) [20, 3].

3 Extended clique width

3.1 Extended k-expressions

In this section we introduce a more general version of k-expressions, and accordingly a more general version of clique width. New k-expressions require two more operators on k-labeled graphs. The first one does not have analogues in k-expressions. Let $\mathbb{G}$ be a k-labeled graph and r a positive integer parameter. The idea behind the *inflation* operator is the following. For each vertex v of $\mathbb{G}$ we add up to r new copies of v. The set of new copies of v depends only on the label of v, and is given by the vector $\overrightarrow{M}$ defined below. Let $v_{i_1}, \dots, v_{i_\ell}$ be the added copies of v, and v itself is considered as v_0. Next, some new edges are introduced: whether or not edge $v_i w_j$ is added depends only on whether $vw \in V(\mathcal{G})$, the labels of v, w, and the numbers i, j. These connections are given by the set $\mathcal{S}$ defined below. Finally, the new copies obtain labels, and the label of v_i only depends on the label of v in $\mathbb{G}$ and i. This step is governed by the vector $\overrightarrow{\sigma}$ in the definition below.

We now proceed to a formal definition. Fix natural k, r. By $\overrightarrow{M}$ we denote a vector $(M_1, \dots, M_k)$ where each M_i is a subset of $[[r]]$ containing 0. For such a vector $\overrightarrow{M}$, let

$$\mathcal{L}(\overrightarrow{M}) = \{(i_1, j_1, i_2, j_2) \mid i_1, i_2 \in [k], j_1 \in M_{i_1}, j_2 \in M_{i_2}\}.$$

▶ **Definition 12.** *Let $\overrightarrow{M} = (M_1, \dots, M_k)$, $\{0\} \subseteq M_i \subseteq [[r]]$ for $i \in [k]$, $\sigma_j : [k] \to [k]$ for $j \leq [[r]]$, where σ_0 is the identity mapping, and $\mathcal{S} \subseteq \mathcal{L}(\overrightarrow{M})$. Also, $\mathcal{S}$ is required to be a symmetric set, that is, if $(i_1, j_1, i_2, j_2) \in \mathcal{S}$ then $(i_2, j_2, i_1, j_1) \in \mathcal{S}$. Operator $\beta_{\overrightarrow{M}, \overrightarrow{\sigma}, \mathcal{S}}$ transforms a k-labeled graph $\mathbb{G} = (V, E, \pi)$ to a k-labeled graph $\mathbb{G}' = (V', E', \pi')$ as follows:*

- *$V' = \bigcup_{i=1}^{k} C_i$, where $C_i = \{a_j | j \in M_i,\ a \in V \text{ and } \pi(a) = i\}$. The vertices a_0, $a \in V$, are called original vertices of $\mathbb{G}' = \beta_{\overrightarrow{M}, \overrightarrow{\sigma}, \mathcal{S}}(\mathbb{G})$ and are identified with their corresponding vertices from V;*
- *$a_j b_{j'} \in E'$ if and only if $ab \in E$, and $(\pi(a), j, \pi(b), j') \in \mathcal{S}$ or $j = j' = 0$;*
- *$\pi'(a_j) = \sigma_j(\pi(a))$*

The second operator combines disjoint union with a sequence of adding edges operators.

▶ **Definition 13.** *Let $\mathcal{T} \subseteq [k] \times [k]$. Operator $\eta_{\mathcal{T}}$ takes two k-labeled graphs as input and produces a k-labeled graph as output. For k-labeled graphs $\mathbb{G}_1 = (V_1, E_1, \pi_1)$, and $\mathbb{G}_2 = (V_2, E_2, \pi_2)$, V_1, V_2 disjoint, the k-labeled graph $\eta_{\mathcal{T}}(\mathbb{G}_1, \mathbb{G}_2) = (V, E, \pi)$, is defined as follows:*

- *$V = V_1 \cup V_2$;*
- *$E = E_1 \cup E_2 \cup \{(a, b) \mid a \in V_1, b \in V_2, \pi_1(a) = i, \pi_2(b) = j,\ (i, j) \in \mathcal{T}\}$;*
- *$\pi(a) = \pi_1(a)$ if $a \in V_1$ and $\pi(a) = \pi_2(a)$ if $a \in V_2$.*

We refer to this operator as the connect operator.

An *extended* (k, r)*-expression* is a (properly formed) expression that involves operators $\cdot_i$ $(i \in [k])$, $\rho_{i \to j}$ $(i, j \in [k])$, $\beta_{\overrightarrow{M}, \overrightarrow{\sigma}, \mathcal{S}}$, and $\eta_{\mathcal{T}}$, where $\overrightarrow{M}, \overrightarrow{\sigma}, \mathcal{S}, \mathcal{T}$ are as in Definitions 12, 13. Similar to k-expressions, extended (k, r)-expressions represent k-labeled graphs, as well as usual graphs. For an example of extended (k, r)-expression see the construction of hypercubes in Section 3.2.

Note that if $\mathbb{G}_1$ and $\mathbb{G}_2$ are two isomorphic k-labeled graphs, and $\mathbb{G}_1$ is represented by an extended (k, r)-expression Φ, then Φ is an extended (k, r)-expression representing $\mathbb{G}_2$ as well.

A graph $G = (V, E)$ is said to have *extended clique width* (k, r) if the pair (k, r) is minimal such that there is a k-labeling π of G and an extended (k, r)-expression Φ that represents $\mathbb{G} = (V, E, \pi)$. If such a π exists we also say that Φ represents G. Note that an extended clique width of a graph is not unique, as pairs of numbers can be incomparable. However, for our purposes it will usually be enough to assume that $k = r$: just replace both parameters with $\max(k, r)$. The class of all graphs of extended clique width at most (k, r) is denoted by $\mathcal{X}_{k,r}$. A class $\mathcal{G}$ of graphs has bounded extended clique width if $\mathcal{G} \subseteq \mathcal{X}_{k,k}$ for some k.

The connect operator is clearly a substitute for the operator η_{ij} from Definition 10 of clique width. In particular, graphs of extended clique width $(k, 0)$ are very close to graphs of clique width k.

▶ **Proposition 14.** *Any graph G that can be represented by a k-expression, can also be represented by an extended $(k,0)$-expression. Therefore, $\mathcal{C}_k \subseteq \mathcal{X}_{k,0}$, that is, every graph that has clique width k also has extended clique width at most $(k,0)$.*

As is easily seen, the connect operator can be expressed through disjoint union and adding edges. However, we will need properties similar to the safety of k-expressions. Unfortunately, the inflation operator does not allow for an equally clean and easy definition of safety, as in the case of k-expressions, and we use the connect operator instead.

Let $\mathbb{G} = \eta_{\mathcal{T}}(\mathbb{G}_1, \mathbb{G}_2)$. It is straightforward from the definition that $\mathbb{G}[V(\mathbb{G}_1)]$ is equal to $\mathbb{G}_1$ and $\mathbb{G}[V(\mathbb{G}_2)]$ is equal to $\mathbb{G}_2$, that is, $\eta_{\mathcal{T}}$ does not add edges inside $\mathbb{G}_1, \mathbb{G}_2$. Also, if $\mathbb{G} = \beta_{\overrightarrow{M},\overrightarrow{\sigma},\mathcal{S}}(\mathbb{G}_1)$, then again $\mathbb{G}[V(\mathbb{G}_1)]$ is equal to $\mathbb{G}_1$. Similar to k-expressions we say that an extended (k,r)-expression Φ is *safe* if for each of its subexpressions $\eta_{\mathcal{T}}(\Phi_1, \Phi_2)$ and $\beta_{\overrightarrow{M},\overrightarrow{\sigma},\mathcal{S}}(\Phi_1)$ such that Φ_1, Φ_2 represent graphs $\mathbb{G}_1, \mathbb{G}_2$, respectively, it holds $\mathbb{G}_i = \mathbb{G}[V(\mathbb{G}_i)]$ for $i = 1, 2$. The following property is straightforward.

▶ **Lemma 15.** *Any extended (k,r)-expression is safe.*

An extended (k,r)-expression representing G (if one exists) can be found in plain exponential time.

▶ **Theorem 16.** *There is an algorithm running in time $O^*((4\max(k,r)+4)^{|V(G)|})$ that given a graph G outputs an extended (k,r)-expression for G if one exists, or reports "NO" otherwise.*

Proof (Sketch). One of the ingredients of our algorithm is the problem of deciding whether two k-labeled graphs are isomorphic. k-labeled graphs $\mathbb{G} = (V_1, E_1, \pi_1), \mathbb{H} = (V_2, E_2, \pi_2)$ are isomorphic if there exists an isomorphism φ from the graph $G = (V_1, E_1)$ to $H = (V_2, E_2)$ such that $\pi_1(a) = \pi_2(\varphi(a))$ for $a \in V_1$. We show that this problem can be reduced to the ordinary Graph Isomorphism problem and use the celebrated result by Babai [1] that there is an algorithm that, given graphs G and H, decides whether there exists an isomorphism between G and H in time $O(2^{\log(|V(G)|)^{O(1)}})$.

▶ **Lemma 17.** *There is a polynomial time reduction from the problem of deciding the isomorphism of k-labeled graphs to Graph Isomorphism.*

We now describe the main part of the algorithm. Create an array N of size $(k+1)^n$ whose entries $N(\mathbb{G}')$ are labeled with a k-labeling $\mathbb{G}'$ of a subgraph G' of G. For every entry $N(\mathbb{G}')$ the k-labeled graph $\mathbb{G}'$ either has an extended (k,r)-expression or it does not. The goal is to set the value of each entry $N(\mathbb{G}')$ to some extended (k,r)-expression for $\mathbb{G}'$ if it has one and to "no" otherwise. Then either for some labeling $\mathbb{G}$ of G the entry $N(\mathbb{G})$ contains a (k,r)-expression for G, or G does not have extended clique width at most (k,r).

Now we consider more detailed possibilities for each $\mathbb{G}'$. There are four cases. Case 1 takes place if $\mathbb{G}'$ has an extended (k,r)-expression that ends with an inflation operator; Case 2 takes place if $\mathbb{G}'$ has an extended (k,r)-expression that ends with a connect operator; Case 3 takes place if $\mathbb{G}'$ has an extended (k,r)-expression that ends with a sequence of relabeling operators; and, finally, Case 4 takes place if $\mathbb{G}'$ does not have an extended (k,r)-expression.

All one-element k-labeled graphs are obviously represented by an extended (k,r)-expression. Let us suppose the values of each entry $N(\mathbb{G}')$, where $\mathbb{G}'$ contains at most $n-1$ vertices is set correctly. Then, we want to set the correct values for entries of the array whose associated k-labeled graph has exactly n vertices. We use the dynamic programming approach that consists of two phases. In Phase 1, for each entry $N(\mathbb{G}')$ such that $\mathbb{G}'$ has n vertices, we check if $\mathbb{G}'$ satisfies the conditions of Case 1. Then for each k-labeled graph like this that does not satisfy the conditions of Case 1 we check if it falls in Case 2. In Phase 2, by relabeling $\mathbb{G}'$ for which $N(\mathbb{G}')$ is assigned a value, we find a new extended (k,r)-expression for $\mathbb{G}'$ that do not satisfy the conditions of Cases 1 and 2, but satisfy the conditions of Case 3. In the end, for each $\mathbb{G}'$ that belongs to none of Cases 1, 2, or 3, we set the value $N(\mathbb{G}')$ to "no" because it does not have an extended (k,r)-expression. In the rest of this proof, for a k-labeled graph $\mathbb{G}'$, we show how to check if it satisfies the conditions of each of Cases 1 and 2.

Let $\mathbb{G}' = (V', E', \pi')$ be a k-labeled graph with $|V'| = n$ and it has an extended (k,r)-expression that ends with an inflation operator. Then there is an induced subgraph $\mathbb{G}'_1$ of $\mathbb{G}'$ such that the result of application of an inflation operator to $\mathbb{G}'_1$ is isomorphic to $\mathbb{G}'$, and $\mathbb{G}'_1$ has an extended (k,r)-expression. Thus, there exist $\sigma_i : [k] \to [k]$, $i \in [[r]]$, $\overrightarrow{M}$, $M_i \subseteq [[r]]$, $i \in [k]$, $\mathcal{S} \subseteq \mathcal{L}(\overrightarrow{M})$, and a set $V'_1 \subset V'$, such that

(A) $\mathbb{G}'_2 = (V'_2, E'_2, \pi'_2) = \beta_{\overrightarrow{M},\overrightarrow{\sigma},\mathcal{S}}(\mathbb{G}'[V'_1])$ is isomorphic to $\mathbb{G}'$, and

(B) $\mathbb{G}'[V'_1]$ has an extended (k,r)-expression.

Conversely, if there exist $V'_1 \subset V'$, $\sigma_i : [k] \to [k]$, $i \in [[r]]$, $\overrightarrow{M}$, $M_i \subseteq [[r]]$, $i \in [k]$, $\mathcal{S} \subseteq \mathcal{L}(\overrightarrow{M})$ satisfying conditions (A),(B), then $\mathbb{G}'_2$ has an extended (k,r)-expression that ends with an inflation operator. As $\mathbb{G}'_2$ and $\mathbb{G}'$ are isomorphic, $\mathbb{G}'$ has an extended k-expression that ends with an inflation operator as well. Thus, the sufficient and necessary conditions for $\mathbb{G}'$ to have an extended (k,r)-expression that ends with an inflation operator, is that there exist $V'_1 \subset V'$, $\sigma_i : [k] \to [k]$, $i \in [[r]]$, $\overrightarrow{M}$, $M_i \subseteq [[r]]$, $i \in [k]$, $\mathcal{S} \subseteq \mathcal{L}(\overrightarrow{M})$ satisfying (A),(B).

The algorithm now searches through all possible selection of V'_1, $\overrightarrow{\sigma}$, $\mathcal{S}$, to check if conditions (A),(B) satisfied for any of them. Let us evaluate the running time of this procedure. Checking condition (A) takes time $O(2^{log((k+2)n)^{O(1)}})$ by Lemma 17, while condition (B) can be verified by looking up the existing entry $N(\mathbb{G}'[V'_1])$ in $O(1)$ time. There are 2^n choices for V'_1 and k^{rk} choices for $\overrightarrow{\sigma}$. Vector $\overrightarrow{M}$ can be chosen in 2^{rk} ways, and so $\mathcal{L}(\overrightarrow{M})$ has at most $2^{2rk}k^2$ elements. Thus, $\mathcal{S}$ can be chosen in at most $2^{2^{2rk}k^2}$ ways. Thus, the total running time of filling up $N(\mathbb{G}')$ in this case is upper bounded by

$$2^{|V(G)|} \times k^{rk} \times 2^{2rk}k^2 \times 2^{2^{2rk}k^2} \times O(2^{log((k+2)n)^{O(1)}}) = O^*(2^{2|V(G)|}).$$

Now let us suppose that $\mathbb{G}' = (V', E', \pi')$ has an extended (k,r)-expression that ends with a connect operator. Then due to the safety of extended (k,r)-expressions there exist two induced subgraphs $\mathbb{G}'_1$ and $\mathbb{G}'_2$ of $\mathbb{G}'$ such that, first, they both are represented by extended (k,r)-expressions, and, second, there is $\mathcal{T} \subseteq [k]^2$, such that $\eta_{\mathcal{T}}(\mathbb{G}'_1, \mathbb{G}'_2)$ is identical to $\mathbb{G}'$. Thus to find an extended (k,r)-expression for $\mathbb{G}'$ it suffices to go through all partitions of V' into sets V'_1 and V'_2 and for each partition check the following two conditions. First, check if $\mathbb{G}'_1 = \mathbb{G}'[V'_1]$ and $\mathbb{G}'_2 = \mathbb{G}'[V'_2]$ have an extended (k,r)-expression by looking up the entries $N(\mathbb{G}'_1), N(\mathbb{G}'_2)$. Second, check if there is $\mathcal{T} \subseteq [k]^2$ such that $\eta_{\mathcal{T}}(\mathbb{G}'_1, \mathbb{G}'_2)$ is identical to $\mathbb{G}$. Since there are at most $2^{|V(G)|}$ ways to partition V' into V'_1 and V'_2, takes time $O(2^{|V(G)|})$ to check if $\mathbb{G}'$ falls into Case 2.

So far we have registered an extended (k,r)-expression for every $\mathbb{G}'$ that satisfies the conditions of Case 1 or Case 2. Now, start Phase 2 and check whether any of the remaining k-labeled graphs $\mathbb{G}'$ satisfies the conditions of Case 3. In order to do that we go through all k-labeled graphs $\mathbb{G}'$ with n vertices and such that $N(\mathbb{G}')$ contains an extended (k,r)-expression Φ, that is initially for all $\mathbb{G}'$ that fall into Cases 1,2. Then we consider every possible relabeling ρ_{ij} in turn. If $\rho_{ij}(\mathbb{G}')$ is a k-labeled graph such that $N(\mathbb{G}')$ does not have an extended (k,r)-expression, then we set $N(\rho_{ij}(\mathbb{G}')) = \rho_{ij}(\Phi)$. We repeat this process for each k-labeled graph $\mathbb{G}'$, until no new entries can be filled. The time required for Phase 2 in total, for all $\mathbb{G}'$, not only those with n vertices is bounded by number of all k-labelings of all subgraphs of G times the number of possible operators ρ_{ij}. As is easily seen, the time required for Phase 2 in total is

$$(k+1)^{|V(G)|} \times k^2 = O^*((k+1)^{|V(G)|})$$

Time complexity: The array we construct has $(k+1)^{|V(G)|}$ entries. The time required to complete Phase 1 for all the entries is bounded by $O^*(4^{|V(G)|} \times (k+1)^{|V(G)|})$. The time to complete Phase 2 for all entries is bounded by $O^*((k+1)^{|V(G)|})$. Thus the total running time is $O^*((4k+4)^{|V(G)|})$. ◀

Next we explore what kind of graphs and k-labeled graphs can be represented by extended (k,r)-expressions.

3.2 Graph classes of bounded extended but not regular clique width

In this section we show that not all graphs of bounded extended clique width also have bounded clique width. Specifically, we consider the classes Hypercubes of hypercubes, Grids of rectangular grids, and $\mathcal{K}(\mathcal{H})$ of sudivisions of cliques by graphs from a class $\mathcal{H}$. All these classes have unbounded clique width, as mentioned in Section 2.2.

▶ **Theorem 18.**
(1) Hypercubes *has extended clique width at most (2,1).*
(2) Grids *has extended clique width at most (6,1).*
(3) *If $\mathcal{H}$ is a class of graphs of extended clique width (k,r), then $\mathcal{K}(\mathcal{H})$ has extended clique width at most $(k+5, \max(r,1))$.*

Proof. We present extended (2,1)-expressions for hypercubes and extended (6,1)-expressions for grids. Extended expressions for subdivide cliques are more involved, and the reader is referred to the full version of the paper [7].

(1) Let HC_n denote an n-dimensional hypercube. An extended (2,1)-expression Φ_n representing HC_n is constructed by induction on the dimensionality of the hypercube. The base cases of induction are HC_0 and HC_1. An extended (2,1)-expression for HC_0 is $\cdot_1$, and an extended (2,1)-expression for HC_1 is $\eta_{\{(1,2)\}}(\cdot_1, \cdot_2)$.

Suppose that for $m \le n$ the graph HC_m has an extended (2,1)-expression. Let Φ_n be an extended (2,1)-expression for HC_n. Let $\overrightarrow{M} = (\{0,1\},\{0,1\})$, let σ_0 be the identity mapping on [2], let $\sigma_1 : [2] \to [2]$ be given by $\sigma_1(1) = 2, \sigma_1(2) = 1$, and let $\mathcal{S} = \{(1,1,1,1),(2,1,2,1),(1,1,2,1),(2,1,1,1),(1,0,2,1),(2,1,1,0),(2,0,1,1),(1,1,2,0)\}$. Then it is not hard to see that $\beta_{\overrightarrow{M},\vec{\sigma},\mathcal{S}}\Phi_n$ is an extended (2,1)-expression for HC_{n+1}.

(2) Let us denote the vertex set of an $n \times m$-grid by $G_{n,m} = [n] \times [m]$. We proceed by induction on n, m. First, observe that a 2×2-grid labeled in an arbitrary way with 4 labels can be represented by a 4-expression in a straightforward manner. We choose the labeling π_{22} of $G_{2,2}$ given by $\pi_{22}(1,1) = 1, \pi_{22}(2,1) = 2, \pi_{22}(2,1) = 3, \pi_{22}(2,2) = 4$.

Next, we construct a 6-expression (not an extended one) for a $2 \times m$-grid labeled in a specific way. The labeling π'_{2m} of $G_{2,m}$ we achieve is given by $\pi'_{2m}(1,1) = \cdots = \pi'_{2m}(1,m-1) = 1$, $\pi'_{2m}(2,1) = \cdots = \pi'_{2m}(2,m-1) = 2$, $\pi'_{2m}(1,m) = 3$, $\pi'_{2m}(2,m) = 4$. Suppose we have constructed a 6-expression representing a $2 \times (m-1)$-grid labeled this way. Then add vertices $(1,m)$ and $(2,m)$ labeled 5 and 6, respectively, and apply operators $\eta_{53}, \eta_{56}, \eta_{64}$, and $\rho_{3\to1}, \rho_{4\to2}, \rho_{5\to3}, \eta_{6\to4}$. It is straightforward that the resulting labeled graph is a $2 \times m$-grid labeled in the required way.

Now starting with the labeled $2 \times m$-grid constructed in the previous step we show by induction that a $n \times m$-grid with labeling π_{nm} can be represented by an extended (4,1)-expression, where π_{nm} is given by $\pi_{nm}(i,j) = 3$ for $i \leq n-2$ and $j \in [m]$, $\pi_{nm}(n-1,j) = 1, \pi_{nm}(n,j) = 2$ for $j \in [m]$. The base case for induction, the grid $G_{2,m}$ labeled with π_{2m} can be obtained from the labeled grid constructed in the previous paragraph by applying operators $\rho_{3\to1}$ and $\rho_{4\to2}$. Suppose that an extended (4,1)-expression representing $G_{n-1,m}$ labeled with π_{n-1m} exists. For the induction step we consider inflation operator with the following parameters: $k = 4$, $r = 1$, $\overrightarrow{M} = (\{0,1\},\{0\},\{0\},\{0\})$, $\mathcal{S} = \{(1,1,2,0),(2,0,1,1),(1,1,1,1)\}$, σ_0 is the identity mapping on [4] and $\sigma_1(i) = i$, except $\sigma_1(1) = 4$. The operator $\beta_{\overrightarrow{M},\overrightarrow{\sigma},\mathcal{S}}$ applied to $G_{n-1,m}$ labeled with π_{n-1m} works as follows: it creates an extra copy of each vertex with label 1, that is, of $n-2$-nd row, and connects each new vertex a_1 to every vertex with label 2, a is connected to. In other words, if $a = (n-2,i)$, then a_1 plays the role of (n,i) and is properly connected to the only vertex with label 2 vertex $(n-2,i)$ is connected to, that is, $(n-1,i)$. Also, $\beta_{\overrightarrow{M},\sigma,\mathcal{S}}$ connects vertices $(n,i),(n,i+1)$. Finally, the vertices of the form (n,i) are assigned label 4. In order to obtain a grid labeled with π_{nm} it suffices to apply operators $\rho_{1\to3}$, $\rho_{2\to1}$ and $\rho_{4\to2}$. ◀

4 Counting homomorphism to labeled graphs given an extended k-expression

In this section we prove our main result.

▶ **Theorem 19.** *Let G and H be two graphs, and let k-labeled graph $\mathbb{H}$ be a k-labeling of graph H. Given an extended (k,r)-expression Φ for $\mathbb{H}$,* $\mathsf{hom}(G,H)$ *can be found in time $O^*((2(\max(k,r)+1))^{2|V(G)|})$*

The following notation and terminology will be used throughout this section. Let $\mathsf{HOM}(G,H)$ denote the set of all homomorphisms from G to H. Let $X \subseteq V(G)$, and let $\chi : X \to [k]$ be a label function. A mapping φ from X to k-labeled graph $\mathbb{H} = (V,E,\pi)$ is said to be consistent with χ if for every $x \in X$ it holds $\pi(\varphi(x)) = \chi(x)$. Let $\mathsf{hom}_\chi(G[X],\mathbb{H})$, $\mathsf{HOM}_\chi(G[X],\mathbb{H})$, $\mathsf{map}_\chi(G[X],\mathbb{H})$, and $\mathsf{MAP}_\chi(G[X],\mathbb{H})$, denote the number of homomorphisms from $G[X]$ to $\mathbb{H}$ consistent with χ, the set of all homomorphisms from $G[X]$ to $\mathbb{H}$ consistent with χ, the number of all mappings from $G[X]$ to $\mathbb{H}$ consistent with χ, and the set of all mappings from $G[X]$ to $\mathbb{H}$ consistent with χ, respectively.

Observing that an extended (k,r)-expression can be naturally viewed as an extended $(\max(k,r),\max(k,r))$-expression, in what follows we assume $k = r$. Let Φ be an extended (k,k)-expression for a k-labeling $\mathbb{H}$ of the graph H. We proceed by induction on the structure of Φ. More precisely, our algorithm will compute entries of an array $\mathsf{hom}(\mathbb{G}[X],\mathbb{H}')$, where $X \subseteq V(G)$ and $\mathbb{G}[X]$ is a k-labeling of $G[X]$, and $\mathbb{H}'$ is the k-labeled graph represented by a subexpression of Φ. Since the labeling of $\mathbb{H}'$ is important in this inductive process, we also cannot avoid labeling the graph $\mathbb{G}$. Operator $\cdot_i$ creating a graph $\mathbb{H}'$ with a single vertex labeled i gives the base case of induction. In this case $\mathsf{hom}(\mathbb{G}[X],\mathbb{H}') = 1$ if all vertices of X are labeled i and $\mathbb{G}[X]$ has no edges; otherwise $\mathsf{hom}(\mathbb{G}[X],\mathbb{H}') = 0$. Finally, after computing the numbers $\mathsf{hom}(\mathbb{G},\mathbb{H})$ for all the k-labelings $\mathbb{G}$ of G, we complete using the following observation.

▶ **Observation 20.** *Let G and H be graphs, and let k-labeled graph $\mathbb{H}$ be a k-labeling of H. Then*

$$\mathsf{hom}(G,H) = \sum_{\chi:V(G)\to[k]} \mathsf{hom}_\chi(G,\mathbb{H})$$

It therefore suffices to show how to compute $\mathsf{hom}(\mathbb{G}[X], \mathbb{H}')$, where $\mathbb{G}[X]$ is an arbitrary k-labeling of $G[X]$, $X \subseteq V(G)$, and $\mathbb{H}'$ is represented by a subexpression Φ' of Φ, provided $\mathsf{hom}(\mathbb{G}[Y], \mathbb{H}'')$ is known for all $Y \subset X$, all labelings $\mathbb{G}[Y]$ of $G[Y]$, and $\mathbb{H}''$ represented by a subexpression Φ'' of Φ' with $\Phi'' \neq \Phi'$. We consider 3 cases depending on the last operator of Φ'. In the cases of the relabeling and connect operators the argument is similar to that for k-expressions. Here we only consider the inflation operator.

4.1 Inflation operator

In this part, we show how to make a recursive step in the case when the last operator of Φ' is an inflation operator. Before explaining this step, we need several definitions.

A *retraction* is a homomorphism ψ from a graph G_2 to its subgraph G_1 such that $\psi(v) = v$ for each vertex v of G_1. In this case the subgraph G_1 is called a *retract* of G_2. A retraction from a k-labeled graph $\mathbb{G}_2 = (V_2, E_2, \pi_2)$ to a k-labeled graph $\mathbb{G}_1 = (V_1, E_1, \pi_1)$ is defined to be a retraction from $G_2 = (V_2, E_2)$ to $G_1 = (V_1, E_1)$ preserving the label function π_2, that is, $\pi_2(v) = \pi_1(\psi(v))$ for all $v \in V_2$.

It will be convenient for us to subdivide operator $\beta_{\overrightarrow{M},\overrightarrow{\sigma},\mathcal{S}}$ into two steps: the first one is expansion of the original graph using $\overrightarrow{M}$ and $\mathcal{S}$, and the second is relabeling of some vertices of the resulting graph using $\overrightarrow{\sigma}$. More specifically, let $\mathbb{H} = (V, E, \pi)$ be a k-labeled graph, $\overrightarrow{M}$, $M_i \subseteq [[k]]$ for $i \in [k]$ (recall that we assume $k = r$), $\mathcal{S} \subseteq \mathcal{L}(\overrightarrow{M})$, and $\sigma_i : [k] \to [k]$, $i \in [[k]]$. Then $\mathbb{H}' = (V', E', \pi') = \alpha_{\overrightarrow{M},\mathcal{S}}(\mathbb{H})$ is given by

- $V' = \bigcup_{i=1}^{k} C_i$, where $C_i = \{a_j | j \in M_i,\ a \in V \text{ and } \pi_1(a) = i\}$. The vertices a_0, $a \in V$, are called original vertices of $\mathbb{H}' = \alpha_{\overrightarrow{M},\mathcal{S}}(\mathbb{H})$ and are identified with their corresponding vertices from V;
- $(a_j, b_{j'}) \in E'$ if and only if $(a, b) \in E$, and $(\pi(a), j, \pi(b), j') \in \mathcal{S}$ or $j = j' = 0$;
- $\pi'(a_j) = \pi(a)$.

Then, $\mathbb{H}'' = (V'', E'', \pi'') = \beta_{\overrightarrow{M},\overrightarrow{\sigma},\mathcal{S}}(\mathbb{H})$, that is, $V'' = V'$, $E'' = E'$, and $\pi''(a_j) = \sigma_j(\pi(a))$ for $a \in V''$ and $j \in M_{\pi(a)}$.

As is easily seen, $\mathbb{H}$ is an induced subgraph of $\mathbb{H}'$, and a retract. Indeed, the mapping μ that maps every $a_j \in V(\mathbb{H}')$ to a (recall that a_j is a "copy" of some $a \in V(\mathbb{H})$) is a retraction.

The objective is to find a method to express the number of homomorphisms from induced subgraphs of G to $\mathbb{H}''$ given those from induced subgraphs of G to $\mathbb{H}$.

▶ **Lemma 21.** *Let $Y \subseteq V(G)$ and let γ be a function $Y \to [k]$. There is an algorithm that given $\mathsf{hom}_\zeta(G[X], \mathbb{H})$ for all functions ζ from a subset $X \subset Y$ to $[k]$ as input, finds $\mathsf{hom}_\gamma(G[Y], \mathbb{H}'')$ in time $O((2(k+1))^{|V(G)|})$.*

We break this down into two steps. The main result of Step I, which is summarized in Lemma 22, finds an equality for the number of homomorphisms from G to $\mathbb{H}'$. Then, the result for Step II analogous to Lemma 22 finds the number of homomorphisms from induced subgraphs of G to $\mathbb{H}''$ given those for G and $\mathbb{H}'$. As Step II is substantially simpler than Step I, we omit it here.

Step I

Let $\mathbb{H}' = (V', E', \pi') = \alpha_{\overrightarrow{M},\mathcal{S}}(\mathbb{H})$ and $\mathbb{H} = (V, E, \pi)$. Let Y be a subset of $V(G)$ and γ a function $Y \to [k]$. Also, set

$$\mathbb{W}(\gamma) = \{\omega |\ \omega : Y \to [[k]] \text{ and } \forall a \in Y,\ \omega(a) \in M_{\gamma(a)}\}.$$

For $\omega \in \mathbb{W}(\gamma)$, let

$$\mathsf{HOM}_\gamma(G[Y], \mathbb{H}', \omega) = \{\varphi \mid \ \varphi \in \mathsf{HOM}_\gamma(G[Y], \mathbb{H}') \text{ and } \forall a \in Y, \exists b \in V(\mathbb{H}) \ \text{ s. t. } \varphi(a) = b_{\omega(a)}\}.$$

For the rest of Step I, let X' and X'' be two disjoint subsets of $V(G)$ and let $\chi' : X' \to [k]$ and $\chi'' : X'' \to [k]$ be arbitrary functions. Also let $X = X' \uplus X''$ and let $\chi = \chi' \uplus \chi''$.

Let $\mathsf{HOM}_{\chi',\chi''}(G[X], \mathbb{H}')$ denote the set of all elements of $\mathsf{HOM}_\chi(G[X], \mathbb{H}')$ that map a vertex a from X to an original vertex of $\mathbb{H}'$ (recall that any vertex of $\mathbb{H}$ is called an original vertex of $\mathbb{H}'$) if and only if $a \in X'$.

For any $\varphi \in \mathsf{HOM}_{\chi',\chi''}(G[X], \mathbb{H}')$, there is a unique $\omega \in \mathbb{W}(\chi)$ such that φ is also an element of $\mathsf{HOM}_\chi(G[X], \mathbb{H}', \omega)$. Let us call ω the *consistent function* of φ. We then partition $\mathsf{HOM}_{\chi',\chi''}(G[X], \mathbb{H}')$ into smaller subsets and count the elements in each smaller subset. The partition splits $\mathsf{HOM}_{\chi',\chi''}(G[X], \mathbb{H}')$ into sets of homomorphisms that all share the same consistent function $\omega \in \mathbb{W}(\chi)$. As is easily seen, $\mathsf{HOM}_\chi(G[X], \mathbb{H}', \omega) \cap \mathsf{HOM}_{\chi',\chi''}(G[X], \mathbb{H}')$ is such a subset.

Let $\mathcal{B}(\chi', \chi'')$ be the set of all $\omega \in \mathbb{W}(\chi)$ such that ω satisfies the following properties:

(b.1) $\omega \in \mathbb{W}(\chi)$ and $\omega(x) = 0$ if and only if $x \in X'$.

(b.2) For every $a, b \in X$ such that at least one of them is not an element of X', and $ab \in E(G)$ it holds that $(\chi(a), \omega(a), \chi(b), \omega(b)) \in \mathcal{S}$.

Now, as the set of homomorphisms is subdivided into sufficiently small fragments, it is possible to show that the number of elements in $HOM_\chi(G[X], \mathbb{H}')$ such that ω is their consistent function is the same for any $\omega \in \mathcal{B}(\chi', \chi'')$ and it is zero otherwise.

▶ **Lemma 22.** *Let G, $\mathbb{H}$, $\mathbb{H}'$, X', X'', $X = X' \uplus X''$, χ', χ'', and $\chi = \chi' \uplus \chi''$ be defined as above, then*

$$|\mathsf{HOM}_{\chi',\chi''}(G[X], \mathbb{H}')| = |\mathcal{B}(\chi', \chi'')| \times \mathsf{hom}_\chi(G[X], \mathbb{H}).$$

To evaluate the running time of this procedure, note that the algorithm has to enumerate all possible partitions X', X'' of X, and all mappings that can be in $\mathbb{W}(\chi)$. Overall, it amounts to the number of mappings from X to a $k+1$ element set. The number of choices of X is $2^{|V(G)|}$. Thus the running time is bounded by $O((2(k+1))^{|V(G)|})$. Lemma 21 now follows from Lemma 22 and a similar result for Step II.

4.2 Putting pieces together

We are now in a position to prove Theorem 19.

Proof of Theorem 19. By Observation 20, $\mathsf{hom}(G, H)$ equals the sum of $\mathsf{hom}_\chi(G, \mathbb{H})$ over all k-label functions $\chi : V(G) \to [k]$. For each χ we need to compute $\mathsf{hom}_\chi(G, \mathbb{H})$. This computation is done through dynamic programming and requires finding all the numbers of the form $\mathsf{hom}_\chi(G[X], \mathbb{H}')$, where $X \subseteq V(G)$ and $\mathbb{H}'$ is a graph represented by a subexpression of Φ. By Lemma 21 and similar results for relabeling and connect operators computing each such value from the previous values takes $O((2k+1)^{|V(G)|})$ time. There are $k^{|V(G)|}$ label functions χ, and $2^{|V(G)|}$ subsets of $V(G)$. As the number of subexpressions of Φ introduces only a polynomial factor, the running time of the algorithm is $O^*((2(k+1)^{2|V(G)|})$. ◀

5 Beyond bounded extended clique width

In this section we study plain exponential classes that do not have bounded extended clique width. We start with showing that the class of all graphs with degrees less than a constant does not have bounded extended clique width, and how it can be combined with any plain exponential class to produce a new plain exponential class. Then we present two more plain exponential classes of graphs that so far not representable as derivatives of graph with bounded degree and/or bounded extended clique width.

5.1 Bounded degrees

To prove that some graph class does not have bounded extended clique width we first identify two nontrivial properties of graphs whose extended clique width is at most (k, k). Let G be a graph and $\mathcal{N}(v)$ denote the neighborhood of $v \in V(G)$. Also, let H be an induced subgraph of G, and $\mathcal{N}_H(v) = \mathcal{N}(v) \cap H$ for $v \in V(G)$.

▶ **Lemma 23.** *Let G be a connected graph and $|V(G)| = n$. If G has extended clique width at most (k, k) then the following two conditions hold:*

(1) *For any $\frac{k^2}{n} < \alpha \leq \frac{1}{2}$, there exists a subset $W \subseteq V(G)$ with $\frac{\alpha n}{k+1} \leq |W| \leq \alpha n$ such that there are at most 2^k subsets $U_1, \ldots, U_\ell$ of W with the following property: for every $v \in V(G) - W$ either $\mathcal{N}_H(v) = U_i$ for some $i \in [\ell]$, or $\mathcal{N}_H(v) = \mathcal{N}_H(w) \cap U_i$ for some $w \in V(H)$ and $i \in [\ell]$.*

(2) *If in addition the maximal degree of G is d, there is a constant $\delta(d, k)$ that only depends on d and k, such that for any β, $\delta(d, k) < \beta < \frac{1}{2}$, there are subsets $U \subseteq W \subseteq V$ such that $|W| \geq d|U|$, and $\frac{\beta n}{k+1} < |W| \leq \beta n$. Also, there is a partition Π of W into $|U|$ classes such that every vertex from $W - U$ only has neighbors in at most d blocks of Π.*

Then to prove that the class of all graphs whose degrees are bounded by a constant, does not have bounded extended clique width, we prove that a random d-regular graph does not satisfy the property from Lemma 23(2) with high probability, concluding that $\mathcal{D}_d$ does not have bounded extended clique width.

▶ **Lemma 24.** *Let $d > 3$. The probability that a random d-regular graph with n vertices satisfies the condition of Lemma 23(2) is $o(1)$.*

Classes of bounded degree can be combined with any plain exponential class to form another plain exponential class, as the following theorem shows. Let G_1 and G_2 be graphs. The *Cartesian product* of G_1 and G_2, denoted by $G_1 \square G_2$, is the graph whose vertex set is $V(G_1) \times V(G_2)$, and vertices (u_1, v_1) and (u_2, v_2) of $G_1 \square G_2$ are connected with an edge if and only if $u_1 = u_2$ and $v_1 v_2 \in E(G_2)$, or $v_1 = v_2$ and $u_1 u_2 \in E(G_1)$. For classes $\mathcal{G}, \mathcal{H}$ of graphs $\mathcal{G} \square \mathcal{H}$ denotes the class $\{G \square H \mid G \in \mathcal{G}, H \in \mathcal{H}\}$.

▶ **Theorem 25.** *If $\mathcal{D}$ is in plain exponential class of graphs and $\mathcal{B}$ has bounded degree, then $\mathcal{B} \square \mathcal{D}$ is also plain exponential.*

Proof. Let d be a bound on the degree of graphs from $\mathcal{B}$. Let $H = B \square D$, $B \in \mathcal{B}$, $D \in \mathcal{D}$, and let G be a graph. We are concerned with the number $\mathsf{hom}(G, H)$. Without loss of generality assume $V(B) = [r]$. Let P be a r-partition of $V(G)$. Then a mapping $h : V(G) \to V(H)$ is said to be consistent with P if for every $v \in V(G)$ such that $v \in P_i$ it holds $h(v) \in \{(i, e) | e \in V(D)\}$. In other words, h maps vertices of each set $P_i \in V(G)$ to vertices of the same copy of D in the Cartesian product.

Our algorithm will find a set $\mathcal{P}$ of r-partitions of $V(G)$ such that if a homomorphism $h : G \to H$ is consistent with an r-partition P, then $P \in \mathcal{P}$. Every r-partition can be viewed as a mapping from $V(G)$ to $V(B)$. Since B has degree at most d, partitions from $\mathcal{P}$ can be enumerated using a process similar to that in Example 7. Order the vertices $v_1, \dots, v_n$ of G in such a way that each vertex except for the first one is adjacent to one of the preceding vertices. Then a brute force algorithm is organized as follows: Assign images from $V(B)$ to $v_1, \dots, v_n$ in turn. Clearly, there are $V(B)$ possibilities to map v_1. Suppose that images from $V(B)$ are assigned to $v_1, \dots, v_{j-1}$ by a mapping $\pi : \{v_1, \dots, v_{j-1}\} \to V(B)$. We claim that there are just $d + 1$ possibilities to extend π on v_j if we want to keep the possibility that a homomorphism consistent with the obtained r-partition exists. By the choice of the order $v_1, \dots, v_n$, there is v_i, $i < j$, adjacent with v_j. It is possible to assing $\pi(v_j) = \pi(v_i)$. In this case a consistent homomorphism may map the edge v_iv_j to an edge of the form $(\pi(v_i), e_1)(\pi(v_i), e_2)$ for some $e_1e_2 \in E(D)$. Otherwise v_iv_j should be mapped to an edge of the form $(\pi(v_i), e)(\pi(v_j), e)$. In this case there are at most d possibilities for $\pi(v_j)$. Thus, the algorithm enumerates all the required r-partitions in time $O^*((d + 1)^n)$.

Now, let P be one of the r-partitions of $V(G)$ generated in the previous step. Let G'_P be a graph that is obtained by contracting every edge of $V(G)$ whose ends are in different blocks of P. We claim that $\mathsf{hom}(G'_P, D)$ is equal to the number of homomorphisms of G to $B\square D$ that are consistent with P.

Let $x \in V(G)$ and let $y \in V(G'_P)$. We use the notation $x \in y$, if y is the result of contraction of x with 0 or more other vertices of G. Also, the set of all homomorphism from G to $H = B\square D$ that are consistent with P is denoted by $HOM(P, G, H)$. We define a mapping φ from elements of $\mathsf{HOM}(G'_P, D)$ to elements of $\mathsf{HOM}(P, G, H)$ as follows: for every $h' \in \mathsf{HOM}(G'_P, D)$ set $\varphi(h') = h$, where h is given by $h(x) = (i, e)$, for $x \in P_i$ and $e = h'(y)$, $x \in y$.

We show that φ is bijective. First, we show that it is injective. If $h'_1, h'_2 \in \mathsf{HOM}(G'_P, D)$ are two different mappings, there is an element $y \in V(G'_P)$ such that $h'_1(y) \neq h'_2(y)$. Therefore, for every $x \in V(G)$ with $x \in y$, $\varphi(h'_1)(x) \neq \varphi(h'_2)(x)$.

Next we prove that φ is also surjective. We define a function $\varphi^{-1} : \mathsf{HOM}(P, G, H) \to \mathsf{MAP}(G'_P, E)$ such that $(\varphi^{-1} \circ \varphi)$ is the identity mapping. Then to complete the proof of surjectivity it only remains to show that the range of φ^{-1} is $\mathsf{HOM}(G'_P, D)$.

Note that for any homomorphism from G to H consistent with P, any $y \in V(G'_P)$, and any $w_1, w_2 \in y$, if $h(w_1) = (i, e)$, then $h(w_2) = (j, e)$ for some $j \in [r]$. We define φ^{-1} as follows. For $h \in \mathsf{HOM}(P, G, H)$ set $\varphi^{-1}(h) = h'$ such that h' is given by $h'(y) = e$, where y is such that for every $x \in y$, $h(x) = (j, e)$ for some $j \in [r]$.

It is straightforward that $(\varphi^{-1} \circ \varphi)$ is the identity function. Now observe that for any $y_1, y_2 \in V(G'_P)$ with $y_1y_2 \in E(G'_P)$ there are $x_1 \in y_1$ and $x_2 \in y_2$ such that $x_1x_2 \in E(G)$. Since y_1 and y_2 are not contracted in G'_P, x_1, x_2 are in the same block P_j of P for some $j \in [r]$. Therefore, $h(x_1) = (j, e_1)$, $h(x_2) = (j, e_2)$, and $e_1e_2 \in E(D)$. Hence $h'(y_1)h'(y_2) = e_1e_2$ is an edge of E. Thus, h' maps an edge of G to an edge of H, and it is a homomorphism. The surjectivity of φ follows.

Finally, since all the r-partitions of G for which there may exist a consistent homomorphism can be enumerated in plain exponential time, and $\mathsf{hom}(G'_P, D)$ can also be found in plain-exponential time for each such partition P, the overall algorithm runs in plain exponential time. ◀

5.2 Subdivided Cliques

Recall that the *subdivision* of an edge uv by a graph H is a graph with vertex set $V(H) \cup \{u, v\}$ and edge set $E(H) \cup \bigcup\limits_{t \in V(H)} \{ut, vt\}$. The subdivision of a graph G by a graph H is the graph obtained by replacing every edge uv of G with its subdivision by a copy of H (a disjoint copy for each edge). Let $\mathcal{K}(\mathcal{H})$ denote the class of subdivisions of cliques by graphs from a class $\mathcal{H}$.

The following theorem is the main result of this section.

▶ **Theorem 26.** *Let $\mathcal{H}$ be a plain exponential class of graphs. Then $\mathcal{K}(\mathcal{H})$ is also plain exponential.*

More precisely, if $\#\textsc{GraphHom}(-, \mathcal{H})$ can be solved in time $O^*(c^{|V(G)|+|V(H)|})$, c constant, for any given graphs G and $H \in \mathcal{H}$, then $\#\textsc{GraphHom}(-, \mathcal{K}(\mathcal{H}))$ can be solved in time $O^*(c_1^{2(|V(G)|+|V(H)|})$, where $c_1 = \max(c, 2)$.

Theorem 18(3) claims that if $\mathcal{H}$ is of bounded extended clique width, then so is $\mathcal{K}(\mathcal{H})$, and then that $\mathcal{K}(\mathcal{H})$ is plain exponential follows from Theorem 19. However, in Theorem 26 $\mathcal{H}$ does not have to be of bounded extended clique width.

5.3 Kneser Graphs

Kneser graphs give another example of a plain exponential class of graphs.

The *Kneser graph* $\mathrm{KG}_{n,k}$ is the graph whose vertex set is the set of k-element subsets of a set with n elements, and two vertices are adjacent if and only if the two corresponding sets are disjoint. By Kneser_k we denote the class of all Kneser graphs for a fixed k. The class Kneser_k is plain exponential, as it follows from the results of [2, 21]. Here we give an alternative algorithm for $\textsc{GraphHom}(-, \mathsf{Kneser}_k)$.

Let G be a graph, and $G^{(k)}$ denote the graph obtained by replacing each of its vertices with a clique of size k and replacing each of its edges with a complete bipartite graph on $k + k$ vertices. For $a \in V(G)$ let $\psi(a)$ denote the set of vertices of the clique replacing v in $G^{(k)}$.

First, we introduce a many to one correspondence between elements of $\mathsf{HOM}(G^{(k)}, K_n)$ and $\mathsf{HOM}(G, \mathrm{KG}_{n,k})$. Let $\tau : \mathsf{HOM}(G^{(k)}, K_n) \to \mathsf{HOM}(G, \mathrm{KG}_{n,k})$ be defined by setting $\tau(\varphi) : V(G) \to \mathrm{KG}_{n,k}$ to be the mapping $v \mapsto \{\varphi(u) | u \in \psi(v)\}$. Notice that the cardinality of $\{\varphi(u) | u \in \psi(v)\}$ equals k because $G^{(k)}[\psi(v)]$ is a k-clique and φ is a homomorphism from $G^{(k)}$ to K_n. Therefore $\tau(\varphi)(v)$ is always a vertex of $\mathrm{KG}_{n,k}$.

It can be shown that for $\varphi \in \mathsf{HOM}(G^{(k)}, K_n)$, $\tau(\varphi)$ is a homomorphism, and moreover for any element σ of $\mathsf{HOM}(G, \mathrm{KG}_{n,k})$, $\tau(\varphi) = \sigma$ for exactly $(k!)^{|V(G)|}$ homomorphisms $\varphi \in \mathsf{HOM}(G^{(k)}, K_n)$. Therefore

$$|\mathsf{HOM}(G, \mathrm{KG}_{n,k})| = \frac{\mathsf{HOM}(G^{(k)}, K_n)}{(k!)^{|V(G)|}}.$$

Since there is an algorithm that computes $\mathsf{hom}(G^{(k)}, K_n)$ in time $O^*(2^{k|V(G)|})$, there is an algorithm that computes $\mathsf{hom}(G, \mathrm{KG}_{n,k})$ in the same time.

▶ Remark 27. The running time of the algorithm above is not plain exponential if k is not a constant. Bonamy et al. [2] proved that the class $\mathsf{Kneser} = \bigcup_{k \in \mathbb{N}} \mathsf{Kneser}_k$ is not plain exponential unless the ETH fails.

Interestingly, the class of Kneser graphs does not have bounded extended clique width.

▶ **Theorem 28.** *The class* Kneser$_2$ *does not have bounded extended clique width.*

To prove this result we use the property of graphs with bounded extended clique width from Lemma 23 (1).

References

1 László Babai. Graph isomorphism in quasipolynomial time [extended abstract]. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 684–697, 2016.

2 Marthe Bonamy, Lukasz Kowalik, Michal Pilipczuk, Arkadiusz Socala, and Marcin Wrochna. Tight lower bounds for the complexity of multicoloring. *TOCT*, 11(3):13:1–13:19, 2019.

3 Flavia Bonomo, Luciano N. Grippo, Martin Milanic, and Martín Darío Safe. Graph classes with and without powers of bounded clique-width. *Discrete Applied Mathematics*, 199:3–15, 2016.

4 Andreas Brandstädt, Feodor F. Dragan, Hoàng-Oanh Le, and Raffaele Mosca. New graph classes of bounded clique-width. *Theory Comput. Syst.*, 38(5):623–645, 2005.

5 Jianer Chen, Xiuzhen Huang, Iyad A. Kanj, and Ge Xia. Strong computational lower bounds via parameterized complexity. *J. Comput. Syst. Sci.*, 72(8):1346–1367, 2006.

6 Marek Cygan, Fedor V. Fomin, Alexander Golovnev, Alexander S. Kulikov, Ivan Mihajlin, Jakub Pachocki, and Arkadiusz Socala. Tight bounds for graph homomorphism and subgraph isomorphism. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1643–1649, 2016.

7 Amineh Dadsetan and Andrei A. Bulatov. Counting homomorphisms in plain exponential time. *CoRR*, abs/1810.03087, 2018. arXiv:1810.03087.

8 Víctor Dalmau and Peter Jonsson. The complexity of counting homomorphisms seen from the other side. *Theor. Comput. Sci.*, 329(1-3):315–323, 2004.

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

10 Fedor V. Fomin, Serge Gaspers, and Saket Saurabh. Improved exact algorithms for counting 3- and 4-colorings. In *Computing and Combinatorics, 13th Annual International Conference, COCOON 2007, Banff, Canada, July 16-19, 2007, Proceedings*, pages 65–74, 2007.

11 Fedor V. Fomin, Pinar Heggernes, and Dieter Kratsch. Exact algorithms for graph homomorphisms. *Theory Comput. Syst.*, 41(2):381–393, 2007.

12 Martin Charles Golumbic and Udi Rotics. On the clique-width of some perfect graph classes. *Int. J. Found. Comput. Sci.*, 11(3):423–443, 2000.

13 Martin Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *J. ACM*, 54(1):1:1–1:24, 2007.

14 P. Hell and Nešetřil. *Graphs and homomorphisms*, volume 28 of *Oxford Lecture Series in Mathematics and its Applications*. Oxford University Press, 2004.

15 P. Hell and J. Nešetřil. On the complexity of H-coloring. *Journal of Combinatorial Theory, Ser.B*, 48:92–110, 1990.

16 Russell Impagliazzo and Ramamohan Paturi. Complexity of k-SAT. In *Proceedings of the 14th Annual IEEE Conference on Computational Complexity, Atlanta, Georgia, USA, May 4-6, 1999*, pages 237–240, 1999.

17 Mikko Koivisto. An $O^*(2^n)$ algorithm for graph coloring and other partitioning problems via inclusion–exclusion. In *47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), 21-24 October 2006, Berkeley, California, USA, Proceedings*, pages 583–590, 2006.

18 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Lower bounds based on the exponential time hypothesis. *Bulletin of the EATCS*, 105:41–72, 2011.

19 László Lovász. Kneser's conjecture, chromatic number, and homotopy. *J. Comb. Theory, Ser. A*, 25(3):319–324, 1978.

20 Johann A. Makowsky and Udi Rotics. On the clique-width of graphs with few P4's. *International Journal of Foundations of Computer Science*, 10(03):329–348, 1999.

21 Jesper Nederlof. Inclusion exclusion for hard problems. Master's thesis, Department of Information and Computer Science, Utrecht University, 2008. URL: `http://www.staff.science.uu.nl/~neder003/MScThesis.pdf`.

22 Patrick Traxler. The time complexity of constraint satisfaction. In *Parameterized and Exact Computation, Third International Workshop, IWPEC 2008, Victoria, Canada, May 14-16, 2008. Proceedings*, pages 190–201, 2008.

23 Magnus Wahlström. New plain-exponential time classes for graph homomorphism. In *Computer Science - Theory and Applications, Fourth International Computer Science Symposium in Russia, CSR 2009, Novosibirsk, Russia, August 18-23, 2009. Proceedings*, pages 346–355, 2009.

From Holant to Quantum Entanglement and Back

Jin-Yi Cai
Department of Computer Sciences, University of Wisconsin-Madison, Madison, WI, USA
jyc@cs.wisc.edu

Zhiguo Fu
School of Information Science and Technology and KLAS,
Northeast Normal University, Changchun, China
fuzg432@nenu.edu.cn

Shuai Shao
Department of Computer Sciences, University of Wisconsin-Madison, Madison, WI, USA
sh@cs.wisc.edu

Abstract

Holant problems are intimately connected with quantum theory as tensor networks. We first use techniques from Holant theory to derive new and improved results for quantum entanglement theory. We discover two particular entangled states $|\Psi_6\rangle$ of 6 qubits and $|\Psi_8\rangle$ of 8 qubits respectively, that have extraordinary closure properties in terms of the Bell property. Then we use entanglement properties of constraint functions to derive a new complexity dichotomy for all real-valued Holant problems containing a signature of odd arity. The signatures need not be symmetric, and no auxiliary signatures are assumed.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness

Keywords and phrases Holant problem, Quantum entanglement, SLOCC equivalence, Bell property

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.22

Category Track A: Algorithms, Complexity and Games

Related Version A full version of this paper is available at `https://arxiv.org/abs/2004.05706`.

Funding *Jin-Yi Cai*: Jin-Yi Cai is supported by NSF CCF-1714275.
Zhiguo Fu: Zhiguo Fu is supported by NSFC-61872076.
Shuai Shao: Shuai Shao is supported by NSF CCF-1714275.

Acknowledgements We want to thank anonymous reviewers for their helpful comments.

1 Introduction

1.1 Holant problems

Holant problems are a broad class of sum-of-products computations. It generalizes other frameworks such as counting constraint satisfaction problems (#CSP) and counting graph homomorphisms (#GH). Both have been well studied and full complexity dichotomies have been established [9, 26, 7, 13, 11, 25, 8, 31, 12]. On the other hand, the understanding of Holant problems, even restricted to the Boolean domain, is still limited. In this paper, we focus on Holant problems defined over the Boolean domain.

Holant problems are parameterized by a set of constraint functions, also called signatures. A signature (over the Boolean domain) of arity $n > 0$ is a map $\mathbb{Z}_2^n \to \mathbb{C}$. Let $\mathcal{F}$ be any fixed set of signatures. A signature grid $\Omega = (G, \pi)$ over $\mathcal{F}$ is a tuple, where $G = (V, E)$ is a graph without isolated vertices, π labels each $v \in V$ with a signature $f_v \in \mathcal{F}$ of arity $\deg(v)$, and labels the incident edges $E(v)$ at v with input variables of f_v. We consider all 0-1 edge assignments σ, and each gives an evaluation $\prod_{v\in V} f_v(\sigma|_{E(v)})$, where $\sigma|_{E(v)}$ denotes the restriction of σ to $E(v)$.

▶ **Definition 1** (Holant problems). *The input to the problem* $\operatorname{Holant}(\mathcal{F})$ *is a signature grid* $\Omega = (G, \pi)$ *over* $\mathcal{F}$*. The output is the partition function*

$$\operatorname{Holant}_\Omega = \sum_{\sigma: E(G)\to\{0,1\}} \prod_{v\in V(G)} f_v(\sigma|_{E_{(v)}}).$$

Bipartite Holant problems $\operatorname{Holant}(\mathcal{F} \mid \mathcal{G})$ *are Holant problems over bipartite graphs* $H = (U, V, E)$*, where each vertex in* U *or* V *is labeled by a signature in* $\mathcal{F}$ *or* $\mathcal{G}$ *respectively. We say* $\mathcal{F}$ *is on the left hand side* (LHS) *and* $\mathcal{G}$ *is on the right hand side* (RHS)*.*

Weighted #CSP is a special class of Holant problems. So are all weighted #GH. Other problems expressible as Holant problems include counting matchings and perfect matchings [43], counting weighted Eulerian orientations [38, 15], computing the partition functions of six-vertex models [41, 16] and eight-vertex models [4, 14], and a host of other vertex models from statistical physics [5]. It is proved that counting perfect matchings cannot be expressed by #GH [28, 17]. Thus, Holant problems are provably more expressive.

Progress has been made in the complexity classification of Holant problems. When all signatures are restricted to be symmetric, a full dichotomy is proved [18]. When asymmetric signatures are allowed, some dichotomies are proved for special families of Holant problems by assuming that certain auxiliary signatures are available, e.g., Holant^*, Holant^+ and Holant^c [20, 2, 22, 3]. Without assuming auxiliary signatures a Holant dichotomy is established only for non-negative real-valued signatures [37].

1.2 Quantum entanglement theory

Holant problems can be viewed as tensor networks in quantum theory. The partition function $\operatorname{Holant}_\Omega$ can be used in a (strong) simulation of quantum circuits [44]. A signature grid is just a tensor network, where each signature is a tensor with its inputs associated with its incident edges and the Holant value of the signature grid is obtained by contracting all edges. In this sense, a signature of arity n represents a state of n *qubits*. In quantum theory, the basic component of a system is a qubit. The (pure) state $|\Psi\rangle$ of n qubits is described by a vector in $\mathbb{C}^{2^n}$. (The standard notion requires quantum states to have norm 1, but in this paper, normalization by a nonzero scalar makes no difference for complexity, so we work with states having arbitrary nonzero norms.) A nonzero n-ary signature f is synonymous with an n-qubit state $|f\rangle = \sum_{x\in\{0,1\}^n} f(x)|x\rangle$. In this paper, we use them interchangeably. When f is a zero signature (i.e., $f \equiv 0$), we agree that $|f\rangle$ is a null state, denoted by $\mathfrak{N}$.

A core concept in quantum theory is *entanglement*. It is perhaps the most distinguishing characteristic feature separating quantum and classical physics.

▶ **Definition 2** (Quantum entanglement). *A state of n qubits (*$n > 1$*, representing a multiple system) is* entangled *if it cannot be decomposed as a tensor product of single-qubit states (individual systems). It is* genuinely entangled *if it cannot be decomposed as a tensor product of states of proper subsystems. It exhibits* multipartite entanglement *if it involves a genuinely entangled state of subsystem of more than two qubits (i.e., it cannot be decomposed as a tensor product of single-qubit states and 2-qubit states).*

Today, entanglement is recognized as an important resource in quantum computing and quantum information theory. It has been shown that quantum computing speedups essentially depend on unbounded entanglement [34]. While in quantum information theory, an entangled state is shared by several parties, one can perform operations on a subsystem locally without access to the other subsystems. This set-up is commonly used in quantum teleportation and quantum key distribution [27, 6]. For different information-theoretic tasks, different types of entanglement can be used [40]. The classification of them under *stochastic local operation with classical communication* (SLOCC) equivalence was proposed in 2000 by Dür et al. [24], and is an area of active research [46, 39, 36, 35, 1, 30]. Yet so far, even the classification of entangled 4-qubit states is not completely settled. For more about quantum entanglement theory, we refer to the survey [33].

1.3 Existing dichotomies inspired by entanglement theory

There are many natural connections between Holant problems and quantum theory. The introduction of Holant problems is inspired by holographic transformations [45]. Such a holographic transformation applied separately on each qubit i with a matrix A_i is just a SLOCC in quantum theory. Also, many known P-time computable signature sets for Holant problems can be clearly described in the quantum literature [19, 2] and they correspond directly to sets of states that are of independent interest in quantum theory [23, 32].

Going beyond that, Backens recently applied knowledge from the theory of quantum entanglement, directly to the study of Holant problems and derived new dichotomy results [2, 3]. We give a short description for these results in this subsection. We use $\langle\Phi|$ to denote the Hermitian adjoint (complex conjugate) of $|\Phi\rangle$, and $\langle\Phi|\Psi\rangle$ to denote the (complex) inner product of two n-qubit states.

▶ **Definition 3** (Projection). *The projection of the i-th qubit of an n-qubit ($n \geqslant 2$) state $|\Psi\rangle$ onto a single-qubit state $|\theta\rangle = a|0\rangle + b|1\rangle$ is defined as $\langle\theta|_i|\Psi\rangle = \bar{a}|\Psi_i^0\rangle + \bar{b}|\Psi_i^1\rangle$ where $\bar{a}$ and $\bar{b}$ are complex conjugates of a and b, and $|\Psi_i^0\rangle$ and $|\Psi_i^1\rangle$ are states of the remaining $n-1$ qubits when the i-th qubit of $|\Psi\rangle$ is set to 0 and 1 respectively.*

▶ **Theorem 4** ([42, 29]). *Let $|\Psi\rangle$ be a genuinely entangled n-qubit ($n \geqslant 3$) state. For any two qubits of $|\Psi\rangle$, there exist projections of the other $n-2$ qubits onto $n-2$ many single-qubit states that result in an entangled 2-qubit state.*

This result was presented to show that any pure entangled multipartite quantum state violates some Bell's inequality [42]. The original proof [42] was flawed and was corrected recently [29]. Theorem 4 shows that two particle entanglement can be realized via performing local projections on a genuinely entangled multiparticle state. It is observed in [2] that the theorem holds even when restricted to only local projections onto computational or Hadamard basis states, i.e., $|0\rangle$, $|1\rangle$, $|+\rangle = |0\rangle + |1\rangle$ and $|-\rangle = |0\rangle - |1\rangle$.

Based on Theorem 4 and the inductive entanglement classification under SLOCC equivalence [36, 35, 1], Backens showed that beyond entangled 2-qubit states, genuinely entangled 3-qubit states can be realized via local projections onto computational or Hadamard basis states (Theorem 12 in [2]). This theorem is equivalent to the following inductive statement.

▶ **Theorem 5** ([2]). *Let $|\Psi\rangle$ be an n-qubit ($n \geqslant 4$) state exhibiting multipartite entanglement. Then, there exists some i and some $|\theta\rangle \in \{|0\rangle, |1\rangle, |+\rangle, |-\rangle\}$ such that $\langle\theta|_i|\Psi\rangle$ exhibits multipartite entanglement.*

▶ Remark 6. This result shows that multipartite entanglement of an n-qubit ($n \geqslant 4$) state can be *preserved* under projections onto states $|0\rangle$, $|1\rangle$, $|+\rangle$ and $|-\rangle$.

The Holant$^+(\mathcal{F})$ problem is defined as Holant$(\mathcal{F} \cup \{|0\rangle, |1\rangle, |+\rangle, |-\rangle\})$, where single qubit states $|0\rangle, |1\rangle, |+\rangle$ and $|-\rangle$ represent unary signatures $\Delta_0 = (1, 0)$, $\Delta_1 = (0, 1)$, $\Delta_+ = (1, 1)$ and $\Delta_- = (1, -1)$ in the Holant framework. According to Theorem 5, we know that in the framework of Holant$^+$ problems, a genuinely entangled 3-qubit state can always be realized from an n-qubit ($n \geqslant 4$) state exhibiting multipartite entanglement. Then, using a genuinely entangled 3-qubit state, a full dichotomy was proved for Holant$^+$ problems [2]. Later, it was generalized to Holant^c problems [22, 3] where Holant$^c(\mathcal{F})$ is defined as Holant$(\mathcal{F} \cup \{|0\rangle, |1\rangle\})$.

1.4 Our results

In this paper, we consider when multipartite entanglement can be preserved under projections onto only computational basis states, i.e., $|0\rangle$ or $|1\rangle$. We have the following result.

▶ **Theorem 7.** *Let $|\Psi\rangle$ be an n-qubit ($n \geqslant 4$) state exhibiting multipartite entanglement and $\langle 0^n|\Psi\rangle \neq 0$. If $n \geqslant 5$ and $|\Psi\rangle$ is not of the form $a|0^n\rangle + b|1^n\rangle$, or $n = 4$ and $|\Psi\rangle$ is not of the form $a|0000\rangle + b|1111\rangle + c|0011\rangle + d|1100\rangle$ (up to a permutation of the four qubits) where a, b, c and d can possibly be zero, then there exists some i such that $|\Psi_i^0\rangle$ or $|\Psi_i^1\rangle$ exhibits multipartite entanglement.*

Under SLOCC equivalence, without loss of generality, we may assume that $\langle 0^n|\Psi\rangle \neq 0$. The other conditions are all necessary to ensure the preservation of multipartite entanglement under projections to $|0\rangle$ and $|1\rangle$. Thus Theorem 7 is a strengthening of Theorem 5. More importantly, our approach is in the opposite direction to Backens'. While Backens proved results in quantum entanglement theory to apply it to the complexity classification of Holant problems, we prove new results in quantum entanglement theory by employing the machinery from Holant problems. We prove Theorem 7 using a technique developed for Holant problems called the interplay between the *unique prime factorization* of signatures and *gadget constructions*. This technique is at the heart of a standard approach (arity reduction) to build inductive arguments for Holant problems [15]. The new result in quantum entanglement theory sheds light on the classification of entanglement under SLOCC equivalence.

Going one step further, we ask whether we can restrict projections onto only one state $|0\rangle$, while multipartite entanglement is still preserved. The answer is no. Then, one way to salvage the situation is to consider the self-loop gadget using one of the Bell states, $|\phi^+\rangle = |00\rangle + |11\rangle$ together with projections onto $|0\rangle$.

▶ **Definition 8** (Self-loop). *The self-loop on the i-th and j-th qubits of a state $|\Psi\rangle$ by the Bell state $|\phi^+\rangle = |00\rangle + |11\rangle$ is defined as $\langle \phi^+|_{ij}|\Psi\rangle = |\Psi_{ij}^{00}\rangle + |\Psi_{ij}^{11}\rangle$, where $|\Psi_{ij}^{00}\rangle$ and $|\Psi_{ij}^{11}\rangle$ are states of $n-2$ qubits when setting the i-th and j-th qubits of $|\Psi\rangle$ to 00 and 11 respectively.*

▶ **Lemma 9.** *Let $|\Psi\rangle$ be an n-qubit ($n \geqslant 4$) state exhibiting multipartite entanglement. There exists some choice of three or four of the n qubits such that by performing self-loops by $|\phi^+\rangle$ and projections onto $|0\rangle$ of the other qubits, we get*

- *a 3-qubit state exhibiting multipartite entanglement, or*
- *a* GHZ *type 4-qubit state, i.e.,* $|\mathrm{GHZ}_4\rangle = |0000\rangle + |1111\rangle$, *or*
- *the state $|1\rangle$.*

Why do we consider $|\phi^+\rangle$ and $|0\rangle$? The state $|\phi^+\rangle$ is synonymous with the binary EQUALITY signature $=_2$. It is always available in the Holant framework as it means *merging* two dangling edges in a graph. Moreover, we can show that $|0\rangle$ is realizable from any state of odd number of qubits under some mild assumptions. Then, we can apply Lemma 9 to get a new dichotomy for Holant problems where at least one signature of odd arity is present.

▶ **Theorem 10.** *Let $\mathcal{F}$ be a set of real-valued signatures containing at least one signature of odd arity. If $\mathcal{F}$ satisfies the tractability condition* (T) *in Theorem 21, then* $\mathrm{Holant}(\mathcal{F})$ *is polynomial-time computable; otherwise,* $\mathrm{Holant}(\mathcal{F})$ *is #P-hard.*

▶ Remark 11. Theorem 7 and Lemma 9 hold for complex-valued n-qubit states. However, Theorem 10 is restricted to real-valued signatures, in which the Hermitian conjugate and the complex inner product can be represented by a *mating* gadget in the Holant framework.

1.5 Surprising discovery of two extraordinary quantum states

What about signature sets containing only signatures of even arity, in which $|0\rangle$ cannot be realized. Since $|\phi^+\rangle$ is always available, we consider whether multipartite entanglement is preserved under self-loops by $|\phi^+\rangle$ alone. Given an n-qubit ($n \geqslant 6$ is even) state $|\Psi\rangle$ exhibiting multipartite entanglement, are there some i and j such that performing a self-loop by $|\phi^+\rangle$ on the i-th and j-th qubits of $|\Psi\rangle$ results in an $(n-2)$-qubit state exhibiting multipartite entanglement? (By the definition of multipartite entanglement, for even n it must be $n \geq 6$.)

The answer is no. Here we made a quite surprising discovery: There *exist* genuinely entangled 6-qubit and 8-qubit states such that multipartite entanglement is *not* preserved under self-loops. Furthermore, it is not preserved under self-loops not only by $|\phi^+\rangle$, but also by all four Bell states, $|\phi^+\rangle, |\psi^+\rangle = |01\rangle + |10\rangle, |\phi^-\rangle = |00\rangle - |11\rangle$, and $|\psi^-\rangle = |01\rangle - |10\rangle$. The self loop of the i-th and j-th qubits of $|\Psi\rangle$ by $|\psi^+\rangle$ is defined as $\langle\psi^+|_{ij}|\Psi\rangle = |\Psi^{01}_{ij}\rangle + |\Psi^{10}_{ij}\rangle$. Similarly, we can define $\langle\phi^-|_{ij}|\Psi\rangle$ and $\langle\psi^-|_{ij}|\Psi\rangle$.

▶ **Definition 12** (Bell property). *Let $|\Psi\rangle$ be a genuinely entangled state. We say that it satisfies the Bell property if for any two qubits i and j of $|\Psi\rangle$ and any Bell state $|\phi\rangle$, $\langle\phi|_{ij}|\Psi\rangle$ is a tensor product of Bell states. It satisfies the strong Bell property if for any i and j and any Bell state $|\phi\rangle$, $\langle\phi|_{ij}|\Psi\rangle$ is a tensor product of the Bell state $|\phi\rangle$, i.e., $\langle\phi|_{ij}|\Psi\rangle = |\phi\rangle \otimes \cdots \otimes |\phi\rangle$.*

▶ **Theorem 13.** *There exist genuinely entangled 6-qubit states that satisfy the Bell property, and genuinely entangled 8-qubit states that satisfy the strong Bell property.*

We first give an 8-qubit state $|\Psi_8\rangle$ that satisfies the strong Bell property.

$$\begin{aligned}|\Psi_8\rangle =& |00000000\rangle + |00001111\rangle + |00110011\rangle + |00111100\rangle + |01010101\rangle + |01011010\rangle + |10011001\rangle + |10010110\rangle \\ &+ |01101001\rangle + |01100110\rangle + |10100101\rangle + |10101010\rangle + |11000011\rangle + |11001100\rangle + |11110000\rangle + |11111111\rangle.\end{aligned}$$

$|\Psi_8\rangle$ can be represented by an 8-ary signature Ψ_8. Let $\mathcal{S}(\Psi_8)$ be the support of Ψ_8, i.e., $\mathcal{S}(\Psi_8) = \{\alpha \in \mathbb{Z}_2^8 \mid \Psi_8(\alpha) \neq 0\}$. $\mathcal{S}(\Psi_8)$ has the following structure: the sums of the first four variables, and the last four variables are both even; the assignment of the first four variables are either identical to, or complement of the assignment of the last four variables. While it is not obvious from this description that the support set is an affine subspace of $\mathbb{Z}_2^8$, but *it is*.

$$\begin{aligned}\mathcal{S}(\Psi_8) = \big\{(x_1, x_2, \ldots, x_8) \in \mathbb{Z}_2^8 \,|& x_1 + x_2 + x_3 + x_4 \equiv 0,\ x_1 + x_2 + x_5 + x_6 \equiv 0, \\ & x_1 + x_3 + x_5 + x_7 \equiv 0,\ x_2 + x_3 + x_5 + x_8 \equiv 0,\ \bmod 2\big\}.\end{aligned}$$

In other words, take 4 variables x_1, x_2, x_3, x_5, (these are not the first 4 variables in the description above), then on the support the remaining 4 variables are mod 2 sums of $\binom{4}{3}$ subsets of $\{x_1, x_2, x_3, x_5\}$. This will imply that $|\Psi_8\rangle$ is genuinely entangled. Also, one can check that $\langle\phi|_{12}|\Psi_8\rangle = |\phi\rangle^{\otimes 3}$ for any Bell state $|\phi\rangle$. Due to the symmetry of $|\Psi_8\rangle$, the same result holds by replacing $\{1, 2\}$ with any $\{i, j\}$. Thus, $|\Psi_8\rangle$ satisfies the strong Bell property.

The 6-qubit state $|\Psi_6\rangle$ satisfying the Bell property has 32 nonzero coefficients. We give it in the signature form.

$$\Psi_6(x_1, \ldots, x_6) = \chi_{\mathcal{S}(\Psi_6)} \cdot (-1)^{x_1x_4 + x_2x_5 + x_3x_6 + x_4x_5 + x_5x_6 + x_4x_6},$$

where $\chi_{\mathcal{S}(\Psi_6)}$ is the indicator function on the support $\mathcal{S}(\Psi_6) = \{(x_1, \ldots, x_6) \in \mathbb{Z}_2^6 \mid \sum_{i=1}^6 x_i = 0 \bmod 2\}$ (even parity). Such a support will imply that $|\Psi_6\rangle$ is genuinely entangled. We can write Ψ_6 as the following 8-by-8 matrix where the assignment of the first three variables in lexicographic order (from 000 to 111) is the row index and the assignment of the last three variables in lexicographic order is the column index.

$$M_{123,456}(\Psi_6) = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & -1 & 1 & 0 & 1 & 0 & 0 & -1 \\ 0 & 1 & -1 & 0 & 1 & 0 & 0 & -1 \\ -1 & 0 & 0 & -1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & -1 & 0 & 0 & -1 \\ -1 & 0 & 0 & 1 & 0 & -1 & 1 & 0 \\ -1 & 0 & 0 & 1 & 0 & 1 & -1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}.$$

By the symmetry of $|\Psi_6\rangle$, one can check that $|\Psi_6\rangle$ satisfies the Bell property by verifying $\langle\phi|_{12}|\Psi_6\rangle$, $\langle\phi|_{45}|\Psi_6\rangle$ and $\langle\phi|_{14}|\Psi_6\rangle$ are tensor products of Bell states for any bell state $|\phi\rangle$.

We can use Pauli operations to generate more states satisfying the Bell property. Consider the following four Pauli operators

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad Y = \begin{bmatrix} 0 & -\mathfrak{i} \\ \mathfrak{i} & 0 \end{bmatrix} \quad \text{and} \quad Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}.$$

A Pauli operation on an n-qubit state $|\Psi\rangle$ is defined as $P_1 \otimes P_2 \otimes \ldots \otimes P_n|\Psi\rangle$ (which produces another n-qubit state) where each P_i is a Pauli operator. Let $|\Psi_6\rangle$ and $|\Psi_8\rangle$ be states described above. Let $\mathfrak{P}_6$ and $\mathfrak{P}_8$ denote the sets of states realized by performing Pauli operations on $|\Psi_6\rangle$ and $|\Psi_8\rangle$ respectively.

▶ **Theorem 14.** *Every state in $\mathfrak{P}_6$ or $\mathfrak{P}_8$ satisfies the Bell property.*

Due to the existence of these 6-qubit and 8-qubit states with such extraordinary properties, it remains as a difficult task to achieve a full dichotomy for real-valued Holant problems. On the other hand, we hope such states can be further investigated and perhaps applied to quantum computing or quantum information theory.

The paper is organized as follows. In Section 2, we give a proof of our main quantum entanglement result (Theorem 7) by using the theory of signatures. Then we give some preliminaries for Holant problems in Section 3, and a proof sketch for our dichotomy result (Theorem 10) in Section 4.

2 Preservation of Multipartite Entanglement under Projections

We use the theory of signatures to prove Theorem 7. Recall that by our definition, a signature always has arity at least one. A nonzero signature g *divides* f denoted by $g \mid f$, if there is a signature h such that $f = g \otimes h$ (with possibly a permutation of variables) or there is a constant λ such that $f = \lambda \cdot g$. In the latter case, if $\lambda \neq 0$, then we also have $f \mid g$ since $g = \frac{1}{\lambda} \cdot f$. For nonzero signatures f and g, we use $f \sim g$ to denote both $g \mid f$ and $f \mid g$. A nonzero signature f is irreducible if f cannot be written as $g \otimes h$ for some signatures g and h. This is equivalent to saying that $|f\rangle$ is a genuinely entangled state of multiple qubits or $|f\rangle$ is a single-qubit state. Let $\mathcal{T}_1$ denote the set of tensor products of unary signatures and $\mathcal{T}$ denote the set of tensor products of unary and binary signatures. Then a state $|f\rangle$ of multiple qubits is entangled iff $f \notin \mathcal{T}_1$, and $|f\rangle$ exhibits multipartite entanglement iff $f \notin \mathcal{T}$. In terms of the above division relation, the unique prime factorization (UPF) of signatures is established (see Lemma 2.13 of [15]). The following result is a direct corollary.

▶ **Corollary 15.** *Let f be a nonzero n-ary signature. Suppose that there are two irreducible signatures g on variables in $A \subseteq [n]$ and h on variables in $B \subseteq [n]$ such that $g \mid f$ and $h \mid f$. Then, either A is disjoint with B, or $A = B$ and $g \sim h$.*

We use f_i^0 and f_i^1 to denote the signature forms of quantum states $|f_i^0\rangle$ and $|f_i^1\rangle$ realized by projections onto $|0\rangle$ and $|1\rangle$. In the Holant framework, these signatures are realized by a *pinning* gadget, i.e., connecting the variable x_i of f with unary signatures $\Delta_0 = (1, 0)$ and $\Delta_1 = (0, 1)$ respectively. We may further pick a variable x_j of f_i^c and pin it to the value d ($c, d \in \{0, 1\}$). Obviously, the pinning gadgets on different variables x_i and x_j commute. Thus, we have $(f_i^c)_j^d = (f_j^d)_i^c$. We denote it by f_{ij}^{cd}.

Suppose that $g \mid f$ where g is on variables in A. Then for any variable x_i of f that is not in A, we have $g \mid f_i^0$ and $g \mid f_i^1$ (By definition, the division relation holds even if f_i^0 or f_i^1 is a zero signature). Thus, the division relation is unchanged under pinning gadgets on variables out of A. The following lemma shows that a stronger converse is also true.

▶ **Lemma 16.** *Let f be a signature of arity $n \geqslant 2$. If there exists a signature g on variables in $A \subsetneq [n]$ and some x_i not in A such that $g \mid f_i^0$ and $g \mid f_i^1$, then $g \mid f$.*

Now, we are ready to prove Theorem 7. We restate it in terms of signatures. We use 0^n and 1^n to denote the n-bit all-0 and all-1 strings, and $\mathcal{S}(f)$ to denote the support of f.

▶ **Theorem 17.** *Let f be an n-ary ($n \geqslant 4$) signature, $f \notin \mathcal{T}$ and $f(0^n) \neq 0$. If $n \geqslant 5$ and $\mathcal{S}(f) \not\subseteq \{0^n, 1^n\}$, or $n = 4$ and $\mathcal{S}(f) \not\subseteq \{0000, 1111, 0011, 1100\}$ up to any permutation of four variables, then there exists some i such that f_i^0 or f_i^1 is not in $\mathcal{T}$.*

Proof. Since $f(0^n) \neq 0$, we have $f_i^0 \not\equiv 0$ and $f_{ij}^{00} \not\equiv 0$ (not identically 0) for all indices i and j. Also, since the support $\mathcal{S}(f) \not\subseteq \{0^n, 1^n\}$, there exist some s and t such that $f_{st}^{01} \not\equiv 0$. For a contradiction, we assume $f_i^0, f_i^1 \in \mathcal{T}$ for all i. We consider the following two possible cases.

Case 1. For all indices i, $f_i^0 \in \mathcal{T}_1$ (i.e., tensor product of unary signatures).

We will show that in this case, there is a unary signature $a(x_u)$ on some variable x_u, such that $a(x_u) \mid f$. This will lead to a contradiction.

Recall that there exist some s and t such that $f_{st}^{01} \not\equiv 0$. Then, clearly $f_t^1 \not\equiv 0$. Since $f_t^1 \in \mathcal{T}$, in the UPF of f_t^1, the variable x_s may appear in a unary signature or an irreducible binary signature. In both cases, since f has arity at least 4, we can pick a variable x_u such that x_u and x_s appear in two distinct irreducible signatures in the UPF of f_t^1 (i.e., x_u and x_s are not entangled in f_t^1). Then, we show that x_u must appear in a unary signature in the UPF of f_t^1. Otherwise, there is an irreducible binary signature $b(x_u, x_v)$ such that $b(x_u, x_v) \mid f_t^1$. Since x_u is not entangled with x_s in f_t^1, we have $v \neq s$. Then, $b(x_u, x_v) \mid f_{st}^{01}$. On the other hand, we consider f_s^0. By our assumption, $f_s^0 \in \mathcal{T}_1$ and hence there exists some unary signature $a'(x_u)$ such that $a'(x_u) \mid f_s^0$. Then, $a'(x_u) \mid f_{st}^{01}$. Since $f_{st}^{01} \not\equiv 0$, by Corollary 15, $b(x_u, x_v) \sim a'(x_u)$. Contradiction. Thus, there exists some $a(x_u)$ such that $a(x_u) \mid f_t^1$.

Now we show that $a(x_u) \mid f_t^0$. First, we show that $a(x_u) \mid f_s^0$. Since $f_s^0 \in \mathcal{T}_1$, there exists some unary signature $a'(x_u)$ such that $a'(x_u) \mid f_s^0$, and then $a'(x_u) \mid f_{st}^{01}$. Also, we have $a(x_u) \mid f_{st}^{01}$ since $a(x_u) \mid f_t^1$. Since $f_{st}^{01} \not\equiv 0$, by Corollary 15, we have $a(x_u) \sim a'(x_u)$. Thus, $a(x_u) \mid f_s^0$. Since $f_t^0 \in \mathcal{T}_1$, there exists a unary signature $a''(x_u)$ such that $a''(x_u) \mid f_t^0$, and then $a''(x_u) \mid f_{st}^{00}$. Also, we have $a(x_u) \mid f_{st}^{00}$ since $a(x_u) \mid f_s^0$. Remember that $f_{st}^{00} \not\equiv 0$. Then, by Corollary 15, we have $a(x_u) \sim a''(x_u)$. Thus, $a(x_u) \mid f_t^0$.

Since $a(x_u) \mid f_t^0$ and $a(x_u) \mid f_t^1$, by Lemma 16, we have $a(x_u) \mid f$. In other words, $f = a(x_u) \otimes g$ where g is a nonzero signature of arity $n - 1$ on variables other than x_u. Since $f \notin \mathcal{T}$, we have $g \notin \mathcal{T}$. Consider f_u^0. We know that it is a nonzero signature and hence $f_u^0 \sim g$. Thus, $f_u^0 \notin \mathcal{T}$. We have reached a contradiction.

Case 2. There exists some index k and an irreducible binary signature $b(x_v, x_w)$ such that $b(x_v, x_w) \mid f_k^0$.

We will show that in this case, $b(x_v, x_w) \mid f$. First, we show that $b(x_v, x_w) \mid f_i^0$ for all $i \notin \{v, w\}$. We already have $b(x_v, x_w) \mid f_k^0$. Consider f_i^0 for all indices $i \notin \{v, w, k\}$. Since $f_i^0 \in \mathcal{T}$ and $f_i^0 \not\equiv 0$, there is either a unary signature $a(x_v)$ or an irreducible binary signature $b'(x_v, x_{w'})$ for some $w' \notin \{i, v\}$ that appears in the UPF of f_i^0, i.e., $a(x_v) \mid f_i^0$ or $b'(x_v, x_{w'}) \mid f_i^0$. In the former case, we have $a(x_v) \mid f_{ik}^{00}$. In the latter case and if $w' \neq k$, we have $b'(x_v, x_{w'}) \mid f_{ik}^{00}$. In the latter case and if $w' = k$, then let $a'(x_v)$ be the unary signature realized from $b'(x_v, x_{w'})$ by pinning $x_{w'} = x_k$ to 0, we get $a'(x_v) \mid f_{ik}^{00}$. On the other hand, since $b(x_v, x_w) \mid f_k^0$, we have $b(x_v, x_w) \mid f_{ik}^{00}$. Since $f_{ik}^{00} \not\equiv 0$, by Corollary 15, we know that the two cases that $a(x_v) \mid f_{ik}^{00}$ and $a'(x_v) \mid f_{ik}^{00}$ cannot occur. Thus, $w' \neq k$ and $b'(x_v, x_{w'}) \mid f_{ik}^{00}$. By Corollary 15, $w' = w$, and $b(x_v, x_w) \sim b'(x_v, x_{w'})$. Thus, $b(x_v, x_w) \mid f_i^0$ for all $i \notin \{v, w\}$.

Then we want to show that there exists some $j \notin \{v, w\}$ such that $b(x_v, x_w) \mid f_j^1$.

- We first consider the case that there exist some indices i and j where $\{i, j\}$ is disjoint with $\{v, w\}$ such that $f_{ij}^{01} \not\equiv 0$. We show that $b(x_v, x_w) \mid f_j^1$. Since $b(x_v, x_w) \mid f_i^0$, we have $b(x_v, x_w) \mid f_{ij}^{01}$. By assumption $f_{ij}^{01} \not\equiv 0$, and then clearly $f_j^1 \not\equiv 0$. Recall that $f_j^1 \in \mathcal{T}$. Again, there is either a unary signature $a(x_v)$ or an irreducible binary signature $b'(x_v, x_{w'})$ that appears in the UPF of f_j^1, i.e., $a(x_v) \mid f_j^1$ or $b'(x_v, x_{w'}) \mid f_j^1$. In the first case since $i \neq v$, we can pin x_i of f_j^1 to 0, and we get $a(x_v) \mid f_{ij}^{01}$. In the second case and if $w' = i$, again we can get $a'(x_v) \mid f_{ij}^{01}$, where $a'(x_v) = b'(x_v, 0)$, obtained from pinning x_i to 0. But $f_{ij}^{01} \not\equiv 0$ and $b(x_v, x_w) \mid f_{ij}^{01}$. Then, in the UPF of f_{ij}^{01}, it does not have a unary signature on x_v as a factor. Thus, it must be the case that $b'(x_v, x_{w'}) \mid f_j^1$ where $w' \neq i$. Then, we have $b'(x_v, x_{w'}) \mid f_{ij}^{01}$. Since $b(x_v, x_w) \mid f_{ij}^{01}$ and $f_{ij}^{01} \not\equiv 0$, by Corollary 15, $w' = w$ and $b'(x_v, x_{w'}) \sim b(x_v, x_w)$, and thus $b(x_v, x_w) \mid f_j^1$. Then, by Lemma 16, we have $b(x_v, x_w) \mid f$. In other words, $f = b(x_v, x_w) \otimes h$ where h is a nonzero signature of arity $n - 2$ on variables other than x_v and x_w. Since $f \notin \mathcal{T}$, we have $h \notin \mathcal{T}$. Then consider f_v^0. We know that it is a nonzero signature and $h \mid f_v^0$. Thus, $f_v^0 \notin \mathcal{T}$. Contradiction.
- Then we consider the case that $f_{ij}^{01} \equiv 0$ for all indices $\{i, j\}$ that are disjoint with $\{v, w\}$. Consider an n-bit input α of f. We write α as $\alpha_v \alpha_w \beta$ where α_v is the input on variable x_v, α_w is the input on variable x_w, and β is the input on the other $n - 2$ variables. Then, $f(\alpha) = 0$ if β is not the all-0 or all-1 bit string in $\{0, 1\}^{n-2}$. It follows that f has at most eight nonzero entries. We list all its entries by the following 4-by-2^{n-2} matrix $M_{vw}(f)$ with $(x_v, x_w) \in \{0, 1\}^2$ as the row index (in the order 00, 01, 10, 11) and the assignment of the other variables in lexicographic order as the column index.

$$M_{vw}(f) = \begin{bmatrix} c_1 & 0 & \dots & \dots & 0 & c_2 \\ c_3 & 0 & \dots & \dots & 0 & c_4 \\ c_5 & 0 & \dots & \dots & 0 & c_6 \\ c_7 & 0 & \dots & \dots & 0 & c_8 \end{bmatrix}.$$

Here, $c_1 = f(0^n) \neq 0$. Consider signatures f_v^0 and f_v^1. They have the following matrix forms with the variable $x_w \in \{0, 1\}$ as the row index.

$$M_w(f_v^0) = \begin{bmatrix} c_1 & 0 & \dots & \dots & 0 & c_2 \\ c_3 & 0 & \dots & \dots & 0 & c_4 \end{bmatrix} \quad \text{and} \quad M_w(f_v^1) = \begin{bmatrix} c_5 & 0 & \dots & \dots & 0 & c_6 \\ c_7 & 0 & \dots & \dots & 0 & c_8 \end{bmatrix}.$$

Also consider signatures f_w^0 and f_w^1. They have the following matrix forms with the variable $x_v \in \{0, 1\}$ as the row index.

$$M_v(f_w^0) = \begin{bmatrix} c_1 & 0 & \dots & \dots & 0 & c_2 \\ c_5 & 0 & \dots & \dots & 0 & c_6 \end{bmatrix} \quad \text{and} \quad M_v(f_w^1) = \begin{bmatrix} c_3 & 0 & \dots & \dots & 0 & c_4 \\ c_7 & 0 & \dots & \dots & 0 & c_8 \end{bmatrix}.$$

Consider f_v^0. Since f has arity at least 4, f_v^0 has arity at least 3. Since $f_v^0 \in \mathcal{T}$, the variable x_w either appears in a unary factor $a(x_w)$ of f_v^0 or an irreducible binary factor $b(x_w, x_{w'})$ of f_v^0. In the latter case, we can pick another variable x_r of f_v^0 where $r \neq w$ or w', and we consider f_{vr}^{00}. We know that $f_{vr}^{00} \not\equiv 0$ since $c_1 \neq 0$ and $b(x_w, x_{w'}) \mid f_{vr}^{00}$ since $b(x_w, x_{w'}) \mid f_v^0$. Notice that the column with c_2 and c_4 does not appear in $M_w(f_{vr}^{00})$. Thus, the signature f_{vr}^{00} is of the form $(c_1, c_3) \otimes (1,0)^{\otimes(n-2)}$ which is a tensor product of unary signatures. Contradiction. Thus, there is a unary signature $a(x_w)$ such that $a(x_w) \mid f_v^0$. Then, we have $c_1c_4 = c_2c_3$. Similarly by considering f_w^0, we have $c_1c_6 = c_2c_5$. Now, we consider f_v^1, and prove $c_5c_8 = c_6c_7$. If $c_5 = c_7 = 0$, then clearly we have $c_5c_8 = c_6c_7 = 0$. Otherwise, for any $r \neq w$ or v, we have $f_{vr}^{10} = (c_5, c_7) \otimes (1,0)^{\otimes(n-2)} \not\equiv 0$ which is a tensor product of unary signatures. If there is a binary signature $b(x_w, x_{w'})$ such that $b(x_w, x_{w'}) \mid f_v^1$, then we can find some $r \neq w, w'$ such that $b(x_w, x_{w'}) \mid f_{vr}^{10}$. Contradiction. Thus, there is a unary signature $a(x_w)$ such that $a(x_w) \mid f_v^1$. Then, we have $c_5c_8 = c_6c_7$. Similarly by considering f_w^1, we have $c_3c_8 = c_4c_7$.

- Suppose $n \geqslant 5$. Then f_v^0 has arity at least 4. We first show that $c_2 = 0$. We consider $f_{vw}^{00} = [c_1, 0, \ldots\ldots, 0, c_2]$. Since $f_v^0 \in \mathcal{T}$, we have $f_{vw}^{00} \in \mathcal{T}$. Note that f_{vw}^{00} has arity at least 3. Since $c_1 \neq 0$, the only possible value of c_2 to make $f_{vw}^{00} \in \mathcal{T}$ is 0. Thus, $c_2 = 0$. Since $c_1c_4 = c_2c_3 = 0$ and $c_1 \neq 0$, we have $c_4 = 0$. Also, since $c_1c_6 = c_2c_5 = 0$ and $c_1 \neq 0$, we have $c_6 = 0$. If $c_8 = 0$, then $f = b(x_v, x_w) \otimes (1,0)^{\otimes(n-2)} \in \mathcal{T}$. A contradiction with $f \notin \mathcal{T}$. Thus, we have $c_8 \neq 0$. Since $c_5c_8 = c_6c_7 = 0$ and $c_8 \neq 0$, we have $c_5 = 0$. Also since $c_3c_8 = c_4c_7 = 0$ and $c_8 \neq 0$, we have $c_3 = 0$. Consider $f_{vw}^{11} = [c_7, 0, \ldots\ldots, 0, c_8]$. Since $f_{vw}^{11} \in \mathcal{T}$ and it has arity at least 3, and $c_8 \neq 0$, we have $c_7 = 0$. Thus, f has only two nonzero entries that are on the all-0 input and the all-1 input. A contradiction with our assumption that $\mathcal{S}(f) \nsubseteq \{0^n, 1^n\}$.
- Suppose $n = 4$. If $c_2 = 0$, then with the same proof as in the case that $n \geqslant 5$, we have $c_4 = c_6 = 0$, $c_8 \neq 0$ and then $c_3 = c_5 = 0$. Thus, $\mathcal{S}(f) \subseteq \{0000, 1111, 1100\}$. Contradiction. Otherwise, $c_2 \neq 0$. Suppose that $c_2 = kc_1$. Then $c_4 = kc_3$ since $c_1c_4 = c_2c_3$ and $c_6 = kc_5$ since $c_1c_6 = c_2c_5$. If c_3 and c_4 are not zero, then $c_8 = kc_7$ since $c_3c_8 = c_4c_7$. Then, $f = b(x_v, x_w) \otimes (1,0,0,k) \in \mathcal{T}$. Contradiction. Thus, $c_3 = c_4 = 0$. Similarly, if c_5 and c_6 are not zero, then we still have $c_8 = kc_7$ since $c_5c_8 = c_6c_7$. Then, we have $f \in \mathcal{T}$. Contradiction. Thus, $c_5 = c_6 = 0$. Then, $\mathcal{S}(f) \subseteq \{0000, 1111, 0011, 1100\}$. Contradiction.

Therefore, there exists some i such that f_i^0 or f_i^1 is not in $\mathcal{T}$. ◀

Our result can be used in the classification of entanglement under SLOCC equivalence. An n-qubit state $|\Psi\rangle$ is equivalent to another n-qubit state $|\Phi\rangle$ under SLOCC if there exist some invertible 2-by-2 matrices M_1, M_2, …, M_n such that $|\Psi\rangle = M_1 \otimes M_2 \otimes \ldots \otimes M_n|\Phi\rangle$. Physicists are interested in the classification of SLOCC equivalence classes. For 2-qubit states there are two SLOCC classes, and for 3-qubit states there are six SLOCC classes [24]. However, for states of 4 or more qubits there are infinitely many SLOCC classes [24]. Then, the goal is to categorize these classes into some finitely many families with common physical or mathematical properties. Depending on which properties are used, there are different approaches. One powerful approach that can possibly handle states of a high number of qubits is by induction [36, 35, 1, 30]. In this approach, the classification of n-qubit states relies on the classification of $(n-1)$-qubit states.

Consider an n-qubit state $|\Psi\rangle$. We can pick some index i and write $|\Psi\rangle$ as $|\Psi\rangle = |0\rangle|\Psi_i^0\rangle + |1\rangle|\Psi_i^1\rangle$. Families of entanglement classes of $|\Psi\rangle$ can be defined according to the types of entanglements found in the linear span$\{|\Psi_i^0\rangle, |\Psi_i^1\rangle\}$ which is related to the entanglement types of $|\Psi_i^0\rangle$ and $|\Psi_i^1\rangle$ themselves. Theorem 7 gives a direct relation between the entanglement types of $|\Psi\rangle$ and $\{|\Psi_i^0\rangle, |\Psi_i^1\rangle\}$. For example, consider a 5-qubit state exhibiting multipartite entanglement. First, by performing SLOCC using the matrix $N_2 = \left[\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right]$ on this state, we can always get a state $|\Psi\rangle$ where the coefficient of $|0^5\rangle$ is nonzero. If $|\Psi\rangle$ has the form $a|0^5\rangle + b|1^5\rangle$, then it is equivalent to $|\mathrm{GHZ}_5\rangle = |0^5\rangle + |1^5\rangle$. Otherwise, we can apply Theorem 7. There exists some i such that $|\Psi_i^0\rangle$ or $|\Psi_i^1\rangle$ exhibits multipartite entanglement. Then, in order to classify the state $|\Psi\rangle$, we only need to consider possible entanglement types of $\{|\Psi_i^0\rangle, |\Psi_i^1\rangle\}$ where at least one state exhibits multipartite entanglement. This eliminates many cases compared to considering all entanglement types of $\{|\Psi_i^0\rangle, |\Psi_i^1\rangle\}$.

3 Preliminaries for Holant Problems

3.1 Definitions and Notations

Let f be a complex-valued signature. If $\overline{f(\alpha)} = f(\overline{\alpha})$ for all α where $\overline{f(\alpha)}$ denotes the complex conjugation of $f(\alpha)$ and $\overline{\alpha}$ denotes the bit-wise complement of α, we say f satisfies *arrow reversal symmetry* (ARS). We may use f^α to denote $f(\alpha)$.

We use $=_n$ to denote the EQUALITY signature of arity n, which takes value 1 on the all-0 or all-1 inputs and 0 elsewhere. Note that $=_2$ represents the Bell state $|\phi^+\rangle$. Let $\mathcal{EQ} = \{=_1, =_2, \ldots, =_n, \ldots\}$ denote the set of all EQUALITY signatures. Then $\#\mathrm{CSP}(\mathcal{F})$ is exactly $\mathrm{Holant}(\mathcal{EQ} \mid \mathcal{F})$. Also, let $\mathcal{EQ}_k = \{=_k, =_{2k}, \ldots, =_{nk}, \ldots\}$, and we define $\#\mathrm{CSP}_k(\mathcal{F})$ to be $\mathrm{Holant}(\mathcal{EQ}_k \mid \mathcal{F})$. The following two reductions are known [10]:

$$\#\mathrm{CSP}(\mathcal{F}) \leqslant_T \mathrm{Holant}(=_3, \mathcal{F}) \quad \text{and} \quad \#\mathrm{CSP}_2(\mathcal{F}) \leqslant_T \mathrm{Holant}(=_4, \mathcal{F}).$$

Here, $\leqslant_T$ denotes P-time Turing reduction. We use $\neq_2$ to denote the binary DISEQUALITY signature with truth table $(0,1,1,0)$. It represents the Bell state $|\psi^+\rangle$.

A signature f of arity $n \geqslant 2$ can be expressed as a $2 \times 2^{n-1}$ matrix $M_i(f)$, which lists the 2^n entries of f with variable $x_i \in \{0,1\}$ as row index and the assignments of the other $n-1$ variables in lexicographic order as column index, i.e. $M_i(f) = \left[\begin{smallmatrix} f^{0,00\ldots0} & f^{0,00\ldots1} & \ldots & f^{0,11\ldots1} \\ f^{1,00\ldots0} & f^{1,00\ldots1} & \ldots & f^{1,11\ldots1} \end{smallmatrix}\right] = \left[\begin{smallmatrix} \mathbf{f}_i^0 \\ \mathbf{f}_i^1 \end{smallmatrix}\right]$, where $\mathbf{f}_i^a$ denotes the row vector indexed by $x_i = a$ in $M_i(f)$. For $=_2$, it has the 2-by-2 signature matrix $M(=_2) = I_2 = \left[\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right]$. For $\neq_2$, $M(\neq_2) = N_2 = \left[\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right]$.

3.2 Holographic Transformation

For an invertible matrix $T \in \mathbf{GL}_2(\mathbb{C})$ and a signature f of arity n, written as a column vector $f \in \mathbb{C}^{2^n}$, we denote by $Tf = T^{\otimes n} f$ the transformed signature. For a signature set $\mathcal{F}$, define $T\mathcal{F} = \{Tf \mid f \in \mathcal{F}\}$ to be the set of transformed signatures. For signatures written as row vectors we define fT^{-1} and $\mathcal{F}T^{-1}$ similarly.

Let $T \in \mathbf{GL}_2(\mathbb{C})$. The holographic transformation defined by T is the following operation: given a signature grid $\Omega = (H, \pi)$ of $\mathrm{Holant}(\mathcal{F} \mid \mathcal{G})$, for the same bipartite graph H, we get a new signature grid $\Omega' = (H, \pi')$ of $\mathrm{Holant}\left(\mathcal{F}T^{-1} \mid T\mathcal{G}\right)$ by replacing each signature in $\mathcal{F}$ or $\mathcal{G}$ with the corresponding signature in $\mathcal{F}T^{-1}$ or $T\mathcal{G}$.

▶ **Theorem 18** (Valiant's Holant Theorem [45]). *For any $T \in \boldsymbol{GL}_2(\mathbb{C})$,*

$$\mathrm{Holant}(\mathcal{F} \mid \mathcal{G}) \equiv_T \mathrm{Holant}(\mathcal{F}T^{-1} \mid T\mathcal{G}).$$

$\mathrm{Holant}(\mathcal{F})$ is equivalent to its bipartite form $\mathrm{Holant}\,(=_2\mid \mathcal{F})$. A particular holographic transformation that will be commonly used is the transformation defined by $Z^{-1} = \frac{1}{\sqrt{2}} \left[\begin{smallmatrix} 1 & -\mathfrak{i} \\ 1 & \mathfrak{i} \end{smallmatrix}\right]$, with $Z = \frac{1}{\sqrt{2}} \left[\begin{smallmatrix} 1 & 1 \\ \mathfrak{i} & -\mathfrak{i} \end{smallmatrix}\right]$. Since $(=_2)Z = (\neq_2)$, we have $\mathrm{Holant}\,(=_2\mid \mathcal{F}) \equiv_T \mathrm{Holant}\left(\neq_2\mid Z^{-1}\mathcal{F}\right)$. We denote $Z^{-1}\mathcal{F}$ by $\widehat{\mathcal{F}}$ and $Z^{-1}f$ by $\widehat{f}$. The following relation between f and $\widehat{f}$ is known.

▶ **Lemma 19** (Lemma A.2 in [15]). *f is a real valued signature iff $\widehat{f}$ satisfies* ARS.

3.3 Gadget Construction

One basic reduction for Holant problems is gadget construction. We say a signature f is *realizable* from a signature set $\mathcal{F}$ (by gadget construction) if there is a graph $G = (V, E, D)$ with internal edges E and dangling edges D where each vertex $v \in V$ is labeled by a signature f_v from $\mathcal{F}$, and the graph defines the signature f by its sum-of-products with inputs on the dangling edges. If f is realizable from a set $\mathcal{F}$, then $\text{Holant}(f, \mathcal{F}) \equiv_T \text{Holant}(\mathcal{F})$.

A basic gadget construction is *merging*; we connect two variables x_i and x_j of f using $=_2$. We use $\partial_{ij} f = f_{ij}^{00} + f_{ij}^{11}$ to denote a signature realized by merging, where f_{ij}^{ab} denotes the signature obtained by setting $(x_i, x_j) = (a, b) \in \{0,1\}^2$. The merging operation using $=_2$ is synonymous with performing a self-loop by the Bell state $|\phi^+\rangle$.

A gadget construction often used in this paper is *mating*. Given a *real*-valued signature f of arity $n > m \geqslant 1$, we connect two copies of f in the following manner: Fix a set S of $n - m$ variables among all n variables of f. For each $x_k \in S$, connect x_k of one copy of f with x_k of the other copy using $=_2$. The variables that are not in S are called dangling variables. For $m = 1$, there is one dangling variable x_i. Then, the mating construction realizes a binary signature, denoted by $\mathfrak{m}_i f$. It can be represented by matrix multiplication. We have

$$M(\mathfrak{m}_i f) = M_i(f) I_2^{\otimes(n-1)} M_i^{\mathrm{T}}(f) = \begin{bmatrix} \mathbf{f}_i^0 \\ \mathbf{f}_i^1 \end{bmatrix} \begin{bmatrix} \mathbf{f}_i^{0\mathrm{T}} & \mathbf{f}_i^{1\mathrm{T}} \end{bmatrix} = \begin{bmatrix} |\mathbf{f}_i^0|^2 & \langle \mathbf{f}_i^0, \mathbf{f}_i^1 \rangle \\ \langle \mathbf{f}_i^0, \mathbf{f}_i^1 \rangle & |\mathbf{f}_i^1|^2 \end{bmatrix}. \tag{3.1}$$

The (complex) inner product $\langle \cdot, \cdot \rangle$ uses complex conjugation. But since f is real-valued, this is the same as the usual dot product. $|\mathbf{f}|$ denotes its 2-norm. In the setting of $\text{Holant}(\neq_2 | \widehat{\mathcal{F}})$, the above mating operation is equivalent to connecting variables in S using $\neq_2$. We denote the resulting signature by $\widehat{\mathfrak{m}}_i \widehat{f} = \widehat{\mathfrak{m}_i f}$. Note that $\widehat{f}$ satisfies ARS since f is real. Thus,

$$N_2^{\otimes(n-1)} \widehat{\mathbf{f}}_i^{0\,\mathrm{T}} = (\widehat{f}^{0,11\ldots1}, \widehat{f}^{0,11\ldots0}, \ldots, \widehat{f}^{0,00\ldots0})^{\mathrm{T}} = (\overline{\widehat{f}^{1,00\ldots0}}, \overline{\widehat{f}^{1,00\ldots1}}, \ldots, \overline{\widehat{f}^{1,11\ldots1}})^{\mathrm{T}} = \overline{\widehat{\mathbf{f}}_i^1}^{\mathrm{T}}.$$

Then, we have

$$M(\widehat{\mathfrak{m}}_i \widehat{f}) = \begin{bmatrix} \widehat{\mathbf{f}}_i^0 \\ \widehat{\mathbf{f}}_i^1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}^{\otimes(n-1)} \begin{bmatrix} \widehat{\mathbf{f}}_i^{0\,\mathrm{T}} & \widehat{\mathbf{f}}_i^{1\,\mathrm{T}} \end{bmatrix} = \begin{bmatrix} \widehat{\mathbf{f}}_i^0 \\ \widehat{\mathbf{f}}_i^1 \end{bmatrix} \begin{bmatrix} \overline{\widehat{\mathbf{f}}_i^1}^{\mathrm{T}} & \overline{\widehat{\mathbf{f}}_i^0}^{\mathrm{T}} \end{bmatrix} = \begin{bmatrix} \langle \widehat{\mathbf{f}}_i^0, \widehat{\mathbf{f}}_i^1 \rangle & |\widehat{\mathbf{f}}_i^0|^2 \\ |\widehat{\mathbf{f}}_i^1|^2 & \langle \widehat{\mathbf{f}}_i^1, \widehat{\mathbf{f}}_i^0 \rangle \end{bmatrix}. \tag{3.2}$$

Here, due to ARS, the complex inner product can also be represented by mating using $\neq_2$.

3.4 Known results

We give some known signature sets that define tractable, i.e., polynomial time computable, counting problems. There are three families: product-type signatures, affine signatures and local affine signatures denoted by $\mathscr{P}$, $\mathscr{A}$ and $\mathscr{L}$ respectively. Please see the full paper or [22] for definitions and more details. Problems defined by $\mathscr{T}$ are also tractable.

▶ **Definition 20.** *We say a signature set $\mathcal{F}$ is $\mathscr{C}$-transformable if there exists a $T \in \mathbf{GL}_2(\mathbb{C})$ such that $(=_2)(T^{-1})^{\otimes 2} \in \mathscr{C}$ and $T\mathcal{F} \subseteq \mathscr{C}$.*

By Theorem 18, if $\text{Holant}(\mathscr{C})$ is tractable, then $\text{Holant}(\mathcal{F})$ is tractable for any $\mathscr{C}$-transformable set $\mathcal{F}$. The following tractable results are known [22, 3].

▶ **Theorem 21.** *For any complex-valued signature set $\mathcal{F}$,* Holant$(\mathcal{F})$ *is P-time computable if*

$$\mathcal{F} \subseteq \mathscr{T}, \quad \mathcal{F} \text{ is } \mathscr{P}\text{-transformable}, \quad \mathcal{F} \text{ is } \mathscr{A}\text{-transformable}, \quad \text{or } \mathcal{F} \text{ is } \mathscr{L}\text{-transformable}. \tag{T}$$

Based on dichotomy results of #CSP and Holant^c [21, 22, 3], the following #P-hardness results are known.

▶ **Theorem 22.** *Let $\mathcal{F}$ be a set of real-valued signatures. If $\mathcal{F}$ does not satisfy condition* (T)*, then* $\#\mathrm{CSP}(\mathcal{F})$*,* $\#\mathrm{CSP}_2(\mathcal{F})$ *and* $\mathrm{Holant}^c(\mathcal{F})$ *are #P-hard.*

For reducible signatures, the following reduction was proved by Lin and Wang [37].

▶ **Lemma 23.** *If a nonzero real-valued signature f has a factorization $g \otimes h$ where g and h are also real-valued signatures, then* $\mathrm{Holant}(g, h, \mathcal{F}) \equiv_T \mathrm{Holant}(f, \mathcal{F})$*. In this case, we say that g and h are realizable from f by factorization.*

4 Proof Sketch for Theorem 10

We give a proof sketch for Theorem 10.

By Theorem 21, if $\mathcal{F}$ satisfies condition (T), then $\mathrm{Holant}(\mathcal{F})$ is tractable. We prove #P-hardness when $\mathcal{F}$ does not satisfy condition (T). First, we show that under some holographic transformations, either one can use a signature of odd arity in $\mathcal{F}$ to realize the unary signature $\Delta_0 = (1, 0)$, or one can realize an equality signature $=_k$ for some $k \geqslant 3$.

▶ **Lemma 24.** *Let $\mathcal{F}$ be a set of real-valued signatures containing a signature of odd arity. Then* $\mathrm{Holant}(\neq_2|=_k, \widehat{\mathcal{F}}) \leqslant_T \mathrm{Holant}(\mathcal{F})$ *for some $k \geqslant 3$, or* $\mathrm{Holant}(\Delta_0, Q\mathcal{F}) \leqslant_T \mathrm{Holant}(\mathcal{F})$ *for some real orthogonal 2-by-2 matrix $Q \in \mathbf{O}_2(\mathbb{R})$.*

We first prove the #P-hardness of $\mathrm{Holant}(\neq_2|=_k, \widehat{\mathcal{F}})$ given $k \geqslant 3$ and $\mathcal{F}$ does not satisfy condition (T). We give the following reduction.

▶ **Lemma 25.** *If $k \geqslant 3$, then* $\#\mathrm{CSP}_k(\neq_2, \mathcal{G}) \equiv_T \mathrm{Holant}(\mathcal{EQ}_k\ |\neq_2, \mathcal{G}) \leqslant_T \mathrm{Holant}(\neq_2|=_k, \mathcal{G})$*.*

Proof. The first equivalence is by definition. For the second reduction, we show that $=_{nk}$ can be realized on the LHS by induction on n. First, we connect one variable of each of k copies of $\neq_2$ on the LHS with the k variables of $=_k$ on the RHS (Figure 1a). This gadget realizes $=_k$ on the LHS.

Then, suppose that $=_{nk}$ is realizable on the LHS. We take one copy of $=_{nk}$ and two copies of $=_k$ on the LHS, and one copy of $=_k$ on the RHS. Remember that $k \geqslant 3$. We connect two variables of $=_k$ on the RHS with one variable of each of the two copies of $=_k$ on the LHS, and connect the other $k - 2$ variables of $=_k$ on the RHS with $k - 2$ variables of $=_{nk}$ on the LHS (Figure 1b). This gadget realizes $=_{(n+1)k}$ on the LHS.

Also, connecting $k - 1$ variables of one copy of $=_k$ on the RHS with $k - 1$ variables of another copy of $=_k$ on the RHS using $\neq_2$ on the LHS realizes $\neq_2$ on the RHS. ◀

Then, we give a dichotomy of $\#\mathrm{CSP}_k(\neq_2, \mathcal{G})$ for any *complex*-valued signature set $\mathcal{G}$. This result should be of independent interest. Let $\rho_k = e^{\frac{\mathrm{i}\pi}{2k}}$ be a $4k$-th primitive root of unity, $T_k = \left[\begin{smallmatrix} 1 & 0 \\ 0 & \rho_k \end{smallmatrix}\right]$, and $\mathcal{A}_k^d = T_k^d \mathcal{A} = \{T_k^d f \mid f \in \mathcal{A}\}$ where $d \in [k]$.

▶ **Theorem 26.** *Let $\mathcal{G}$ be a set of complex-valued signatures. If $\mathcal{G} \subseteq \mathcal{P}$ or $\mathcal{G} \subseteq \mathcal{A}_k^d$ for some $d \in [k]$. then* $\#\mathrm{CSP}_k(\neq_2, \mathcal{G})$ *is tractable; otherwise,* $\#\mathrm{CSP}_k(\neq_2, \mathcal{G})$ *is* # P*-hard.*

When $\mathcal{F}$ does not satisfy condition (T), we can show that $\widehat{\mathcal{F}} \notin \mathcal{P}$ and $\widehat{\mathcal{F}} \notin \mathcal{A}_k^d$ for any $d \in [k]$. Combining with Lemma 25 and Theorem 26, we have the following result.

▶ **Lemma 27.** *Let $\mathcal{F}$ be a set of complex-valued signatures. If $\mathcal{F}$ does not satisfy condition* (T) *and $k \geqslant 3$, then* $\mathrm{Holant}(\neq_2|=_k, \widehat{\mathcal{F}})$ *is #P-hard.*

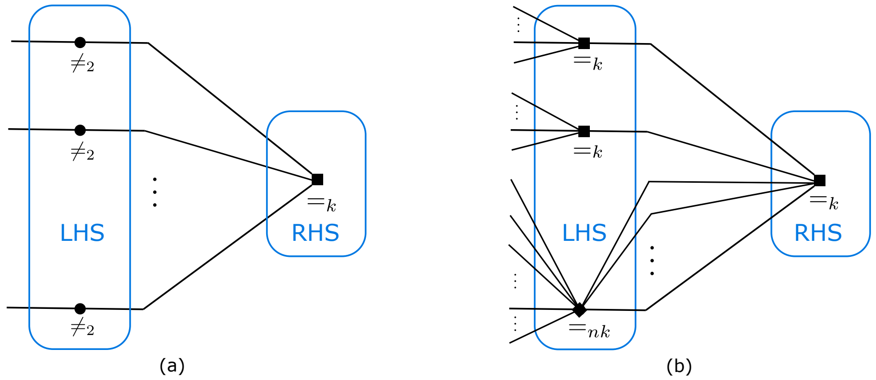


Figure 1 Gadgets realizing $=_k$ and $=_{(n+1)k}$ on the LHS.

Next, we prove the #P-hardness of Holant$(\Delta_0, Q\mathcal{F})$ given $\mathcal{F}$ is a real-valued signature set not satisfying condition (T) and $Q \in \mathbf{O}_2(\mathbb{R})$. Under this condition, we can show that $Q\mathcal{F}$ is also a real-valued signature set not satisfying condition (T). Thus, in order to prove Holant$(\Delta_0, Q\mathcal{F})$ is #P-hard, it suffices to show that Holant$(\Delta_0, \mathcal{F})$ is #P-hard for any real-valued signature set $\mathcal{F}$ not satisfying condition (T). Here, we will apply our entanglement result (Lemma 9) to the proof of #P-hardness. If Δ_1 is realizable from Holant$(\Delta_0, \mathcal{F})$, then we reduce Holant$(\Delta_0, \mathcal{F})$ from Holant$^c(\mathcal{F})$ and we are done by the dichotomy of Holant$^c(\mathcal{F})$. By using Δ_0, we first give two conditions that Δ_1 can be easily realized by pinning or interpolation. We show that either Holant$^c(\mathcal{F}) \leqslant_T$ Holant$(\Delta_0, \mathcal{F})$, or every irreducible $f \in \mathcal{F}$ satisfies the following important *first order orthogonality* condition.

▶ **Definition 28** (First order orthogonality). *Let f be a complex-valued signature of arity $n \geqslant 2$, we say that it satisfies* first order orthogonality (1st-Orth) *if there exists some $\mu \neq 0$ such that for all indices $i \in [n]$, the entries of f satisfy the following equations*

$$|\boldsymbol{f}_i^0|^2 = |\boldsymbol{f}_i^1|^2 = \mu, \text{ and } \langle \boldsymbol{f}_i^0, \boldsymbol{f}_i^1 \rangle = 0.$$

To restate it in the quantum terminology, let $|\Psi\rangle$ be a normalized n-qubit ($n \geqslant 2$) state, i.e., $\langle \Psi|\Psi\rangle = 1$. Then it satisfies first order orthogonality if for every i-th qubit of $|\Psi\rangle$, $\langle \Psi_i^0|\Psi_i^0\rangle = \langle \Psi_i^1|\Psi_i^1\rangle = 1/2$ and $\langle \Psi_i^0|\Psi_i^1\rangle = 0$.

▶ Remark 29. When f is a real-valued signature, the inner product is just the ordinary dot product which can be represented by mating using $=_2$. Thus, f satisfies 1st-Orth iff there is some real $\mu \neq 0$ such that for all indices i, $M(\mathfrak{m}_i f) = \mu I_2$. On the other hand, when $\widehat{f}$ is a signature with ARS, by equation (3.2), the complex inner product can also be represented by mating using $\neq_2$. Thus, $\widehat{f}$ satisfies 1st-Orth iff there is some real $\mu \neq 0$ such that for all i, $M(\widehat{\mathfrak{m}}_i \widehat{f}) = \mu N_2$. Moreover, f satisfies 1st-Orth iff $\widehat{f}$ satisfies it. Although 1st-Orth is well-defined for any complex-valued signature, the properties of $\mathfrak{m}_i f$ and $\widehat{\mathfrak{m}}_i \widehat{f}$ crucially depend on f being real (equivalently $\widehat{f}$ satisfying ARS).

Back to the proof of the #P-hardness of Holant$(\Delta_0, \mathcal{F})$. Since $\mathcal{F}$ does not satisfy condition (T), $\mathcal{F} \not\subseteq \mathscr{T}$. Hence, there is a signature $f \in \mathcal{F}$ of arity $n \geqslant 3$ such that $f \notin \mathscr{T}$. In other words, $\mathcal{F}$ contains an n-qubit state exhibiting multipartite entanglement. We will prove #P-hardness by induction on n. We first consider the base case that $n = 3$. We show that an irreducible ternary signature (a genuinely entangled 3-qubit state) satisfying first order orthogonality has some special forms, from which one can realize $=_3$ or $=_4$ after some holographic transformations. Then, we can reduce the problem from #CSP$(\mathcal{F})$, or

#CSP$_2(\mathcal{F})$, or Holant$(\neq_2|=_3, \widehat{\mathcal{F}})$, to Holant$(\Delta_0, \mathcal{F})$. This allows us to finish the proof by invoking existing dichotomy results for #CSP$(\mathcal{F})$, or #CSP$_2(\mathcal{F})$, or the #P-hardness result we showed above for Holant$(\neq_2|=_k, \widehat{\mathcal{F}})$ where $k \geqslant 3$.

▶ **Lemma 30.** *Let $\mathcal{F}$ be a set of real-valued signatures containing a ternary signature $f \notin \mathfrak{T}$. If $\mathcal{F}$ does not satisfy condition* (T)*, then* Holant$(\Delta_0, \mathcal{F})$ *is #P-hard.*

Then, we consider the inductive step. The general strategy is that we start with a signature $f \in \mathcal{F}$ of arity $n \geqslant 4$ that is not in $\mathfrak{T}$, and realize a signature g of arity $n-1$ or $n-2$ also not in $\mathfrak{T}$, by pinning or merging. (By the definition of $\mathfrak{T}$, when $n = 4$, this g must have arity 3.) By a sequence of reductions (that is constant in length independent of the problem instance size), we can realize a signature h of arity 3 that is not in $\mathfrak{T}$. Then we are done. In other words, given an n-qubit state with multipartite entanglement, we want to show that multipartite entanglement is preserved under projections onto $|0\rangle$ and self-loops by $|\phi^+\rangle$. Lemma 9 says that the preservation holds, or $|1\rangle$ or $|\mathrm{GHZ}_4\rangle$ is realizable. We give an inductive restatement of Lemma 9 in the Holant framework.

▶ **Lemma 31.** *Let $f \in \mathcal{F}$ be a signature of arity $n \geqslant 4$ and $f \notin \mathfrak{T}$. Then one of the following alternatives must hold:*

- Δ_1 *is realizable:* Holant$(\Delta_0, \Delta_1, \mathcal{F}) \leqslant_T$ Holant$(\Delta_0, \mathcal{F})$*, or*
- $=_4$ *is realizable:* Holant$(=_4, \mathcal{F}) \leqslant_T$ Holant$(\Delta_0, \mathcal{F})$*, or*
- *a signature $g \notin \mathfrak{T}$ of arity $n-1$ or $n-2$ is realizable:* Holant$(\Delta_0, g, \mathcal{F}) \leqslant_T$ Holant$(\Delta_0, \mathcal{F})$.

Proof Sketch. For all indices i and all pairs of indices $\{j, k\}$, consider f_i^0 and $\partial_{jk} f$. If there exists i or $\{j, k\}$ such that f_i^0 or $\partial_{jk} f \notin \mathfrak{T}$, then we can realize $g = f_i^0$ or $\partial_{jk} f$ which has arity $n-1$ or $n-2$, and we are done. Otherwise, f_i^0 and $\partial_{jk} f \in \mathfrak{T}$ for all i and all $\{j, k\}$. Under this assumption, our goal is to show that we can realize Δ_1, or there is a unary signature $a(x_u)$ or a binary signature $b(x_v, x_w)$ such that $a(x_u) \mid f$ or $b(x_v, x_w) \mid f$. Then, we have $f = a(x_u) \otimes g$ or $f = b(x_v, x_w) \otimes g$ for some g of arity $n-1$ or $n-2$. We know g can be realized from f by factorization. By the definition of $\mathfrak{T}$, we have $g \notin \mathfrak{T}$ since $f \notin \mathfrak{T}$, and we are done. When $n \geqslant 5$, the above induction proof can be achieved by the interplay of the unique prime factorization, and the commutivity of f_i^0 (pinning) and $\partial_{jk} f$ (merging) gadgets on disjoint indices. For $n = 4$, there is the additional case that $=_4$ can be realized. Thus for $n = 4$, it requires more work; we need to combine the induction proof and first order orthogonality to handle it. ◀

Remember that Holant$(\Delta_0, \Delta_1, \mathcal{F})$ is just Holant$^c(\mathcal{F})$ and #CSP$_2(\mathcal{F}) \leqslant_T$ Holant$(=_4, \mathcal{F})$. By Theorem 22, #CSP$_2(\mathcal{F})$ and Holant$^c(\mathcal{F})$ are both #P-hard when $\mathcal{F}$ does not satisfy condition (T). Combining with Lemmas 30 and 31, we have the following result.

▶ **Lemma 32.** *Let $\mathcal{F}$ be a set of real-valued signatures. If $\mathcal{F}$ does not satisfy condition* (T)*, then* Holant$(\Delta_0, \mathcal{F})$ *is #P-hard.*

Finally, combining Theorem 21 and Lemmas 24, 27 and 32, we finished the proof of Theorem 10.

References

1 Miriam Backens. The inductive entanglement classification yields ten rather than eight classes of four-qubit entangled states. *Physical Review A*, 97:022329, 2017.

2 Miriam Backens. A new holant dichotomy inspired by quantum computation. In *Proceedings of the 44th International Colloquium on Automata, Languages, and Programming*, pages 16:1–16:14, 2017.

3 Miriam Backens. A complete dichotomy for complex-valued holant^c. In *Proceedings of the 45th International Colloquium on Automata, Languages, and Programming*, pages 12:1–12:14, 2018.

4 Rodney J Baxter. Eight-vertex model in lattice statistics. *Physical Review Letters*, 26(14):832, 1971.

5 Rodney J Baxter. The six and eight-vertex models revisited. *Journal of statistical physics*, 116(1-4):43–66, 2004.

6 Charles H Bennett, Gilles Brassard, Claude Crépeau, Richard Jozsa, Asher Peres, and William K Wootters. Teleporting an unknown quantum state via dual classical and einstein-podolsky-rosen channels. *Physical review letters*, 70(13):1895, 1993.

7 Andrei Bulatov, Martin Dyer, Leslie Ann Goldberg, Markus Jalsenius, Mark Jerrum, and David Richerby. The complexity of weighted and unweighted #csp. *Journal of Computer and System Sciences*, 78(2):681–688, 2012.

8 Andrei Bulatov and Martin Grohe. The complexity of partition functions. *Theoretical Computer Science*, 348(2-3):148–186, 2005.

9 Andrei A Bulatov. The complexity of the counting constraint satisfaction problem. *Journal of the ACM (JACM)*, 60(5):1–41, 2013.

10 Jin-Yi Cai and Xi Chen. *Complexity Dichotomies for Counting Problems: Volume 1, Boolean Domain*. Cambridge University Press, 2017.

11 Jin-Yi Cai and Xi Chen. Complexity of counting csp with complex weights. *Journal of the ACM (JACM)*, 64(3):1–39, 2017.

12 Jin-Yi Cai, Xi Chen, and Pinyan Lu. Graph homomorphisms with complex values: A dichotomy theorem. *SIAM Journal on Computing*, 42(3):924–1029, 2013.

13 Jin-Yi Cai, Xi Chen, and Pinyan Lu. Nonnegative weighted# csp: An effective complexity dichotomy. *SIAM Journal on Computing*, 45(6):2177–2198, 2016.

14 Jin-Yi Cai and Zhiguo Fu. Complexity classification of the eight-vertex model. *arXiv preprint*, 2017. `arXiv:1702.07938`.

15 Jin-Yi Cai, Zhiguo Fu, and Shuai Shao. Complexity of counting weighted eulerian orientations with ars. *arXiv preprint*, 2019. `arXiv:1904.02362`.

16 Jin-Yi Cai, Zhiguo Fu, and Mingji Xia. Complexity classification of the six-vertex model. *Information and Computation*, 259:130–141, 2018.

17 Jin-Yi Cai and Artem Govorov. Perfect matchings, rank of connection tensors and graph homomorphisms. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 476–495. SIAM, 2019.

18 Jin-Yi Cai, Heng Guo, and Tyson Williams. A complete dichotomy rises from the capture of vanishing signatures. *SIAM Journal on Computing*, 45(5):1671–1728, 2016.

19 Jin-Yi Cai, Heng Guo, and Tyson Williams. Clifford gates in the holant framework. *Theoretical Computer Science*, 745:163–171, 2018.

20 Jin-Yi Cai, Pinyan Lu, and Mingji Xia. Dichotomy for holant* problems of boolean domain. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, pages 1714–1728. SIAM, 2011.

21 Jin-Yi Cai, Pinyan Lu, and Mingji Xia. The complexity of complex weighted boolean #csp. *Journal of Computer and System Sciences*, 80(1):217–236, 2014.

22 Jin-Yi Cai, Pinyan Lu, and Mingji Xia. Dichotomy for real holant^c problems. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1802–1821. SIAM, 2018.

23 Jeroen Dehaene and Bart De Moor. Clifford group, stabilizer states, and linear and quadratic operations over gf (2). *Physical Review A*, 68(4):042318, 2003.

24 Wolfgang Dür, Guifre Vidal, and J Ignacio Cirac. Three qubits can be entangled in two inequivalent ways. *Physical Review A*, 62(6):062314, 2000.

25 Martin Dyer and Catherine Greenhill. The complexity of counting graph homomorphisms. *Random Structures & Algorithms*, 17(3-4):260–289, 2000.

26 Martin Dyer and David Richerby. An effective dichotomy for the counting constraint satisfaction problem. *SIAM Journal on Computing*, 42(3):1245–1274, 2013.

27 Artur K Ekert. Quantum cryptography based on bell's theorem. *Physical review letters*, 67(6):661, 1991.

28 Michael Freedman, László Lovász, and Alexander Schrijver. Reflection positivity, rank connectivity, and homomorphism of graphs. *Journal of the American Mathematical Society*, 20(1):37–51, 2007.

29 Mariami Gachechiladze and Otfried Gühne. Completing the proof of "generic quantum nonlocality". *Physics Letters A*, 381(15):1281–1285, 2017.

30 Masoud Gharahi Ghahi and Stefano Mancini. Comment on "inductive entanglement classification of four qubits under stochastic local operations and classical communication". *Physical Review A*, 98(6):066301, 2018.

31 Leslie Ann Goldberg, Martin Grohe, Mark Jerrum, and Marc Thurley. A complexity dichotomy for partition functions with mixed signs. *SIAM Journal on Computing*, 39(7):3336–3402, 2010.

32 Daniel Gottesman. The heisenberg representation of quantum computers, talk at. In *International Conference on Group Theoretic Methods in Physics*. Citeseer, 1998.

33 Ryszard Horodecki, Paweł Horodecki, Michał Horodecki, and Karol Horodecki. Quantum entanglement. *Reviews of modern physics*, 81(2):865, 2009.

34 Richard Jozsa and Noah Linden. On the role of entanglement in quantum-computational speed-up. *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 459(2036):2011–2032, 2003.

35 Lucas Lamata, Juan León, D Salgado, and E Solano. Inductive entanglement classification of four qubits under stochastic local operations and classical communication. *Physical Review A*, 75(2):022318, 2007.

36 Lucas Lamata, Juan León, D Salgado, and Enrique Solano. Inductive classification of multipartite entanglement under stochastic local operations and classical communication. *Physical Review A*, 74(5):052336, 2006.

37 Jiabao Lin and Hanpin Wang. The complexity of boolean holant problems with nonnegative weights. *SIAM Journal on Computing*, 47(3):798–828, 2018.

38 Milena Mihail and Peter Winkler. On the number of eulerian orientations of a graph. *Algorithmica*, 16(4-5):402–414, 1996.

39 Akimasa Miyake and Frank Verstraete. Multipartite entanglement in 2×2×n quantum systems. *Physical Review A*, 69(1):012101, 2004.

40 Michael A Nielsen and Isaac Chuang. Quantum computation and quantum information, 2002.

41 Linus Pauling. The structure and entropy of ice and of other crystals with some randomness of atomic arrangement. *Journal of the American Chemical Society*, 57(12):2680–2684, 1935.

42 Sandu Popescu and Daniel Rohrlich. Generic quantum nonlocality. *Physics Letters A*, 166(5-6):293–297, 1992.

43 Leslie G Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.

44 Leslie G Valiant. Quantum circuits that can be simulated classically in polynomial time. *SIAM Journal on Computing*, 31(4):1229–1254, 2002.

45 Leslie G Valiant. Holographic algorithms. *SIAM Journal on Computing*, 37(5):1565–1594, 2008.

46 Frank Verstraete, Jeroen Dehaene, Bart De Moor, and Henri Verschelde. Four qubits can be entangled in nine different ways. *Physical Review A*, 65(5):052112, 2002.

Counting Perfect Matchings and the Eight-Vertex Model

Jin-Yi Cai
University of Wisconsin-Madison, Madison, WI, USA
http://pages.cs.wisc.edu/~jyc/
jyc@cs.wisc.edu

Tianyu Liu
University of Wisconsin-Madison, Madison, WI, USA
http://pages.cs.wisc.edu/~tl/
tl@cs.wisc.edu

Abstract

We study the approximation complexity of the partition function of the eight-vertex model on general 4-regular graphs. For the first time, we relate the approximability of the eight-vertex model to the complexity of approximately counting perfect matchings, a central open problem in this field. Our results extend those in [8].

In a region of the parameter space where no previous approximation complexity was known, we show that approximating the partition function is at least as hard as approximately counting perfect matchings via approximation-preserving reductions. In another region of the parameter space which is larger than the region that is previously known to admit Fully Polynomial Randomized Approximation Scheme (FPRAS), we show that computing the partition function can be reduced to counting perfect matchings (which is valid for both exact and approximate counting). Moreover, we give a complete characterization of nonnegatively weighted (not necessarily planar) 4-ary matchgates, which has been open for several years. The key ingredient of our proof is a *geometric* lemma.

We also identify a region of the parameter space where approximating the partition function on *planar* 4-regular graphs is feasible but on *general* 4-regular graphs is equivalent to approximately counting perfect matchings. To our best knowledge, these are the first problems that exhibit this dichotomic behavior between the planar and the nonplanar settings in approximate counting.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness

Keywords and phrases Approximate complexity, the eight-vertex model, counting perfect matchings

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.23

Category Track A: Algorithms, Complexity and Games

Related Version A full version of the paper is available at https://arxiv.org/abs/1904.10493.

Funding Supported by NSF CCF-1714275.

1 Introduction

The eight-vertex model is defined over 4-regular graphs, the states of which are the set of *even orientations*, i.e. those with an even number of arrows into (and out of) each vertex. There are eight permitted types of local configurations around a vertex – hence the name eight-vertex model (see Figure 1).

Classically, the eight-vertex model is defined by statistical physicists on a square lattice region where each vertex of the lattice is connected by an edge to four nearest neighbors. In general, the eight configurations 1 to 8 in Figure 1 are associated with eight possible weights $w_1, \ldots, w_8$. By physical considerations, the total weight of a state remains unchanged if all arrows are flipped, assuming there is no external electric field. In this case we write $w_1 = w_2 = a$, $w_3 = w_4 = b$, $w_5 = w_6 = c$, and $w_7 = w_8 = d$. This complementary invariance is known as *arrow reversal symmetry* or *zero field assumption*.

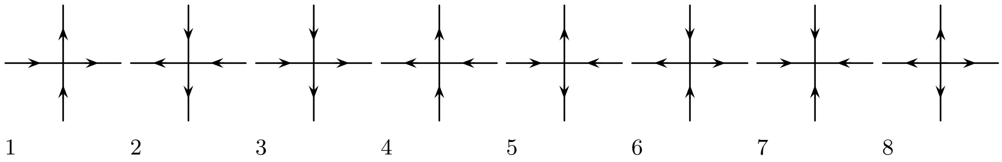


Figure 1 Valid configurations of the eight-vertex model.

Even in the zero-field setting, this model is already enormously expressive: the special case when $d = 0$ is the zero-field six-vertex model which has sub-models such as the ice ($a = b = c$), KDP, and Rys F models; on the square lattice, some other important models such as the dimer and zero-field Ising models can be reduced to the eight-vertex model [2]. After it was introduced in 1970 by Sutherland [19] and by Fan and Wu [9], Baxter [1, 2] achieved a good understanding of the zero-field case in the thermodynamic limit on the square lattice (in physics it is called an "exactly solved model").

In this paper, we assume the arrow reversal symmetry and further assume that $a, b, c, d \geq 0$, as is the case in *classical* physics. Given a 4-regular graph G, we label four incident edges of each vertex from 1 to 4. The *partition function* of the eight-vertex model with parameters (a, b, c, d) on G is defined as

$$Z_{\mathrm{EV}}(G; a, b, c, d) = \sum_{\tau \in \mathcal{O}_{\mathbf{e}}(G)} a^{n_1+n_2} b^{n_3+n_4} c^{n_5+n_6} d^{n_7+n_8}, \tag{1}$$

where $\mathcal{O}_{\mathbf{e}}(G)$ is the set of all even orientations of G, and n_i is the number of vertices in type i in G ($1 \leq i \leq 8$, locally depicted as in Figure 1 where the 4 edges are oriented counterclockwise starting from the edge on the left) under an even orientation $\tau \in \mathcal{O}_{\mathbf{e}}(G)$.

In terms of the exact computational complexity, a complexity dichotomy is given for the eight-vertex model on 4-regular graphs for all eight parameters [6]. This is studied in the context of a classification program for the complexity of counting problems [5], where the eight-vertex model serves as important basic cases for Holant problems defined by not necessarily symmetric constraint functions. It is shown that every setting is either P-time computable (and some are surprising) or #P-hard. However, most cases for P-time tractability are due to nontrivial cancellations. In our setting where a, b, c, d are nonnegative, the problem of computing the partition function of the eight-vertex model exactly is #P-hard unless: (1) $a = b = c = d$ (this is equivalent to the unweighted case); (2) at least three of a, b, c, d are zero; or (3) two of a, b, c, d are zero and the other two are equal. In addition, on planar graphs it is also P-time computable for parameter settings (a, b, c, d) with $a^2 + b^2 = c^2 + d^2$, using the *FKT algorithm*.

Since exact computation is hard in most cases, one natural question is what is the approximate complexity of counting and sampling of the eight-vertex model. To our best knowledge, prior to [8], there is only one previous result in this regard due to Greenberg and Randall [12]. They showed that on square lattice regions a specific Markov chain (which flips the orientations of all four edges along a uniformly picked face at each step) is torpidly mixing when d is large. This means that when sinks and sources have large weights, this particular chain *cannot* be used to approximately count or sample eight-vertex configurations on the square lattice according to the Gibbs measure. Recently, similar torpid mixing results have been achieved for the six-vertex model on the square lattice [17].

The paper [8] gave the first classification results for the approximate complexity of the eight-vertex model on general and planar 4-regular graphs, and they conform to phase transition in physics. In order to state the results, we adopt the following notations assuming $a, b, c, d \geq 0$.

- $\mathcal{X} = \{\ (a,b,c,d) \mid a \le b+c+d,\ \ b \le a+c+d,\ \ c \le b+c+d,\ \ d \le a+b+c\}$;
- $\mathcal{Y} = \{\ (a,b,c,d) \mid a+d \le b+c,\ \ b+d \le a+c,\ \ c+d \le a+b\}$;
- $\mathcal{Z} = \{\ (a,b,c,d) \mid a^2 \le b^2+c^2+d^2,\ \ b^2 \le a^2+c^2+d^2,\ \ c^2 \le a^2+b^2+d^2,\ \ d^2 \le a^2+b^2+c^2\}$.

▶ Remark 1. $\mathcal{Y} \subset \mathcal{X}$. $\mathcal{Z} \subset \mathcal{X}$.

Physicists have shown an *order-disorder phase transition* for the eight-vertex model on the square lattice between parameter settings outside $\mathcal{X}$ and those inside $\mathcal{X}$ (see Baxter's book [3] for more details). In [8], it was shown that: (1) approximating the partition function of the eight-vertex model on general 4-regular graphs outside $\mathcal{X}$ is NP-hard, (2) there is an FPRAS[1] for general 4-regular graphs in the region $\mathcal{Y} \bigcap \mathcal{Z}$, and (3) there is an FPRAS for planar 4-regular graphs in the extra region $\{(a,b,c,d) \mid a+d \le b+c,\ \ b+d \le a+c,\ \ c+d \ge a+b\} \bigcap \mathcal{Z}$.

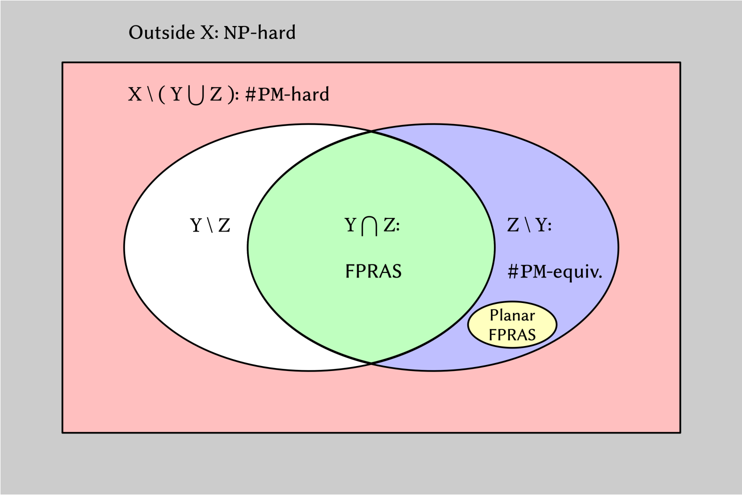


Figure 2 A Venn diagram of the approximation complexity of the eight-vertex model.

In this paper we make further progress in the classification of the approximate complexity of the eight-vertex model on 4-regular graphs in terms of the parameters (see Figure 2). For the first time, the complexity of approximating the partition function of the eight-vertex model ($\#\mathrm{EV}(a,b,c,d)$) is related to that of approximately counting perfect matchings ($\#\mathrm{PM}$).

▶ **Theorem 2.** *For any four positive numbers $a,b,c,d > 0$ such that $(a,b,c,d) \notin \mathcal{Y}$, the problem $\#\mathrm{EV}(a,b,c,d)$ is at least as hard to approximate as counting perfect matchings:*

$$\#\mathrm{PM} \le_{\mathrm{AP}} \#\mathrm{EV}(a,b,c,d).$$

▶ Remark 3. The theorem is stated for the case where all four parameters are positive. The same proof also works for the case when there is exactly one zero among the nonnegative values $\{a,b,c\}$. A complete account for four nonnegative values $\{a,b,c,d\}$ is given in the Table 1. There is a symmetry among a,b,c for the eight-vertex model on general (not necessarily planar) 4-regular graphs, so for simplicity in this table we assume $a \le b \le c$.

[1] Suppose $f : \Sigma^* \to \mathbb{R}$ is a function mapping problem instances to real numbers. A *fully polynomial randomized approximation scheme (FPRAS)* [14] for a problem is a randomized algorithm that takes as input an instance x and $\varepsilon > 0$, running in time polynomial in n (the input length) and ε^{-1}, and outputs a number Y (a random variable) such that $\Pr\left[(1-\varepsilon)f(x) \le Y \le (1+\varepsilon)f(x)\right] \ge \frac{3}{4}$.

Table 1 Approximation complexity of the eight-vertex model with $(a, b, c, d) \notin d\text{-}\mathtt{SUM}^2$.

	$d = 0$	$d > 0$
$a = b = c = 0$	P-time computable (trivial)	P-time computable (trivial)
$a = b = 0, c > 0$	P-time computable (trivial)	$c = d$: P-time computable [6] $c \neq d$: NP-hard [8]
$a = 0, b, c > 0$	NP-hard [8]	#PM-hard (in this paper)
$a, b, c > 0$	NP-hard [7]	#PM-hard (in this paper)

The proof of Theorem 2 is in Section 3. Our proof for the hardness result has several ingredients:

1. We express the eight-vertex model on a 4-regular graph G as an edge-2-coloring problem on G using Valiant's *holographic transformation* [22].
2. We show that a modified version of the edge-2-coloring problem on G is equivalent to the zero-field Ising model on its *crossing-circuit graph* denoted by $\widetilde{G}$. Thus known #PM-equivalence result for the Ising model [11, Lemma 7] directly transfers to the modified version of the edge-2-coloring problem under certain parameter settings.
3. We further show that for any parameter setting outside $\mathcal{Y}$, approximating the partition function of the eight-vertex model is at least as hard as the #PM-equivalent modified edge-2-coloring problem via approximation-preserving reductions.

▶ **Theorem 4.** *For any* $(a, b, c, d) \in \mathcal{Z}$*,*

$$\#\mathrm{EV}(a, b, c, d) \leq_{\mathrm{AP}} \#\mathrm{PM}.$$

The proof of Theorem 4 is in Section 4. To prove the #PM-easiness result, we again express the eight-vertex model in the Holant framework (see Section 2) and show that the constraint functions of the eight-vertex model in $\mathcal{Z}$ can be implemented by constant-size *matchgates* with nonnegatively weighted edges (Definition 13). We note that allowing nonnegative edge-weights does not add more computational power to the unweighted #PM [18, Proposition 5]. The crucial ingredient of our proof is a *geometric* lemma (Lemma 18) in 3-dimensional space.

This matchgate expressibility is tight: no constraint functions of the eight-vertex model with parameter settings outside the region $\mathcal{Z}$ can be implemented by a matchgate (Lemma 19). Moreover, the general version of our result also works for the eight-vertex model without the arrow reversal symmetry. It is open if approximately computing the partition function in $\mathcal{X} \setminus (\mathcal{Y} \bigcup \mathcal{Z})$ is #PM-equivalent or not.

As part of this work, we give a complete characterization of the constraint functions that can be expressed by 4-ary matchgates in Theorem 15. This solves an important question that has been open for several years [18, 4]. We believe it is of independent interest.

▶ **Corollary 5.** *For any four positive numbers* $a, b, c, d > 0$ *such that* $(a, b, c, d) \in \mathcal{Z} \setminus \mathcal{Y}$*,*

$$\#\mathrm{EV}(a, b, c, d) \equiv_{\mathrm{AP}} \#\mathrm{PM}.$$

Note that for the eight-vertex model in the region[3] $\{(a,b,c,d) \mid a+d \le b+c, \;\; b+d \le a+c, \;\; c+d > a+b\} \bigcap \mathcal{Z}$, computing $Z_{\mathrm{EV}}(a,b,c,d)$ is (1) #P-complete in exact computation [6], (2) #PM-equivalent in approximate computation on general 4-regular graphs (Corollary 5), and (3) admits an FPRAS in approximate computation on planar 4-regular graphs [8]. To our best knowledge, these are the first identified problems having these three properties. Previously the combined results of [10] and [13] proved that counting k-colorings for certain range of parameters is FPRASable on general graphs but NP-hard in approximate complexity. The complexity result in this paper is different because the complexity we described in item (2) above is #PM-equivalent (neither harder nor easier). We note that the complexity status of #PM is a long standing open problem in the field (neither known to be FPRASable nor known to be NP-hard to approximate). These results contrast with, in the setting of approximately counting, the FKT algorithm for exact counting which shows that the #P-hard problem #PM can be computed in polynomial time on planar graphs.

2 Preliminaries

Given a 4-regular graph $G=(V,E)$, the *edge-vertex incidence graph* $G'=(U_E,U_V,E')$ is a bipartite graph where $(u_e,u_v) \in U_E \times U_V$ is an edge in E' iff $e \in E$ in G is incident to $v \in V$. We model an orientation $(w \to v)$ on an edge $e=\{w,v\} \in E$ from w into v in G by assigning 1 to $(u_e,u_w) \in E'$ and 0 to $(u_e,u_v) \in E'$ in G'. A configuration of the eight-vertex model on G is an *edge-2-coloring* on G', namely $\sigma : E' \to \{0,1\}$, where for each $u_e \in U_E$ its two incident edges are assigned 01 or 10, and for each $u_v \in U_V$ the sum of values $\sum_{i=1}^{4} \sigma(e_i) \equiv 0 \pmod 2$, over the four incident edges of u_v. Thus we model the even orientation rule of G on all $v \in V$ by requiring “two-0-two-1/four-0/four-1” locally at each vertex $u_v \in U_V$.

The “one-0-one-1” requirement on the two edges incident to a vertex in U_E is a binary Disequality constraint, denoted by $(\neq_2)$. The values of a 4-ary *constraint function*, or a *signature* f can be listed in a matrix $M(f) = \begin{bmatrix} f_{0000} & f_{0010} & f_{0001} & f_{0011} \\ f_{0100} & f_{0110} & f_{0101} & f_{0111} \\ f_{1000} & f_{1010} & f_{1001} & f_{1011} \\ f_{1100} & f_{1110} & f_{1101} & f_{1111} \end{bmatrix}$, called the *constraint matrix* of f. For the eight-vertex model satisfying the even orientation rule and arrow reversal symmetry, the signature f at every vertex $v \in U_V$ in G' has the form $M(f) = \begin{bmatrix} d & 0 & 0 & a \\ 0 & b & c & 0 \\ 0 & c & b & 0 \\ a & 0 & 0 & d \end{bmatrix}$, if we draw a vertex with incident edges labeled 1, 2, 3, and 4 locally as the left, down, right, and up edges respectively according to Figure 1. Thus computing the partition function $Z_{\mathrm{EV}}(G;a,b,c,d)$ is equivalent to evaluating

$$Z'(G';f) := \sum_{\sigma : E' \to \{0,1\}} \prod_{u \in U_E} (\neq_2)\left(\sigma|_{E'(u)}\right) \prod_{u \in U_V} f\left(\sigma|_{E'(u)}\right).$$

where $E'(u)$ denotes the incident edges of $u \in U_E \cup U_V$. In fact, in this way we express the partition function of the eight-vertex model as the Holant sum in the framework for Holant problems:

$$Z_{\mathrm{EV}}(G;a,b,c,d) = \mathrm{Holant}\left(G'; \neq_2 \mid f\right)$$

where we use $\mathrm{Holant}(H; g \mid f)$ to denote the sum $\sum_{\sigma : E \to \{0,1\}} \prod_{u \in U} g\left(\sigma|_{E(u)}\right) \prod_{u \in V} f\left(\sigma|_{E(u)}\right)$ on a bipartite graph $H=(U,V,E)$. Each vertex in U (or V) is assigned the signature g (or f, respectively). The signature g is considered as a row vector (or covariant tensor),

[3] This region is the intersection of $\overline{\mathcal{Y}}$ and the extra region $\{(a,b,c,d) \mid a+d \le b+c, \;\; b+d \le a+c, \;\; c+d \ge a+b\}$ where there is an FPRAS for planar graphs. Note that the strict inequality $c+d > a+b$ is needed.

whereas the signature f is considered as a column vector (or contravariant tensor). (See [5] for more on Holant problems.) The following proposition says that an invertible holographic transformation does not change the complexity of the Holant problem in the bipartite setting.

▶ **Proposition 6** ([22]). *Suppose $T \in \mathbb{C}^2$ is an invertible matrix. Let $d_1 = \operatorname{arity}(g)$ and $d_2 = \operatorname{arity}(f)$. Define $g' = g\left(T^{-1}\right)^{\otimes d_1}$ and $f' = T^{\otimes d_2} f$. Then for any bipartite graph H, $\operatorname{Holant}(H; g \mid f) = \operatorname{Holant}(H; g' \mid f')$.*

We denote $\operatorname{Holant}(G; f) = \operatorname{Holant}(G'; =_2 \mid f)$ and use $\operatorname{Holant}(f)$ to denote the problem whose input is a graph G and output is $\operatorname{Holant}(G; f)$; this is equivalent to the usual definition.

3 #PerfectMatchings-hardness

Our proof strategy for Theorem 2 is as follows. In Lemma 7, we express the eight-vertex model on a 4-regular graph G as a Holant problem; this is an equivalent form of the orientation problem expressed as an edge-2-coloring problem on G, and is achieved using a holographic transformation. In Lemma 8, we give an approximation-preserving reduction to show that this edge-2-coloring problem is at least as hard as a modified version of the edge-2-coloring problem where weights at some input originally in the support are dropped off. In Lemma 9, we establish the equivalence between this modified version of the edge-2-coloring problem and the zero-field Ising model. Thus a known result for the Ising model (Proposition 11) indicates the #PM-equivalence of this modified version of the edge-2-coloring problem under certain parameter settings (Corollary 12). It can be deduced from Lemma 7, Lemma 8, and Corollary 12 that for any (a, b, c, d) with $a + d > b + c$ (and symmetrically $b + d > a + c$ or $c + d > a + b$), approximately computing the partition function is at least as hard as the #PM-equivalent modified edge-2-coloring problem under approximation-preserving reductions.

▶ **Lemma 7.**

$$2^{|V(G)|} \cdot Z_{\mathrm{EV}}(G; a, b, c, d) = \operatorname{Holant}\left(G; \begin{bmatrix} a{+}b{+}c{+}d & 0 & 0 & -a{+}b{+}c{-}d \\ 0 & a{-}b{+}c{-}d & a{+}b{-}c{-}d & 0 \\ 0 & a{+}b{-}c{-}d & a{-}b{+}c{-}d & 0 \\ -a{+}b{+}c{-}d & 0 & 0 & a{+}b{+}c{+}d \end{bmatrix}\right).$$

Proof. Using the binary disequality function ($\neq_2$) for the orientation of any edge, we can express the partition function of the eight-vertex model G as a Holant problem on its edge-vertex incidence graph G',

$$Z_{\mathrm{EV}}(G; a, b, c, d) = \operatorname{Holant}\left(G'; \neq_2 \mid f\right),$$

where f is the 4-ary signature with $M(f) = \left[\begin{smallmatrix} d & 0 & 0 & a \\ 0 & b & c & 0 \\ 0 & c & b & 0 \\ a & 0 & 0 & d \end{smallmatrix}\right]$. Note that, writing the truth table of $(\neq_2) = (0, 1, 1, 0)$ as a vector and multiplied by a tensor power of the matrix Z^{-1}, where $Z = \frac{1}{\sqrt{2}}\left[\begin{smallmatrix} 1 & 1 \\ i & -i \end{smallmatrix}\right]$ we get $(\neq_2)(Z^{-1})^{\otimes 2} = (1, 0, 0, 1)$, which is exactly the truth table of the binary equality function ($=_2$). Then according to Proposition 6, by the Z-transformation, we get

$$\begin{aligned} \operatorname{Holant}\left(G'; \neq_2 \mid f\right) &= \operatorname{Holant}\left(G'; \neq_2 \cdot \left(Z^{-1}\right)^{\otimes 2} \mid Z^{\otimes 4} \cdot f\right) \\ &= \operatorname{Holant}\left(G'; =_2 \mid Z^{\otimes 4} f\right) \\ &= \operatorname{Holant}\left(G;\ Z^{\otimes 4} f\right), \end{aligned}$$

and a direct calculation shows that $M(Z^{\otimes 4} f) = \frac{1}{2}\begin{bmatrix} a{+}b{+}c{+}d & 0 & 0 & -a{+}b{+}c{-}d \\ 0 & a{-}b{+}c{-}d & a{+}b{-}c{-}d & 0 \\ 0 & a{+}b{-}c{-}d & a{-}b{+}c{-}d & 0 \\ -a{+}b{+}c{-}d & 0 & 0 & a{+}b{+}c{+}d \end{bmatrix}$. ◀

▶ **Lemma 8.** *Suppose $d > 0$ and at most one of a, b, c is zero. Then*

$$\operatorname{Holant}\left(\begin{bmatrix} a+b+c+d & 0 & 0 & -a+b+c-d \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -a+b+c-d & 0 & 0 & a+b+c+d \end{bmatrix}\right) \leq_{\mathrm{AP}} \operatorname{Holant}\left(\begin{bmatrix} a+b+c+d & 0 & 0 & -a+b+c-d \\ 0 & a-b+c-d & a+b-c-d & 0 \\ 0 & a+b-c-d & a-b+c-d & 0 \\ -a+b+c-d & 0 & 0 & a+b+c+d \end{bmatrix}\right).$$

Proof. This task can be reduced to

$$\operatorname{Holant}\left(\neq_2 \mid \begin{bmatrix} d & 0 & 0 & a \\ 0 & b & c & 0 \\ 0 & c & b & 0 \\ a & 0 & 0 & d \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}\right) \leq_{\mathrm{AP}} \operatorname{Holant}\left(\neq_2 \mid \begin{bmatrix} d & 0 & 0 & a \\ 0 & b & c & 0 \\ 0 & c & b & 0 \\ a & 0 & 0 & d \end{bmatrix}\right) \tag{2}$$

and the analysis can be found in the full version of this paper.

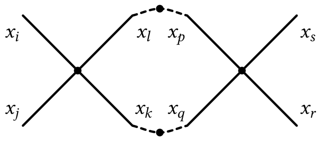


Figure 3 A gadget construction.

Next we show how to get (2). Given the signature f with matrix $\begin{bmatrix} d & 0 & 0 & a \\ 0 & b & c & 0 \\ 0 & c & b & 0 \\ a & 0 & 0 & d \end{bmatrix}$ in #EV(a,b,c,d), we construct a 4-ary signature $\check{f}$ with constraint matrix $\begin{bmatrix} \check{d} & 0 & 0 & \check{a} \\ 0 & \check{b} & \check{c} & 0 \\ 0 & \check{c} & \check{b} & 0 \\ \check{a} & 0 & 0 & \check{d} \end{bmatrix}$ using a polynomial number of vertices and edges such that $\check{a}$, $\check{b}$, $\check{c}$, and $\check{d}$ are all exponentially close to 1 after normalization, i.e., to be 2^{-n^C} close to 1, for any $C > 0$, with a construction of $n^{O(1)}$ size in polynomial time.

We assume we start with the following condition:

$$0 < d \leq a \leq b \leq c. \tag{3}$$

If this is not the case, we can obtain a 4-ary construction that realizes this condition using constantly many vertices. With some preliminary construction we can further assume $1 \leq d \leq a \leq b \leq c \leq \frac{3}{2}d$ initially. (See the full version of this paper for details.) Note that starting with the signature f with matrix $M(f) = \begin{bmatrix} d & 0 & 0 & a \\ 0 & b & c & 0 \\ 0 & c & b & 0 \\ a & 0 & 0 & d \end{bmatrix}$, we can arbitrarily permute a, b, c by relabeling the edges, and so we get signatures f_1 with $M(f_1) = \begin{bmatrix} d & 0 & 0 & b \\ 0 & a & c & 0 \\ 0 & c & a & 0 \\ b & 0 & 0 & d \end{bmatrix}$ and f_2 with $M(f_2) = \begin{bmatrix} d & 0 & 0 & c \\ 0 & a & b & 0 \\ 0 & b & a & 0 \\ c & 0 & 0 & d \end{bmatrix}$. There are two constructions G_1 and G_2 which we use as basic steps; both constructions start with a signature f with parameters satisfying (3).

1. $\boldsymbol{G_1}$: connect two vertices with signatures f_1 and f_2 respectively as in Figure 3. Since we are in the orientation view, we place the signature $(\neq_2)$ on the two degree 2 vertices connecting the two degree 4 vertices. Then the signature g_1 of the construction G_1 is obtained by matrix multiplication $M(g_1) = M_{x_i x_j, x_s x_r}(g_1) = M(f_1) \cdot N \cdot M(f_2)$, where $N = \begin{bmatrix} & & & 1 \\ & & 1 & \\ & 1 & & \\ 1 & & & \end{bmatrix}$. Thus

$$M(g_1) = \begin{bmatrix} (b+c)d & 0 & 0 & bc+d^2 \\ 0 & a(b+c) & a^2+bc & 0 \\ 0 & a^2+bc & a(b+c) & 0 \\ bc+d^2 & 0 & 0 & (b+c)d \end{bmatrix}.$$

The signature g_1 has four new parameters, denoted by

$$(a_1,\ b_1,\ c_1,\ d_1) = (a(b+c),\ bc+d^2,\ a^2+bc,\ (b+c)d).$$

We make the following observations and all of them can be easily verified using (3):

- d_1 is the weight on sink and source and $0 < d_1 \leq a_1, b_1, c_1$.
- $c_1 = \max(a_1, b_1, c_1, d_1)$.
- $\frac{a_1}{d_1} = \frac{a}{d}, \frac{b_1}{d_1} \leq \frac{b}{d}, \frac{c_1}{d_1} \leq \frac{c}{d}$.
- $c_1 d_1 \leq a_1 b_1$ because $c_1 d_1 - a_1 b_1 = -(b+c)(a-d)(bc-ad) \leq 0$.

2. ***G*$_2$**: connect two vertices with signatures f_2 as in Figure 3. Denote the signature of G_2 by g_2. We have

$$M(g_2) = M(f_2) \cdot N \cdot M(f_2) = \begin{bmatrix} 2cd & 0 & 0 & c^2+d^2 \\ 0 & 2ab & a^2+b^2 & 0 \\ 0 & a^2+b^2 & 2ab & 0 \\ c^2+d^2 & 0 & 0 & 2cd \end{bmatrix}.$$

The signature g_2 has four new parameters, denoted by

$$(a_2,\ b_2,\ c_2,\ d_2) = (2ab,\ a^2+b^2,\ c^2+d^2,\ 2cd).$$

The following observations can also be easily verified using (3):

- d_2 is the weight on sink and source and if $cd \leq ab$, then $0 < d_2 \leq a_2, b_2, c_2$.
- $\frac{a_2}{d_2} \leq \frac{a}{d}, \frac{b_2}{d_2} \leq \frac{b}{d}, \frac{c_2}{d_2} \leq \frac{c}{d}$.
- $\frac{c_2 - d_2}{d_2} = \frac{(c-d)^2}{2cd} \leq \frac{1}{2}\left(\frac{c-d}{d}\right)^2 \leq \left(\frac{c-d}{d}\right)^2$.

Based on the two basic constructions above, we construct the signature $\check{f}$ in logarithmically many rounds recursively, each of the $O(\log n)$ rounds uses the signature constructed in the previous round. We now describe a single round in this construction, which consists of two steps. In step 1 we use a signature with some parameter setting (a, b, c, d) satisfying (3) and apply G_1 to two copies of the signature. If the resulting parameter $b_1 < a_1$ we switch the roles of a_1 and b_1, and obtain $(a'_1, b'_1, c'_1, d'_1) = (b_1, a_1, c_1, d_1)$, again satisfying (3), as well as $c_1 d_1 \leq a_1 b_1$. In step 2, we apply G_2 to two copies of the signature constructed in step 1 (with the switching of the roles of a_1 and b_1 if it is needed). Denote the parameters of the resulting signature by (a^*, b^*, c^*, d^*). Altogether each round uses four copies of the signature from the previous round, starting with the initial given signature. Therefore in polynomial time we can afford to carry out $C \log n$ rounds for any constant C. Note that, if we consider the normalized quantities $(\frac{a}{d}, \frac{b}{d}, \frac{c}{d}, \frac{d}{d})$, then the respective quantities in each step G_1 and G_2 do not increase their distances to 1, i.e.,

$$0 \leq \frac{a^*}{d^*} - 1 \leq \frac{a}{d} - 1, \quad 0 \leq \frac{b^*}{d^*} - 1 \leq \frac{b}{d} - 1, \quad 0 \leq \frac{c^*}{d^*} - 1 \leq \frac{c}{d} - 1.$$

This is true even if the G_2 construction in step 2 is applied in the case when the roles of a_1 and b_1 are switched for the signature from step 1, when that switch is required $(b_1 < a_1)$ as described. More importantly, based on the properties of G_1 and G_2, we know that the (normalized) gap between d and the previous largest entry c shrinks quadratically fast, as measured by the new c^* normalized with d^*. More precisely,

$$0 \leq \frac{c^*}{d^*} - 1 \leq \left(\frac{c}{d} - 1\right)^2.$$

Note that c^* may no longer be the largest among a^*, b^*, c^*; however we will permute them to get $\widetilde{a}, \widetilde{b}, \widetilde{c}$ so that (3) is still satisfied before proceeding to the next round. This completes the description of our construction in one round which obtains $(\widetilde{a}, \widetilde{b}, \widetilde{c}, \widetilde{d})$ from (a, b, c, d).

We will construct the final signature $\check{f}$ by $O(\log n)$ rounds of this construction. Also we will follow each value a, b, c individually as they get transformed through each round. To state it formally, starting with the normalized triple $(\frac{a}{d}, \frac{b}{d}, \frac{c}{d})$, we define a successor triple $(\frac{a^*}{d^*}, \frac{b^*}{d^*}, \frac{c^*}{d^*})$, so that each entry has the respective successor (e.g., the entry $\frac{a}{d}$ has successor $\frac{a^*}{d^*}$). This is well-defined because (a_1, b_1, c_1, d_1) and (a_2, b_2, c_2, d_2) are homogeneous functions of (a, b, c, d). Note that even though from one round to the next, we may have to rename a^*, b^*, c^* so that the permutated triple $\widetilde{a}, \widetilde{b}, \widetilde{c}$ satisfies (3), the successor sequence as the rounds progress stays with an individual value. E.g., starting from (a, b, c, d), if after one round $a^* = \max(a^*, b^*, c^*, d^*) = \widetilde{c}$, then the successor of $\frac{a}{d}$ after two rounds is $\frac{(\widetilde{c})^*}{(d^*)^*}$. Now define $(\alpha_k, \beta_k, \gamma_k)$ to be the (ordered) triple $(\frac{a}{d}, \frac{b}{d}, \frac{c}{d})$ for $k = 1$, or its successor triple, at the beginning of the k-th round for $k > 1$.

Let $\check{f}$ be the 4-ary signature constructed after $3(k+1)$ rounds. By the *Pigeonhole Principle*, after $3(k+1)$ rounds, at least one of a, b, c has the property that in at least $k+1$ many rounds (let $1 \le i_0 < i_1 < \ldots < i_k \le 3(k+1)$ be $k+1$ such rounds) the corresponding $\frac{a}{d}, \frac{b}{d}, \frac{c}{d}$ or its successors are the maximum (normalized) value in that round, and thus its next successor gets shrunken quadratically in that round. Suppose this is a; the same proof works if it is b or c. Let α_i be the maximum (normalized) value at the beginning of round i in $k+1$ rounds, where $i \in \{i_0, \ldots, i_k\}$. Since initially we have $1 \le d \le a \le b \le c \le \frac{3}{2}d$,

$$0 \le \alpha_{i_1} - 1 \le \alpha_{i_0+1} - 1 \le (\alpha_{i_0} - 1)^2 \le \frac{1}{2^2}.$$

Then

$$0 \le \alpha_{i_2} - 1 \le \alpha_{i_1+1} - 1 \le (\alpha_{i_1} - 1)^2 \le \frac{1}{2^{2^2}}.$$

By induction $0 \le \alpha_{i_k} - 1 \le \frac{1}{2^{2^k}}$. At the end of $3(k+1)$ rounds, if $\check{f}$ has parameters $(\check{a}, \check{b}, \check{c}, \check{d})$, then

$$0 \le \frac{\max(\check{a}, \check{b}, \check{c})}{\check{d}} - 1 \le \alpha_{i_k} - 1 \le \frac{1}{2^{2^k}}.$$

Therefore, after logarithmically many rounds, using polynomially many vertices, we can get a 4-ary construction with parameters $\check{a}$, $\check{b}$, $\check{c}$, and $\check{d}$ that are exponentially close to 1 after normalizing by $\check{d}$. Thus (2) is proved. ◀

Problem : ISING(β).
Instance : Graph $G = (V, E)$.
Output: $Z_{\text{ISING}}(G; \beta) := \sum_{\sigma: V \to \{0,1\}} \beta^{\text{mono}(\sigma)}$, where mono($\sigma$) denotes the number of edges $\{u, v\}$ such that $\sigma(u) = \sigma(v)$.

▶ **Lemma 9.** *The Ising problem* ISING $\left(\frac{w}{x}\right)$ *is equivalent to the Holant problem* Holant $\left(\begin{bmatrix} w & 0 & 0 & x \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ x & 0 & 0 & w \end{bmatrix}\right)$. *In particular,* ISING $\left(\frac{w}{x}\right) \equiv_{\text{AP}}$ Holant $\left(\begin{bmatrix} w & 0 & 0 & x \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ x & 0 & 0 & w \end{bmatrix}\right)$.

▶ Remark 10. A non-homogenized form of the Ising model is $\widetilde{Z}_{\text{ISING}}(G; x, w) := \sum_{\sigma: V \to \{0,1\}} w^{\text{mono}(\sigma)} x^{|E| - \text{mono}(\sigma)}$. If $x \neq 0$ then $\widetilde{Z}_{\text{ISING}}(G; x, w) = x^{|E|} Z_{\text{ISING}}(G; \frac{w}{x})$. If $x = 0$ then in $\widetilde{Z}_{\text{ISING}}$ all vertices in each component must take the same assignment (all 0 or all 1). In this case both $\widetilde{Z}_{\text{ISING}}(G; x, w)$ and the Holant problem in Lemma 9 are trivially solvable in polynomial time.

Proof. For the problem Holant $\left(\begin{bmatrix} w & 0 & 0 & x \\ 0 & y & z & 0 \\ 0 & z & y & 0 \\ x & 0 & 0 & w \end{bmatrix}\right)$, the roles of x, y, z are interchangeable by relabeling the edges. For example, if the signature $f(x_1, x_2, x_3, x_4)$ has the constraint matrix $\begin{bmatrix} w & 0 & 0 & x \\ 0 & y & z & 0 \\ 0 & z & y & 0 \\ x & 0 & 0 & w \end{bmatrix}$, then the signature $f(x_1, x_3, x_2, x_4)$ has the constraint matrix $\begin{bmatrix} w & 0 & 0 & z \\ 0 & y & x & 0 \\ 0 & x & y & 0 \\ z & 0 & 0 & w \end{bmatrix}$. It follows that

$$\text{Holant}\left(\begin{bmatrix} w & 0 & 0 & x \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ x & 0 & 0 & w \end{bmatrix}\right) \quad \text{and} \quad \text{Holant}\left(\begin{bmatrix} w & 0 & 0 & 0 \\ 0 & 0 & x & 0 \\ 0 & x & 0 & 0 \\ 0 & 0 & 0 & w \end{bmatrix}\right)$$

are exactly the same problem. So to prove the lemma it suffices to prove the equivalence of

$$\textsc{Ising}\left(\frac{w}{z}\right) \quad \text{and} \quad \text{Holant}\left(\begin{bmatrix} w & 0 & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & z & 0 & 0 \\ 0 & 0 & 0 & w \end{bmatrix}\right).$$

First we show that Holant $\left(\begin{bmatrix} w & 0 & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & z & 0 & 0 \\ 0 & 0 & 0 & w \end{bmatrix}\right)$ can be expressed as $\textsc{Ising}\left(\frac{w}{z}\right)$.

Given a 4-regular graph $G = (V, E)$ as an instance of Holant $\left(\begin{bmatrix} w & 0 & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & z & 0 & 0 \\ 0 & 0 & 0 & w \end{bmatrix}\right)$, we can partition E into a set $\mathcal{C}$ of *circuits* (in which vertices may repeat but edges cannot) in the following way: at every vertex $v \in V$, denote the four edges incident to v by e_1, e_2, e_3, e_4 in a cyclic order according to the local labeling of the signature function; we make e_1 and e_3 into adjacent edges in a single circuit, and similarly we make e_2 and e_4 into adjacent edges in a single circuit (note that these may be the same circuit). We say each circuit in $\mathcal{C}$ is a *crossing circuit* of G. For the graph G, we define its *crossing-circuit graph* $\widetilde{G} = (\mathcal{C}, \widetilde{E})$, with possible multiloops and multiedges, as follows: its vertex set $\mathcal{C}$ consists of the crossing circuits; for every $v \in V$, if circuits C_1 and C_2 intersect at v, then there is an edge $\widetilde{e}_v \in \widetilde{E}$ labeled by v. Note that it is possible that $C_1 = C_2$, and for such a self-intersectison point the edge $\widetilde{e}_v$ is a loop. Each $C \in \mathcal{C}$ may have multiple loops, and for distinct circuits C_1 and C_2 there may be multiple edges between them. The edge set $\widetilde{E}$ of $\widetilde{G}$ is in 1-1 correspondence with V of G.

Observe that the problem Holant $\left(\begin{bmatrix} w & 0 & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & z & 0 & 0 \\ 0 & 0 & 0 & w \end{bmatrix}\right)$ requires that every valid configuration σ (that contributes a non-zero term) obeys the following rule at each vertex v:

- Assuming e_1, e_2, e_3, e_4 are the four edges incident to v in cyclic order, then $\sigma(e_1) = \sigma(e_3)$ (denoted by b_1) and $\sigma(e_2) = \sigma(e_4)$ (denoted by b_2). That is to say, all edges in a crossing circuit must have the same assignment (either all 0 or all 1). Therefore, the valid configurations σ on the edges of G are in 1-1 correspondence with $0, 1$-assignments σ' on the vertices of $\widetilde{G}$.
- Under σ, the local weight on v is w if $b_1 = b_2$ and is z otherwise. Suppose crossing circuits C_1 and C_2 intersect at v (they could be identical). Then in $\widetilde{G}$, σ' has local weight w on the edge $\widetilde{e}_v$ if $\sigma'(C_1) = \sigma'(C_2)$ and has local weight z otherwise.

This means

$$\text{Holant}\left(G; \begin{bmatrix} w & 0 & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & z & 0 & 0 \\ 0 & 0 & 0 & w \end{bmatrix}\right) = z^{|V(G)|} \cdot Z_{\textsc{Ising}}\left(\widetilde{G}; \frac{w}{z}\right).$$

Next we show that $\textsc{Ising}\left(\frac{w}{z}\right)$ can be expressed as Holant $\left(\begin{bmatrix} w & 0 & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & z & 0 & 0 \\ 0 & 0 & 0 & w \end{bmatrix}\right)$. Note that every graph $G = (V, E)$ (without isolated vertices) is the crossing-circuit graph of some 4-regular graph $\overline{G}$. To define $\overline{G}$ from G, one only needs to do the following: (1) transform each vertex $v \in V$ into a closed cycle C_v; (2) for each loop at $v \in V$, make a self-intersection on C_v; and (3) for each non-loop edge $\{u, v\} \in E$ (u and v are two distinct vertices), make C_u and C_v

intersect in a "crossing" way at a vertex in $\overline{G}$ (by first creating a vertex p on C_u and another vertex p' on C_v, then merging p and p' with local labeling $1, 3$ on C_u and $2, 4$ on C_v). Then the above proof holds for the reverse direction. ◀

▶ **Proposition 11** ([11, Lemma 7]). *Suppose $\beta < -1$. Then $\#\mathrm{PM} \equiv_{\mathrm{AP}} \textsc{Ising}(\beta)$.*

▶ **Corollary 12.** *Suppose $x \neq 0$ and $\frac{w}{x} < -1$. Then $\#\mathrm{PM} \equiv_{\mathrm{AP}} \mathrm{Holant}\left(\left[\begin{smallmatrix} w & 0 & 0 & x \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ x & 0 & 0 & w \end{smallmatrix}\right]\right)$.*

When $a+d > b+c$ we have $\frac{a+b+c+d}{-a+b+c-d} < -1$, so by Corollary 12, Lemma 8, and Lemma 7,

$$\begin{aligned}
\#\mathrm{PM} &\equiv_{\mathrm{AP}} \mathrm{Holant}\left(\left[\begin{smallmatrix} a+b+c+d & 0 & 0 & -a+b+c-d \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -a+b+c-d & 0 & 0 & a+b+c+d \end{smallmatrix}\right]\right) \\
&\leq_{\mathrm{AP}} \mathrm{Holant}\left(\left[\begin{smallmatrix} a+b+c+d & 0 & 0 & -a+b+c-d \\ 0 & a-b+c-d & a+b-c-d & 0 \\ 0 & a+b-c-d & a-b+c-d & 0 \\ -a+b+c-d & 0 & 0 & a+b+c+d \end{smallmatrix}\right]\right) \\
&\equiv_{\mathrm{AP}} \#\mathrm{EV}(a, b, c, d).
\end{aligned}$$

By the symmetry of a, b, c, we have proved Theorem 2.

4 #PerfectMatchings-easiness

In this section, we address two problems:

1. What are the signatures that can be realized by 4-ary matchgates (Definition 13)?
   Although the set of signatures that can be realized by planar matchgates with complex edge weights have been completely characterized [5], the set of signatures that can be realized by general (not necessarily planar) matchgates with nonnegative real edge weights is not fully understood, even for matchgates of arity 4. This type of matchgates plays a crucial role in the study of the approximate complexity of counting problems [18, 4], as we will see in this paper.
   In Theorem 15, we give a complete characterization of signatures of arity 4 that can be realized by matchgates with nonnegative real edges. Our method is primarily geometric.
2. Theorem 2 shows that for positive parameters $(a, b, c, d) \notin \mathcal{Y}$ the problem $\#\mathrm{EV}(a, b, c, d)$ is at least as hard as counting perfect matchings approximately. Here we ask the reverse question: For what parameter settings (a, b, c, d) does $\#\mathrm{EV}(a, b, c, d) \leq_{\mathrm{AP}} \#\mathrm{PM}$?
   We know that

   $$Z_{\mathrm{EV}}(G; a, b, c, d) = \mathrm{Holant}\left(G'; {\neq_2} \mid f\right),$$

   where f is the 4-ary signature with $M(f) = \left[\begin{smallmatrix} d & 0 & 0 & a \\ 0 & b & c & 0 \\ 0 & c & b & 0 \\ a & 0 & 0 & d \end{smallmatrix}\right]$. Considering the fact that $(\neq_2)$ can be easily realized by a matchgate (a vertex with two dangling edges), Theorem 4 is a direct consequence of Lemma 17 which says that any signature in $\mathscr{Z}$ is realizable by some *4-ary matchgate* of constant size (with nonnegative edge weights, but not necessarily planar) (see Definition 13). Our theorem works for the eight-vertex model with parameter settings $\mathcal{S}^{\mathrm{E}}_{\leq 2}$ (defined below) not necessarily satisfying the arrow reversal symmetry.
   Moreover, Lemma 19 indicates that our result is tight in the sense that $\mathcal{S}^{\mathrm{E}}_{\leq 2}$ captures precisely the set of all signatures that can be realized by 4-ary matchgates (with even support, i.e., nonzero only on inputs of even Hamming weight). A similar statement holds for $\mathcal{S}^{\mathrm{O}}_{\leq 2}$. the corresponding set with odd support.

▶ **Definition 13.** *We use the term* a k-ary matchgate *to denote a graph* Γ *having* k *"dangling" edges, labelled* $i_1, \ldots, i_k$*. Each dangling edge has weight* 1 *and each non-dangling edge* e *is equipped with a nonnegative weight* w_e*. A configuration is a* $0,1$*-assignment to the edges. A configuration is a perfect matching if every vertex has exactly one incident edge assigned* 1*. A* k*-ary matchgate implements the signature* f*, where* $f(b_1, \ldots, b_k)$ *for* $(b_1, \ldots, b_k) \in \{0,1\}^k$ *is the sum, over perfect matchings, of the product of the weight of edges with assignment* 1*, where the dangling edge* i_j *is assigned* b_j*, and the empty product has weight* 1*.*

▶ Remark 14. Contrary to Definition 13 which does not require planarity, planar matchgates with complex edge weights has been completely characterized [21, 5]. As computing the weighted sum of perfect matchings is in polynomial time over planar graphs by the *FKT algorithm* [20, 15, 16], problems that can be locally expressed by planar matchgates are tractable over planar graphs.

▶ **Notation.**

$$\mathcal{S}^{\mathrm{E}}_{\leq^2} = \{f \mid M(f) = \begin{bmatrix} d_1 & 0 & 0 & a_1 \\ 0 & b_1 & c_1 & 0 \\ 0 & c_2 & b_2 & 0 \\ a_2 & 0 & 0 & d_2 \end{bmatrix} \textit{ satisfying } \begin{cases} a_1a_2 \leq b_1b_2 + c_1c_2 + d_1d_2 \\ b_1b_2 \leq a_1a_2 + c_1c_2 + d_1d_2 \\ c_1c_2 \leq a_1a_2 + b_1b_2 + d_1d_2 \\ d_1d_2 \leq a_1a_2 + b_1b_2 + c_1c_2 \end{cases}, a_1, \cdots, d_2 \geq 0.\},$$

$$\mathcal{S}^{\mathrm{O}}_{\leq^2} = \{f \mid M(f) = \begin{bmatrix} 0 & d_1 & a_1 & 0 \\ b_1 & 0 & 0 & c_1 \\ c_2 & 0 & 0 & b_2 \\ 0 & a_2 & d_2 & 0 \end{bmatrix} \textit{ satisfying } \begin{cases} a_1a_2 \leq b_1b_2 + c_1c_2 + d_1d_2 \\ b_1b_2 \leq a_1a_2 + c_1c_2 + d_1d_2 \\ c_1c_2 \leq a_1a_2 + b_1b_2 + d_1d_2 \\ d_1d_2 \leq a_1a_2 + b_1b_2 + c_1c_2 \end{cases}, a_1, \cdots, d_2 \geq 0.\}.$$

▶ **Theorem 15.** *Denote by* $\mathcal{M}$ *the set of signatures that can be realized by 4-ary matchgates. Then* $\mathcal{M} = \mathcal{S}^{\mathrm{E}}_{\leq^2} \bigcup \mathcal{S}^{\mathrm{O}}_{\leq^2}$.

▶ Remark 16. Note that any signature in $\mathcal{M}$ must satisfy either *even parity* (nonzero only on inputs of even Hamming weight) or *odd parity* (nonzero only on inputs of odd Hamming weight). Theorem 15 for the even parity part ($\mathcal{S}^{\mathrm{E}}_{\leq^2}$) is a combination of Lemma 17 and Lemma 19. The odd parity part can be proved similarly.

▶ **Lemma 17.** *Suppose* $f \in \mathcal{S}^{\mathrm{E}}_{\leq^2}$*. Then there is a 4-ary matchgate of constant size whose signature is* f*.*

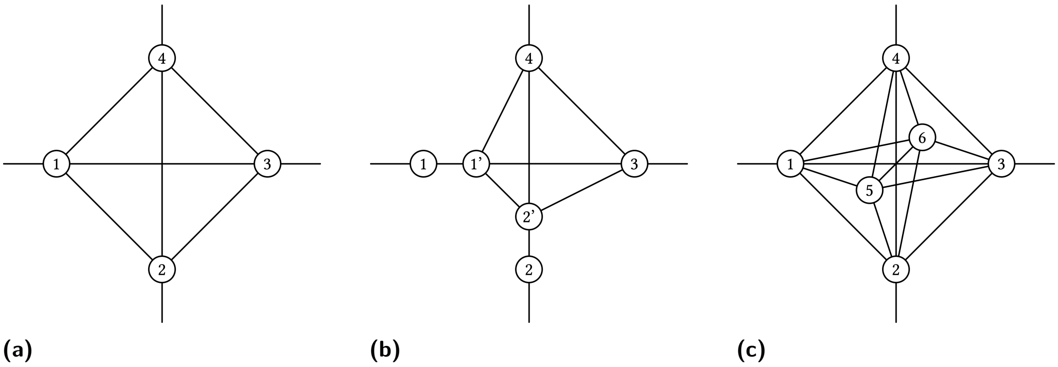


Figure 4 4-ary matchgates.

Proof. We first note that if any of the four inequalities in the definition of $\mathcal{S}_{\leq^2}$ is an equality, then the remaining three inequalities automatically hold, since the 8 values $a_1, \ldots, d_2$ are all nonnegative.

Given a signature $\begin{bmatrix} d_1 & 0 & 0 & a_1 \\ 0 & b_1 & c_1 & 0 \\ 0 & c_2 & b_2 & 0 \\ a_2 & 0 & 0 & d_2 \end{bmatrix}$, first we construct a matchgate for $d_1d_2 = a_1a_2+b_1b_2+c_1c_2$. If $d_1d_2 = 0$ then all four products $a_1a_2 = b_2b_2 = c_1c_2 = d_1d_2 = 0$, and one can easily adapt from the following proof to show that the signature is realizable as a matchgate signature. So it suffices to implement the normalized version $\begin{bmatrix} a_1a_2+b_1b_2+c_1c_2 & 0 & 0 & a_1 \\ 0 & b_1 & c_1 & 0 \\ 0 & c_2 & b_2 & 0 \\ a_2 & 0 & 0 & 1 \end{bmatrix}$. Our construction is a weighted K_4 depicted in Figure 4a. Let e_1, e_2, e_3, e_4 be the dangling edges incident to vertices $1, 2, 3, 4$, respectively. Denote by w_{ij} the weight on the edge between vertex i and vertex j. One can check that the following weight assignment meets our need: $w_{12} = a_1, w_{34} = a_2, w_{14} = b_1, w_{23} = b_2, w_{13} = c_1, w_{24} = c_2$.

For $a_1a_2 = b_1b_2 + c_1c_2 + d_1d_2$, without loss of generality we assume $a_1a_2 \neq 0$ and we normalize $a_1 = 1$. Then our construction is shown in Figure 4b where we set $w_{11'} = 1, w_{22'} = 1, w_{1'2'} = d_2, w_{34} = d_1, w_{1'4} = c_2, w_{2'3} = c_1, w_{1'3} = b_2, w_{2'4} = b_1$. One can verify that it realizes the normalized signature $\begin{bmatrix} d_1 & 0 & 0 & 1 \\ 0 & b_1 & c_1 & 0 \\ 0 & c_2 & b_2 & 0 \\ b_1b_2+c_1c_2+d_1d_2 & 0 & 0 & d_2 \end{bmatrix}$. The construction for $b_1b_2 = a_1a_2 + c_1c_2 + d_1d_2$ and $c_1c_2 = a_1a_2 + b_1b_2 + d_1d_2$ are symmetric to the above case.

It remains to show that the interior

$$\begin{cases} a_1a_2 & < & b_1b_2 + c_1c_2 + d_1d_2 \\ b_1b_2 & < & a_1a_2 + c_1c_2 + d_1d_2 \\ c_1c_2 & < & a_1a_2 + b_1b_2 + d_1d_2 \\ d_1d_2 & < & a_1a_2 + b_1b_2 + c_1c_2 \end{cases} \tag{4}$$

can all be reached. We first deal with the case when all eight parameters are strictly positive and leave the other cases to the end of this proof. We use a weighted K_6 to be our matchgate depicted in Figure 4c, and set $w_{12} = r_1, w_{34} = r_2, w_{14} = s_1, w_{23} = s_2, w_{13} = t_1, w_{24} = t_2, w_{15} = p_1, w_{25} = p_2, w_{35} = p_3, w_{45} = p_4, w_{16} = q_1, w_{26} = q_2, w_{36} = q_3, w_{46} = q_4, w_{56} = 1$. Then the matchgate has a singature with the following parameters

$$\begin{aligned} a'_1 &= r_1 + p_1q_2 + p_2q_1, & a'_2 &= r_2 + p_3q_4 + p_4q_3, \\ b'_1 &= s_1 + p_1q_4 + p_4q_1, & b'_2 &= s_2 + p_2q_3 + p_3q_2, \\ c'_1 &= t_1 + p_1q_3 + p_3q_1, & c'_2 &= t_2 + p_2q_4 + p_4q_2, \\ d'_1 &= (r_1r_2 + s_1s_2 + t_1t_2)+ & d'_2 &= 1, \\ &\quad (p_3q_4 + p_4q_3)r_1 + (p_1q_2 + p_2q_1)r_2+ \\ &\quad (p_2q_3 + p_3q_2)s_1 + (p_1q_4 + p_4q_1)s_2+ \\ &\quad (p_2q_4 + p_4q_2)t_1 + (p_1q_3 + p_3q_1)t_2. \end{aligned}$$

Note that all the edge weights have to be nonnegative. By properly setting the edge weights in the matchgate, we show that we can achieve any relative ratios among the eight given positive values $a_1, a_2, b_1, b_2, c_1, c_2, d_1, d_2$ that satisfy (4). Our first step is to achieve any relative ratios among the four product values a_1a_2, b_1b_2, c_1c_2, d_1d_2 satisfying (4); and the second step is to adjust the relative ratio within the pairs $\{a_1, a_2\}$, $\{b_1, b_2\}$, $\{d_1, d_2\}$ and $\{c_1, c_2\}$ without affecting the product values. This can be justified by the observation that, by a scaling a global positive constant can be easily achieved, and all appearances of a_1 and a_2 in (4) are as a product a_1a_2, and similarly for $b_1, b_2, c_1, c_2, d_1, d_2$.

For the fourteen edge weights $r_1, \ldots, q_4$ to be determined, let

$$\begin{cases} A' = p_1p_2q_3q_4 + p_3p_4q_1q_2, & R = r_1r_2 + r_1(p_3q_4 + p_4q_3) + r_2(p_1q_2 + p_2q_1), \\ B' = p_1p_4q_2q_3 + p_2p_3q_1q_4, & S = s_1s_2 + s_1(p_2q_3 + p_3q_2) + s_2(p_1q_4 + p_4q_1), \\ C' = p_1p_3q_2q_4 + p_2p_4q_1q_3, & T = t_1t_2 + t_1(p_2q_4 + p_4q_2) + t_2(p_1q_3 + p_3q_1), \end{cases} \tag{5}$$

and define

$$\begin{cases} A & = & A' + S + T \\ B & = & B' + R + T \\ C & = & C' + R + S \\ D & = & A' + B' + C'. \end{cases} \tag{6}$$

Note that A', B', C', R, S, T are all nonnegative and so are A, B, C, D.

Our goal is to choose the fourteen edge weights $r_1, \ldots, q_4$ so that A, B, C, D are all positive and satisfy

$$\begin{cases} A & = \frac{1}{2}(b_1b_2 + c_1c_2 + d_1d_2 - a_1a_2) \\ B & = \frac{1}{2}(a_1a_2 + c_1c_2 + d_1d_2 - b_1b_2) \\ C & = \frac{1}{2}(a_1a_2 + b_1b_2 + d_1d_2 - c_1c_2) \\ D & = \frac{1}{2}(a_1a_2 + b_1b_2 + c_1c_2 - d_1d_2). \end{cases} \tag{7}$$

Note that, by definition, the left-side of (7) is precisely the right-side of (7) when $a_1, \ldots, d_2$ are replaced by $a'_1, \ldots, d'_2$ respectively. Denote the products $a_1a_2, b_1b_2, c_1c_2, d_1d_2$ by a^{**}, b^{**}, c^{**}, d^{**} respectively. Then (7) is a set of four linear equations $M \cdot \begin{bmatrix} a^{**} \\ b^{**} \\ c^{**} \\ d^{**} \end{bmatrix} = \begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix}$, where $M = \frac{1}{2}\begin{bmatrix} -1 & 1 & 1 & 1 \\ 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 1 \\ 1 & 1 & 1 & -1 \end{bmatrix}$. Note that M is invertible and $M^{-1} = M$, so (7) is equivalent to $M \cdot \begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix} = \begin{bmatrix} a^{**} \\ b^{**} \\ c^{**} \\ d^{**} \end{bmatrix}$, having an identical form. Since the requirement (4) in terms of $a^{**}, b^{**}, c^{**}, d^{**}$ translates into the requirement A, B, C, D being strictly positive via M, it is not surprising that the requirement $a^{**}, b^{**}, c^{**}, d^{**}$ being strictly positive translates into the requirement

$$\begin{cases} A < B + C + D \\ B < A + C + D \\ C < A + B + D \\ D < A + B + C, \end{cases} \tag{8}$$

and that A, B, C, D are positive is the same as (4).

Furthermore, let $\begin{cases} X=S+T \\ Y=R+T \\ Z=R+S \end{cases}$, then the requirement R, S, T being positive is equivalent to the requirement $\begin{cases} Y+Z>X \\ X+Z>Y \\ X+Y>Z \end{cases}$. This is because $\begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} R \\ S \\ T \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$ is the same as $\frac{1}{2}\begin{bmatrix} -1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & 1 & -1 \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} R \\ S \\ T \end{bmatrix}$.

The crucial ingredient of our proof is a *geometric* lemma in 3-dimensional space. Suppose $a^{**}, b^{**}, c^{**}, d^{**}$ are positive and they satisfy (4). This defines $\begin{bmatrix} \tilde{A} \\ \tilde{B} \\ \tilde{C} \\ \tilde{D} \end{bmatrix} = M \cdot \begin{bmatrix} a^{**} \\ b^{**} \\ c^{**} \\ d^{**} \end{bmatrix}$. By a scaling we may assume $\tilde{D} = 1$. Hence $(\tilde{A}, \tilde{B}, \tilde{C}, \tilde{D})$ are positive and satisfy (8). Thus $(\tilde{A}, \tilde{B}, \tilde{C})$ belongs to the set U in the statement of Lemma 18.

By Lemma 18, there exist (strictly) positive tuples $(\tilde{A}', \tilde{B}', \tilde{C}')$ and $(\tilde{X}, \tilde{Y}, \tilde{Z})$ such that

$$(\tilde{A}, \tilde{B}, \tilde{C}) = (\tilde{A}', \tilde{B}', \tilde{C}') + (\tilde{X}, \tilde{Y}, \tilde{Z}),$$

satisfying $\tilde{A}' + \tilde{B}' + \tilde{C}' = 1$ and $\begin{cases} \tilde{Y}+\tilde{Z}>\tilde{X} \\ \tilde{X}+\tilde{Z}>\tilde{Y} \\ \tilde{X}+\tilde{Y}>\tilde{Z} \end{cases}$. By the previous observation this indicates that there exist (strictly) positive $\tilde{A}', \tilde{B}', \tilde{C}', \tilde{R}, \tilde{S}, \tilde{T}$ such that $\begin{cases} \tilde{A}=\tilde{A}'+\tilde{S}+\tilde{T} \\ \tilde{B}=\tilde{B}'+\tilde{R}+\tilde{T} \\ \tilde{C}=\tilde{C}'+\tilde{R}+\tilde{S} \\ \tilde{D}=\tilde{A}'+\tilde{B}'+\tilde{C}' \end{cases}$.

We first set p_i, q_i $(1 \le i \le 4)$ so that $(A', B', C') = c \cdot (\tilde{A}', \tilde{B}', \tilde{C}')$ for some constant c. To achieve this, set $q_1 = q_2 = q_3 = q_4 = 1$, and let o_1, o_2, o_3 be positive, and then set $p_1 = \sqrt{\frac{o_2 o_3}{o_1}}$, $p_2 = \sqrt{\frac{o_3 o_1}{o_2}}$, $p_3 = \sqrt{\frac{o_1 o_2}{o_3}}$, and $p_4 = \frac{1}{p_1 p_2 p_3}$. We have $p_1 p_2 p_3 p_4 = 1$, and $p_2 p_3 = o_1, p_3 p_1 = o_2, p_1 p_2 = o_3$. Then set $\begin{cases} A'=p_1p_2+\frac{1}{p_1p_2}=o_3+\frac{1}{o_3} \\ B'=p_2p_3+\frac{1}{p_2p_3}=o_1+\frac{1}{o_1} \\ C'=p_3p_1+\frac{1}{p_3p_1}=o_2+\frac{1}{o_2} \end{cases}$, which can be independently any positive numbers at least 2, by choosing o_1, o_2, o_3 to be suitable positive numbers. This allows us to get A', B', C' such that $(A', B', C') = c \cdot (\tilde{A}', \tilde{B}', \tilde{C}')$ for some constant c. Then it is obvious that $r_1, r_2, s_1, s_2, t_1, t_2$ can be set so that $(R, S, T) = c \cdot (\tilde{R}, \tilde{S}, \tilde{T})$. Compute A, B, C, D according to (6). As a consequence, $(A, B, C, D) = c \cdot (\tilde{A}, \tilde{B}, \tilde{C}, \tilde{D})$ is a valid solution.

To adjust the relative ratio between $\{d_1, d_2\}$, say increasing $\frac{d_2}{d_1}$ by δ, while keeping all product values and the relative ratios within the other three pairs, just increase $r_1, r_2, s_1, s_2, t_1, t_2$ by $\delta^{1/2}$ and increase p_i, q_i $(1 \le i \le 4)$ by $\delta^{1/4}$. Similarly, to increase $\frac{a_2}{a_1}$ by δ alone without affecting the other products and ratios, just increase r_2 by $\delta^{1/2}$ and p_3, p_4, q_3, q_4 by $\delta^{1/4}$, and decrease r_1 by $\delta^{1/2}$ and p_1, p_2, q_1, q_2 by $\delta^{1/4}$. The other cases are symmetric.

Finally we deal with the cases when there are zeros among the eight parameters. Note that at most one of $a_1a_2, b_1b_2, c_1c_2, d_1d_2$ is zero, because if at least two products are zero, say $a_1a_2 = b_1b_2 = 0$, then (4) forces a contradiction that $c_1c_2 < d_1d_2$ and $d_1d_2 < c_1c_2$. In the case $d_1d_2 = 0$:

- $d_1 = 0, d_2 \ne 0$: We make the modification that $w_{12} = w_{34} = w_{14} = w_{23} = w_{13} = w_{24} = 0$, i.e. $r_1, r_2, s_1, s_2, t_1, t_2 = 0$.
- $d_1 = d_2 = 0$: We make the further modification that $w_{56} = 0$.
- $d_1 \ne 0, d_2 = 0$: We connect the four dangling edges in Figure 4c to four degree 2 vertices, respectively. This switches the role of d_1 and d_2 in the previous proof.

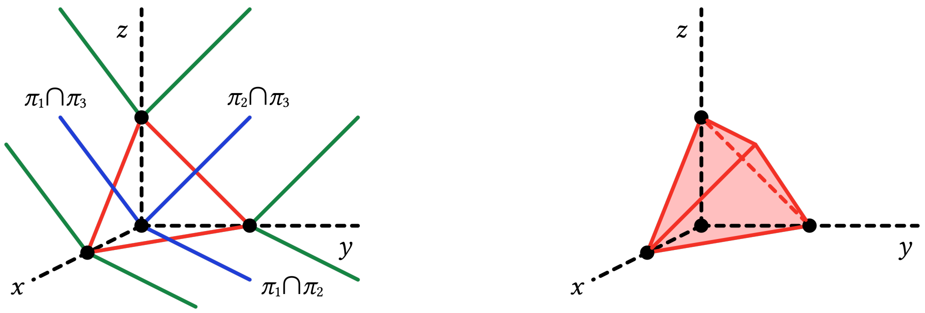
One can check our proof is still valid in the above three cases. If $a_1a_2 = 0$, then we connect the dangling edges on vertices 1, 2 to two degree 2 vertices (similar to the operation from Figure 4a to Figure 4b). This switches the role of d_1, d_2 with a_2, a_1 and the proof folllows. The proofs for $b_1b_2 = 0$ and $c_1c_2 = 0$ are symmetric. ◀

Now we give the crucial geometric lemma.

▶ **Lemma 18.** *Let* $U = \{(x, y, z) \in \mathbb{R}^3_{>0} \mid x < y+z+1, y < x+z+1, z < x+y+1, 1 < x+y+z\}$, $V = \{(x, y, z) \in \mathbb{R}^3_{>0} \mid x + y + z = 1\}$, *and* $W = \{(x, y, z) \in \mathbb{R}^3_{>0} \mid y + z > x, x + z > y, x + y > z\}$. *Then* U *is the Minkowski sum of* V *and* W, *namely,* U *consists of precisely those points* $\boldsymbol{u} \in \mathbb{R}^3$, *such that* $\boldsymbol{u} = \boldsymbol{v} + \boldsymbol{w}$ *for some* $\boldsymbol{v} \in V$ *and* $\boldsymbol{w} \in W$. *The same statement is true for the closures of* U, V *and* W *(in the topology of Euclidean space).*

Proof. Observe that U, V, and W are the interiors of a polyhedron with 7 facets, a regular triangle, and a polyhedron with 3 facets, respectively.

The polyhedron for W is the intersection of three half spaces bounded by three planes, $\begin{cases} (\pi_1): y+z \ge x \\ (\pi_2): x+z \ge y \\ (\pi_3): x+y \ge z \end{cases}$, where the planes are defined by equalities. Note that these inequalities imply that $x, y, z \ge 0$, thus this polyhedron has only three facets. We can find the intersection of each pair of the three planes for W as $\begin{cases} \pi_1 \cap \pi_2 : x=y \ge 0, z=0 \\ \pi_1 \cap \pi_3 : x=z \ge 0, y=0 \\ \pi_2 \cap \pi_3 : y=z \ge 0, x=0 \end{cases}$. Note that these intersections lie on the planes $z = 0$, $y = 0$, and $x = 0$, respectively.



(a) The blue rays are the intersections of the three facets for W. The red triangle is the boundary for V. The green rays together with the red triangle are the intersections of the seven facets for U.

(b) The tetrahedron in U that is left uncovered by sliding W along the boundary of V, but is covered by the rays from the simplex V in the direction of $(1,1,1)$.

Figure 5

Let $\boldsymbol{e}_1 = (1,0,0), \boldsymbol{e}_2 = (0,1,0), \boldsymbol{e}_3 = (0,0,1)$. Then the triangle for V is just the convex hull of $\boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3$. Suppose we shift the origin of W from $\boldsymbol{0}$ to $\boldsymbol{e}_1$ and denote the resulting (interior of a) polyhedron by $W^{\boldsymbol{e}_1}$, then we have the defining inequalities $\begin{cases} (\pi_1^{\boldsymbol{e}_1}): y+z \geq (x-1) \\ (\pi_2^{\boldsymbol{e}_1}): (x-1)+z \geq y \\ (\pi_3^{\boldsymbol{e}_1}): (x-1)+y \geq z \end{cases}$, where the shifted planes are defined by the corresponding equalities. By symmetry, if we shift the origin of W to $\boldsymbol{e}_2$ and to $\boldsymbol{e}_3$, we have respectively $W^{\boldsymbol{e}_2}$ with $\begin{cases} (\pi_1^{\boldsymbol{e}_2}): (y-1)+z \geq x \\ (\pi_2^{\boldsymbol{e}_2}): x+z \geq (y-1) \\ (\pi_3^{\boldsymbol{e}_2}): x+(y-1) \geq z \end{cases}$, and $W^{\boldsymbol{e}_3}$ with $\begin{cases} (\pi_1^{\boldsymbol{e}_3}): y+(z-1) \geq x \\ (\pi_2^{\boldsymbol{e}_3}): x+(z-1) \geq y \\ (\pi_3^{\boldsymbol{e}_3}): x+y \geq (z-1) \end{cases}$. Note that the shifted planes $\pi_1^{\boldsymbol{e}_1}$, $\pi_2^{\boldsymbol{e}_2}$, and $\pi_3^{\boldsymbol{e}_3}$ contain three distinct facets of U, and they coincide exactly with a facet of $W^{\boldsymbol{e}_1}$, $W^{\boldsymbol{e}_2}$, and $W^{\boldsymbol{e}_3}$, respectively.

By sliding W with its origin along the line $x+y=1, z=0$ from $\boldsymbol{e}_1$ to $\boldsymbol{e}_2$, we have a partial coverage of U by the shifting copies of W from $W^{\boldsymbol{e}_1}$ and $W^{\boldsymbol{e}_2}$:

- The shifted ray of $\pi_1 \cap \pi_2 : x = y \geq 0, z = 0$ moves from $\pi_1^{\boldsymbol{e}_1} \cap \pi_2^{\boldsymbol{e}_1} : (x-1) = y \geq 0, z = 0$ to $\pi_1^{\boldsymbol{e}_2} \cap \pi_2^{\boldsymbol{e}_2} : x = (y-1) \geq 0, z = 0$. Notice that this is a parallel transport, and stays on the plane $z = 0$, and thus it swipes another facet of U on $z = 0$ bounded by the two lines $x - y = -1$ and $x - y = 1$.
- The shifted ray of $\pi_1 \cap \pi_3 : x = z \geq 0, y = 0$ moves from $\pi_1^{\boldsymbol{e}_1} \cap \pi_3^{\boldsymbol{e}_1} : (x-1) = z \geq 0, y = 0$ to $\pi_1^{\boldsymbol{e}_2} \cap \pi_3^{\boldsymbol{e}_2} : x = z \geq 0, (y-1) = 0$; the shifted ray of $\pi_2 \cap \pi_3 : y = z \geq 0, x = 0$ moves from $\pi_2^{\boldsymbol{e}_1} \cap \pi_3^{\boldsymbol{e}_1} : y = z \geq 0, (x-1) = 0$ to $\pi_2^{\boldsymbol{e}_2} \cap \pi_3^{\boldsymbol{e}_2} : (y-1) = z \geq 0, x = 0$. Notice that both stay on the plane $x + y - z = 1$ which is $\pi_3^{\boldsymbol{e}_1} = \pi_3^{\boldsymbol{e}_2}$.

It follows that the part of U satisfying $x + y - z > 1$ is covered by the Minkowski sum of W and the line segment on $x + y = 1, z = 0$ from $\boldsymbol{e}_1$ to $\boldsymbol{e}_2$ (which is a side of the triangle V).

Symmetrically, after sliding W with its origin from $\boldsymbol{e}_2$ to $\boldsymbol{e}_3$ along the line $y+z = 1, x = 0$ we get the parallel tranport from $W^{\boldsymbol{e}_2}$ to $W^{\boldsymbol{e}_3}$. Also after sliding W with its origin from $\boldsymbol{e}_3$ back to $\boldsymbol{e}_1$ along the line segment $x + z = 1, y = 0$ we get the parallel tranport from $W^{\boldsymbol{e}_3}$ to $W^{\boldsymbol{e}_1}$. After these, the only subset in U that is left uncovered by shifting copies of W is $U \cap \{(x,y,z) \mid \begin{cases} -x+y+z \leq 1 \\ x-y+z \leq 1 \\ x+y-z \leq 1 \end{cases} \} = \{(x,y,z) \in \mathbb{R}^3_{>0} \mid \begin{cases} x+y+z \geq 1 \\ -x+y+z \leq 1 \\ x-y+z \leq 1 \\ x+y-z \leq 1 \end{cases} \}$ – a *tetrahedron* (Figure 5b). However this subset can be covered by the rays $\{\boldsymbol{v} + \lambda(1,1,1) \mid \boldsymbol{v} \in V, \lambda > 0\}$. Note that $\lambda(1,1,1) \in W$ for all $\lambda > 0$.

Finally regarding the closures $\overline{U}, \overline{V}$ and $\overline{W}$, for $v_n \in V$ and $w_n \in W$, if $v_n \to v \in \overline{V}$ and $w_n \to w \in \overline{W}$, then $u_n = v_n + w_n \in U$, and $u_n \to v + w$. So $v + w \in \overline{U}$. Conversely, if $u_n \to u \in \overline{U}$, where $u_n \in U$, then $u_n = v_n + w_n$ for some $v_n \in V$ and $w_n \in W$. As V is bounded, there is a convergent subsequence $\{v_{n_k}\}$, such that $v = \lim_{k\to\infty} v_{n_k} \in \overline{V}$. Then $w_{n_k} = u_{n_k} - v_{n_k}$ also converges to some $w \in \overline{W}$, and then $u = \lim_{k\to\infty}(v_{n_k} + w_{n_k}) = v + w$, is a sum of points from $\overline{V}$ and $\overline{W}$.

This completes the proof. ◀

▶ **Lemma 19.** *Suppose f is the signature of a 4-ary matchgate with* $M(f) = \begin{bmatrix} d_1 & 0 & 0 & a_1 \\ 0 & b_1 & c_1 & 0 \\ 0 & c_2 & b_2 & 0 \\ a_2 & 0 & 0 & d_2 \end{bmatrix}$. *Then $f \in \mathcal{S}^{\mathrm{E}}_{\leq 2}$. In particular, if f satisfies arrow reversal symmetry, $f \in \mathcal{Z}$.*

▶ Remark 20. The last part $d_1 d_2 \ \leq\ a_1 a_2 + b_1 b_2 + c_1 c_2$ was proved in [4, Lemma 56]. The proofs for other three parts are symmetric and similar to the proof for the last part. For completeness, here we give the proof for the first part $a_1 a_2 \ \leq\ b_1 b_2 + c_1 c_2 + d_1 d_2$.

Proof. Consider a 4-ary matchgate Γ with signature f. Given that $M(f) = \begin{bmatrix} d_1 & 0 & 0 & a_1 \\ 0 & b_1 & c_1 & 0 \\ 0 & c_2 & b_2 & 0 \\ a_2 & 0 & 0 & d_2 \end{bmatrix}$, $a_1 a_2 \ \leq\ b_1 b_2 + c_1 c_2 + d_1 d_2$ is equivalent as

$$f(0011)f(1100) \leq f(0110)f(1001) + f(0101)f(1010) + f(0000)f(1111). \tag{9}$$

Let $I = \{i_1, i_2, i_3, i_4\}$ be the set of dangling edges of Γ. For $X \subseteq I$, let M_X denote the set of perfect matchings that include dangling edges in X (by assigning them 1) and exclude dangling edges in $I \setminus X$ (by assigning them 0). We exhibit an injective map

$$\mu : M_{\{i_1,i_2\}} \times M_{\{i_3,i_4\}} \to [M_{\{i_2,i_3\}} \times M_{\{i_1,i_4\}}] \bigcup [M_{\{i_2,i_4\}} \times M_{\{i_1,i_3\}}] \bigcup [M_{\emptyset} \times M_I]$$

which is weight-preserving in the sense that for matchings m_1, m_2, m_3, m_4 with $\mu(m_1, m_2) = (m_3, m_4)$, we have $w(m_1)w(m_2) = w(m_3)w(m_4)$. The existence of μ implies (9).

Given $(m_1, m_2) \in M_{\{i_1,i_2\}} \times M_{\{i_3,i_4\}}$, consider $m_1 \oplus m_2$ and note that this is a collection of cycles together with two paths. Let π be the path connecting the dangling edge i_1 to some other dangling edge; let π' be the path connecting the remaining two dangling edges. Let $m_3 := m_1 \oplus \pi$ and $m_4 := m_2 \oplus \pi$. Then we have the following

- If π connects i_1 to i_2, then $m_3 \in M_{\emptyset}$ and $m_4 \in M_I$;
- If π connects i_1 to i_3, then $m_3 \in M_{\{i_2,i_3\}}$ and $m_4 \in M_{\{i_1,i_4\}}$;
- If π connects i_1 to i_4, then $m_3 \in M_{\{i_2,i_4\}}$ and $m_4 \in M_{\{i_1,i_3\}}$.

The construction is invertible, since if (m_3, m_4) is in the image of the above mapping, then $m_3 \oplus m_4 = m_1 \oplus m_2$. From $m_1 \oplus m_2$, we can recover π (as the unique path that connects i_1 to one of the other dangling edges in $\{i_2, i_3, i_4\}$). Then we can recover m_1 and m_2 as $m_3 \oplus \pi$ and $m_4 \oplus \pi$ respectively. Therefore, $\mu : (m_1, m_2) \to (m_3, m_4)$ is an injection.

To see that μ is weight-preserving, observe that the each of the edges in π appears in exactly one of m_1 and m_2 and in exactly one of m_3 and m_4 and that $m_i \setminus \pi = m_{i+2} \setminus \pi$ for $i \in \{1, 2\}$. Hence,

$$w(m_1)w(m_2) = \prod_{e \in m_1 \setminus \pi} w_e \cdot \prod_{e \in m_2 \setminus \pi} w_e \cdot \prod_{e \in \pi} w_e = \prod_{e \in m_3 \setminus \pi} w_e \cdot \prod_{e \in m_4 \setminus \pi} w_e \cdot \prod_{e \in \pi} w_e = w(m_3)w(m_4).$$

◀

References

1 R. J. Baxter. Eight-vertex model in lattice statistics. *Phys. Rev. Lett.*, 26:832–833, April 1971. doi:10.1103/PhysRevLett.26.832.

2 R. J. Baxter. Partition function of the eight-vertex lattice model. *Annals of Physics*, 70(1):193–228, 1972. doi:10.1016/0003-4916(72)90335-1.

3 R. J. Baxter. *Exactly Solved Models in Statistical Mechanics.* Academic Press Inc., San Diego, CA, USA, 1982.

4 Andrei Bulatov, Leslie Ann Goldberg, Mark Jerrum, David Richerby, and Stanislav Živný. Functional clones and expressibility of partition functions. *Theoretical Computer Science*, 687:11–39, 2017. doi:10.1016/j.tcs.2017.05.001.

5 Jin-Yi Cai and Xi Chen. *Complexity Dichotomies for Counting Problems*, volume 1. Cambridge University Press, 2017. doi:10.1017/9781107477063.

6 Jin-Yi Cai and Zhiguo Fu. Complexity classification of the eight-vertex model. *CoRR*, abs/1702.07938, 2017. arXiv:1702.07938.

7 Jin-Yi Cai, Tianyu Liu, and Pinyan Lu. Approximability of the six-vertex model. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '19, pages 2248–2261, 2019. doi:10.1137/1.9781611975482.136.

8 Jin-Yi Cai, Tianyu Liu, Pinyan Lu, and Jing Yu. Approximability of the eight-vertex model. *CoRR*, abs/1811.03126, 2018. arXiv:1811.03126.

9 Chungpeng Fan and F. Y. Wu. General lattice model of phase transitions. *Phys. Rev. B*, 2:723–733, August 1970. doi:10.1103/PhysRevB.2.723.

10 Andreas Galanis, Daniel Štefankovič, and Eric Vigoda. Inapproximability for antiferromagnetic spin systems in the tree nonuniqueness region. *J. ACM*, 62(6), December 2015. doi:10.1145/2785964.

11 Leslie Ann Goldberg and Mark Jerrum. Inapproximability of the Tutte polynomial. *Information and Computation*, 206(7):908–929, 2008. doi:10.1016/j.ic.2008.04.003.

12 Sam Greenberg and Dana Randall. Slow mixing of Markov chains using fault lines and fat contours. *Algorithmica*, 58(4):911–927, December 2010. doi:10.1007/s00453-008-9246-3.

13 Thomas P. Hayes, Juan C. Vera, and Eric Vigoda. Randomly coloring planar graphs with fewer colors than the maximum degree. *Random Structures & Algorithms*, 47(4):731–759, 2015. doi:10.1002/rsa.20560.

14 Richard M. Karp and Michael Luby. Monte-Carlo algorithms for enumeration and reliability problems. In *Proceedings of the 24th Annual Symposium on Foundations of Computer Science*, SFCS '83, pages 56–64, Washington, DC, USA, 1983. IEEE Computer Society. doi:10.1109/SFCS.1983.35.

15 P.W. Kasteleyn. The statistics of dimers on a lattice: I. The number of dimer arrangements on a quadratic lattice. *Physica*, 27(12):1209–1225, 1961. doi:10.1016/0031-8914(61)90063-5.

16 P.W. Kasteleyn. Graph theory and crystal physics. In F. Harary, editor, *Graph Theory and Theoretical Physics*, pages 43–110. Academic Press, 1967.

17 Tianyu Liu. Torpid mixing of Markov chains for the six-vertex model on $\mathbb{Z}^2$. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, APPROX/RANDOM 2018, pages 52:1–52:15, 2018. doi:10.4230/LIPIcs.APPROX-RANDOM.2018.52.

18 Colin McQuillan. Approximating Holant problems by winding. *CoRR*, abs/1301.2880, 2013. arXiv:1301.2880.

19 Bill Sutherland. Two dimensional hydrogen bonded crystals without the ice rule. *Journal of Mathematical Physics*, 11(11):3183–3186, 1970. doi:10.1063/1.1665111.

20 H. N. V. Temperley and Michael E. Fisher. Dimer problem in statistical mechanics-an exact result. *The Philosophical Magazine: A Journal of Theoretical Experimental and Applied Physics*, 6(68):1061–1063, 1961. doi:10.1080/14786436108243366.

21 Leslie Valiant. Quantum circuits that can be simulated classically in polynomial time. *SIAM Journal on Computing*, 31(4):1229–1254, 2002. doi:10.1137/S0097539700377025.

22 Leslie G. Valiant. Holographic algorithms. *SIAM J. Comput.*, 37(5):1565–1594, February 2008. doi:10.1137/070682575.

Roundtrip Spanners with $(2k-1)$ Stretch

Ruoxu Cen
Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China
cenbo@aliyun.com

Ran Duan
Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China
duanran@mail.tsinghua.edu.cn

Yong Gu
Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China
guyong12@mails.tsinghua.edu.cn

Abstract

A roundtrip spanner of a directed graph G is a subgraph of G preserving roundtrip distances approximately for all pairs of vertices. Despite extensive research, there is still a small stretch gap between roundtrip spanners in directed graphs and undirected graphs. For a directed graph with real edge weights in $[1, W]$, we first propose a new deterministic algorithm that constructs a roundtrip spanner with $(2k-1)$ stretch and $O(kn^{1+1/k}\log(nW))$ edges for every integer $k > 1$, then remove the dependence of size on W to give a roundtrip spanner with $(2k-1)$ stretch and $O(kn^{1+1/k}\log n)$ edges. While keeping the edge size small, our result improves the previous $2k+\epsilon$ stretch roundtrip spanners in directed graphs [Roditty, Thorup, Zwick'02; Zhu, Lam'18], and almost matches the undirected $(2k-1)$-spanner with $O(n^{1+1/k})$ edges [Althöfer et al. '93] when k is a constant, which is optimal under Erdös conjecture.

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

Keywords and phrases Graph theory, Deterministic algorithm, Roundtrip spanners

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.24

Category Track A: Algorithms, Complexity and Games

1 Introduction

A t-spanner of a graph G is a subgraph of G in which the distance between every pair of vertices is at most t times their distance in G, where t is called the stretch of the spanner. Sparse spanner is an important choice to implicitly representing all-pair distances [19], and spanners also have application backgrounds in distributed systems (see [14]). For undirected graphs, $(2k-1)$-spanner with $O(n^{1+1/k})$ edges is proposed and conjectured to be optimal [2, 17]. However, directed graphs may not have sparse spanners with respect to the normal distance measure. For instance, in a bipartite graph with two sides U and V, if there is a directed edge from every vertex in U to every vertex in V, then removing any edge (u, v) in this graph will destroy the reachability from u to v, so its only spanner is itself, which has $O(n^2)$ edges. To circumvent this obstacle, one can approximate the optimal spanner in terms of edge size (e.g. in [9, 3]), or one can define directed spanners on different distance measures. This paper will study directed sparse spanners on roundtrip distances.

Roundtrip distance is a natural metric with good property. Cowen and Wagner [7, 8] first introduce it into directed spanners. Formally, roundtrip distance between vertices u, v in G is defined as $d_G(u \leftrightarrows v) = d_G(u \to v) + d_G(v \to u)$, where $d_G(u \to v)$ is the length of shortest path from u to v in G. For a directed graph $G = (V, E)$, a subgraph $G' = (V, E')$ $(E' \subseteq E)$ is called a t-roundtrip spanner of G if for all $u, v \in G$, $d_{G'}(u \leftrightarrows v) \le t \cdot d_G(u \leftrightarrows v)$, where t is called the stretch of the roundtrip spanner.

In a directed graph $G=(V,E)$ ($n=|V|, m=|E|$) with real edge weights in $[1,W]$, Roditty et al. [16] give a $(2k+\epsilon)$-spanner of $O(\min\{(k^2/\epsilon)n^{1+1/k}\log(nW),\ (k/\epsilon)^2 n^{1+1/k}(\log n)^{2-1/k}\})$ edges. Recently, Zhu and Lam [18] derandomize it and improve the size of the spanner to $O((k/\epsilon)n^{1+1/k}\log(nW))$ edges, while the stretch is also $2k+\epsilon$. We make a step further based on these works and reduce the stretch to $2k-1$. Formally, we state our main results in the following theorems.

▶ **Theorem 1.** *For any directed graph G with real edge weights in $[1,W]$ and integer $k \geq 1$, there exists a $(2k-1)$-roundtrip spanner of G with $O(kn^{1+1/k}\log(nW))$ edges, which can be constructed in $\tilde{O}(kmn\log W)$ time*[1]*.*

By a similar scaling method in [16], we can make the size of the spanner independent of the maximum edge weight W to obtain a $(2k-1)$-spanner with strongly subquadratic space.

▶ **Theorem 2.** *For any directed graph G with real edge weights in $[1,W]$ and integer $k \geq 1$, there exists a $(2k-1)$-roundtrip spanner of G with $O(kn^{1+1/k}\log n)$ edges, which can be constructed in $\tilde{O}(kmn\log W)$ time.*

Actually, our result almost matches the lower bound following girth conjecture. The girth conjecture, implicitly mentioned by Erdös [11], says that for any k, there exists a graph with n vertices and $\Omega(n^{1+1/k})$ edges whose girth (minimum cycle) is at least $2k+2$. This conjecture implies that no algorithm can construct a spanner of $O(n^{1+1/k})$ size and less than $2k-1$ stretch for all undirected graph with n vertices [17]. This lower bound also holds for roundtrip spanners on directed graphs.

Our approach is based on the scaling constructions of the $(2k+\epsilon)$-stretch roundtrip spanners in [16, 18]. To reduce the stretch, we construct inward and outward shortest path trees from vertices in a hitting set [1, 10] of size $O(n^{1/k})$, and carefully choose the order to process vertices in order to make the stretch exactly $2k-1$. To further make the size of the spanner strongly subquatratic, we use a similar approach as in [16] to contract small edges in every scale, and treat vertices with different radii of balls of size $n^{1-1/k}$ differently.

1.1 Related Works

The construction time in this paper is $\tilde{O}(kmn\log W)$. However, there exist roundtrip spanners with $o(mn)$ construction time but larger stretches. Pachoci et al. [13] proposes an algorithm which can construct $O(k\log n)$-roundtrip spanner with $O(n^{1+1/k}\log^2 n)$ edges. Its construction time is $O(mn^{1/k}\log^5 n)$, which breaks the cubic time barrier. Very recently, Chechik et al. [6] give an algorithm which constructs $O(k\log\log n)$-roundtrip spanners with $\tilde{O}(n^{1+1/k})$ edges in $\tilde{O}(m^{1+1/k})$ time.

For spanners defined with respect to normal directed distance, researchers aim to approximate the k-spanner with minimum number of edges. Dinitz and Krauthgamer [9] achieve $\tilde{O}(n^{2/3})$ approximation in terms of edge size, and Bermen et al. [3] improves the approximation ratio to $\tilde{O}(n^{1/2})$.

Another type of directed spanners is transitive-closure spanner, introduced by Bhattacharyya et al. [5]. In this setting the answer may not be a subgraph of G, but a subgraph of the transitive closure of G. In other words, selecting edges outside the graph is permitted. The tradeoff is between diameter (maximum distance) and edge size. One of Bhattacharyya et al.'s results is spanners with diameter k and $O((n\log n)^{1-1/k})$ approximation of optimal edge size [5], using a combination of linear programming rounding and sampling. Berman et al. [4] improves the approximation ratio to $O(n^{1-1/\lceil k/2\rceil}\log n)$. We refer to Raskhodnikova [15] as a review of transitive-closure spanners.

[1] $\tilde{O}(\cdot)$ hides $\log n$ factors.

1.2 Organization

In Section 2, the notations and basic concepts used in this paper will be discussed. In Section 3 we describe the construction of the $(2k-1)$-roundtrip spanner with $O(kn^{1+1/k}\log(nW))$ edges, thus proving Theorem 1. Then in Section 4 we improve the size of the spanner to $O(kn^{1+1/k}\log n)$ and still keep the stretch to $(2k-1)$, thus proving Theorem 2. The conclusion and further direction are discussed in Section 5.

2 Preliminaries

In this paper we consider a directed graph $G=(V,E)$ with non-negative real edge weights w where $w(e)\in[1,W]$ for all $e\in E$. Denote $G[U]$ to be the subgraph of G induced by $U\subseteq V$, i.e. $G[U]=(U,E\cap(U\times U))$. A roundtrip path between nodes u and v is a cycle (not necessarily simple) passing through u and v. The roundtrip distance between u and v is the minimum length of roundtrip paths between u and v. Denote $d_U(u\leftrightarrows v)$ to be the roundtrip distance between u and v in $G[U]$. (Sometimes we may also use $d_U(u\leftrightarrows v)$ to denote a roundtrip shortest path between u,v in $G[U]$.) It satisfies:

- For $u,v\in U$, $d_U(u\leftrightarrows u)=0$ and $d_U(u\leftrightarrows v)=d_U(v\leftrightarrows u)$.
- For $u,v\in U$, $d_U(u\leftrightarrows v)=d_U(u\to v)+d_U(v\to u)$.
- For $u,v,w\in U$, $d_U(u\leftrightarrows v)\le d_U(u\leftrightarrows w)+d_U(w\leftrightarrows v)$.

Here $d_U(u\to v)$ is the one-way distance from u to v in $G[U]$. We use $d(u\leftrightarrows v)$ to denote the roundtrip distance between u and v in the original graph $G=(V,E)$.

In G, a t-roundtrip spanner of G is a subgraph H of G on the same vertex set V such that the roundtrip distance between any pair of $u,v\in V$ in H is at most $t\cdot d(u\leftrightarrows v)$. t is called the *stretch* of the spanner.

For a subset of vertices $U\subseteq V$, given a center $u\in U$ and a radius R, define roundtrip ball $Ball_U(u,R)$ to be the set of vertices whose roundtrip distance on $G[U]$ to center u is strictly smaller than the radius R. Formally, $Ball_U(u,R)=\{v\in U: d_U(u\leftrightarrows v)<R\}$. Then the size of the ball, denoted by $|Ball_U(u,R)|$, is the number of vertices in it. Similarly we define $\overline{Ball}_U(u,R)=\{v\in U: d_U(u\leftrightarrows v)\le R\}$. Subroutine $\texttt{InOutTrees}(U,u,R)$ calculates the edge set of an inward and an outward shortest path tree centered at u spanning vertices in $Ball_U(u,R)$ on $G[U]$. (That is, the shortest path tree from u to all vertices in $Ball_U(u,R)$ and the shortest path tree from all vertices in $Ball_U(u,R)$ to u.) It is easy to see that the shortest path trees will not contain vertices outside $Ball_U(u,R)$:

▶ **Lemma 3.** *The inward and outward shortest path trees returned by* $\texttt{InOutTrees}(U,u,R)$ *only contain vertices in* $Ball_U(u,R)$*.*

Proof. For any $v\in Ball_U(u,R)$, let C be a cycle containing u and v such that the length of C is less than R. Then for any vertex $w\in C$, $d_U(u\leftrightarrows w)<R$, so w must be also in the trees returned by $\texttt{InOutTrees}(U,u,R)$. ◀

For all notations above, we can omit the subscript V when the roundtrip distance is considered in the original graph $G=(V,E)$. Our algorithm relies on the following well-known theorem to calculate hitting sets deterministically.

▶ **Theorem 4** (Cf. Aingworth et al. [1], Dor et al. [10]). *For universe V and its subsets $S_1,S_2,\ldots,S_n$, if $|V|=n$ and the size of each S_i is greater than p, then there exists a hitting set $H\subseteq V$ intersecting all S_i, whose size $|H|\le(n\ln n)/p$, and such a set H can be found in $O(np)$ time deterministically.*

3 A $(2k-1)$-Roundtrip Spanner Algorithm

In this section we introduce our main algorithm constructing a $(2k-1)$-roundtrip spanner with $O(kn^{1+1/k}\log(nW))$ edges for any G. We may assume $k \geq 2$ in the following analysis, since the result is trivial for $k=1$.

Our approach combines the ideas of [16] and [18]. In [18], given a length L, we pick an arbitrary vertex u and find the smallest integer h such that $|\overline{Ball}(u,(h+1)L)| < n^{1/k}|\overline{Ball}(u,h\cdot L)|$, then we include the inward and outward shortest path tree centered at u spanning $\overline{Ball}(u,(h+1)L)$ and remove vertices in $\overline{Ball}(u,h\cdot L)$ from V. We can see that $h \leq k$, so the stretch is $2k$ for u,v with roundtrip distance L, and by a scaling approach the final stretch is $2k+\epsilon$. We observe that if $h=k-1$, $|\overline{Ball}(u,(k-1)L)| \geq n^{(k-1)/k}$, so by Theorem 4 we can preprocess the graph by choosing a hitting set H with size $O(n^{1/k}\log n)$ and construct inward and outward shortest path trees centered at all vertices in H, then we do not need to include the shortest path trees spanning $\overline{Ball}(u,k\cdot L)$. The stretch can then be decreased to $2k-1+\epsilon$. To make the stretch equal $2k-1$, instead of arbitrarily selecting u each time, we carefully define the order to select u.

3.1 Preprocessing

We first define a radius $R(u)$ for each vertex u. It is crucial for the processing order of vertices.

▶ **Definition 5.** *For all $u \in V$, we define $R(u)$ to be the maximum length R such that $|Ball(u,R)| < n^{1-1/k}$, that is, if we sort the vertices by their roundtrip distance to u in G by increasing order, $R(u)$ is the roundtrip distance from u to the $\lceil n^{1-1/k}\rceil$-th vertex.*

For any $u \in V$, $|\overline{Ball}(u,R(u))| \geq n^{1-1/k}$. By Theorem 4, we can find a hitting set H intersecting all sets in $\{\overline{Ball}(u,R(u)) : u \in V\}$, such that $|H| = O(n^{1/k}\log n)$. For all $t \in H$, we build an inward and an outward shortest path tree of G centered at t, and denote the set of edges of these trees by E_0 and include them in the final spanner. This step generates $O(n^{1+1/k}\log n)$ edges in total, and it is easy to obtain the following statement:

▶ **Lemma 6.** *For $u,v \in V$ such that $d(u \leftrightarrows v) \geq R(u)/(k-1)$, the roundtrip distance between u and v in the graph (V,E_0) is at most $(2k-1)d(u \leftrightarrows v)$.*

Proof. Find the vertex $t \in H$ such that $t \in \overline{Ball}(u,R(u))$, that is, $d(u \leftrightarrows t) \leq R(u)$. Then the inward and outward shortest path trees from t will include $d(u \leftrightarrows t)$ and $d(t \leftrightarrows v)$. By $R(u) \leq (k-1)d(u \leftrightarrows v)$, we have $d(u \leftrightarrows t) \leq (k-1)d(u \leftrightarrows v)$ and $d(t \leftrightarrows v) \leq d(t \leftrightarrows u) + d(u \leftrightarrows v) \leq k \cdot d(u \leftrightarrows v)$. So the roundtrip distance of u and v in E_0 is at most $d(u \leftrightarrows t) + d(t \leftrightarrows v) \leq (2k-1)d(u \leftrightarrows v)$. ◀

3.2 Approximating a Length Interval

Instead of approximating all roundtrip distances at once, we start with an easier subproblem of approximating all pairs of vertices whose roundtrip distances are within an interval $[L/(1+\epsilon), L)$. Parameter ϵ is a real number in $(0, 1/(2k-2)]$. The procedure $\texttt{Cover}(G,k,L,\epsilon)$ described in Algorithm 1 will return a set of edges which gives a $(2k-2)(1+\epsilon)$-approximation of roundtrip distance $d(u \leftrightarrows v)$ if $R(u)/(k-1) > d(u \leftrightarrows v)$, for $d(u \leftrightarrows v) \in [L/(1+\epsilon), L)$.

Note that in Algorithm 1, initially $U = V$ and the balls are considered in $G[U] = G$. In the end of every iteration we remove a ball from U, and the following balls are based on the roundtrip distances in $G[U]$. However, $R(u)$ does not need to change during the algorithm and can still be based on roundtrip distances in the original graph G. The analysis for the size of the returned set $\hat{E}$ and the stretch are as follows.

Algorithm 1 Cover$(G(V,E),k,L,\epsilon)$.

```
1: $U \leftarrow V, \hat{E} = \emptyset$
2: while $U \neq \emptyset$ do
3:     $u \leftarrow \arg\max_{u \in U} R(u)$
4:     $step \leftarrow \min\{R(u)/(k-1), L\}$
5:     $h \leftarrow$ minimum positive integer satisfying $|Ball_U(u, h \cdot step)| < n^{h/k}$
6:     Add InOutTrees$(U, u, h \cdot step)$ to $\hat{E}$
7:     Remove $\overline{Ball}_U(u,(h-1)step)$ from $U$
8: end while
9: return $\hat{E}$
```

▶ **Lemma 7.** *The returned edge set of* `Cover`*(G,k,L,ϵ) has $O(n^{1+1/k})$ size.*

Proof. When processing a vertex u, by the selection of h in line 5, $|Ball_U(u, h \cdot step)| < n^{h/k}$ and $|\overline{Ball}_U(u,(h-1)step)| \geq n^{(h-1)/k}$. When $h \geq 2$ it is because of h's minimality, and when $h = 1$ it is because $u \in \overline{Ball}_U(u, 0)$. So each time `InOutTrees` is called, the size of ball to build shortest path trees is no more than $n^{1/k}$ times the size of ball to remove. During an execution of `Cover`(G,k,L,ϵ), each vertex is removed once from U. Therefore the total number of edges added in $\hat{E}$ is $O(n^{1+1/k})$. ◀

We can also see that if the procedure `Cover`$(G[U],k,L,\epsilon)$ is run on a subgraph $G[U]$ induced on a subset $U \subseteq V$, then the size of $\hat{E}$ is bounded by $O(|U|n^{1/k})$. It is also easy to see that h is at most $k-1$:

▶ **Lemma 8.** *The h selected at line 5 in* `Cover`*(G,k,L,ϵ) satisfies $h \leq k-1$.*

Proof. In $G[U]$, the ball $Ball_U(u,(k-1)step)$ must have size no greater than $Ball(u,(k-1)step)$ since the distances in $G[U]$ cannot decrease while some vertices are removed. Since $|Ball(u, R(u))| < n^{1-1/k}$ and $step \leq R(u)/(k-1)$, we get $|Ball_U(u,(k-1)step)| \leq |Ball(u,(k-1)step)| < n^{1-1/k}$, thus $h \leq k-1$. ◀

Next we analyze the roundtrip distance stretch in $\hat{E}$. Note that in order to make the final stretch $2k-1$, for the roundtrip distance approximated by edges in $\hat{E}$ we can make the stretch $(2k-2)(1+\epsilon)$, but for the roundtrip distance approximated by E_0 we need to make the stretch at most $2k-1$ as E_0 stays the same.

▶ **Lemma 9.** *For any pair of vertices u, v such that $d(u \leftrightarrows v) \in [L/(1+\epsilon), L)$, either* `Cover`*$(G,k,L,\epsilon)$'s returned edge set $\hat{E}$ can form a cycle passing through u, v with length at most $(2k-2)(1+\epsilon)d(u \leftrightarrows v)$, or $R(u) \leq (k-1)d(u \leftrightarrows v)$, in which case the E_0 built in Section 3.1 can form a detour cycle with length at most $(2k-1)d(u \leftrightarrows v)$ by Lemma 6.*

Proof. Consider any pair of vertices u, v with roundtrip distance $d = d(u \leftrightarrows v) \in [L/(1+\epsilon), L)$, and a shortest cycle P going through u, v with length d.

During `Cover`(G,k,L,ϵ), consider the vertices on P that are first removed from U. Suppose w is one of the first removed vertices, and w is removed as a member of $\overline{Ball}_{U_c}(c,(h_c-1)step_c)$ centered at c. This is to say $d_{U_c}(c \leftrightarrows w) \leq (h_c-1)step_c$.

Case 1: $step_c > d$. Then

$$d_{U_c}(c \leftrightarrows u) \leq d_{U_c}(c \leftrightarrows w) + d_{U_c}(w \leftrightarrows u) \leq (h_c-1)step_c + d < h_c step_c,$$

and $u \in Ball_{U_c}(c, h_c step_c)$. The second inequality holds because U_c is the remaining vertex set before removing w, so by definition of w, all vertices on P are in U_c. Symmetrically

$v \in Ball_{U_c}(c, h_c step_c)$. `InOutTrees`$(U_c, c, h_c step_c)$ builds a detour cycle passing through u, v with length $< 2h_c step_c$. By Lemma 8, we have $h_c \le k-1$. Also $step_c \le L \le (1+\epsilon)d$, therefore we build a detour of length $< 2(k-1)step_c \le (2k-2)(1+\epsilon)d$ in $\hat{E}$.

Case 2: $step_c \le d$. Because $d < L$, this case can only occur when $step_c = R(c)/(k-1)$. Because c is chosen before u, $R(u) \le R(c) = (k-1)step_c \le (k-1)d$. By Lemma 6, E_0 can give a $(2k-1)$-approximation of d. ◀

3.3 Main Construction

Now we can proceed to prove the main theorem based on a scaling on lengths of the cycles from 1 to $2nW$.

▶ **Theorem 10.** *For any directed graph G with real edge weights in $[1, W]$, there exists a polynomial time constructible $(2k-1)$-roundtrip spanner of G with $O(kn^{1+1/k}\log(nW))$ edges.*

Proof. Note that the roundtrip distance between any pair of vertices must be in the range $[1, 2(n-1)W]$. First do the preprocessing in Section 3.1. Then divide the range of roundtrip distance $[1, 2nW)$ into intervals $[(1+\epsilon)^{p-1}, (1+\epsilon)^p)$, where $\epsilon = 1/(2k-2)$. Call `Cover`$(G, k, (1+\epsilon)^p, \epsilon)$ for $p = 0, \cdots, \lfloor \log_{1+\epsilon}(2nW) \rfloor + 1$, and merge all returned edges with E_0 to form a spanner.

First we prove that the edge size is $O(kn^{1+1/k}\log(nW))$. Preprocessing adds $O(n^{1+1/k} \cdot \log n)$ edges. `Cover`$(G, k, (1+\epsilon)^p, \epsilon)$ is called for $\log_{1+1/(2k-2)}(2nW) = O(k\log(nW))$ times. By Lemma 7, each call generates $O(n^{1+1/k})$ edges. So the total number of edges in the roundtrip spanner is $O(kn^{1+1/k}\log(nW))$.

Next we prove the stretch is $2k-1$. For any pair of vertices u, v with roundtrip distance d, let $p = \lfloor \log_{1+\epsilon} d \rfloor + 1$, then $d \in [(1+\epsilon)^{p-1}, (1+\epsilon)^p)$. By Lemma 9, either the returned edge set of `Cover`$(G, k, (1+\epsilon)^p, \epsilon)$ can form a detour cycle passing through u, v of length at most $(2k-2)(1+\epsilon)d = (2k-1)d$, or the edges in E_0 can form a detour cycle passing through u, v of length at most $(2k-1)d$.

In conclusion this algorithm can construct a $(2k-1)$-roundtrip spanner with $O(kn^{1+1/k} \cdot \log(nW))$ edges. ◀

3.4 Construction Time

The running time of the algorithm in the proof of Theorem 10 is $O(kn(m + n\log n)\log(nW))$. It is also easy to see that the algorithm is deterministic. Next we analyze construction time in detail.

In preprocessing, for any $u \in V$, $R(u)$ can be calculated by running Dijkstra searches with Fibonacci heap [12] starting at u, so calculating $R(\cdot)$ takes $O(n(m + n\log n))$ time. Finding H takes $O(n^{2-1/k})$ time by Theorem 4. Building E_0 takes $O(n^{1/k}\log n \cdot (m + n\log n))$ time.

A `Cover` call's while loop runs at most n times since each time at least one node is removed. In a loop, u can be found in $O(n)$ time, and all other operations regarding roundtrip balls can be done in $O(m + n\log n)$ time by Dijkstra searches starting at u on $G[U]$. Therefore a `Cover` call takes $O(n(m + n\log n))$ time.

`Cover` is called $O(k\log(nW))$ times. Combined with the preprocessing time, the total construction time is $O(kn(m + n\log n)\log(nW))$.

4 Removing the Dependence on W

In this section we prove Theorem 2. The size of the roundtrip spanner in Section 3 is dependent on the maximum edge weight W. In this section we remove the dependence by designing the scaling approach more carefully. Our idea is similar to that in [16]. When we consider the roundtrip distances in the range $[L/(1+\epsilon), L)$, all cycles with length $\leq L/n^3$ have little effect so we can contract them into one node, and all edges with length $> (2k-1)L$ cannot be in any $(2k-1)L$ detour cycles, so they can be deleted. Thus, an edge with length l can only be in $O(\log_{1+\epsilon} n)$ iterations for L between $l/(2k-1)$ and $l \cdot n^3$ (based on the girth of this edge). However, the stretch will be a little longer if we directly apply the algorithm in Section 3 on the contracted graph.

To overcome this obstacle, we only apply the vertex contraction when $R(u)$ is large (larger than $2(k-1)L$). By making the "step" a little larger than L and ϵ smaller, when $d < L < step$, the stretch is still bounded by $(2k-1)$. When $R(u) \leq 2(k-1)L$, we first delete all node v with $R(v) < L/8$, then simply apply the algorithm in Section 3 in the original graph. Since every node u can only be in the second part when $R(u)/2(k-1) \leq L \leq 8R(u)$, the number of edges added in the second part is also strongly polynomial.

First we define the girth of an edge:

▶ **Definition 11.** *We define the* girth of an edge e *in* G *to be the length of shortest directed cycle containing* e*, and denote it by* $g(e)$*.*

It is easy to see that for $e = (u, v)$, $d(u \leftrightarrows v) \leq g(e)$. In $O(n(m + n \log n))$ time we can compute $g(e)$ for all edges e in G [12].

Algorithm 2 approximates roundtrip distance $d(u \leftrightarrows v) \in [L/(1+\epsilon), L)$. In the p-th iteration of the algorithm, $G_p[U_p]$ is always the subgraph contracted from the subgraph $G[U]$. Given $v_p \in U_p$, let $C(v_p)$ be the set of vertices in U that are contracted into v_p. We can see the second part of this algorithm (after line 12) is the same as Algorithm 1 in Section 3.

For the contracted subgraph $G_p[U_p]$, we give new definitions for balls and `InOutTrees`. Given two vertices $u_p, v_p \in U_p$, define

$$\hat{d}_{U_p}(u_p, v_p) = \min_{u \in C(u_p), v \in C(v_p)} d_U(u, v)$$

Balls in $G_p[U_p]$ are defined as follows.

$$Ball_{U_p}(u_p, r) = \{v_p \in U_p : \hat{d}_{U_p}(u_p, v_p) < r\}$$

$$\overline{Ball}_{U_p}(u_p, r) = \{v_p \in U_p : \hat{d}_{U_p}(u_p, v_p) \leq r\}$$

In Line 9, `NewInOutTrees`$(U_p, u_p, h \cdot step)$ is formed by only keeping the edges between different contracted vertices in `InOutTrees`$(U, u, h \cdot step)$ from u (see Line 6). In the inward tree or outward tree of `InOutTrees`$(U, u, h \cdot step)$, if after contraction there are multiple edges from or to a contracted vertex, respectively, only keep one of them. We can see the number of edges added to $\hat{E}$ is bounded by $O(|Ball_{U_p}(u_p, h \cdot step)|)$. Also in U_p, the roundtrip distance from u_p to vertices in $Ball_{U_p}(u_p, h \cdot step)$ by edges in `NewInOutTrees`$(U_p, u_p, h \cdot step)$ is at most $h \cdot step$.

In line 3, we can delete long edges since obviously they cannot be included in $\hat{E}$.

The main algorithm is shown in Algorithm 3.

▶ **Lemma 12.** *For* $k \leq n$ *and* $n \geq 12$*, Algorithm* Spanner(G, k) *constructs a* $(2k-1)$*-roundtrip spanner of* G*.*

Algorithm 2 Cover2(G, k, p, ϵ).

```
1:  $L \leftarrow (1+\epsilon)^p$
2:  Contract all edges $e$ with $g(e) \le L/n^3$ in $G$ to form a graph $G_p$, let its vertex set be $V_p$
3:  (Delete edges $e$ with $g(e) > 2(k-1)L$ from $G_p$)
4:  $U \leftarrow V, U_p \leftarrow V_p, \hat{E} \leftarrow \emptyset$
5:  while $U \neq \emptyset$ and $\max_{u \in U} R(u) \ge 2(k-1)L$ do
6:      $u \leftarrow \arg\max_{u \in U} R(u)$, let $u_p$ be the corresponding vertex in $U_p$
7:      $step \leftarrow (1+1/n^2)L$
8:      $h \leftarrow$ minimum positive integer satisfying $|Ball_{U_p}(u_p, h \cdot step)| < n^{h/k}$
9:      Add NewInOutTrees$(U_p, u_p, h \cdot step)$ to $\hat{E}$
10:     Remove $\overline{Ball}_{U_p}(u_p, (h-1)step)$ from $U_p$, remove corresponding vertices from $U$
11: end while
12: Remove all vertices $u$ from $U$ with $R(u) < L/8$
13: while $U \neq \emptyset$ do
14:     $u \leftarrow \arg\max_{u \in U} R(u)$
15:     $step \leftarrow \min\{R(u)/(k-1), L\}$
16:     $h \leftarrow$ minimum positive integer satisfying $|Ball_U(u, h \cdot step)| < n^{h/k}$
17:     Add InOutTrees$(U, u, h \cdot step)$ to $\hat{E}$
18:     Remove $\overline{Ball}_U(u, (h-1)step)$ from $U$
19: end while
20: return $\hat{E}$
```

Algorithm 3 Spanner$(G(V, E), k)$.

```
Do the preprocessing in Section 3.1. Let $E_0$ be the added edges
$\epsilon \leftarrow \frac{1}{4(k-1)}$.
$\hat{E} \leftarrow E_0$
for $p \leftarrow 0$ to $\lfloor \log_{1+\epsilon}(2nW) \rfloor + 1$ do
    $\hat{E} \leftarrow \hat{E} \cup$ Cover2$(G, k, p, \epsilon)$
end for
return $H(V, \hat{E})$
```

Proof. For any pair of vertices u, v with roundtrip distance $d = d(u \leftrightarrows v)$ on G, there exists a p, such that $d \in [(1+\epsilon)^{p-1}, (1+\epsilon)^p)$. Let $L = (1+\epsilon)^p$. If $R(u) \le (k-1)d$ or $R(v) \le (k-1)d$, by Lemma 6, E_0 contains a roundtrip cycle between u and v with length at most $(2k-1)d$. So we assume $R(u) > (k-1)d$ and $R(v) > (k-1)d$. Also, if there is a vertex w on the shortest cycle containing u and v with $R(w) < L/8$, then there will be a vertex $t \in H$ so that $d(w \leftrightarrows t) < L/8$, so the roundtrip distance in E_0 will be $d(u \leftrightarrows t) + d(t \leftrightarrows v) < L/4 + 2d \le (1+\epsilon)d/4 + 2d < (2k-1)d$ for $k \ge 2$, so Line 12 cannot impact the correctness.

Consider the iteration p of Algorithm 2, let u_p, v_p be the contracted vertices of u, v respectively. Let P be a shortest cycle going through u, v in G and P' be the contracted cycle going through u_p, v_p in G_p. It is easy to see that each vertex on P corresponds to some vertex on P'. Similar as Lemma 8, in Line 8 and Line 16, we have $(k-1) \cdot step \le R(u)$. It is easy to see that $|Ball_{U_p}(u_p, (k-1) \cdot step)| \le |Ball_U(u, (k-1) \cdot step)| < n^{1-1/k}$, which implies $h \le k-1$.

We prove it by the induction on p. When p is small, there is no contracted vertex in G_p. By the same argument as in Lemma 9, either $\texttt{Cover2}(G, k, L, \epsilon)$'s returned edge set $\hat{E}$ contains a roundtrip cycle between u and v with length at most

$$2h_c \cdot \mathit{step}_c \le 2(k-1)(1+1/n^2)(1+\epsilon)d = (2k-3/2)(1+1/n^2)d \le (2k-1)d$$

($k \ge 2$, $k \le n$ and $n \ge 12$) since $\mathit{step}_c \le (1+1/n^2)L$ in Line 7 and Line 15 and $h_c \le k-1$, or E_0 contains a cycle between u and v with length at most $(2k-1)d$. Next we assume that vertices of G contracted in the same vertex in G_p are already connected in $\hat{E}$, and has the $(2k-1)$-stretch.

During $\texttt{Cover2}(G, k, p, \epsilon)$, if some vertices in P' are removed from U_p in Line 10, like Lemma 9, suppose $w_p \in U_p$ is one of the first removed vertices, and w_p is removed as a member of $\overline{\mathit{Ball}}_{U_c}(c, (h_c-1)\mathit{step}_c)$ centered at c. Let $w' \in C(w_p)$ be one vertex on P, since there are at most n original vertices contracted and $\mathit{step}_c = (1+1/n^2)L$, we have $d_U(c \leftrightarrows u) \le d_{U_p}(c \leftrightarrows w_p) + d_U(w' \leftrightarrows u) + n \cdot L/n^3 \le (h_c-1)\mathit{step}_c + d_U(u \leftrightarrows v) + L/n^2 < (h_c-1)\mathit{step}_c + L + L/n^2 = h_c\mathit{step}_c$, and symmetrically $d_U(c \leftrightarrows v) < h_c\mathit{step}_c$. Thus $\texttt{NewInOutTrees}(U_c, c, h_c\mathit{step}_c)$ builds a roundtrip cycle passing through u_p, v_p of length $< 2h_c\mathit{step}_c$ in current contracted graph. It follows that $d_{G_p[\hat{E}]}(u_p, v_p) < 2h_c\mathit{step}_c \le 2(k-1)(1+1/n^2)L$. Since there are at most n contracted vertices in the roundtrip cycle between u_p and v_p, and $w(e) \le g(e)$ for every contracted edge e, we have

$$d_{G[\hat{E}]}(u,v) \le 2(k-1)(1+1/n^2)L + n \cdot (2k-1)L/n^3 \le (2k-3/2)(1+3/n^2)d \le (2k-1)d.$$

($k \ge 2$, $k \le n$ and $n \ge 12$.)

If there is no vertex in P' removed from U_p in Line 10 and Line 12, then all vertices w in P have $L/8 \le R(w) < 2(k-1)L$. By the same argument as in Lemma 9, the second part of Algorithm 2 also ensures that $\hat{E} \cup E_0$ contains a roundtrip cycle passing through u, v with length at most $(2k-1)d$. ◀

▶ **Lemma 13.** *The subgraph returned by algorithm* Spanner(G, k) *has* $O(kn^{1+1/k} \log n)$ *edges.*

Proof. Preprocessing adds $O(n^{1+1/k} \log n)$ edges as in Section 3.1. The edges added in Line 17 is bounded as follows. Consider Algorithm 2, after Line 12, the subgraph only consists of vertices with $R(u) \in [L/8, 2(k-1)L]$, so each vertex belongs to at most $\log_{1+\epsilon} 16k$ such iterations. Thus the total number of edges added after Line 12 is at most $n^{1+1/k} \log_{1+\epsilon} 16k = O(kn^{1+1/k} \log k)$ edges. Next we count the edges added in Line 9.

We remove the directions of all edges in G to get an undirected graph G', and remove the directions of all edges in every G_p to get an undirected graph G'_p, but define the weight of an edge e in G' and every G'_p to be the girth $g(e)$ in G. Let F be a minimum spanning forest of G' w.r.t. the girth $g(e)$. We can see that in iteration p, if we remove edges in F with $g(e) > 2(k-1)(1+\epsilon)^p$ and contract edges e with $g(e) \le (1+\epsilon)^p/n^3$ in F, then the connected components in F will just be the connected components in G'_p, which are the strongly connected components in G_p. This is because of the cycle property of MST: If an edge $e = (u,v)$ in G'_p has $g(e) \le (1+\epsilon)^p/n^3$, then in F all edges f on the path connecting u, v have $g(f) \le (1+\epsilon)^p/n^3$, thus u, v are already contracted in F; If an edge $e = (u,v)$ in G'_p has $g(e) \le 2(k-1)(1+\epsilon)^p$, then in F all edges f on the path connecting u, v have $g(f) \le 2(k-1)(1+\epsilon)^p$, so u, v are in the same component in F.

So the total size of connected components $\{C : |C| \ge 2\}$ in G'_p is at most 2 times the number of edges e in F with $(1+\epsilon)^p/n^3 < g(e) \le 2(k-1)(1+\epsilon)^p$, and every edge in F can be in at most $\log_{1+\epsilon} 2(k-1)n^3 = O(k \log n)$ number of different G'_p. Thus, the total size of connected components with size at least 2 in all G'_p is bounded by $O(kn \log n)$. By a similar argument of Lemma 7, in each call of Cover2(G, k, p, ϵ), line 9 will add $|C|n^{1/k}$ new edges to $\hat{E}$, for every connected component C with $|C| \geq 2$ in G'_p. Thus the total number of edges in the subgraph returned by Spanner(G, k) is bounded by $O(kn^{1+1/k}\log n)$. ◀

Construction Time

The analysis of Spanner's running time is similar to Section 3.4. Compared with Cover, Cover2 adds operations of building G_p. We also need to calculate $g(\cdot)$ in preprocessing, which can done by n Dijkstra searches. G_p can be built in $O(m)$ time. Cover2 is called $\log_{1+\epsilon'}(2nW) = O(k\log(nW))$ times. Therefore the total construction time is still $O(kn(m+n\log n)\log(nW))$.

5 Conclusion

In this paper we discuss the construction of $(2k-1)$-roundtrip spanners with $O(kn^{1+1/k}\log n)$ edges. An important and interesting further direction is whether we can find truly subcubic algorithm constructing such spanners.

References

1 Donald Aingworth, Chandra Chekuri, Piotr Indyk, and Rajeev Motwani. Fast estimation of diameter and shortest paths (without matrix multiplication). *SIAM Journal on Computing*, 28(4):1167–1181, 1999.

2 Ingo Althöfer, Gautam Das, David Dobkin, Deborah Joseph, and José Soares. On sparse spanners of weighted graphs. *Discrete & Computational Geometry*, 9(1):81–100, January 1993. doi:10.1007/BF02189308.

3 Piotr Berman, Arnab Bhattacharyya, Konstantin Makarychev, Sofya Raskhodnikova, and Grigory Yaroslavtsev. Improved approximation for the directed spanner problem. In *International Colloquium on Automata, Languages, and Programming*, pages 1–12, Berlin, Heidelberg, 2011. Springer.

4 Piotr Berman, Sofya Raskhodnikova, and Ge Ruan. Finding sparser directed spanners. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 424–435, Dagstuhl, 2010. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.FSTTCS.2010.424.

5 A. Bhattacharyya, E. Grigorescu, K. Jung, S. Raskhodnikova, and D. Woodruff. Transitive-closure spanners. *SIAM Journal on Computing*, 41(6):1380–1425, 2012. doi:10.1137/110826655.

6 Shiri Chechik, Yang P. Liu, Omer Rotem, and Aaron Sidford. Improved Girth Approximation and Roundtrip Spanners. *arXiv e-prints*, page arXiv:1907.10779, July 2019. arXiv:1907.10779.

7 Lenore J Cowen and Christopher G Wagner. Compact roundtrip routing for digraphs. In *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 885–886, Philadelphia, 1999. SIAM.

8 Lenore J Cowen and Christopher G Wagner. Compact roundtrip routing in directed networks. In *Proceedings of the 19th Annual ACM Symposium on Principles of Distributed Computing*, pages 51–59, New York, 2000. ACM.

9 Michael Dinitz and Robert Krauthgamer. Directed spanners via flow-based linear programs. In *Proceedings of the 43rd Annual ACM Symposium on Theory of Computing*, pages 323–332, New York, 2011. ACM. doi:10.1145/1993636.1993680.

10 Dorit Dor, Shay Halperin, and Uri Zwick. All-pairs almost shortest paths. *SIAM Journal on Computing*, 29(5):1740–1759, 2000.

11 Paul Erdös. Extremal problems in graph theory. In *Theory of Graphs and Its Applications (Proc. Sympos. Smolenice, 1963)*, pages 29–36, Prague, 1964. Publ. House Czechoslovak Acad. Sci.

12 Michael L Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34(3):596–615, 1987.

13 Jakub Pachocki, Liam Roditty, Aaron Sidford, Roei Tov, and Virginia Vassilevska Williams. Approximating cycles in directed graphs: Fast algorithms for girth and roundtrip spanners. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1374–1392, Philadelphia, 2018. SIAM.

14 David Peleg. *Distributed Computing: A Locality-Sensitive Approach*. Society for Industrial and Applied Mathematics, USA, 2000.

15 Sofya Raskhodnikova. Transitive-closure spanners: A survey. In *Property Testing*, pages 167–196. Springer, Berlin, Heidelberg, 2010.

16 Liam Roditty, Mikkel Thorup, and Uri Zwick. Roundtrip spanners and roundtrip routing in directed graphs. *ACM Trans. Algorithms*, 4(3):29:1–29:17, July 2008. doi:10.1145/1367064.1367069.

17 Mikkel Thorup and Uri Zwick. Approximate distance oracles. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, pages 183–192, New York, 2001. ACM.

18 Chun Jiang Zhu and Kam-Yiu Lam. Deterministic improved round-trip spanners. *Information Processing Letters*, 129:57–60, 2018.

19 Uri Zwick. Exact and approximate distances in graphs: a survey. In *European Symposium on Algorithms*, pages 33–48, Berlin, Heidelberg, 2001. Springer.

New Extremal Bounds for Reachability and Strong-Connectivity Preservers Under Failures

Diptarka Chakraborty
National University of Singapore, Singapore
diptarka@comp.nus.edu.sg

Keerti Choudhary
Tel Aviv University, Israel
keerti.choudhary@cs.tau.ac.il

Abstract

In this paper, we consider the question of computing sparse subgraphs for any input directed graph $G = (V, E)$ on n vertices and m edges, that preserves reachability and/or strong connectivity structures.

- We show $O(n + \min\{|\mathcal{P}|\sqrt{n}, n\sqrt{|\mathcal{P}|}\})$ bound on a subgraph that is an 1-fault-tolerant reachability preserver for a given vertex-pair set $\mathcal{P} \subseteq V \times V$, i.e., it preserves reachability between any pair of vertices in $\mathcal{P}$ under single edge (or vertex) failure. Our result is a significant improvement over the previous best $O(n|\mathcal{P}|)$ bound obtained as a corollary of single-source reachability preserver construction. We prove our upper bound by exploiting the special structure of single fault-tolerant reachability preserver for any pair, and then considering the interaction among such structures for different pairs.
- In the lower bound side, we show that a 2-fault-tolerant reachability preserver for a vertex-pair set $\mathcal{P} \subseteq V \times V$ of size $\Omega(n^{\epsilon})$, for even any arbitrarily small ϵ, requires at least $\Omega(n^{1+\epsilon/8})$ edges. This refutes the existence of linear-sized dual fault-tolerant preservers for reachability for any polynomial sized vertex-pair set.
- We also present the first sub-quadratic bound of at most $\widetilde{O}(k\ 2^k\ n^{2-1/k})$ size, for strong-connectivity preservers of directed graphs under k failures. To the best of our knowledge no non-trivial bound for this problem was known before, for a general k. We get our result by adopting the color-coding technique of Alon, Yuster, and Zwick [JACM'95].

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Preservers, Strong-connectivity, Reachability, Fault-tolerant, Graph sparsification

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.25

Category Track A: Algorithms, Complexity and Games

Related Version A full version of the paper is available at `https://arxiv.org/abs/2004.12890`.

Funding *Diptarka Chakraborty*: Supported in part by NUS ODPRT Grant, WBS No. R-252-000-A94-133.
Keerti Choudhary: Supported in part by ERC Grant No. 803118.

Acknowledgements Authors would like to thank Robert Krauthgamer and Greg Bodwin for some useful discussion.

1 Introduction

One of the major problems in computer science, especially in the era of big data, is to *sparsify* input graph while preserving certain properties of it. Let $\wp$ be any property defined over a graph. Given a graph $G = (V, E)$, a subgraph $H = (V, E_H)$, where $E_H \subseteq E$, is said to preserve property $\wp$ if the property $\wp$ is satisfied by the subgraph H if and only if it is satisfied

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 25; pp. 25:1–25:20
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

by graph G. Reachability and strong-connectivity are two fundamental graph properties that we consider in this paper. In case of reachability, given a directed graph G and a set $\mathcal{P}$ of vertex-pairs the objective is to find a subgraph H with as few edges as possible, so that for any pair $(s, t) \in \mathcal{P}$ there is a path from s to t in H iff so is in G. This problem has been studied extensively [16, 10, 1]. In case of strong-connectivity, given a directed graph G the objective is to find a subgraph H with as few edges as possible so that the strongly-connected components in G and H are identical. A folklore result shows that for any n-node graph we can have a strong-connectivity preserving subgraph with at most $2n$ edges.

In this paper we study the above two problems under the possibility of edge or vertex failures. In the real world networks are prone to failures. Most of the time such failures are unavoidable and also unpredictable in physical systems like communication or road networks. Due to this reason edge (or vertex) failure model draws a huge attention of the researchers in the recent past. In most of the scenarios such failures are much smaller in number in comparison to the size of the graph. Thus it is natural to associate a parameter to capture the number of edge (or vertex) failures, and then try to build fault-tolerant data-structures of size depending on this failure parameter for various graph theoretic problems. Many natural graph theoretic questions like connectivity [30, 28, 5, 25], finding shortest paths [18], graph-structures preserving approximate distances [26, 17, 13, 19, 7, 8, 11, 4] etc. have been studied in the presence of edge (or vertex) failures.

The main focus of this paper is to understand the extremal structure of paths in directed graphs under possible (bounded) edge failures. More specifically, our goal is to show existence (or non-existence) of subgraphs of certain size[1] that preserves reachability and strong-connectivity in the presence of a small number of edge failures.

▶ **Definition 1** (Fault-tolerant Strong-Connectivity Preserver (FT-SCC Preserver)). *For any graph $G = (V, E)$, a subgraph H of G is said to be k-fault-tolerant strong-connectivity preserver (k-FT-SCC preserver) if for each set F $(\subseteq E)$ of k edge failures, the strongly-connected components in $G - F$ and $H - F$ are identical.*

We would like to emphasize that so far there is no non-trivial bound on the size of k-FT-SCC preserver. In this paper we show an upper bound of $\widetilde{O}(k\ 2^k\ n^{2-1/k})$ on k-FT-SCC preserver of any n-node (directed) graph. Moreover we show that we can find such a subgraph efficiently.

▶ **Theorem 2.** *There is a polynomial time (randomized) algorithm that given any directed graph $G = (V, E)$ on n vertices and $k \geq 1$, computes a k-FT-SCC preserver of G containing at most $\widetilde{O}(k\ 2^k\ n^{2-1/k})$ edges with probability at least $1 - 1/n^4$.*

As a direct application we get a data-structure (oracle) of size sub-quadratic in n, for reporting strongly-connected components after k failures. The best earlier known bound for $k > 1$ failures was $\Omega(2^k n^2)$ [6], whereas for $k = 1$ it was known by Georgiadis, Italiano, and Parotsidis [24] that $O(n)$ space and query-time bound is achievable. So ours is the first truly sub-quadratic (i.e., $O(n^{2-\epsilon})$ for some $\epsilon > 0$) sized strong-connectivity oracle for any constant number of failures. One may observe that there are graphs with n nodes for which any k-FT-SCC preserver must be of size $\Omega(2^k n)$.

Next we study the extremal bounds of fault-tolerant reachability preserving subgraphs.

[1] Throughout this paper by *size* of a subgraph we mean the number of edges present in that subgraph.

▶ **Definition 3** (Fault-tolerant Pairwise Reachability Preserver). *For any graph $G = (V, E)$ and a set $\mathcal{P}$ of vertex-pairs, a subgraph H of G is said to be k-fault-tolerant pairwise reachability preserver for $\mathcal{P}$, denoted by $FTRS(\mathcal{P}, G)$, if for each F $(\subseteq E)$ of k edge failure, the reachability relations between pairs in $\mathcal{P}$ agree on $G - F$ and $H - F$.*

Baswana, Choudhary and Roditty [5] provided a polynomial time algorithm that given any n-node directed graph constructs an $O(2^k n)$-sized subgraph that preserves reachability from a fixed source vertex to all other vertices under k edge failures. As a corollary, to preserve reachability between arbitrary $\mathcal{P}$ pairs, we get an $O(2^k n|\mathcal{P}|)$-sized k-fault-tolerant pairwise reachability preserver. Clearly the bound is extremely bad for large sized set $\mathcal{P}$. So far we do not know any better bound even for small values of k. On the other hand in standard static setting (i.e., when $k = 0$) we know existence of $O\big(n + (n|\mathcal{P}|)^{2/3}\big)$-sized pairwise reachability preserver [1].

An important question is how much the size of a reachability preserver varies when we go from standard static (i.e., without any failure) setting to single failure setting, and then further from single failure to dual failure setting. It is also natural to ask the following question: What is the bound on the number of pairs in $\mathcal{P}$ so that it is possible to obtain linear sized single-fault-tolerant pairwise reachability preservers. Here we show that this is possible as long as $|\mathcal{P}| = O(\sqrt{n})$. Note that this is also the current best known limit for the standard static setting [1]. So one cannot hope to improve our bound without improving the bound for static setting. Below we state our upper bound result.

▶ **Theorem 4.** *For any directed graph $G = (V, E)$ with n vertices, and a set $\mathcal{P}$ of vertex-pairs, there exists a single-fault pairwise reachability preserver $FTRS(\mathcal{P}, G)$ having at most $O\big(n + \min(|\mathcal{P}|\sqrt{n},\ n\sqrt{|\mathcal{P}|})\big)$ edges. Furthermore, we can find such a subgraph in polynomial time.*

Our construction of SCC preservers plays a significant role in obtaining such a sparse reachability preserver. One may wonder whether the above result can be generalized to multiple failures, at least for constantly many failures. Unfortunately, we observe a striking difference between single and multiple (even for two) failures scenario in the context of pairwise reachability preservers.

▶ **Theorem 5.** *For every n, p with $p = O(n^{2/3})$, there is an infinite family of n-node directed graphs and pair-sets $\mathcal{P}$ of size p, for which a dual fault-tolerant pairwise reachability preserver requires at least $\Omega(n|\mathcal{P}|^{\frac{1}{8}})$ edges.*

This shows linear size reachability preserver is not possible under dual failures in general pairwise setting with the number of pairs being $\Omega(n^\epsilon)$ for any small $\epsilon > 0$. As a consequence we get a polynomial separation in size of a pairwise reachability preserver between single and dual failures. This is in sharp contrast with single-source all destinations setting, wherein, the size only doubles each time we increase the count of failure by value one [5].

It is worth mentioning that in this paper we show fault-tolerant structures with respect to edge failures only, however, all our results hold for vertex-failures as well.

1.1 Related Work

A simple version of reachability preserver is when there is a single source vertex s and we would like to preserve reachability from s to all other vertices. Baswana et al. [5] provided an efficient construction of a k-fault-tolerant single-source reachability preserver of size $O(2^k n)$. Further they showed that this upper bound on size of a preserver is tight up to some constant factor. As an immediate corollary of their result, we get a k-fault-tolerant pairwise reachability preserver of size $O(2^k n|\mathcal{P}|)$ (by applying the algorithm of [5] to find subgraph for each source vertex in pairs of $\mathcal{P}$, and then taking the union of all these subgraphs). We do not know whether this bound is tight for general k. However for standard static (with no faulty edges) setting much better bound is known. We know that even to preserve all the pairwise distances, not just reachability, there is a subgraph of size $O\big(n + \min(n^{2/3}|\mathcal{P}|, n\sqrt{|\mathcal{P}|})\ \big)$ [16, 10]. Later Abboud and Bodwin [1] showed that for any directed graph $G = (V, E)$ given a set S of source vertices and a pair-set $\mathcal{P} \subseteq S \times V$ we can construct a pairwise reachability preserver of size $O\big(n + \min(\sqrt{n|\mathcal{P}||S|}, (n|\mathcal{P}|)^{2/3})\ \big)$. It is further shown that for any integer $d \geq 2$ there is an infinite family of n-node graphs and vertex-pair sets $\mathcal{P}$ for which any pairwise reachability preserver must be of size $\Omega\big(n^{2/(d+1)}|\mathcal{P}|^{(d-1)/d}\big)$. Note, for undirected graph storing spanning forests is sufficient to preserve pairwise reachability information, and thus we can always get a linear size reachability preserver for undirected graphs. We would like to emphasize that all the results provided in this paper hold for directed graphs. A problem similar to constructing reachability preserver is to construct a data-structure (aka. oracle) that can answer queries of the form whether a vertex is reachable from a fixed source vertex s after multiple edge (or vertex) failures. As an application of [5] we get such an oracle of size $O(2^k n)$ for k edge (or vertex) failures with query time $O(2^k n)$. For just dual failures we have an $O(n)$ size oracle with $O(1)$ query time due to [15]. In a recent work, Brand and Saranurak [31] obtained a k-fault-tolerant $\mathcal{O}(n^2)$ sized reachability oracle that has $O(k^\omega)$ query time, where ω is the constant of matrix-multiplication.

Finding strongly connected components (SCCs) under edge failures is another important problem. One specific problem is given a directed graph G to build a data-structure (oracle) that for any vertices u, v and a set of edges F of size k can answer whether u and v are in the same SCC in $G - F$. Using k-fault-tolerant reachability preserver of [5] we can get such an oracle of size $O(2^k n^2)$ with query time $O(2^k n)$ (see [6]). Moreover, [6] provides us an algorithm that computes all the SCCs in $G - F$ in time $O(2^k n \log^2 n)$ by using a data-structure of size $O(2^k n^2)$. Georgiadis, Italiano and Parotsidis [24] also studied this problem of computing all the SCCs under single edge failure, and gave a solution with $O(n)$ query time using a data-structure of size only $O(n)$. However so far we do not know any solution for computing all the SCCs after more than one edge failures using a data structure of size $O(n^{2-\epsilon})$ for any $\epsilon > 0$. In this paper we give construction of first such truly sub-quadratic sized data-structure as long as there are only constantly many failures. For undirected graphs, the optimal bound of $O(kn)$ edges for k-fault-tolerant connectivity preserver directly follows from k-edge (vertex) connectivity certificate constructions provided by Nagamochi and Ibaraki [27]. In contrast, for directed graphs, the only known result for "sparse" certificate of k-edge (vertex) strong-connectivity is for $k = 2$, due to a series of works by Georgiadis et al. [22, 23, 21]. Our truly sub-quadratic sized k-FT-SCC preservers also in turn provides the *first* truly sub-quadratic sized k-edge (vertex) strong-connectivity certificates for directed graphs, for $k \geq 3$ (see Section 6).

Other closely related problems that have been studied in the fault-tolerant model include computing distance preservers [18, 29, 28], depth-first-search tree [3], spanners [13, 19], approximate distance preservers [7, 30, 9], approximate distance oracles [20, 14], compact routing schemes [14, 12].

1.2 Technical Overview

SCC preserver. Our starting point is a simple construction, which is motivated from some of the techniques used in [24], of linear (in number of vertices) sized single fault-tolerant SCC oracle. Then by using that construction as a basic building block we provide a construction of fault-tolerant SCC preserver for k edge failures. Using the ideas inspired by color-coding technique of Alon, Yuster and Zwick [2], we show a generic procedure that converts any r-fault-tolerant (or even non-fault-tolerant) SCC preserver construction into a $(k+r)$-FT-SCC preserver construction. Our technique, especially the first step of our conversion procedure, is quite similar to that used in [19] to convert any spanner to a r-fault-tolerant spanner with the same *stretch*. The first step alone cannot serve our purpose fully, mostly because it could work (with high probability) only when the SCCs after k faults are of "small size". To mitigate this issue we have to handle the large sized SCCs (after k failures) with a completely different technique. As a consequence our whole proof becomes slightly more intricate than that in [19].

Our conversion procedure works as follows. In the first step, we sample a set J of edges and treat J as failure set, and then compute r-fault-tolerant SCC preserver of $G - J$ (residual graph after removing edges in J). Do this multiple times and take union of all those r-FT SCC preservers for different random choices of J. Now in the second step, we sample a set W of "a few" vertices of G, and then for each $w \in W$ compute single-source FTRS with w as the source and single-destination FTRS by treating w as the destination. Then take union of all these FTRS subgraphs. Finally we claim that with high probability the union of subgraphs produced by first and the second step is a $(k + r)$-FT-SCC preserver.

Our correctness proof proceeds as a win-win analysis. For the sake of simplicity let us provide a high level proof sketch for the case when $r = 0$. For any set F of edge failures, we distinguish two cases depending on whether a SCC C in $G - F$ is small or not. Our choice on size of J ensures that we over-samples F during the first step. Hence if C is of small size with high probability at least for one random choice of J, C will be completely inside one strongly connected component after removing J (that also includes F) from G (in other words, J "separates" C from F), leading to C also being a SCC in the final subgraph (after failure of edges in F). Next we turn to the case when C is of large size. In that scenario it is not difficult to show that with high probability W (chosen at random during second step) and C have some common vertex, and hence C will be preserved due to inclusion of single-source and single-destination FTRS structure. Our techniques hold even when we are able to "partially separate" C from F, and that helps us in proving our result for any r (see Section 3 for the details). So we get that any improvement in size of r-FT-SCC preserver will directly improve the size of $(k + r)$-FT-SCC preserver.

FT-Reachability-Preserver. The construction of $O(n + \sqrt{n}|\mathcal{P}|)$-sized reachability-preserver for a pair-set $\mathcal{P}$ uses the fact the preservers on general digraphs are reducible to preserver on DAGs (since the SCCs can always be compressed into "supernodes", and there is a linear size certificate for strong-connectivity). In a DAG, it is not very difficult to ensure that paths between two given pairs meet and diverge only once, which in turn provides a cap on the maximum number of edges in a preserver. Our approach to FT-reachability is to try to adapt the constructions for non-faulty setting [1]. However, one major hindrance we face is that we cannot directly compress SCCs into "supernodes" as they can destroy 2-connectivity structures, and thus the problem is not reducible to DAGs. We start by observing that FTRS(p) for a pair $p = (s, t)$ is just union of two "maximally disjoint" $s - t$ paths. The interactions between FTRS(p) for different pairs p help us in achieving our bound of $O(n + \sqrt{n}|\mathcal{P}|)$. Our second upper bound of $O(n\sqrt{|\mathcal{P}|})$ is much simpler than the first one. Again we consider union of FTRS(p) structures for all $p \in \mathcal{P}$, and then consider all the vertices that appear in more than $\sqrt{|\mathcal{P}|}$ "maximally disjoint" paths in total (in all FTRS structures). Next we remove all those paths and add single-source and single-destination FTRS structure from those selected vertices. Then we use the properties of single-source and single-destination FTRS to show that the final subgraph will be a FTRS for the pair-set $\mathcal{P}$.

These structures as well as linear bound on preserver size for small sized $\mathcal{P}$ (at most $O(\sqrt{n})$ pairs), is not expendable beyond single failure due to the fact that k-FTRS(p) for any $k > 1$ cannot be represented as union of $o(n)$ paths (see [15]). In the latter part of this paper we also show that this is not a drawback of our approach, instead in some sense it is unavoidable, by proving a size lower bound of any k-fault-tolerant pairwise preserver, for $k \geq 2$. Our lower bound for dual failure is inspired by the following observation: If for a pair $p = (s, t)$, $Q_p = (q_1, \dots, q_\ell)$ and $R_p = (r_1, \dots, r_t)$ are two vertex-disjoint paths from s to t. Then by playing with failures on Q and R, we can force multiple paths originating from Q and terminating to R to be present in our 2-FTRS for p. Note that a 2-FTRS, for a single pair p, would still be linear in size. However, as the number of pairs increases achieving sparsity is tricky. To obtain a 2-FTRS lower-bound for multiple pairs we embed in between the paths Q_p and R_p, the "hard" distance preserver graph given by Coppersmith and Elkin [16]. We start with a lower bound distance preserver graph G over pair-set $\mathcal{P}$, and perform its layering L number of times, for some parameter L. Inspired by techniques of Bodwin et al. [10, 1], we are able to show that all the paths in the "hard" instance graph from [16] can be assumed to have equal distance between the relevant pairs. Thus our layered embedded structure also acts as a non-faulty reachability preserver among pairs with end-points respectively on paths Q_p and R_p.

2 Preliminaries and Tools

Given a directed graph $G = (V, E)$ on $n = |V|$ vertices and $m = |E|$ edges, the following notations will be used throughout the paper.

- $H[A]$: The subgraph of H induced by vertices in set A.
- G^R : The graph obtained by reversing all the edges in graph G.
- $H - F$: For a set of edges F, the graph obtained by deleting the edges in F from graph H.
- $\pi(x, y, H)$: The shortest path from x to y in graph H.
- $P \circ Q$: The concatenation of two paths P and Q, i.e., a path that first follows P and then Q.
- $T(v)$: The subtree of a directed tree T rooted at a vertex $v \in T$.
- $cert(C, H)$: An arbitrarily chosen certificate of at most $2(|C| - 1)$ edges corresponding to a strongly connected component C in H.

Our algorithm for computing *pairwise-reachability* and *strong-connectivity* preservers in a fault tolerant environment employs the concept of a *single-source* FTRS which is a sparse subgraph that preserves reachability from a designated source vertex even after the failure of at most k edges in G. Observe that in case of no failure, a directed reachability tree has $n - 1$ edges and is able to preserve reachability from the source which is a also the root. An FTRS with respect to a given source is formally defined as follows.

▶ **Definition 6** (FTRS). *Let $\mathcal{P} \subseteq V \times V$ be any set of pairs of vertices. A subgraph H of G is said to be a k-Fault-Tolerant Reachability-Subgraph of G for $\mathcal{P}$ if for any pair $(s, t) \in \mathcal{P}$ and for any subset $F \subseteq E$ of k edges, t is reachable from s in $G - F$ if and only if t is reachable from s in $H - F$. Such a subgraph H is denoted by k-FTRS$(\mathcal{P}, G)$, or simply FTRS$(\mathcal{P}, G)$ when $k = 1$.*

Baswana et al. [5] provide a construction of sparse FTRS for any general $k \geq 1$ when there is a designated source vertex.

▶ **Theorem 7** ([5]). *For any directed graph $G = (V, E)$, a designated source vertex $s \in V$, and an integer $k \geq 1$, there exists a (sparse) subgraph H of G which is a k-FTRS$(\{s\} \times V, G)$ and contains at most $2^k n$ edges. Moreover, such a subgraph is computable in $O(2^k mn)$ time, where n and m are respectively the number of vertices and edges in graph G.*

Our constructions will require the knowledge of the vertices reachable from a vertex s as well as the vertices that can reach s. So we will be using FTRS defined with respect to a source vertex ($\{s\} \times V$ case), as well as FTRS defined with respect to a destination vertex ($V \times \{s\}$ case).

In this paper, we consider fault-tolerant structures with respect to edge failures only. Vertex failures can be handled by simply splitting a vertex v into an edge (v_{in}, v_{out}), where the incoming and outgoing edges of v are respectively directed into v_{in} and directed out of v_{out}.

3 Strong-connectivity Preservers

We show a construction of strong-connectivity preservers that are able to preserve strong-connectivity relation between vertices in $G = (V, E)$ as long as the number of failures are bounded by k. For our convenience, we assume that G is strongly connected, if not, we may apply our construction to each strongly-connected component (SCC) of G. Although the main contribution of this section is to get a fault-tolerant SCC preserver for general k failures, let us start with the case when there can be at most one edge failure.

3.1 Construction for single failure

We first give a simple construction of an $O(n)$ size FT-SCC preserver for the scenario of $k = 1$. Let s be an arbitrary vertex in G. We initialize H_1 to union of subgraphs FTRS$(\{s\} \times V, G)$ and FTRS$(V \times \{s\}, G)$. The following simple observation describes the significance of H_1 in preserving strong connectivity information.

▶ **Observation 8.** *Given a directed graph G, let H_1 be the union of subgraphs FTRS$(\{s\} \times V, G)$ and FTRS$(V \times \{s\}, G)$. For any vertex x and any edge-failure e, if x and s are strongly connected in $G - \{e\}$, then they are also strongly connected in $H_1 - \{e\}$.*

We now introduce a lemma that will be crucial in preserving strong-connectivity between vertices not (strongly) connected to s after failure.

▶ **Lemma 9.** *For any n-vertex directed graph $G = (V, E)$ and an ordered list $L = (v_1, v_2, \ldots, v_n)$ of vertices of G, in polynomial time we can compute a subgraph $H_0 = H_0(L)$ of G with at most $2n$ edges satisfying the condition that the SCCs of $G[v_1 \cdots v_i]$ are identical to those in $H_0[v_1 \cdots v_i]$, for $1 \leq i \leq n$.*

Proof. For any $i \in \{1, \cdots, n\}$, let $V_i = \{v_1, \ldots, v_i\}$ be the subset of V comprising of first i vertices, and G_i be the subgraph of G induced by the set V_i. We initialize H_0 to be an empty graph on n vertices. The edges of H_0 are incrementally computed in n rounds, wherein, in the i^{th} round we add edges to H_0, so as to ensure that the SCCs of $H_0[v_1 \cdots v_i]$ are identical to those in G_i.

For any $i \in \{1, \cdots, n\}$, let γ_i denote the number of SCCs in graph G_i, and let C_i be the SCC of v_i in graph G_i. Observe that $\gamma_i \leq 1 + \gamma_{i-1}$, where the equality holds for index i if and only if $C_i = \{v_i\}$. If $C_{i,1}, C_{i,2}, \ldots, C_{i,\ell_i}$ is a decomposition of SCC C_i in $G_{i-1} (= G_i - v_i)$, then in round i it suffices to add at most $2\ell_i$ edges corresponding to an out-reachability and an in-reachability tree rooted at v_i and spanning the "super-nodes" (that is obtained by contracting all the edges in a component $C_{i,j}$) $C_{i,1}, C_{i,2}, \ldots, C_{i,\ell_i}$. Now $\ell_i = 1 + \gamma_{i-1} - \gamma_i$. Thus the number of edges in H_0 is at most $2(\ell_2 + \ldots + \ell_n) = 2(n - 1 + \gamma_1 - \gamma_n) \leq 2n$. ◀

Construction procedure of 1-FT-SCC preserver

Consider an arbitrarily chosen vertex s in G. Then by treating s as a source vertex, we compute the directed reachability-tree T rooted at s for graph G. Similarly we compute a reachability-tree T' for reverse graph G^R. Let L (resp. L') represent an ordered list containing the vertices of T (resp. T') sorted in the decreasing order of their depth (where vertices in the same depth are in an arbitrary order). Next we compute the subgraphs $H_0(L)$ and $H_0(L')$ with the property mentioned in Lemma 9, and finally set H to be the union of graphs H_1 (as defined in Observation 8), $H_0(L)$, and $H_0(L')$.

It is easy to see that H contains $O(n)$ edges. We now prove the correctness.

Consider a vertex x in G and a failing edge $e = (a, b)$ such that x and s are not strongly-connected in $G - \{e\}$. Let C_x be the SCC of x in $G - \{e\}$. We will show that C_x must be an SCC in at least one of the graphs: $H_0(L) - \{e\}$ or $H_0(L') - \{e\}$.

Observe that x is either not reachable from s in $G - \{e\}$, or does not have a path to s in $G - \{e\}$. Without loss of generality, we assume that the first case holds. Thus $e = (a, b)$ must lie on the tree-path from s to x in T. Then a is a parent of b in T. Observe that since none of the vertices of C_x can be reachable from s in $G - \{e\}$, the entire SCC C_x must lie in the subtree rooted at b, denoted by T_b. Since L stores vertices of T sorted in the decreasing order of depth, the vertices of subtree T_b (and hence also C_x) appears before a in the list L. This implies that C_x must be an SCC in $H_0(L) - \{(a, b)\}$. This completes the correctness.

So we conclude with the following theorem.

▶ **Theorem 10.** *There is a polynomial time (deterministic) algorithm that given any directed graph $G = (V, E)$ on n vertices, computes an 1-FT-SCC preserver of G with at most $O(n)$ edges.*

3.2 A generic construction

In this section we provide a construction for general k failures.

▶ **Lemma 11.** *If there is an algorithm $\mathcal{A}$ that on every n-node directed graph builds a r-fault-tolerant SCC preserver of size $f(n, r)$, then for any $k = \Omega(r)$, there is a randomized algorithm $\mathcal{B}$ that given a directed graph G and a parameter $\alpha \in [0, 1]$, computes a $(k + r)$-FT-SCC preserver of size $O(k2^{k+r} \cdot n^{2-\alpha} \log n + n^{k\alpha} \cdot \log n \cdot f(n, r))$ with high probability. Moreover, if $\mathcal{A}$ runs in time $T(n)$ then the algorithm $\mathcal{B}$ runs in time poly$(n)T(n)$.*

Description of Procedure $\mathcal{B}$ to compute $(k + r)$-FT-SCC preserver

Let $\alpha \in [0, 1]$ be the input parameter. Procedure $\mathcal{B}$ constructs two graphs H_1 and H_2 as follows.

- $\boldsymbol{H_1}$ **:** Repeat the following for $L = 16 \cdot n^{k\alpha} \cdot \log n$ number of iterations: Independently add each edge of G to a set J with probability $p = \frac{2}{n^\alpha}$, and then use the given algorithm $\mathcal{A}$ to compute a r-FT-SCC preserver of the remaining graph $G - J$. Set H_1 to be union of these r-FT-SCC preservers, taken over all L iterations.
- $\boldsymbol{H_2}$ **:** Let $q = 16(k + r) \cdot n^{-\alpha} \log n$, and W be a uniformly random set of $nq = \big(16(k + r) \cdot n^{1-\alpha} \log n\big)$ vertices in G. Initialize H_2 to be union of $(k + r)$-FTRS$(\{w\} \times V, G)$ and $(k + r)$-FTRS$(V \times \{w\}, G)$, taken over all $w \in W$.

Finally procedure $\mathcal{B}$ outputs a new subgraph H which is union of H_1 and H_2.

Correctness of Procedure $\mathcal{B}$

For a set F of edge failures and a vertex x in G, let $C_{x,F}$ be the SCC containing x in $G-F$. We say that the SCC $C_{x,F}$ is small if it contains at most $\frac{n^\alpha}{4}$ vertices, and large otherwise.

First we will consider the scenario that $C_{x,F}$ is small. Let F_1 and F_2 be any two disjoint subsets of E of size respectively k and r, and let $F = F_1 \cup F_2$. (Observe that C_{x,F_1} might be large even though $C_{x,F}$ is small). Let $cert(C_{x,F}, G-F)$ be an arbitrary certificate of at most $2(|C_{x,F}|-1)$ edges corresponding to a strongly connected component of $C_{x,F}$. We say an iteration *separates* $C_{x,F}$ from F_1 if at that iteration none of the edges of $cert(C_{x,F}, G-F)$ is selected in J, but all the edges of F_1 lie in J. The probability that a particular iteration separates $C_{x,F}$ from F_1 is:

$$(1-p)^{|cert(C_{x,F},G-F)|} \cdot p^{|F_1|} \geq \left(1-\frac{2}{n^\alpha}\right)^{2(n^\alpha/4)} \cdot \left(\frac{2}{n^\alpha}\right)^k \geq \frac{1}{4} \cdot \frac{2^k}{n^{k\alpha}} \ .$$

The probability that none of the iterations is able to separate $C_{x,F}$ from F_1 is at most

$$\left(1-\frac{2^k}{4n^{k\alpha}}\right)^{16n^{k\alpha}\cdot \log n} \leq \frac{1}{n^{4(2^k)}} \ .$$

Now, there are $n^{O(k)}$ (assuming $k = \Omega(r)$) choices for pair (F_1, F_2), and n choices for x, thus a total of $n^{O(k)}$ different choices for the triplet $(C_{x,F}, F_1, F_2)$. By union bound, the probability that none of the L iterations are able to separate $C_{x,F}$ from F_1, for at least one choice of $(C_{x,F}, F_1, F_2)$, is at most: $\frac{n^{O(k)}}{n^{4(2^k)}} \leq \frac{1}{n^5}$.

The next claim is immediate from definition of FT-SCC preserves.

▷ Claim 12. Let $F_1, F_2, J \subseteq E$ where J contains F_1, F be $F_1 \cup F_2$, and C be an SCC in $G-F$ whose certificate is disjoint with J. Further let $\widetilde{H}$ be a r-FT-SCC preserver of $G-J$. Then C is also an SCC in $\widetilde{H} - F_2$ if $|F_2| \leq r$.

The above discussion together with Claim 12 completes the analysis of the scenario when the SCCs are small, and we obtain the following lemma.

▶ **Lemma 13.** *H_1 with high probability preserves small SCCs after $k+r$ failures.*

Next, we consider the scenario that $C_{x,F}$ is large, i.e, contains more than $\frac{n^\alpha}{4}$ vertices. Let F be a set of $k+r$ edge failures, and $C_{x,F}$ be the SCC of x in $G-F$. In order to ensure that the SCC $C_{x,F}$ is preserved in $H_2 - F$ it suffices to ensure that $C_{x,F}$ has non-empty intersection with W. This is because we include in H_2 the $(k+r)$-fault-tolerant in-and-out-reachability preserves of each of the vertices of W in H_2. The probability[2] that none of the vertices of $C_{x,F}$ lie in random set W is at most:

$$(1-q)^{|C_{x,F}|} \leq \left(1-\frac{16(k+r)\log n}{n^\alpha}\right)^{n^\alpha/4} \leq \frac{1}{n^{4(k+r)}} \ .$$

Again, there are a total of $n^{2(k+r)+1}$ different choices for the pair $(C_{x,F}, F)$. By union bound, the probability that for at least one choice of (F, x), there is some large SCC $C_{x,F}$ having empty intersection with W is at most: $\frac{n^{2(k+r)+1}}{n^{4(k+r)}} \leq \frac{1}{n^5}$. This completes the analysis of the scenario when the SCCs are large.

[2] It is easy to verify that the bound of $\frac{1}{n^{4(k+r)}}$ holds for both the scenarios: sampling with replacement by probability q, or just taking W to be a uniformly random subset of vertices of nq size.

▶ **Lemma 14.** *H_2 with high probability preserves large SCCs after $k+r$ failures.*

Now we are ready to prove Lemma 11.

Proof of Lemma 11. Recall, H is the graph obtained by taking the union of the graphs H_1 and H_2. From Lemma 13 and Lemma 14, it follows that our construction results in a valid $(k+r)$-fault-tolerant SCC preserver. The total number of edges in H is $O(k2^{k+r} \cdot n^{2-\alpha}\log n + n^{k\alpha} \cdot \log n \cdot f(n,r))$. ◀

Now as a corollary of Lemma 11, we directly obtain a k-FT-SCC preserver with sub-quadratic in n edges, since we know $f(n,0) = O(n)$. However to get even better bound we use $f(n,1) = O(n)$ by Theorem 10. On substituting $r = 1$ and $\alpha = 1/(k+1)$ in Lemma 11, we obtain that a $(k+1)$-FT-SCC preserver has at most $\widetilde{O}(k\ 2^k\ n^{2-1/(k+1)})$ edges. Thus the following theorem is immediate.

▶ **Theorem 15.** *For every digraph $G = (V,E)$ on n vertices and every $k \geq 1$, there is a polynomial time (randomized) algorithm that with probability at least $1 - 1/n^4$ computes a k-fault-tolerant SCC preserver of G with at most $\widetilde{O}(k\ 2^k\ n^{2-1/k})$ edges.*

4 Reachability Preservers

In this section we will focus on finding a sparse pairwise reachability preserver. Recall that, for a directed graph $G = (V,E)$ and a set $\mathcal{P}$ of vertex-pairs, a 1-fault tolerant reachability subgraph of G is a subgraph H that preserves the reachability information between all pairs of vertices in $\mathcal{P}$ under single edge failure. We denote such a subgraph by FTRS($\mathcal{P}, G$), or simply FTRS($\mathcal{P}$) if the underlying graph G is clear from the context.

4.1 Upper Bound I

Let us start by showing an existential upper-bound on the number of edges present in an optimum sized FTRS.

▶ **Theorem 16.** *For any directed graph $G = (V,E)$ with n vertices, m edges, and a set $\mathcal{P}$ of vertex-pairs, there exists a FTRS($\mathcal{P}, G$) (or simply FTRS($\mathcal{P}$)) that contains $O(n + |\mathcal{P}|\sqrt{n})$ edges.*

As a corollary of the above theorem we get a FTRSof linear (in number of vertices) size whenever number of pairs for which we have to preserve reachability is at most $O(\sqrt{n})$. Note, for each of $O(\sqrt{n})$ pairs if we use Theorem 7 separately we get a FTRSof size $O(n^{3/2})$. Hence our result improves the size of a FTRSby a factor of $\sqrt{n}$. We devote this subsection to prove the above theorem.

Let H_{scc} be an 1-fault tolerant SCC preserver of G as obtained by Theorem 10, and H_{opt} be an optimum sized subgraph of G such that $H := H_{scc} \cup H_{opt}$ is a FTRS($\mathcal{P}$).

For a pair $p = (s,t) \in \mathcal{P}$ let H_p denote an optimum (minimum) sized subgraph of H that is a FTRS(p), i.e., after any single edge failure e, t is reachable from s in $H_p - e$ if and only if that is also the case in $G - e$. An optimum sized subgraph of H that is a FTRS(p), may not be unique. However we arbitrarily choose one such subgraph, and throughout this section refer to that as H_p. The following proposition is immediate from the definition of optimum sized FTRS.

▶ **Proposition 17.** *For any pair $p = (s,t) \in \mathcal{P}$, H_p is union of two $s - t$ paths intersecting only at (s,t)-cut-edges and (s,t)-cut-vertices in H.*

It directly follows from the above proposition that, any *minimal* (s,t)-cut[3] in H_p is of size at most 2. The following property of any (simple) $s-t$ path in H_p will be useful in studying the structure of H (especially in proving Claim 21).

▶ **Observation 18.** *For any pair $p=(s,t)\in\mathcal{P}$ consider a (simple) $s-t$ path Q in H_p. Let C be any minimal (s,t)-cut in H_p. Then Q takes exactly one edge from the set C.*

Proof. The result trivially holds for a minimal (s,t)-cut of size one. Moreover, by minimality of H_p, there cannot exists (s,t)-cut of size three or larger in H_p. So we are left to consider minimal cuts of size two.

Consider a simple $s-t$ path Q in H_p, and let $C=(e,e')$ be a minimal (s,t)-cut of size two in H_p. Let $Z_p=(v_0=s,v_1,\ldots,v_\ell=t)$ be the (s,t)-cut-vertices in H_p, in the order they appear on Q. Let $\mathcal{I}\subseteq[1,\ell]$ be those indices for which (v_{i-1},v_i) is a cut-edge in H_p, and $\mathcal{J}$ be $[1,\ell]\setminus\mathcal{I}$. So for each $i\in\mathcal{J}$, there exists 2-edge-disjoint paths from v_{i-1} to v_i, in H_p; let these be respectively denoted by R_i^0 and R_i^1. By definition of cut-vertices, it is easy to observe that none of the internal vertices of R_i^0 and R_i^1 can lie in Z_p, for $i\in\mathcal{J}$.

Obtain $\widetilde{Q}$ from Q by replacing $R_i^{z_i}$ with $R_i^{1-z_i}$, for $z_i\in\{0,1\}$ and $i\in\mathcal{J}$. So Q and $\widetilde{Q}$, both lie in H_p, and intersect only at (s,t)-cut-edges and cut-vertices. Now for the minimal (s,t)-cut $C=\{e,e'\}$, Q will contain one of the cut-edges, say e, and $\widetilde{Q}$ will contain the other cut edge, i.e. e'. This shows that any simple $s-t$ path in H_p contains exactly one of the edges of a minimal (s,t)-cut of size two. ◀

For each pair $p\in\mathcal{P}$ we define *critical-edge-set* for p, denoted by C_p, as the set of all edges e in H such that $H-e$ is not a FTRS(p). Sometimes we will also use the notation C_p to denote the underlying subgraph formed by edges present in C_p. Note, all the edges of H_{opt} must be in $\cup_{p\in\mathcal{P}}C_p$. Otherwise, if there exists an edge $e\in H_{opt}$ that is not in any of C_p's then we can remove e from H while preserving 1-fault tolerant reachability for all pairs $p\in\mathcal{P}$, leading to a contradiction on the optimality of the size of H_{opt}. So we can deduce the following observation.

▶ **Observation 19.** *For any edge $e\in H$, either $e\in C_p$ for some pair $p\in\mathcal{P}$, or $e\in H_{scc}$.*

Let d_{avg} be the average in-degree of H, i.e., $d_{avg}=|E(H)|/n$. Recall that we use $E(H)$ to denote the set of edges in the subgraph H. Now partition the set of vertices V into two subsets:

1. The set of *light* vertices V_ℓ containing all the vertices whose in-degree in H_{opt} is (strictly) less than $d_{avg}/2$; and
2. The set of *heavy* vertices V_h containing all the vertices whose in-degree in H_{opt} is at least $d_{avg}/2$.

Let E_ℓ and E_h respectively be the set of incoming edges to the vertices in V_ℓ and V_h, in graph H_{opt}. Clearly, $|E_\ell|<(\frac{d_{avg}}{2})n=|E(H)|/2$.

▶ **Lemma 20.** *If $d_{avg}>12$, then for each $p\in\mathcal{P}$, $|E_h\cap C_p|\leq 16|\mathcal{P}|/d_{avg}$.*

We defer the proof of the above lemma to the end of this section. Now assuming the above we show the desired bound on $|E(H)|$, as stated in Theorem 16. First of all, if $d_{avg}\leq 12$ then $E(H)$ is of size $O(n)$. So from now on we assume that $d_{avg}>12$. Note that $|E(H)|=|E(H_{opt})|+|E(H_{scc})|$. By the result of the previous section (see Theorem 10) we know that the graph H_{scc} has at most $O(n)$ edges. Observe that,

[3] A set of edges is said to be an *minimal* (s,t)-cut if and only if it is a valid (s,t)-cut and any of its proper subset is not an (s,t)-cut.

$$
\begin{aligned}
&|E(H_{opt})| = |E_\ell| + |E_h| \leq \frac{|E(H_{opt})|}{2} + |E_h| \\
\Rightarrow &|E(H)| \leq 2(|E_h| + |E(H_{scc})|). \qquad (1)
\end{aligned}
$$

We bound the size of E_h as follows,

$$
\begin{aligned}
|E_h| &\leq \sum_{p\in\mathcal{P}} |E_h \cap C_p| && \text{(since for all } e \in H_{opt},\ e \in C_p \text{ for some } p \in \mathcal{P} \text{ by Observation 19)} \\
&\leq \sum_{p\in\mathcal{P}} \frac{16|\mathcal{P}|}{d_{avg}} && \text{(by Lemma 20)} \\
&= \frac{16|\mathcal{P}|^2 n}{|E(H)|}. && (2)
\end{aligned}
$$

By combining inequalities (1) and (2) we get that $|E(H)| \leq O(n + |\mathcal{P}|\sqrt{n})$.

Now it only remains to prove Lemma 20. Before going into the proof we would first like to make a few important observations regarding the structure of H, which will eventually help us to bound its size. Consider a (simple) path Q from s to t where $(s,t) = p \in \mathcal{P}$, such that all the edges on Q are in H_p. Since Q is a simple path it gives a natural ordering among the vertices present on it. Let us denote this ordering relationship by $<_Q$ ($\leq_Q$ and $>_Q$).

▷ Claim 21. Consider a pair $p = (s,t) \in \mathcal{P}$, and let Q be a (simple) $s - t$ path in H_p. For an edge $e = (u,v) \in C_p$ on Q, suppose there are two vertices $u_1, u_2 <_Q v$, and let Q_1 and Q_2 be (any arbitrary) $u_1 - v$ and $u_2 - v$ path respectively, in $H - \{e\}$. Then Q_1 and Q_2 must share an edge.

Proof. Since $e \in C_p$ if we exclude e from H, the remaining subgraph $H' = H - \{e\}$ will not be a FTRS(p). Observe that e cannot be an (s,t)-cut-edge in H as it violates the existence of paths Q_1 and Q_2 in $H - \{e\}$. So there must exist an edge $f = (u_f, v_f)$ on failure of which there is an $s - t$ path, in $H - \{f\}$, but there is no such path in $H' - \{f\} = H - \{e, f\}$. Since there is no $s - t$ path in $H - \{e, f\}$, $C = \{e, f\}$ must be an (s,t)-cut in H (and also in H_p). Further, since e is not an (s,t)-cut-edge in H, C is a minimal (s,t)-cut in H_p. Let (A, B) be a partition of V induced by the cut C in H such that $s \in A$ and $t \in B$. As Q is a simple $s - t$ path in H_p, by Observation 18 it passes through cut C only once, thereby implying $u_1, u_2 \in A$. Thus, the path Q_i from u_i to v must pass through an edge in C, for $i = 1, 2$. As Q_1 and Q_2 lie in $H - \{e\}$, they both must pass through the edge f, thereby proving the claim. ◁

Now using the above claim we prove the following.

▷ Claim 22. For any pair $p = (s,t) \in \mathcal{P}$, let Q be a (simple) $s - t$ path in H_p. For a vertex v on Q, suppose there are two incoming edges h_1 and h_2 incident on v, which are not part of Q. Further, for $i = 1, 2$, let $p_i = (s_i, t_i)$ be a pair satisfying $h_i \in C_{p_i}$, and let $Q_i \in H_{p_i}$ be an $s_i - t_i$ path containing h_i. If v_i ($\neq v$) is the last vertex in $Q_i[s_i, v]$ that also lies on Q, for $i = 1, 2$, then we cannot simultaneously have $v_1 <_Q v$ as well as $v_2 <_Q v$.

Proof. Let us assume on contrary that $v_1, v_2 <_Q v$. Without loss of generality, we can assume $v_1 \leq_Q v_2$. Let f be an edge on whose failure, any $s_2 - t_2$ path must pass through the edge h_2. Observe h_2 cannot be an (s_2, t_2) cut-edge, as $Q[v_2, v]$ is a $v_2 - v$ path avoiding h_2, thus $C = \{f, h_2\}$ is a minimal (s_2, t_2) cut in H. Note, f must be on $Q[v_2, v]$.

By Claim 21, $Q_1[v_1, v]$ and $Q_2[v_2, v]$ must share an edge. Let $e = (x, y)$ be the last such shared edge. Now even if we exclude h_2 from H we still get the following $s_2 - t_2$ path: $Q_2[s_2, y] \circ Q_1[y, v] \circ Q_2[v, t_2]$ in $H - C$. Since, $C = \{f, h_2\}$ is an (s_2, t_2) cut, this contradicts the assumption that $v_1, v_2 <_Q v$. ◁

▷ Claim 23. For any pair $p = (s, t) \in \mathcal{P}$, let Q be a (simple) $s - t$ path in H_p. For a vertex v on Q, suppose there are two incoming edges h_1 and h_2 incident on v, which are not part of Q. Consider the sets $B_{h_i} := \{q \in \mathcal{P} | h_i \in C_q\}$, for $i \in \{1, 2\}$. Further, for each $i \in \{1, 2\}$, suppose for every pair $p' = (s', t') \in B_{h_i}$, the following holds:

1. Neither there is an $s' - v$ path in $H_{p'}$ with h_i as its last edge, that is internally vertex-disjoint with Q,
2. Nor there exists a vertex, say $v_i <_Q v$, with a $v_i - v$ path in $H_{p'}$ with h_i as its last edge that is edge-disjoint with Q.

Then either $h_1 \notin H_{opt}$ or $h_2 \notin H_{opt}$. (Recall, $H_{opt} = H - H_{scc}$.)

Proof. For $i = 1, 2$, consider a pair $(s_i, t_i) \in B_{h_i}$. Let $Q_i \in H_i$ be an $s_i - t_i$ path containing h_i. Let $u_i \in V - \{v\}$ be the last vertex in $Q_i[s_i, v]$ that also lies on Q. Due to the preconditions mentioned in the statement of the claim, such a u_i must exists, and is necessarily contained in segment $Q[v, t]$. Observe $Q_i[u_i, v]$ is internally vertex disjoint with Q. Next we show that if $u_2 \leq_Q u_1$, $h_2 \notin H_{opt}$.

Suppose $u_2 \leq_Q u_1$. Let f be an edge on whose failure, any $s_2 - t_2$ path must pass through the edge h_2. Observe h_2 cannot be an (s_2, t_2) cut-edge, as $Q[u_2, u_1] \circ Q_1[u_1, v]$ is a $u_2 - v$ path avoiding h_2, thus $C = \{f, h_2\}$ is a minimal (s_2, t_2) cut in H. Observation 18 implies $f \notin Q_2$. Note, f must be either on $Q[u_2, u_1]$ or on $Q_1[u_1, v]$, thereby implying $f \notin Q[v, u_2]$. Thus u_2 and v are strongly connected in $H - \{f\}$ as the cycle $Q_2[u_2, v] \circ Q[v, u_2]$ is intact $H - \{f\}$. Hence H_{scc} contains a $u_2 - v$ path even after the failure of f, and let R' be such a path. Thus even if we exclude h_2 from H_{opt} we still get the following $s_2 - t_2$ path: $Q_2[s_2, u_2] \circ R \circ Q_2[v, t_2]$ in $H - \{f\}$. So due to optimality of H_{opt}, the edge h_2 cannot be in H_{opt} $(= H - H_{scc})$. Similarly when $u_1 <_Q u_2$, $h_1 \notin H_{opt}$, and this completes the proof. ◁

Now we are ready to prove Lemma 20.

Proof of Lemma 20. For the sake of contradiction let us assume that for some pair $p = (s, t) \in \mathcal{P}$, $|E_h \cap C_p| > \frac{16|\mathcal{P}|}{d_{avg}}$. Recall, by Proposition 17 H_p is union of two (simple) $s - t$ paths, say Q and $\widetilde{Q}$, intersecting only at (s, t)-cut-edges in H. At least one of these two paths, say Q, must contain at least $\frac{8|\mathcal{P}|}{d_{avg}}$ edges from $|E_h \cap C_p|$. Now consider the following vertex set

$$Q_h := \{v \in V_h | \text{ there exists an edge } (u, v) \in Q \text{ that is also in } E_h \cap C_p\}.$$

Clearly, $|Q_h| \geq \frac{8|\mathcal{P}|}{d_{avg}}$. Let Q_e denote the subset of edges in $E(H_{opt})$ that are incident on the vertices in Q_h and not part of the path Q. Next consider the following edge-set

$$A := \{(u, v) \in Q_e | \text{ for some } (s', t') \in \mathcal{P} \text{ there exists an } s' - v \text{ path in } H$$
$$\text{with } (u, v) \text{ as its last edge, that is internally vertex-disjoint with } Q\}.$$

Observe, it follows from Proposition 17 that $|A| \leq 2|\mathcal{P}|$. Assuming $d_{avg} > 12$, a simple counting argument shows that there exists a vertex $v \in Q_h$ such that number of edges from $Q_e - A$ that are incident on v is at least 3. If not, then since each vertex in Q_h is by definition heavy, $|A| \geq |Q_h|(\frac{d_{avg}}{2} - 2) \geq \frac{8|\mathcal{P}|}{3}$ assuming $d_{avg} > 12$, which leads to a contradiction.

It implies that there must exist two vertices u_1, u_2 on Q where $u_1 <_Q u_2$, and two edges $h_1, h_2 \in Q_e - A$ (incident on v) such that there are $u_1 - v$ path, say Q_1, with h_1 as its last edge and $u_2 - v$ path, say Q_2, with h_2 as its last edge in H, where both Q_1, Q_2 are edge-disjoint with Q. Now either $u_1, u_2 <_Q v$, or $u_1, u_2 >_Q v$.

Since due to optimality of H_{opt}, $h_2 \in C_{p'}$ for some $p' = (s', t') \in \mathcal{P}$, without loss of generality we can further assume that the path Q_2 is in the subgraph $H_{p'}$. Now if $u_1, u_2 <_Q v$, by Claim 22 there must be an $s' - v$ path in H with h_2 as its last edge that is internally vertex-disjoint with $Q[s, v]$. Thus by the definition of set A, $h_2 \in A$, leading to a contradiction. Now consider the other alternative, i.e., when $u_1, u_2 >_Q v$. In this case without loss of generality we can assume that for $i = 1, 2$ there exists no vertex, say $v_i <_Q v$, with a $v_i - v$ path in H with h_i as its last edge, that is edge-disjoint with Q. Then by Claim 23 either $h_1 \notin H_{opt}$ or $h_2 \notin H_{opt}$, which again leads to a contradiction.

Hence we conclude that for all pairs $p \in \mathcal{P}$, $|E_h \cap C_p| \leq \frac{16|\mathcal{P}|}{d_{avg}}$ for $d_{avg} > 12$. ◀

A Constructive Algorithm. Observe that the size of an minimal subgraph H which is an FTRS$(\mathcal{P}, G)$, must have size at most $O(n + |\mathcal{P}|\sqrt{n})$. Now, a simple constructive algorithm is as follows: We initialize H to G. Next for each pair $(s, t) \in \mathcal{P}$ for each $e \in E(H)$ check if the (s, t)-cut-edges in G and $G - \{e\}$ are identical, if so, remove e from H. The process terminates in polynomial time and results in a graph which is a minimal FTRS$(\mathcal{P}, G)$.

4.2 Upper Bound II

In this subsection, we present our second construction for pairwise reachability preservers.

▶ **Theorem 24.** *For any directed graph $G = (V, E)$ with n vertices, m edges, and a set $\mathcal{P}$ of vertex-pairs, there exists a poly-time computable FTRS$(\mathcal{P}, G)$ containing at most $O(n\sqrt{|\mathcal{P}|})$ edges.*

By Proposition 17, for any pair $p = (s, t) \in \mathcal{P}$, FTRS$(p)$ is union of two $s - t$ paths intersecting only at (s, t) cut-edges and cut-vertices. Let these paths be respectively denoted by $Q_{s,t}$ and $\widetilde{Q}_{s,t}$.

Let $\mathcal{C}$ be the collection $\mathcal{C} = \bigcup_{(s,t)\in\mathcal{P}} \{Q_{s,t}, \widetilde{Q}_{s,t}\}$, and W be initialized to $\emptyset$. For each vertex v, let $\text{FREQ}(v, \mathcal{C})$ denote the total number of paths in $\mathcal{C}$ that contains v. Now we use the following procedure:

1. While there is a vertex w with $\text{FREQ}(w, \mathcal{C}) > \sqrt{|\mathcal{P}|}$, we add w to W, and remove all those paths from $\mathcal{C}$ that contains w.
2. Initialize H to union of subgraphs FTRS(w, G) and FTRS(w, G^R), taken over all $w \in W$.
3. Also add to H the union of edges lying in paths remaining in $\mathcal{C}$.

The size of set W is at most $2\sqrt{|\mathcal{P}|}$ since each time a vertex is added to W at least $\sqrt{|\mathcal{P}|}$ paths are eliminated from $\mathcal{C}$. By Theorem 7 we can bound the size of both FTRS(w, G) and FTRS(w, G^R) by $O(n)$ for each $w \in W$. After step 2, since for each $v \in V$, $\text{FREQ}(v, \mathcal{C})$ is bounded by $\sqrt{|\mathcal{P}|}$, the number of edges in H is at most $O(n\sqrt{|\mathcal{P}|})$. The correctness follows from the following claim.

▷ Claim 25. The subgraph H computed by above process is a FTRS$(\mathcal{P}, G)$.

Proof. Consider a pair $(s, t) \in \mathcal{P}$ and an edge failure $e \in E$. Observe that if e lies in both $Q_{s,t}$ and $\widetilde{Q}_{s,t}$, then e must be an (s, t)-cut. In such a scenario there will be no path from s to t in $G - \{e\}$ as well as in $H - \{e\}$. So let us assume e does not lie in at least one of the paths, $Q_{s,t}$ or $\widetilde{Q}_{s,t}$. Without loss of generality, we assume $Q_{s,t}$ will remain intact in $G-\{e\}$. We will show that there is an $s-t$ path in $H-\{e\}$ even when $Q_{s,t} \notin H$. Recall that if $Q_{s,t} \notin H$, then $Q_{s,t}$ must contain a vertex from set W, let this vertex be w. Since there is a path from s to t in $G-\{e\}$ containing w, there must exists an $s-w$ path, say R_1, in $\mathrm{FTRS}(V \times \{w\}, G) - \{e\}$, and a $w-t$ path, say R_2, in $\mathrm{FTRS}(\{w\} \times V, G) - \{e\}$. The concatenated path $R_1 \circ R_2$ is an $s-t$ path avoiding e. Also $R_1 \circ R_2$ lies in H as we include in H a $\mathrm{FTRS}(\{x\} \times V, G)$ as well as a $\mathrm{FTRS}(V \times \{x\}, G)$, for each $x \in W$. It thus follows that H is a fault-tolerant reachability preserver for the pair (s,t). ◁

5 Lower Bounds for Pairwise Reachability Preserver under Dual Failures

In this section, we provide several lower bound results. In our constructions, we will employ the following lower bound for pairwise distance preservers that was established by Coppersmith and Elkin in [16], and later reformulated in [1] using standard tricks in [10].

▶ **Theorem 26** ([16, 10, 1]). *For any integer $d \geq 2$ there are infinitely many $n \in \mathbb{N}$ such that for any $p \in [n^{f(d)}, n^{f(d+1)}]$, where $f(d) = \frac{2d^2-2d-2}{(d^2+d-2)}$, there exists an n-node undirected unweighted graph $G=(V,E)$ and a pair-set $\mathcal{P} \subseteq V \times V$ of size $|\mathcal{P}| = O(p)$, such that*

- *For each pair $(s,t) \in \mathcal{P}$ there is a unique shortest path in G between s and t,*
- *These paths are all edge-disjoint and have identical length (which we denote by L), and*
- *The edge set of G is precisely the union of these paths and has size $\Omega\big(n^{2d/(d^2+1)}p^{(d^2-d)/(d^2+1)}\big)$.*

Throughout this section, we will reserve $f(d)$ to refer to the function $\frac{2(d^2-d-1)}{(d^2+d-2)}$.

The construction of our lower-bound graph $G=(V_G,E_G)$ is as follows. Let $H = (V_H, E_H)$, $\mathcal{P}_H$ be an instance drawn from Theorem 26. Let L be the common distance between all the pairs in $\mathcal{P}_H$, and $K = L^r$ for some parameter $r \geq 1$ to be fixed later on. For each node u in H, add $2K$ copies of u to G, namely, $u_1, \ldots, u_{2K}$. For each edge (u,v) in H and $i \geq 2$, add edges (u_i, v_{i-1}) and (v_i, u_{i-1}) to G.

Next, for each node $v \in V_H$, add two paths $v_{\text{LEFT}} := (v_{\text{LEFT},1}, \ldots, v_{\text{LEFT},2K})$ and $v_{\text{RIGHT}} := (v_{\text{RIGHT},1}, \ldots, v_{\text{RIGHT},2K})$, each on a set of $2K$ new nodes to G. Also, add an edge from $v_{\text{LEFT},i}$ to v_i, and v_i to $v_{\text{RIGHT},i}$ to G, for $i \in [1, 2K]$.

Finally, for each $(x,y) \in \mathcal{P}_H$, create two new vertices $s_{x,y}$ and $t_{x,y}$, and include $(s_{x,y}, t_{x,y})$ in pair-set $\mathcal{P}_G$; add directed edges from $s_{x,y}$ to $x_{\text{LEFT},1}, y_{\text{RIGHT},1}$ and add directed edges from $x_{\text{LEFT},2K}, y_{\text{RIGHT},2K}$ to $t_{x,y}$. This completes the description of G, and pair-set $\mathcal{P}_G$.

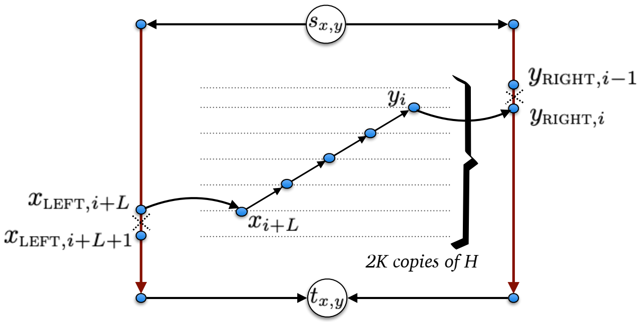


Figure 1 Dual fault-tolerant reachability preserver: depiction of graph G and pair $(s_{x,y}, t_{x,y}) \in \mathcal{P}_G$.

Observe that

1. $|\mathcal{P}_G| = |\mathcal{P}_H|$;
2. $|V_G| = \Theta(|\mathcal{P}_H| + L^r|V_H|) = \Theta(L^r|V_H|)$, whenever $|\mathcal{P}_H| \leq O(L^r|V_H|)$; and
3. $|E_G| = \Theta(|\mathcal{P}_H| + L^r|E_H|) = \Theta(L^{r+1}|\mathcal{P}_H|)$ (since $|E_H| = L|\mathcal{P}_H|$ by the description of H given in Theorem 26).

We first analyze the size of E_G. By Theorem 26, we have

$$L = \frac{|E_H|}{|\mathcal{P}_H|} = \Theta(|V_H|^{\frac{2d}{d^2+1}}\ |\mathcal{P}_H|^{\frac{-(d+1)}{d^2+1}}). \tag{3}$$

Let $n := |V_G| = \Theta(|V_H| \cdot L^r)$ and $m := |E_G| = \Theta(|\mathcal{P}_H| \cdot L^{r+1})$. On multiplying $L^{\frac{2dr}{d^2+1}}$ on both sides of Equation 3, we obtain

$$\begin{aligned} L^{1+\frac{2dr}{d^2+1}} &= \Theta\Big(|V_G|^{\frac{2d}{d^2+1}}\ |\mathcal{P}_G|^{\frac{-(d+1)}{d^2+1}}\Big) \\ \text{or, } L^{r+1} &= \Theta\Big(|V_G|^{\frac{2d(r+1)}{d^2+2rd+1}}\ |\mathcal{P}_G|^{\frac{-(d+1)(r+1)}{d^2+2rd+1}}\Big) \end{aligned}$$

$$\text{Thus, } |E_G| = \Theta(|\mathcal{P}_G|L^{r+1}) = \Theta\Big(|V_G|^{\frac{2d(r+1)}{d^2+2rd+1}}\ |\mathcal{P}_G|^{\frac{d^2+dr-(d+r)}{d^2+2rd+1}}\Big) \tag{4}$$

The size of V_H is given by $|V_H| = \Theta\Big(\frac{|V_G|}{L^r}\Big) = \Theta\Big(n^{\frac{d^2+1}{d^2+2rd+1}}|\mathcal{P}_G|^{\frac{(d+1)r}{d^2+2rd+1}}\Big)$.
So, the condition $|V_H|^{f(d)} \leq |\mathcal{P}_H| \leq |V_H|^{f(d+1)}$ translates to

$$\Big(n^{\frac{d^2+1}{d^2+2rd+1}}|\mathcal{P}_G|^{\frac{(d+1)r}{d^2+2rd+1}}\Big)^{f(d)} \leq |\mathcal{P}_G| \leq \Big(n^{\frac{d^2+1}{d^2+2rd+1}}|\mathcal{P}_G|^{\frac{(d+1)r}{d^2+2rd+1}}\Big)^{f(d+1)}$$

which can be re-stated as: $n^{\frac{(d^2+1)f(d)}{d^2+2rd+1-(d+1)rf(d)}} \leq |\mathcal{P}_G| \leq n^{\frac{(d^2+1)f(d+1)}{d^2+2rd+1-(d+1)rf(d+1)}}$
and on simplification is equivalent to

$$n^{\frac{2(d^2-d-1)}{(d^2+d+2r-2)}} \leq |\mathcal{P}_G| \leq n^{\frac{2(d^2+d-1)}{(d^2+3d+2r)}}. \tag{5}$$

We now prove that a dual fault-tolerant reachability preserver of G requires $\Omega(|E_G|)$ edges. Consider pair $(s_{x,y}, t_{x,y})$ in set $\mathcal{P}_G$, for a pair $(x, y) \in \mathcal{P}_H$. Let $\pi(x, y, H) = (x = w^0, w^1, \ldots, w^L = y)$ be the shortest path between x and y in the undirected graph H.

By construction of G and uniqueness of $\pi(x, y, H)$, it follows that for any $i \in [1, K]$, there exists a unique path from x_{i+L} to y_i in G. Indeed $\pi(x_{i+L}, y_i, G) = (w^0_{i+L}, w^1_{i+L-1}, w^2_{i+L-2}, \ldots, w^L_i)$, is the shortest and the only path starting from $x_{i+L} = w^0_{i+L}$ and terminating to $y_i = w^L_i$ in G. Moreover, there is no path from $x_{i-\alpha+L}$ that terminates to $y_{i+\beta}$ for non-negative integers α, β if at least one of them is greater than 0.

On failures of edges $(x_{\text{LEFT},i+L}, x_{\text{LEFT},i+L+1})$ and $(y_{\text{RIGHT},i-1}, y_{\text{RIGHT},i})$, the concatenated path

$$(s_{x,y}, x_{\text{LEFT},1}, \ldots, x_{\text{LEFT},i+L}) \circ \pi(x_{i+L}, y_i, G) \circ (y_{\text{RIGHT},i}, \ldots, y_{\text{RIGHT},2L}, t_{x,y})$$

is the only path from $s_{x,y}$ to $t_{x,y}$ in the surviving part of G.

Thus, a dual fault-tolerant reachability preserver of G must contain $\pi(x_{i+L}, y_i, G)$, for each $i \in [1, K]$ and $(x, y) \in \mathcal{P}_H$. From the fact that the shortest path between pairs in $\mathcal{P}_H$ are all edge-disjoint in H, it directly follows, a dual fault-tolerant reachability preserver of G must contain $\Omega(KL|\mathcal{P}_H|) = \Omega(K|E_H|) = \Omega(|E_G|)$ edges. The above analysis along with Eq. 4 and Eq. 5 proves our main result, Theorem 27, on dual failure.

▶ **Theorem 27.** *For any integer $d \geq 2$, any real $r \geq 1$, any real $c \in (0,1)$, there are infinitely many $n \in \mathbb{N}$ such that for any $p \in [n^{f(d,r)}, \min\{cn, n^{f(d+1,r)}\}]$, where $f(d,r) = \frac{2d^2-2d-2}{(d^2+d+2r-2)}$, there exists an n-node directed graph $G = (V,E)$ and pair-set $\mathcal{P} \subseteq V \times V$ of size $|\mathcal{P}| = O(p)$, such that any dual fault-tolerant reachability preserver of G for $\mathcal{P}$, must have $\Omega\left(n^{\frac{2d(r+1)}{d^2+2rd+1}} |\mathcal{P}|^{\frac{(d-1)(d+r)}{d^2+2rd+1}}\right)$ edges.*

Note, in the above theorem we need $p \leq cn$ for $c < 1$, to ensure that $|\mathcal{P}_H| \leq O(L^r|V_H|)$ (See the construction of G from H). Some instances of the above theorem are as below.

- $\Omega(n^{8/9}|\mathcal{P}|^{1/3})$ edges for $n^{1/3} \leq |\mathcal{P}| \leq n^{5/6}$ (when $d = 2, r = 1$).
- $\Omega(n^{12/13}|\mathcal{P}|^{4/13})$ edges for $n^{1/4} \leq |\mathcal{P}| \leq n^{5/7}$ (when $d = 2, r = 2$).

Non-existence of Linear-sized Dual Fault-Tolerant Preservers. For $d = 2$, the lower bound turns to be $\Omega\left(n^{\frac{4r+4}{4r+5}} |\mathcal{P}|^{\frac{r+2}{4r+5}}\right)$ on the size of preserver, and bound on $\mathcal{P}$ becomes $[n^{\frac{1}{r+2}}, n^{\frac{5}{r+5}}]$. Let $\epsilon = \frac{2}{(r+2)}$, so $\epsilon \leqslant 2/3$ for $r \geqslant 1$. Now for $|\mathcal{P}| = n^{\epsilon}$, observe $|\mathcal{P}| = n^{\frac{2}{r+2}}$ which lies in range $[n^{\frac{1}{r+2}}, n^{\frac{5}{r+5}}]$, the lower bound on the size of preserver becomes $\Omega(n^{1+\frac{1}{4r+5}})$ which is $\Omega(n^{1+\epsilon/8})$. Thus the following non-linearity result is immediate.

▶ **Theorem 28.** *For every $p = n^{\epsilon}$, for $\epsilon \leq 2/3$, there is an infinite family of n-node directed graphs and pair-sets $\mathcal{P}$ of size $|\mathcal{P}| = p$, for which every dual fault-tolerant reachability preserver of G for $\mathcal{P}$ requires at least $\Omega(n|\mathcal{P}|^{\frac{1}{8}})$ edges.*

Recall, Theorem 7 implies that for any $\mathcal{P}$ of size p there exists a dual fault-tolerant reachability preserver with at most $O(np)$ edges. Our result proves a wide separation in the size pairwise 1-fault-tolerant and 2-fault-tolerant reachability preservers.

6 Application of FT-SCC Preserver in Connectivity Certificates

In this section we present an application of k-FT-SCC preserver for vertex and edge connectivity certificates for digraphs.

▶ **Definition 29** (k-connectivity certificate)**.** *For a graph $G = (V,E)$, a subgraph $H = (V, E_H)$ of G is said to be a k-Edge (Vertex) Connectivity Certificate of G if for each pair of vertices $x, y \in V$, if there are are at least k-edge (vertex) disjoint paths from x to y, and vice versa in G, then the same holds true for graph H as well.*

Georgiadis et al. [22, 23] showed that for any strongly-connected graph we can compute a sparse certificate for 2-vertex connectivity and 2-edge-connectivity comprising of just $O(n)$ edges. However, little is known about extremal size bound of k-connectivity certificates in *digraphs*, for a general k.

We below present a generic reduction from $(k-1)$-FT-SCC preserver to k-connectivity in digraphs.

▶ **Lemma 30.** *Let H be a $(k-1)$-FT-SCC preserver of a digraph G, for some integer $k \geq 1$. Then, H is a k-Edge (Vertex) Connectivity Certificate for G.*[4]

[4] Note that reverse is not always true, i.e. a k-edge(vertex) connectivity certificate is not a $(k-1)$-FT-SCC preserver.

Proof. Let H be a $(k-1)$-FT-SCC preserver of G. Consider a pair of vertices $x, y \in V$, that are at least k-edge (vertex) connected in G.

By Menger's theorem, it follows that for x and y to be k-edge (vertex) connected in G it holds that on removal of any set $F \subseteq E$ of $k-1$ edges (resp. $F \subseteq V$ of $k-1$ vertices) from G, the surviving graph $G-F$ still contains a path from x to y, and a path from y to x, i.e. x and y are strongly connected in $G-F$. Now by definition of $(k-1)$-FT-SCC preserver, we have that on removal of any set F of $k-1$ edges or vertices from H, x and y must be strongly connected in $H-F$. So another application of Menger's theorem, proves that x and y to be k-edge (vertex) connected in H. ◀

Thus, Lemma 30 together with the FT-SCC preserver construction presented in Theorem 15 provides us a k-connectivity certificate of size sub-quadratic in n, for any $k \geq 3$, as follows.

▶ **Theorem 31.** *There is a polynomial time (randomized) algorithm that given any directed graph $G = (V, E)$ on n vertices and $k \geq 2$, computes a k-Edge (Vertex) Connectivity Certificate of G containing at most $\widetilde{O}(k\ 2^k\ n^{2-\frac{1}{k-1}})$ edges with probability at least $1 - 1/n^4$.*

7 Conclusion

In this paper we discuss the problem of sparsifying a directed graph while preserving its strong-connectivity and pairwise reachability structure under edge failures. For SCC preservers, we provide a construction of a truly sub-quadratic (in number of vertices) sized subgraph that preserves SCC components under constantly many edge failures. More specifically, we show an upper bound of $\tilde{O}(k2^k n^{2-1/k})$ on the size for any n-node graph with k faulty edges, whereas we show a lower bound of $\Omega(2^k n)$. We would like to pose the problem of closing this gap between upper and lower bound as an open problem.

In case of reachability preserver, we show an upper bound of $O\big(n+\min(|\mathcal{P}|\sqrt{n},\ n\sqrt{|\mathcal{P}|})\ \big)$ for any n-node graph and a vertex-pair set $\mathcal{P}$ with one faulty edge. This implies linear sized preserver for $O(\sqrt{n})$ many vertex-pairs, which is also known to be the limit for standard non-fault tolerant static setting. Unfortunately we do not know how to generalize our single fault-tolerant pairwise reachability preserver construction to dual fault-tolerant setting. On the contrary, we show a striking difference between single and dual fault tolerant setting by proving a super linear lower bound on dual fault-tolerant reachability preserver for $\Omega(n^\epsilon)$ (for some $\epsilon > 0$) vertex-pairs. One extremely interesting open problem is to get any non-trivial (better than $O(n|\mathcal{P}|)$) upper bound on size of a multiple fault-tolerant pairwise reachability preserver. Other future work lies in improving our size bounds, extending our result to bi-connectivity and other pairwise structures.

References

1 Amir Abboud and Greg Bodwin. Reachability preservers: New extremal bounds and approximation algorithms. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1865–1883, 2018.

2 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995.

3 Surender Baswana, Shreejit Ray Chaudhury, Keerti Choudhary, and Shahbaz Khan. Dynamic DFS in undirected graphs: Breaking the o(m) barrier. *SIAM J. Comput.*, 48(4):1335–1363, 2019.

4 Surender Baswana, Keerti Choudhary, Moazzam Hussain, and Liam Roditty. Approximate single source fault tolerant shortest path. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018*, pages 1901–1915, 2018.

5 Surender Baswana, Keerti Choudhary, and Liam Roditty. Fault tolerant subgraph for single source reachability: generic and optimal. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016*, pages 509–518, 2016.

6 Surender Baswana, Keerti Choudhary, and Liam Roditty. An efficient strongly connected components algorithm in the fault tolerant model. *Algorithmica*, 81(3):967–985, 2019.

7 Surender Baswana and Neelesh Khanna. Approximate shortest paths avoiding a failed vertex: Near optimal data structures for undirected unweighted graphs. *Algorithmica*, 66(1):18–50, 2013.

8 Davide Bilò, Fabrizio Grandoni, Luciano Gualà, Stefano Leucci, and Guido Proietti. Improved purely additive fault-tolerant spanners. In *Algorithms - ESA 2015 - 23rd Annual European Symposium, Proceedings*, pages 167–178, 2015.

9 Davide Bilò, Luciano Gualà, Stefano Leucci, and Guido Proietti. Multiple-edge-fault-tolerant approximate shortest-path trees. In *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016*, pages 18:1–18:14, 2016.

10 Greg Bodwin. Linear size distance preservers. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 600–615, 2017.

11 Greg Bodwin, Fabrizio Grandoni, Merav Parter, and Virginia Vassilevska Williams. Preserving distances in very faulty graphs. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017*, pages 73:1–73:14, 2017.

12 Shiri Chechik. Fault-tolerant compact routing schemes for general graphs. *Inf. Comput.*, 222:36–44, 2013.

13 Shiri Chechik, Michael Langberg, David Peleg, and Liam Roditty. Fault-tolerant spanners for general graphs. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009*, pages 435–444, 2009.

14 Shiri Chechik, Michael Langberg, David Peleg, and Liam Roditty. *f*-sensitivity distance oracles and routing schemes. In *18th Annual European Symposium on Algorithms - ESA (1)*, pages 84–96, 2010.

15 Keerti Choudhary. An optimal dual fault tolerant reachability oracle. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016*, pages 130:1–130:13, 2016.

16 Don Coppersmith and Michael Elkin. Sparse sourcewise and pairwise distance preservers. *SIAM J. Discrete Math.*, 20(2):463–501, 2006.

17 Artur Czumaj and Hairong Zhao. Fault-tolerant geometric spanners. *Discrete & Computational Geometry*, 32(2):207–230, 2004.

18 Camil Demetrescu, Mikkel Thorup, Rezaul Alam Chowdhury, and Vijaya Ramachandran. Oracles for distances avoiding a failed node or link. *SIAM J. Comput.*, 37(5):1299–1318, 2008.

19 Michael Dinitz and Robert Krauthgamer. Fault-tolerant spanners: better and simpler. In *Proceedings of the 30th Annual ACM Symposium on Principles of Distributed Computing, PODC 2011*, pages 169–178, 2011.

20 Ran Duan and Seth Pettie. Dual-failure distance and connectivity oracles. In *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009*, pages 506–515, 2009.

21 Loukas Georgiadis, Giuseppe F. Italiano, Aikaterini Karanasiou, Charis Papadopoulos, and Nikos Parotsidis. Sparse certificates for 2-connectivity in directed graphs. *Theor. Comput. Sci.*, 698:40–66, 2017.

22 Loukas Georgiadis, Giuseppe F. Italiano, Luigi Laura, and Nikos Parotsidis. 2-edge connectivity in directed graphs. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 1988–2005, 2015.

23 Loukas Georgiadis, Giuseppe F. Italiano, Luigi Laura, and Nikos Parotsidis. 2-vertex connectivity in directed graphs. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, pages 605–616, 2015.

24 Loukas Georgiadis, Giuseppe F. Italiano, and Nikos Parotsidis. Strong connectivity in directed graphs under failures, with applications. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 1880–1899, 2017.

25 Manoj Gupta and Shahbaz Khan. Multiple source dual fault tolerant BFS trees. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017*, pages 127:1–127:15, 2017.

26 Tamás Lukovszki. New results of fault tolerant geometric spanners. In *Algorithms and Data Structures, 6th International Workshop, WADS '99, Proceedings*, pages 193–204, 1999.

27 Hiroshi Nagamochi and Toshihide Ibaraki. A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph. *Algorithmica*, 7(5&6):583–596, 1992.

28 Merav Parter. Dual failure resilient BFS structure. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015*, pages 481–490, 2015.

29 Merav Parter and David Peleg. Sparse fault-tolerant BFS trees. In *Algorithms - ESA 2013 - 21st Annual European Symposium, Proceedings*, pages 779–790, 2013.

30 Merav Parter and David Peleg. Fault tolerant approximate BFS structures. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014*, pages 1073–1092, 2014.

31 Jan van den Brand and Thatchaphol Saranurak. Sensitive distance and reachability oracles for large batch updates. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 424–435, 2019.

Matrices of Optimal Tree-Depth and Row-Invariant Parameterized Algorithm for Integer Programming

Timothy F. N. Chan
School of Mathematical Sciences, Monash University, Melbourne, Australia
Mathematics Institute and DIMAP, University of Warwick, Coventry, UK
timothy.chan@monash.edu

Jacob W. Cooper
Faculty of Informatics, Masaryk University, Brno, Czech Republic
jcooper@mail.muni.cz

Martin Koutecký
Computer Science Institute, Charles University, Prague, Czech Republic
koutecky@iuuk.mff.cuni.cz

Daniel Kráľ
Faculty of Informatics, Masaryk University, Brno, Czech Republic
Mathematics Institute, DIMAP and Department of Computer Science,
University of Warwick, Coventry, UK
dkral@fi.muni.cz

Kristýna Pekárková
Faculty of Informatics, Masaryk University, Brno, Czech Republic
kristyna.pekarkova@mail.muni.cz

Abstract

A long line of research on fixed parameter tractability of integer programming culminated with showing that integer programs with n variables and a constraint matrix with tree-depth d and largest entry Δ are solvable in time $g(d,\Delta)\text{poly}(n)$ for some function g, i.e., fixed parameter tractable when parameterized by tree-depth d and Δ. However, the tree-depth of a constraint matrix depends on the positions of its non-zero entries and thus does not reflect its geometric structure. In particular, tree-depth of a constraint matrix is not preserved by row operations, i.e., a given integer program can be equivalent to another with a smaller dual tree-depth.

We prove that the *branch-depth* of the matroid defined by the columns of the constraint matrix is equal to the minimum tree-depth of a row-equivalent matrix. We also design a fixed parameter algorithm parameterized by an integer d and the entry complexity of an input matrix that either outputs a matrix with the smallest dual tree-depth that is row-equivalent to the input matrix or outputs that there is no matrix with dual tree-depth at most d that is row-equivalent to the input matrix. Finally, we use these results to obtain a fixed parameter algorithm for integer programming parameterized by the branch-depth of the input constraint matrix and the entry complexity. The parameterization by branch-depth cannot be replaced by the more permissive notion of branch-width.

2012 ACM Subject Classification Mathematics of computing → Combinatorial optimization; Mathematics of computing → Matroids and greedoids; Mathematics of computing → Combinatorial algorithms

Keywords and phrases Matroid algorithms, width parameters, integer programming, fixed parameter tractability, branch-width, branch-depth

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.26

Category Track A: Algorithms, Complexity and Games

Related Version A full version is available at `https://arxiv.org/abs/1907.06688`.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 26; pp. 26:1–26:19
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Funding The first, second and fourth authors were supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 648509). The second, fourth and fifth authors were supported by the MUNI Award in Science and Humanities of the Grant Agency of Masaryk university. The third author was supported by Charles University project UNCE/SCI/004, and by the project 19-27871X of GA ČR. This publication reflects only its authors' view; the European Research Council Executive Agency is not responsible for any use that may be made of the information it contains.

1 Introduction

Integer programming is a fundamental problem of importance in both theory and practice. It is well-known that integer programming in fixed dimension, i.e., with a bounded number of variables, is polynomially solvable since the work of Lenstra and Kannan [21, 26] from the 1980's. Much subsequent research has focused on studying extensions and speed-ups of the results of Kannan and Lenstra. However, on the side of integer programs with many variables, research has been sparser. Until relatively recently, the most prominent tractable case is that of totally unimodular constraint matrices, i.e., matrices with all subdeterminants equal to 0 and ±1; in this case, all vertices of the feasible region are integral and algorithms for linear programming can be applied.

Besides total unimodularity, many recent results [1, 2, 5, 6, 9, 10, 14, 15] on algorithms for integer programming exploited various structural properties of the constraint matrix yielding efficient algorithms for n-fold IPs, tree-fold IPs, multi-stage stochastic IPs, and IPs with bounded fracture number and bounded tree-width. This research culminated with an algorithm by Levin, Onn and the third author [25] who constructed a fixed parameter algorithm for integer programs with bounded (primal or dual) tree-depth and bounded coefficients. We remark that it is possible to show that the problem is W[1]-hard when parameterized by tree-depth only [10, 24] and NP-hard even for instances with coefficients and tree-width (even path-width) bounded by two [7, Lemma 102] (also cf. [10, 25]).

The tree-depth of a constraint matrix depends on the position of its non-zero entries and thus does not properly reflect the true geometric structure of the integer program. In particular, a matrix with a large (dual) tree-depth may be row-equivalent to another matrix with small (dual) tree-depth that is susceptible to efficient algorithms. We will overcome this drawback with tools from matroid theory. To do so, we consider the branch-depth of the matroid defined by the columns of the constraint matrix and refer to this parameter as to the *branch-depth* of the matrix. Since this matroid is invariant under row operations, the branch-depth of a matrix is *row-invariant*, i.e., preserved by row operations, and captures the true simplicity of the geometric structure of the problem, which can be obfuscated in the case of tree-depth by the choice of the basis.

Our main results can be summarized as follows (we state the results exactly in the next subsection).

- The branch-depth of a matrix A is equal to the minimum dual tree-depth of a matrix row-equivalent to A (Theorem 1).
- There exists a fixed parameter algorithm for computing a matrix with minimum tree-depth that is row-equivalent to an input matrix (Theorem 32).

Our second result is based on a fixed parameter algorithm for computing the branch-depth of a vector matroid represented over a finite field (Theorem 28). Computing decompositions of such matroids is of interest in relation to model checking, in particular monadic second order model checking is fixed parameter tractable for matroids with bounded branch-width that are representable over finite fields [16–18], also see [11]. The existing fixed parameter algorithm [19, 20] for computing the branch-width of such matroids relies on the upper bound on the size of excluded minors for branch-width by Geelen et al. [12] and needs to precompute the (likely very large) list of excluded minors. We could follow a similar path in the setting of branch-depth, however, we decided to design a completely explicit algorithm. While this turned out surprisingly challenging, the hidden constants are significantly better, and we hope that our techniques can be extended to the even more challenging setting of branch-width.

We remark that our first result in conjunction with existing results on approximating branch-depth of a matroid (Theorem 6) yields a fixed parameter algorithm for integer programs with bounded branch-depth (Corollary 4). Since the branch-depth of the constraint matrix is always at most its dual tree-depth (Proposition 12), our algorithm extends the algorithm presented in [25] for integer programs with small dual tree-depth. Since the algorithm from our second result (Theorem 32) preserves that the entry complexity is bounded (cf. Theorem 3), it can also be used a preprocessing step for the algorithm presented in [25], which gives another proof of Corollary 4. Our results on fixed parameter tractability of integer programming cannot be extended to constraint matrices with bounded branch-width (see the discussion at the end of this section); however, Cunningham and Geelen [3] (also cf. [27] for detailed proofs and implementation) provided a slicewise pseudopolynomial algorithm for IPs with non-negative matrices with bounded branch-width, i.e., the problem belongs to the complexity class XP for unary encoding of input.

1.1 Exact statement of our results

To state our results precisely, we need to fix some notation. We consider the general integer programming (IP) problem in the standard form:

$$\min\left\{f(\mathbf{x})\ \middle|\ A\mathbf{x}=\mathbf{b}\,,\ \mathbf{l}\le\mathbf{x}\le\mathbf{u}\,,\ \mathbf{x}\in\mathbb{Z}^n\right\}, \tag{1}$$

where $A\in\mathbb{Z}^{m\times n}$ is an integer $m\times n$ matrix, $\mathbf{b}\in\mathbb{Z}^m$, $\mathbf{l},\mathbf{u}\in(\mathbb{Z}\cup\{\pm\infty\})^n$, and $f:\mathbb{Z}^n\to\mathbb{Z}$ is a separable convex function, i.e., $f(\mathbf{x})=\sum_{i=1}^{n}f_i(x_i)$ where $f_i:\mathbb{Z}\to\mathbb{Z}$ are convex functions. In particular, each $f_i(x_i)$ can be a linear function of x_i. We remark that integer programming is well-known to be NP-hard even when $f(\mathbf{x})\equiv 0$, or when the largest coefficient $\Delta:=\|A\|_\infty$ is 1 (by a reduction from the Vertex Cover problem), or when $m=1$ (by a reduction from the Subset Sum problem). We refer the reader to Section 2 for the definitions of the primal and dual tree-depth of a matrix A and the branch-depth of A.

We now demonstrate the drawback of the parameterization of integer programs by tree-depth that we have mentioned earlier. Consider the following matrices A and A'.

$$A=\begin{pmatrix}1&1&\cdots&1&1\\2&1&\cdots&1&1\\1&2&\ddots&1&1\\\vdots&\ddots&\ddots&\ddots&1\\1&1&\ddots&2&1\\1&1&\cdots&1&2\end{pmatrix}\quad\text{and}\quad A'=\begin{pmatrix}1&1&\cdots&1&1\\1&0&\cdots&0&0\\0&1&\ddots&0&0\\\vdots&\ddots&\ddots&\ddots&0\\0&0&\ddots&1&0\\0&0&\cdots&0&1\end{pmatrix}.$$

The dual tree-depth of the matrix A is equal to the number of its rows while the dual tree-depth of A' is two (its dual graph is a star); we remark that the branch-depth of both matrices A and A' is also equal to two. Since the matrices A and A' are row-equivalent, the integer programs determined by them ought to be of the same computational difficulty. More precisely, consider the following matrix B:

$$B = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 & 0 \\ -1 & 1 & 0 & \cdots & 0 & 0 \\ -1 & 0 & 1 & \ddots & 0 & 0 \\ -1 & \vdots & \ddots & \ddots & \ddots & 0 \\ -1 & 0 & 0 & \ddots & 1 & 0 \\ -1 & 0 & 0 & \cdots & 0 & 1 \end{pmatrix} .$$

Since $A' = BA$, it is possible to replace an integer program of the form (1) with an integer program with a constraint matrix $A' = BA$, right hand side $\mathbf{b}' = B\mathbf{b}$, and bounds $\mathbf{l}' = \mathbf{l}$ and $\mathbf{u}' = \mathbf{u}$, and attempt to solve this new instance of IP which has dual tree-depth two.

In Section 4, we first observe that the branch-depth of a matrix A is at most its dual tree-depth, and prove that the branch-depth of a matrix A is actually equal to the minimum dual tree-depth of a matrix A that is row-equivalent to A:

▶ **Theorem 1.** *Let A be a matrix over a field $\mathbb{F}$. The branch-depth of A is equal to the minimum dual tree-depth of a matrix A' that is row-equivalent to A, i.e., that can be obtained from A by row operations.*

The tools developed to prove Theorem 1 together with existing results on matroid branch-depth yield an algorithm that given a matrix A of small branch-depth yields a matrix B that transforms the matrix A to a row-equivalent matrix with small dual tree-depth. Recall that the *entry complexity of a matrix A*, denoted by $\mathrm{ec}(A)$, is the maximum length of the binary encoding of an entry $A_{i,j}$ (the length of binary encoding a rational number $r = p/q$ with p and q being coprime is $\lceil \log_2 p \rceil + \lceil \log_2 q \rceil + 1$).

▶ **Theorem 2.** *There exist a computable function $g : \mathbb{N} \to \mathbb{N}$ and an FPT-parameter algorithm parameterized by d with running time polynomial in $\mathrm{ec}(A)$, n and m that for an input $m \times n$ integer matrix A and an integer d*

1. *outputs that the branch-depth of A is larger than d, or*
2. *outputs an invertible rational matrix $B \in \mathbb{Q}^{m \times m}$ such that the dual tree-depth of BA is at most 4^d and the entry complexity of BA is $O(g(d)\,\mathrm{ec}(A))$.*

However, we go further and design a fixed parameter algorithm for computing the branch-depth of a vector matroid (Theorem 32) and use this algorithm to prove the following strengthening of Theorem 2.

▶ **Theorem 3.** *There exist a computable function $g' : \mathbb{N}^2 \to \mathbb{N}$ and an FPT-parameter algorithm parameterized by d with running time polynomial in $\mathrm{ec}(A)$, n and m that for an input $m \times n$ integer matrix A and an integer d*

1. *outputs that the branch-depth of A is larger than d, or*
2. *outputs an invertible rational matrix $B \in \mathbb{Q}^{m \times m}$ such that the dual tree-depth of BA is equal to the branch-depth of A and the entry complexity of BA is at most $g'(d, \mathrm{ec}(A))$.*

As explained above, Theorems 2 and 3 allow us to perform row operations to obtain an equivalent integer program with small dual tree-depth from an integer program with small branch-depth. In particular, if the instance of an integer program described as in (1) has bounded branch-depth, then Theorem 3 yields a matrix B such that the instance with $A' = BA$, $\mathbf{b}' = B\mathbf{b}$, $\mathbf{l}' = \mathbf{l}$ and $\mathbf{u}' = \mathbf{u}$ has dual tree-depth equal to branch-depth. To apply the algorithm from [25], we need to transform the matrix A' into an integer matrix. We do so by multiplying each row by the least common multiple of the denominators of the fractions in this row; note that the value of this least common multiple is at most $2^{2^{\mathrm{ec}(A')}}$ since there can be at most $2^{\mathrm{ec}(A')}$ different denominators appearing in the row. In particular, the entry complexity of the resulting integer matrix is bounded by a function of the entry complexity of A'. Also note that since the parameter dependence in the algorithm of [7] is roughly $\mathrm{ec}(A)^{\mathrm{td}_D(A)^2 \cdot 2^{\mathrm{td}_D(A)}}$, improving the exponent by replacing A with a row-equivalent matrix with smaller dual tree-depth likely outweighs the increase in the coefficients, which enters as the base of the exponent. Hence, we obtain the following corollary of the theorem.

▶ **Corollary 4.** *There exists a computable function $g'' : \mathbb{N}^2 \to \mathbb{N}$ such that integer programs with n variables and a constraint matrix A can be solved in time polynomial in $g''(\mathrm{bd}(A), \mathrm{ec}(A))$ and n, where $\mathrm{bd}(A)$ and $\mathrm{ec}(A)$ are the branch-depth and the entry complexity of the matrix A, i.e., integer programming is fixed parameter tractable when parameterized by branch-depth and entry complexity.*

We note that the results of [7,25] give a strongly fixed-parameter algorithm (i.e., an algorithm whose number of arithmetic operations does not depend on the size of the numbers involved) for integer programming in the regimes discussed above if the objective function f is a linear function (i.e., $f(\mathbf{x}) = \mathbf{w}\mathbf{x}$ for some $\mathbf{w} \in \mathbb{Z}^n$). Hence, the corollary above also gives a strongly-polynomial algorithm when f is a linear function.

We also remark that existing hardness results imply that the parameterization both by branch-depth and entry complexity in Corollary 4 is necessary unless FPT =W[1], i.e., it is not sufficient to parameterize instances only by one of the two parameters. Likewise, it is not possible to replace the branch-depth parameter by the more permissive notion of branch-width [3]. In fact, even solving integer programs with constant dual tree-width and constant entry complexity is NP-hard [25] (the dual tree-width of A is an upper bound on the branch-width of the vector matroid formed by columns of A). Let us also mention that Fomin et al. [8] proved lower bounds on the complexity of integer programming parameterized by branch-width under the exponential-time hypothesis.

The algorithm given in Corollary 4 is parameterized by the branch-depth of the vector matroid formed by the columns of the matrix A, i.e., it corresponds to the dual tree-depth of A. It is natural to ask whether the tractability also holds in the setting dual to this one, i.e., when the branch-depth of the vector matroid formed by the rows of the matrix A is bounded. This hope is dismissed in Proposition 14.

2 Notation

In this section, we fix the notation used throughout the paper and present the notions of tree-depth of a graph and of branch-depth of a matroid, including the results concerning them that we will need further. To avoid our presentation becoming cumbersome through adding or subtracting one at various places, we define the *depth* of a rooted tree to be the maximum number of edges on a path from the root to a leaf, and define the *height* of a rooted tree to be the maximum number of vertices on a path from the root to a leaf, i.e., the height of a rooted tree is always equal to its depth increased by one. The *depth of a vertex* in a rooted tree is the number of edges on the path from the root to that particular vertex. The height of a rooted forest F is the maximum height of a rooted tree in F. The *closure* $\mathrm{cl}(F)$ of a rooted forest is the graph obtained by adding edges from each vertex to all its descendants. Finally, the *tree-depth* $\mathrm{td}(G)$ of a graph G is the minimum height of a rooted forest F such that the closure $\mathrm{cl}(F)$ of the rooted forest F contains G as a subgraph. It can be shown that the path-width of a graph G is at most its tree-depth $\mathrm{td}(G)$ decreased by one, and in particular, the tree-width of G is at most its tree-depth decreased by one (in this extended abstract, we do not give the definitions of path-width and tree-width here due to space limitations). We would like to note that the tree-depth as used in [23] is equal to the minimum depth of a rooted tree F such that $G \subseteq \mathrm{cl}(F)$, however, we here follow the definition of tree-depth that is standard; still, we wish to highlight this subtle difference since [23] is one of our main references.

The *primal graph* of an $m \times n$ matrix A is the graph $G_P(A)$ with vertices $\{1, \dots, n\}$, i.e., its vertices correspond to the columns of A, where vertices i and j are connected if A contains a row whose i-th and j-th entries are non-zero. The *primal tree-depth* $\mathrm{td}_P(A)$ of a matrix A is the tree-depth of its primal graph. Analogously, the *dual graph* of A is the graph $G_D(A)$ with vertices $\{1, \dots, m\}$, i.e., its vertices correspond to the rows of A, where vertices i and j are connected if A contains a column whose i-th and j-th entries are non-zero, i.e., the dual graph $G_D(A)$ is isomorphic to the primal graph of the matrix A^T. Finally, the *dual tree-depth* of A, which is denoted by $\mathrm{td}_D(A)$, is the tree-depth of the dual graph $t_D(A)$.

We next introduce the notion of branch-depth of a matroid. To keep our presentation self-contained, we start by recalling the definition of a matroid. A *matroid* M is a pair $(X, \mathcal{I})$, where $\mathcal{I}$ is a non-empty hereditary collection of subsets of X that satisfies the *augmentation axiom*. The collection $\mathcal{I}$ is hereditary if for every $X' \in \mathcal{I}$, $\mathcal{I}$ contains all subsets of X'. The augmentation axiom asserts that for all $X' \in \mathcal{I}$ and $X'' \in \mathcal{I}$ with $|X'| < |X''|$, there exists an element $x \in X'' \setminus X'$ such that $X' \cup \{x\} \in \mathcal{I}$. The sets contained in $\mathcal{I}$ are referred to as *independent*. The *rank* $r(X')$ of a set $X' \subseteq X$ is the size of the maximum independent subset of X'; the rank $r(M)$ of a matroid $M = (X, \mathcal{I})$ is the rank of X and an independent set of size $r(M)$ is a *basis* of M. A *circuit* is a set $X' \subseteq X$ such that X' is not independent but every proper subset of X' is. Two elements of x and x' are said to be *parallel* if $r(\{x\}) = r(\{x'\}) = r(\{x, x'\}) = 1$, and an element x of M is a *loop* if $r(\{x\}) = 0$.

Two particular examples of matroids are graphic matroids and vector matroids. If G is a graph, then the pair $(E(G), \mathcal{I})$ where $\mathcal{I}$ contains all acyclic subsets of edges of G is a matroid and is denoted by $M(G)$; matroids of this kind are called *graphic matroids*. If X is a set of vectors of a vector space and $\mathcal{I}$ contains all subsets of X that are linearly independent, then the pair $(X, \mathcal{I})$ is a matroid; matroids of this kind are *vector matroids*. In the setting of vector matroids, the rank of $X' \subseteq X$ is the dimension of the linear hull of X'. If $(X, \mathcal{I})$ is a vector matroid, we write $\mathcal{L}(X')$ for the linear hull of the vectors contained in $X' \subseteq X$ and abuse the notation by writing $\dim X'$ for $\dim \mathcal{L}(X')$.

In what follows, we will need a notion of a quotient of a vector space, which we now recall. If A is a vector space and K a subspace of A, the *quotient space* A/K is a vector space of dimension $\dim A - \dim K$ obtained from A by considering cosets of A given by K and inheriting addition and scalar multiplication from A; see e.g. [13] for further details if needed. One can show show for every subspace K of A, there exists a subspace B of A with dimension $\dim A - \dim K$ such that each coset contains a single vector from B, i.e., every vector w of A can be uniquely expressed as the sum of a vector w_B of B and a vector w_K of K. We call the vector w_B to be the *quotient* of w by K. Note that the quotient of a vector is not uniquely defined by K, however, it becomes uniquely defined when the subspace B is fixed.

A *depth-decomposition* of a matroid $M = (X, \mathcal{I})$ is a pair (T, f), where T is a rooted tree and f is a mapping from X to the leaves of T such that the number of edges of T is the rank of M and the following holds for every subset $X' \subseteq X$: the rank of X' is at most the number of edges contained in the paths from the root to the vertices $f(x)$, $x \in X'$. If the matroid M contains two parallel elements x and x', we will always assume that $f(x) = f(x')$. The *branch-depth* $\mathrm{bd}(M)$ of a matroid M is the smallest depth of a tree T that forms a depth-decomposition of M. For example, if $M = (X, \mathcal{I})$ is a matroid of rank r, T is a path with r edges rooted at one of its end vertices, and f is a mapping such that $f(x)$ is equal to the non-root end vertex of T for all $x \in X$, then the pair (T, f) is a depth-decomposition of M. In particular, the branch-depth of any matroid M is well-defined and is at most the rank of M. We remark that the notion of branch-depth of a matroid given here is the one defined in [22,23]; another matroid parameter, which is also called branch-depth but is different from the one that we use here, is defined in [4]. Finally, the *branch-depth* $\mathrm{bd}(A)$ of a matrix A is the branch-depth of the vector matroid formed by the columns of A. Since the vector matroid formed by the columns of A and the vector matroid formed by the columns of any matrix row-equivalent to A are the same, the branch-depth of A is invariant under row operations.

An *extended depth-decomposition* of a vector matroid $M = (X, \mathcal{I})$ is a triple (T, f, g) such that (T, f) is a depth-decomposition of M and g is a bijective mapping from the non-root vertices of T to a basis of the linear hull of X that satisfies that every element $x \in X$ is contained in the linear hull of the g-image of the non-root vertices on the path from $f(x)$ to the root of T. We next state and prove a simple proposition on the way that the vectors forming M are expressed as linear combinations of the vectors of the base formed by the g-image.

▶ **Proposition 5.** *Let (T, f, g) be an extended depth-decomposition of a vector matroid M. Let X' be a subset of elements of M and let X'' be an independent subset of X' with $r(X')$ elements. Then for every vector $x' \in X'$, there is a vector $x'' \in X''$ such that x' is contained in the linear hull of the g-images of the vertices on the path from $f(x'')$ to the root.*

If (T, f, g) is an extended depth-decomposition of a matroid M and all g-images are elements of the matroid M, then we say that the extended depth-decomposition is *principal*. Kardoš et al. [23, Corollary 3.17] designed an algorithm that outputs an approximation of an optimal depth-decomposition; we state the result here for the case of vector matroids.

▶ **Theorem 6.** *There exists a polynomial-time algorithm that given a vector matroid M and an integer d, either outputs that the branch-depth of M is larger than d or outputs a principal extended depth-decomposition of M of depth at most 4^d.*

If (T, f, g) is an extended depth-decomposition and u is a vertex of T, then K_u is the linear hull of the g-images of the vertices on the path from u to the root of T; in particular, if u is the root, then K_u contains the zero vector only. It will always be clear from the context for which extended depth-decomposition of M the spaces K_u are defined since the vertex u determines which rooted tree T is considered.

A *branch* of a rooted tree T is a subtree S rooted at a vertex u of T, with u having at least two children, such that S contains exactly u, one child u' of u, and all descendants of u'. In particular, a rooted tree has a branch if and only if it has a vertex with at least two children. A branch S is *primary* if every ancestor of the root of S has exactly one child. Every rooted tree T that is not a rooted path has at least two primary branches and all primary branches are rooted at the same vertex. We write $\widehat{S}$ for the set of elements of the matroid M mapped by f to the leaves of S and $\|S\|$ for the number of edges of S. Let S be a branch of T and $S_1, \ldots, S_k$ be the other branches with the same root. The branch S is *at capacity* if

$$r\left(X \setminus \left(\widehat{S}_1 \cup \cdots \cup \widehat{S}_k\right)\right) = r(M) - \|S_1\| - \|S_2\| - \cdots - \|S_k\|,$$

where X is the set of all elements of the matroid M. Note that if S is primary, then the left side of the equality is $r(\widehat{S})$ and the right side is $h + \|S\|$ in this case, where h is the depth of the root of S In particular, a primary branch S is *at capacity* if and only if the rank of

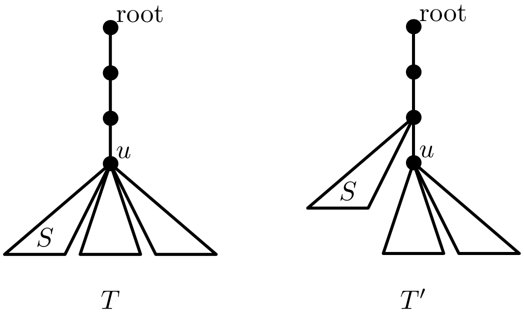


Figure 1 The trees T and T' from the statement of Lemma 8.

$\widehat{S}$ is equal to the sum of $\|S\|$, i.e., if and only if the rank inequality from the definition of a depth-decomposition holds with equality for the set $\widehat{S}$. Finally, a branch S rooted at a vertex u is *solid* if the matroid $(M/K_u)\left[\widehat{S}\right]$ is connected.

3 Optimal extended depth-decompositions

The goal of this section is to show that every vector matroid has an extended depth-decomposition with depth equal to its branch-depth. To do so, we start with noting that branches rooted at the root of the decomposition tree are always at capacity; the proof is left due to space limitations.

▶ **Lemma 7.** *Let (T, f) be a depth-decomposition of a vector matroid M. If T has a branch S rooted at the root of T, then S is at capacity.*

The following lemma is a core of an argument that every matroid has a depth-decomposition of optimal depth such that each primary branch is at capacity. See Figure 1 for the illustration of the operation described in the statement of the lemma.

▶ **Lemma 8.** *Let (T, f) be a depth-decomposition of a vector matroid M. Assume that T contains a primary branch S that is not at capacity. Let u be the root of S, and let T' be the rooted tree obtained from T by changing the root of S to be the parent of u. Then, (T', f) is a depth-decomposition of M.*

Proof. Let X be all elements of M and fix a subset X' of X. We need to show that $\dim X'$ is at most the number e_0 of edges on the paths in T' from the vertices in the f-image of X' to the root. If X' contains an element of $X \setminus \widehat{S}$, then the number of such edges is the same in the trees T and T' and the inequality follows from the fact that (T, f) is a depth-decomposition of M. Hence, we will assume that X' is a subset of $\widehat{S}$. Observe that collectively the primary branches of T different from S contain $r - h - \|S\|$ edges, where h is the depth of the root of S. We derive using the fact that (T, f) is a depth-decomposition the following:

$$\begin{aligned} e_0 + 1 + (r - h - \|S\|) &\geq \dim X' \cup (X \setminus \widehat{S}) \\ &= \dim X' + \dim X \setminus \widehat{S} - \dim \mathcal{L}(X') \cap \mathcal{L}\left(X \setminus \widehat{S}\right) \\ &\geq \dim X' + \dim X \setminus \widehat{S} - \dim \mathcal{L}\left(\widehat{S}\right) \cap \mathcal{L}\left(X \setminus \widehat{S}\right) \\ &= \dim X' + \dim X \setminus \widehat{S} - (\dim \widehat{S} + \dim X \setminus \widehat{S} - \dim X) \\ &= \dim X' - \dim \widehat{S} + r. \end{aligned}$$

This implies that $\dim X'$ is at most

$$e_0 + \dim \widehat{S} + 1 - h - \|S\| \leq e_0,$$

where the inequality follows using that S is not at capacity, i.e., $\dim \widehat{S} < h + \|S\|$. Hence, (T', f) is a depth-decomposition of M. ◀

We can obtain the following by iteratively applying Lemma 8.

▶ **Lemma 9.** *Let (T, f) be a depth-decomposition of a vector matroid $M = (X, \mathcal{I})$ of depth d. There exists a depth-decomposition of M of depth at most d such that every primary branch is at capacity.*

The following lemma describes the way how the structure of the vector matroids is captured by depth-decompositions such that each primary branch is at capacity.

▶ **Lemma 10.** *Let (T, f) be a depth-decomposition of a vector matroid $M = (X, \mathcal{I})$ such that T is not a rooted path and each primary branch of T is at capacity. Let $S_1, \dots, S_k$ be the primary branches of T, and let $A_1, \dots, A_k$ be the linear hulls of $\widehat{S_1}, \dots, \widehat{S_k}$, respectively. Further, let h be the depth of the common root of $S_1, \dots, S_k$ in T. There exists a subspace K of dimension h such that $A_i \cap A_j = K$ for all $1 \leq i < j \leq k$.*

We are now ready to prove the main theorem of this section.

▶ **Theorem 11.** *Let (T, f) be a depth-decomposition of a vector matroid $M = (X, \mathcal{I})$ of depth d. There exists an extended depth-decomposition of M of depth at most d.*

Proof. The proof proceeds by induction on the rank of M. By Lemma 9, we can assume that all primary branches of T are at capacity. If T is a rooted path, we assign elements of a basis of $\mathcal{L}(X)$ to the non-root vertices of T arbitrarily, i.e., we choose g to be any bijection to a basis of $\mathcal{L}(X)$, which yields an extended depth-decomposition (T, f, g) of M. Note that if the rank of M is one, then T is the one-edge rooted path, i.e., this case covers the base of the induction in particular.

We next assume that T is not a rooted path for the rest of the proof. Let $S_1, \dots, S_k$ be the primary branches of T, and let h be the depth of the common root of $S_1, \dots, S_k$. By Lemma 10, there exists a subspace K of dimension h such that the intersection of linear hulls of $\widehat{S_i}$ and $\widehat{S_j}$ is K for all $1 \leq i < j \leq k$; let $b_1, \dots, b_h$ be an arbitrary basis of K.

We define M_i, $i = 1, \dots, k$, to be the matroid such that the elements of M_i are $\widehat{S_i}$ and $X' \subseteq \widehat{S_i}$ is independent if and only if the elements $X' \cup \{b_1, \dots, b_h\}$ are linearly independent. In particular, the rank of $X' \subseteq \widehat{S_i}$ in M_i is equal to $\dim X' \cup K - h$. The matroid M_i can be viewed as obtained by taking the vector matroid with the elements $\widehat{S_i} \cup \{b_1, \dots, b_h\}$ and contracting the elements $b_1, \dots, b_h$. In particular, M_i is a vector matroid, and the vector representation of M_i can be obtained from $\widehat{S_i}$ by taking quotients by K. Note that the rank of M_i is $\dim \widehat{S_i} \cup K - h$, i.e., its rank is smaller than the rank of M and we will be able to eventually apply induction to it.

Let f_i be the restriction of f to $\widehat{S_i}$. We claim that (S_i, f_i) is a depth-decomposition of M_i. Let X' be a subset of $\widehat{S_i}$, and let e_i be the number of edges contained in the union of paths from the elements $f(x)$, $x \in X'$, to the root of S_i. By the definition of M_i, the rank of X' in M_i is equal to $\dim X' \cup K - h$. Choose an arbitrary $j \neq i$, $1 \leq j \leq k$. Since (T, f) is a depth-decomposition of M, the intersection of linear hulls of $\widehat{S_i}$ and $\widehat{S_j}$ is K, and the branch S_j is at capacity, i.e., $\dim \widehat{S_j} = \|S_j\| + h$, we obtain that the rank of X' in M_i is equal to

$$\begin{aligned}\dim X' \cup K - h &= \dim X' \cup \widehat{S_j} - \dim \widehat{S_j}\\ &\le e_i + \|S_j\| + h - \dim \widehat{S_j} = e_i.\end{aligned}$$

Hence, (S_i, f_i) is a depth-decomposition of M_i.

We now apply induction to each matroid M_i and its depth-decomposition (S_i, f_i), $i = 1, \ldots, k$, to obtain extended depth-decompositions (S'_i, f'_i, g_i) of M_i such that the depth of S'_i is at most the depth of S_i. Let T' be a rooted tree obtained from a rooted path of length h by identifying its non-root end with the roots of $S'_1, \ldots, S'_k$. Note that the depth of T' does not exceed the depth of T. Further, let f' be the unique function from X to the leaves of T such that the restriction of f' to the elements of M_i is f_i. Finally, let g be any function from the non-root vertices of T such that the h non-root vertices of the path from the root are mapped to the vectors $b_1, \ldots, b_h$ by g and $g(v) = g_i(v)$ for every non-root vertex v of S_i.

We claim that (T', f', g) is an extended depth-decomposition of M. We first verify that, for every $x \in X$, $f'(x)$ is contained in the linear hull of the g-image of the non-root vertices on the path from $f'(x)$ to the root. Fix $x \in X$ and let i be such that $x \in \widehat{S}_i$. Since (S'_i, f'_i, g_i) is an extended depth-decomposition of M_i, x is contained in the linear hull of K and the g_i-images of the non-root vertices on the path from $f'(x) = f_i(x)$ to the root of S'_i. Hence, x is contained in the linear hull of the g-image of the non-root vertices on the path from $f'(x)$ to the root of T'.

Consider now an arbitrary subset $X' \subseteq X$. We have already established that all elements of X' are contained in the linear hull of the g-image of the non-root vertices on the paths from $f'(x)$, $x \in X'$, to the root of T'. Since the dimension of this linear hull is equal to the number of non-root vertices on such paths, which is equal to the number of edges of the paths, it follows that (T', f') is a depth-decomposition of M. ◀

4 Optimal tree-depth of a matrix

In this section, we relate the optimal dual tree-depth of a matrix A to its branch-depth. We start with observing that the branch-depth of a matrix A is at most its dual tree-depth; the proof is left due to space limitations.

▶ **Proposition 12.** *If A is an $m \times n$ matrix, then $\mathrm{bd}(A) \le \mathrm{td}_D(A)$.*

We next establish the main theorem of this section.

▶ **Theorem 13.** *Let A be an $m \times n$ matrix of rank m, M the vector matroid formed by columns of A, and (T, f, g) an extended depth-decomposition of M. Further, let $\mathrm{Im}(g) = \{w_1, \ldots, w_m\}$. The dual tree-depth of the $m \times n$ matrix A' such that the j-th column of A is equal to*

$$\sum_{i=1}^{m} A'_{ij} w_i$$

is at most the depth of the tree T.

Proof. Let F be the rooted forest obtained from T by removing the root and associate the i-th row of A' with the vertex v of F such that $g(v) = w_i$. Note that the height of F is the depth of T. We will establish that the dual graph $G_D(A')$ is contained in the closure $\mathrm{cl}(F)$ of F. Let i and i', $1 \le i, i' \le m$, be such that the vertices of F associated with the i-th and i'-th rows of A' are adjacent in $G_D(A')$. This means that there exists j, $1 \le j \le n$, such that $A'_{ij} \ne 0$ and $A'_{i'j} \ne 0$. Let v be the leaf of T such that the j-th column of A is mapped by f

to v. The definition of an extended depth-decomposition yields that the j-th column is a linear combination of the g-image of the non-root vertices on the path from v to the root of T. In particular, the path contains the two vertices of T mapped by g to w_i and $w_{i'}$; these two vertices are associated with the i-th and i'-th rows of A'. Hence, the vertices associated with the i-th and i'-th rows are adjacent in $\mathrm{cl}(F)$. We conclude that $G_D(A')$ is a subgraph of $\mathrm{cl}(F)$. ◀

Proposition 12 and Theorem 11 combine to a proof of Theorem 1.

5 Parameterized algorithms for integer programming

The main purpose of this section is to combine Theorem 1 with the existing approximation algorithm for branch-depth (Theorem 6) to obtain an approximation algorithm for computing a row-equivalent matrix with small dual tree-depth if it exists.

Proof of Theorem 2. Let A be an $m \times n$ matrix. Without loss of generality, we can assume that the rows of A are linearly independent, i.e., the rank of A is m. This also implies that the rank of the column space of A is m, in particular, $n \geq m$.

We apply the approximation algorithm described in Theorem 6 to the vector matroid M formed by the columns of the matrix A, and we obtain an extended depth-decomposition (T, f, g) of M. If the depth of T is larger than 4^d, then the branch-depth of A is larger than d; we report this and stop. Let B_g be the matrix with the columns formed by the vectors in $\mathrm{Im}(g)$ and let $B = B_g^{-1}$. Note that the matrix A' from the statement of Theorem 13 is equal to BA. By Theorem 13, the dual tree-depth of A' is at most 4^d. The proof that the entry complexity of A' is at most $O(d \cdot 4^d \cdot \mathrm{ec}(A))$ is left due to space limitations. ◀

Theorem 2 yields Corollary 4, which asserts that integer programming is fixed parameter tractable when parameterized by the branch-depth and the entry complexity of the constraint matrix. We complement this corollary by showing that integer programming is not fixed parameter tractable when parameterized by the "primal" branch-depth.

▶ **Proposition 14.** *Integer programming is* NP*-hard for instances with constraint matrices A satisfying $\mathrm{bd}(A^T) = 1$ and $\mathrm{ec}(A) = 1$, i.e., for instances such that the vector matroid formed by rows of the constraint matrix has branch-depth one.*

6 Structure of extended depth-decompositions

In this section, we present structural results on extended depth-decompositions that we need to design a fixed parameter algorithm to compute a depth-decomposition of a vector matroid with an optimal depth. The proofs of the next two lemmas are left due to space limitations; we note that the first of the two lemmas can be viewed as a generalization of Lemma 10.

▶ **Lemma 15.** *Let (T, f) be a depth-decomposition of a vector matroid M and let U be a set of vertices of T such that every vertex contained in U has at least two children and every ancestor of a vertex in U with at least two children is contained in U. Assume that every branch of T rooted at a vertex from U is at capacity. Every vertex $u \in U$ can be associated with a subspace L_u of the linear hull of the elements of M such that the dimension of L_u is the depth of u and the following holds. Let u be a vertex of U, let $S_1, \ldots, S_k$ be all branches rooted at u, and let $A_1, \ldots, A_k$ be the linear hulls of $\widehat{S_1}, \ldots, \widehat{S_k}$, respectively. If each ancestor of u has a single child, let L_0 be the vector space containing the zero vector only; otherwise, let u' be the nearest ancestor of u with at least two children, and let L_0 be the space $L_{u'}$. It holds that $\mathcal{L}(A_i \cup L_0) \cap \mathcal{L}(A_j \cup L_0) = L_u$ for all $1 \leq i < j \leq k$.*

▶ **Lemma 16.** *Let (T, f) be a depth-decomposition of a vector matroid M, u_1 a vertex of T with at least 2 children, and $u_2, \ldots, u_k$ all ancestors of u_1 with at least two children (listed in the increasing distance from u_1). Assume that every branch rooted at one the vertices $u_1, \ldots, u_k$ is at capacity, and let L_1 be the space L_{u_1} from the statement of Lemma 15 applied with $U = \{u_1, \ldots, u_k\}$. Further, let S_1 be any branch rooted at u_1 and f_1 the restriction of f to S_1. The pair (S_1, f_1) is a depth-decomposition of the vector matroid $(M/L_1)\left[\widehat{S_1}\right]$ and a branch of (S_1, f_1) is at capacity if and only if it is at capacity in (T, f). In addition, if (S_1', f_1') is another depth-decomposition of the vector matroid $(M/L_1)\left[\widehat{S_1}\right]$, then (T', f') is a depth-decomposition of the vector matroid M, where T' is obtained from T by replacing S_1 with S_1', and the function f' is defined as $f'(x) = f_1'(x)$ for $x \in \widehat{S_1}$, and $f'(x) = f(x)$ otherwise.*

Lemmas 15 and 16 allow to extend Lemma 8 to all branches.

▶ **Lemma 17.** *Let (T, f) be a depth-decomposition of a vector matroid M, and S_0 a branch of T rooted at a vertex u_0 such that S_0 is not at capacity. Suppose that every branch rooted at an ancestor of u_0 is at capacity. Let T' be the rooted tree obtained from T by changing the root of S_0 to be the parent of u_0. Then, (T', f) is a depth-decomposition of M.*

Lemmas 15–17 yield an iterative algorithm described in the next theorem.

▶ **Theorem 18.** *There exists a polynomial time algorithm that given a vector matroid M and a depth-decomposition (T, f) of M outputs an extended depth-decomposition (T', f', g) of M such that the depth of T' is at most the depth of T and every branch of T' is at capacity.*

We obtain the following two statements as corollaries of Theorem 18.

▶ **Corollary 19.** *Every vector matroid M has a depth-decomposition (T, f) with depth $\mathrm{bd}(M)$ such that every branch of T is at capacity.*

▶ **Corollary 20.** *If (T, f) is a depth-decomposition of a vector matroid M, then there exists g such that (T, f, g) is an extended depth-decomposition of M.*

We conclude this section with a theorem that asserts that every vector matroid has a depth-decomposition of minimum depth that has a special structure. We need three auxiliary lemmas.

▶ **Lemma 21.** *Let M be a vector matroid and $M_1, \ldots, M_k$ be its components. Further, let (T_i, f_i, g_i) be an extended depth-decomposition of M_i. Let T be the rooted tree obtained from the trees $T_1, \ldots, T_k$ by identifying the roots of the trees, let f be the mapping from the elements of M to the leaves of T such that $f(x) = f_i(x)$ if x belongs to M_i, and let g be the mapping such that $g(v) = g_i(v)$ if v is a non-root vertex of T_i. The triple (T, f, g) is an extended depth-decomposition of M.*

▶ **Lemma 22.** *Let (T, f, g) be an extended depth-decomposition of a vector matroid M, and let u be a vertex with at least two children. If S is a branch rooted at u, then $\widehat{S}$ is a union of components of M/K_u.*

▶ **Lemma 23.** *Let (T, f, g) be an extended depth-decomposition of a vector matroid M, and let u be a vertex with at least two children. Further, let S be a branch rooted at u and (T', f', g') be an extended depth-decomposition of the matroid $(M/K_u)\left[\widehat{S}\right]$. Let T'' be the rooted tree obtained by removing from T the branch S and identifying the root of T' with*

u, setting $f''(x) = f'(x)$ for elements $x \in \widehat{S}$ and $f''(x) = f(x)$ for other elements x of M, and setting $g''(v) = g(v)$ for vertices of T not contained in S and $g''(v) = g'(v) + K_u$ for non-root vertices of T'. The triple (T'', f'', g'') is an extended depth-decomposition of M.

We are now ready to prove the final theorem of this section.

▶ **Theorem 24.** *Every vector matroid M has an extended depth-decomposition (T, f, g) of depth $\mathrm{bd}(M)$ such that every branch of T is both at capacity and solid.*

Proof. We start with a depth-decomposition (T, f, g) of M with depth $\mathrm{td}(M)$ and modify it iteratively as follows. At each iteration, we first apply Theorem 18 to obtain a depth-decomposition such that every branch is at capacity. If every branch is solid, we stop. If there is a branch S that is not solid, we proceed as follows. Since S is not solid, the matroid $(M/K_u)\left[\widehat{S}\right]$ is not connected, where u is the root of S. Let $M_1, \ldots, M_k$ be the components of the matroid $(M/K_u)\left[\widehat{S}\right]$ and let X_u be the set containing all loops of the matroid $(M/K_u)\left[\widehat{S}\right]$. Let (S_i, f_i, g_i) be an extended depth-decomposition of M_i, $i = 1, \ldots, k$, with depth $\mathrm{bd}(M_i)$. Since the branch-depth $\mathrm{bd}(M_i)$ of M_i is at most the branch-depth of $(M/K_u)\left[\widehat{S}\right]$, the depth of each of the trees $S_1, \ldots, S_k$ is at most the depth of S. By Lemmas 21 and 23, it is possible to replace the branch S with the branches $S_1, \ldots, S_k$ rooted at the root of S and assigning the elements of X_u to arbitrary leaves of the branches $S_1, \ldots, S_k$. Note that the depth of the new rooted tree does not exceed the depth of the original rooted tree. In this way, we obtain a new extended depth-decomposition of M, and we proceed to the next iteration. The proof that the procedure described above terminates after at most $r^{\mathrm{bd}(M)+1}$ iterations is left due to space limitations. ◀

7 Algorithm for finite fields

In this section, we design a fixed parameter algorithm for computing a depth-decomposition of a vector matroid over a fixed finite field. To do so, we need to introduce additional notation. Let (T, f, g) be an extended depth-decomposition of a vector matroid M, and let r be the rank of M. Let $u_0, \ldots, u_{2r}$ be a depth-first-search transversal of the tree T (see Figure 2 for illustration). For $i \in \{0, \ldots, 2r\}$, we define A_i to be the linear hull of K_{u_i} and the f-preimage of the leaves among the vertices $u_0, \ldots, u_i$. Similarly, we define B_i to be the linear hull of K_{u_i} and the f-preimage of the leaves among the vertices $u_i, \ldots, u_{2r}$. The sequence $(u_i, A_i, B_i)_{i\in\{0,\ldots,2r\}}$ is called a *transversal sequence* for (T, f, g). Note that $A_i \cap B_i = K_{u_i}$ by the fact that $\mathrm{Im}(g)$ is a basis of the linear hull of elements of M. If (T, f, g) is principal and (T', f', g') is another extended depth-decomposition of M, we say that a branch S of T' is *i-crossed* if $\widehat{S}$ contains the g-image of a vertex on the path from u_i to the root of T.

▶ **Lemma 25.** *Let M be a vector matroid, (T, f, g) a principal extended depth-decomposition of M, and (T', f', g') an extended depth-decomposition of M such that every branch is solid. Further, let $(u_i, A_i, B_i)_{i\in\{0,\ldots,2r\}}$ be a transversal sequence for (T, f, g). If S is a branch of (T', f', g') that is not i-crossed, then $\widehat{S}$ is a subset of A_i or B_i.*

We will design a dynamic programming algorithm, which will be constructing an optimal depth-decomposition of a vector matroid M using the information on the structure of M captured by an extended depth-decomposition of M produced by an approximation algorithm

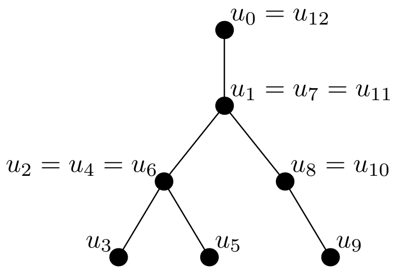


Figure 2 An example of a depth-first-search transversal of a rooted tree.

given in Theorem 6. The depth-decomposition will be constructed iteratively for elements of M in the order in that the leaves that they are assigned to appear in the transversal sequence of the depth-decomposition produced by the approximation algorithm. Since it would not be feasible to store all possible "partial" depth-decompositions, we need a more succinct way of representing an already constructed part of a depth-decomposition, which we now formally introduce.

Let (T, f, g) be a principal extended depth-decomposition of a vector matroid M with rank r over a field $\mathbb{F}$, $(u_i, A_i, B_i)_{i\in\{0,\dots,2r\}}$ a transversal sequence for (T, f, g), and (T', f', g') another extended depth-decomposition of a matroid M. A *frontier* is a tuple (T_0, d, a, b, h) such that d, a, and b are non-negative integers, T_0 is a rooted tree of depth with at most d leaves, and h is a mapping from non-root vertices of T_0 to $\mathbb{F}^{d+a+b}$ such that $\mathrm{Im}(h)$ is a set of linearly independent vectors and for every $j = 1, \dots, d$, there is a leaf of T_0 for which the j-th unit vector is contained in the linear hull of the h-image of the vertices on the path from the leaf to the root of T_0. We will refer the middle a coordinates of images of h as *A-coordinates* and to the last b coordinates as *B-coordinates*.

The *i-frontier* of (T', f', g') with respect to (T, f, g) and $(u_i, A_i, B_i)_{i\in\{0,\dots,2r\}}$ is the frontier (T_0, d, a, b, h) such that

- T_0 is the rooted subtree of T' formed by the paths from the f'-images of U to the root, where U is the set of g-images of the vertices on the path from u_i to the root of T.
- The integer d is the depth of u_i in T.
- The integers a and b are the smallest integers for that there exists an a-dimensional subspace L_A of A_i and a b-dimensional subspace L_B of B_i such that the linear hull of the g'-images of the vertices of T_0 (note that T_0 is a subtree of T') is a subspace of the linear hull of $v^u_1, \dots, v^u_d$, L_A and L_B, where $v^u_1, \dots, v^u_d$ are g-images of the vertices on the path from the root of T to u_i (in this order).
- Finally, h is a mapping from the non-root vertices of T_0 to $\mathbb{F}^{d+a+b}$ that satisfies the following. Let $v^A_1, \dots, v^A_a$ be vectors such that $v^u_1, \dots, v^u_d, v^A_1, \dots, v^A_a$ form a basis of L_A, and let $v^B_1, \dots, v^B_b$ be vectors such that $v^u_1, \dots, v^u_d, v^B_1, \dots, v^B_b$ form a basis of L_B. The value $h(v)$ for a non-root vertex v of T_0 is equal to the coordinates of $f'(v)$ with respect to the (linearly independent) vectors $v^u_1, \dots, v^u_d, v^A_1, \dots, v^A_a, v^B_1, \dots, v^B_a$.

The following lemma justifies the definition of an *i-frontier*. Informally speaking, the lemma says that two depth-decompositions of a vector matroid M can be combined along the same i-frontier, i.e., the i-frontier contains all information that needs to be stored when iteratively constructing a depth-decomposition of M in a dynamic way for the elements of contained in $A_0, A_1, \dots, A_{2r}$.

▶ **Lemma 26.** *Let (T, f, g) be a principal extended depth-decomposition of a vector matroid M, $(u_i, A_i, B_i)_{i\in\{0,\dots,2r\}}$ a transversal sequence for (T, f, g), and (T', f', g') and (T'', f'', g'') two solid extended depth-decompositions of M. Suppose that the i-frontiers of (T', f', g') and (T'', f'', g'') with respect to (T, f, g) and $(u_i, A_i, B_i)_{i\in\{0,\dots,2r\}}$ are the same, and let T_0 be the rooted tree of the i-frontier. Obtain T'_A from T' by removing all branches S with $\widehat{S} \subseteq B_i$ that are not i-crossed, T''_B from T'' by removing all branches S with $\widehat{S} \subseteq A_i$ that are not i-crossed, and T''' by gluing T'_A and T''_B together on the vertices that correspond to each vertex of T_0. Finally, let f''' be a function from the elements of M to the leaves of T''' defined as follows. If $x \in A_i \setminus B_i$, then $f'''(x) = f'(x)$. If $x \in B_i \setminus A_i$, then $f'''(x) = f''(x)$. Lastly, if $x \in A_i \cap B_i = K_{u_i}$, then $f'''(x)$ is any leaf u of T_0 such that $x \in \mathcal{L}(g'(P_u))$. Then (T''', f''') is a depth-decomposition of M.*

Before stating the main result of this section, we need to observe that the number of frontiers for any fixed d is bounded.

▶ **Lemma 27.** *For every integer d and any finite field $\mathbb{F}$, there exist at most $d^{2d+4}|\mathbb{F}|^{2d^4}$ choices of a rooted tree T of depth at most d, integers $d' \le d$, a and b and a mapping h to $\mathbb{F}^{d'+a+b}$ such that (T, d', a, b, h) is a frontier.*

We can now design a dynamic programming algorithm for computing the branch-depth of a matroid represented over a fixed finite field. While there are many technical details that needs to be taken care of, the basic idea of the algorithm is simple: we first obtain an approximate depth-decomposition using Theorem 6 and then proceed computing along its depth-first-search transversal possible frontiers; Lemma 26 guarantees that frontiers capture all information that needs to be carried through dynamic programming, and their number of frontiers is bounded by Lemma 27.

▶ **Theorem 28.** *For the parameterization by a positive integer d and a prime power q, there exists a fixed parameter algorithm that for a vector matroid M over the q-element field either outputs that $\mathrm{bd}(M)$ is larger than d, or outputs a depth-decomposition of M with depth d.*

Proof. We first apply the algorithm from Theorem 6. The algorithm either outputs that the branch-depth of M is larger than d or outputs a principal extended depth-decomposition (T, f, g) of a vector matroid M with depth at most 4^d. For the purpose of the analysis of the algorithm that we present, fix a solid extended depth-decomposition (T_s, f_s, g_s) of M with depth $\mathrm{bd}(M)$, which exists by Theorem 24.

Let r be the rank of the matroid M, and let $(u_i, A_i, B_i)_{i\in\{0,\dots,2r\}}$ be a transversal sequence for (T, f, g). The algorithm then iteratively for $j = 0, \dots, 2r$ computes a list of all frontiers (T_0, d', a, b, h), $d' \le d$ for which there there exists a vector matroid M' with rank $\dim A_i + b$ such that the restrictions of M and M' to the elements contained in the subspace A_i are the same, and a solid extended depth-decomposition (T', f', g') of M' of depth at most d such that (T, d', a, b, h) is the i-frontier of (T', f', g') with respect to (T, f, g) and its transversal sequence.

Note that the number of such frontiers is bounded by a function of d and q only by Lemma 27; the number of edges of T can be shown to be $d' + a + b$. If the branch-depth of M is at most d, then the set of such frontiers is non-empty for every $j = 0, \dots, 2r$: the matroid M' in the i-th iteration can be chosen to be the the union of the restriction of the matroid M to the elements of A_i and the elements of K_u, where u ranges over all the vertices on the path from u_i to the root of T. A solid extended depth-decomposition (T', f', g') can be obtained from (T_s, f_s, g_s) by removing all branches S with $\widehat{S} \subseteq B_i$ that are not i-crossed. So, if the set of the frontiers becomes empty at any of the iterations, the algorithm can stop and output that the branch-depth of M is larger than d.

We now describe the iterations of the algorithm in detail. For $j = 0$, the list of frontiers contains a single element $(R, 0, 0, 0, h)$, where R is the rooted tree that contains the root only and h is the null mapping. We now describe how the algorithm computes the list for $j > 0$ assuming that the list is already available for $j - 1$. The iteration of the algorithm differs according to whether u_j is a parent of u_{j-1} and u_j is a child of u_{j-1}.

We start with describing the case that u_j is a parent of u_{j-1}. Let $d' < d$ be the depth of u_j. Then the depth of u_{j-1} is $d' + 1$. The following is performed for every frontier of the form $(T_0, d' + 1, a, b, h)$ in the list from the previous iteration, and every leaf u of T_0 such that the linear hull of the h-image of the vertices from u to the root contains the $(d' + 1)$-th unit vector. If the leaves of T_0 can be assigned distinct indices $i' = 1, \ldots, d'$ such that the linear hull of the h-image of the vertices from each leaf to the root contains contains the i'-th unit vector, where i' is the index assigned to the leaf, then we include $(T_0, d', a + 1, b, h')$ to the list from the j-th iteration, where h' is a mapping from the non-root vertices of T_0 obtained from h by turning the $(d' + 1)$-th coordinate to be one of the A-coordinates and applying any invertible linear transformation to the $a + 1$ A-coordinates, i.e., we fix such a linear transformation L and set $h'(v) = L(h(v))$ for all vertices v of T_0. If there exists $i' = 1, \ldots, d'$ such that u is the only leaf for which the linear hull of the h-image of the vertices from it to the root contains the i'-th unit vector, we continue with the next choice of (T_0, d', a, b, h') and u. Otherwise, for every $i' = 1, \ldots, d'$ there exists another leaf for which the linear hull of the h-image of the vertices from it to the root contains the i'-th unit vector, which means that we may be able to remove u from T_0. So, let T_0' be the tree obtained from T_0 by removing the path from u to the nearest ancestor with at least two children and turn the $(d' + 1)$-th coordinate to an A-coordinate. If the linear hull of the h-images of the vertices of T_0' restricted to their B-coordinates does not have dimension b, then it is not possible to modify this frontier by removing u and we continue with the next choice of (T_0, d', a, b, h') and u. Otherwise, let a' be the dimension of the linear hull of the h-images of the vertices of T_0' restricted to their A-coordinates and include (T_0, d', a', b, h') to the list from the j-th iteration, where h' is a mapping obtained from h by mapping its A-coordinates by a linear transformation L such that L maps a-dimensional vector space to a'-dimensional vector space and its image has dimension a'.

We next describe the case that u_j is a child of u_{j-1}. Again, let $d' \leq d$ be the depth of u_j. Then the depth of u_{j-1} is $d' - 1$. The following is performed for every frontier of the form $(T_0, d' - 1, a, b, h)$ in the list from the previous iteration. For every leaf u and every $i' = 1, \ldots, b$ such that the unit vector for the i'-th B-coordinate is contained in the linear hull of the h-image of the vertices from u to the root, we turn the i'-th B-coordinate to the d'-th coordinate to obtain h' and include $(T_0, d', a, b - 1, h')$ to the list from the j-th iteration. In addition, we perform the following. For every $\ell = 1, \ldots, d$, we consider a rooted path of length ℓ and identify its root with a vertex T_0 in all possible ways that the depth of the resulting tree T' does not exceed d. Let u be the new leaf of T_0' and let h' be a mapping obtained from h by assigning each of the ℓ new vertices one of the unit vectors for the i'-th B-coordinates for $i' = b + 1, \ldots, b + \ell$. For every invertible linear transformation L to the $b + \ell$ B-coordinates that yields h'' such that the linear hull of the h''-images of the vertices on the path from u to the root of T_0' contains the unit vector for the $(b + \ell)$-th B-coordinate, we turn the $(b + \ell)$-th B-coordinate to the d'-th coordinate to obtain h''' and include $(T_0, d', a', b + \ell - 1, h''')$ to the list from the j-th iteration.

Assume that all the iterations of the algorithm have been performed. If the list of frontiers is empty after any iteration, the branch-depth of M exceeds d and the algorithm reports this. Otherwise, the final list (for $j = 2r$) contains a single element $(R, 0, 0, 0, h)$ where R is the rooted tree that contains the root only. Tracing back to the list for $j = 0$, we obtain a series of frontiers $(T_i, d_i, a_i, b_i, h_i)$, $i = 0, \ldots, 2r$, such that the i-th one can be obtained from the $(i-1)$-th by the operations explained earlier. Note that d_i is the depth of u_i in T. In the frontier $(T_i, d_i, a_i, b_i, h_i)$, the first d_i coordinate of the h-image correspond to the g-image of the vertices on the path from the root of T to u_i (in this order). The way the frontiers were constructed guarantees consistency between the shape of the frontiers and the basis elements displayed by frontiers; in particular, the mappings h_i are projections of the linear hull of elements of M to the subspace corresponding to the i-frontier and the linear transformations used in the steps of the algorithm provides a consistent way of relating these projections with each other. Hence, there exists an extended depth-decomposition (T', f', g') such that $(T_i, d_i, a_i, b_i, h_i)$ is the i-frontier of (T', f', g') with respect to (T, f, g) and $(u_i, A_i, B_i)_{i\in\{0,\ldots,j\}}$. The algorithm computes this extended depth-decomposition (T', f', g') and outputs it. ◀

Theorems 18 and 28 yield the following corollary.

▶ **Corollary 29.** *For the parameterization by a positive integer d and a prime power q, there exists a fixed parameter algorithm that for a vector matroid M over the q-element field either outputs that $\mathrm{bd}(M)$ is larger than d, or computes $\mathrm{bd}(M)$ and outputs an extended depth-decomposition of M with depth $\mathrm{bd}(M)$ such that every branch of the depth-decomposition is at capacity.*

8 Algorithm for rational matrices

In this section, we adopt the algorithm presented in Section 7 to matroids over rationals; the proofs are left due to space limitations. We start with two auxiliary lemmas. We remark that the bound of 2^{2d-1} in Lemma 30 can be replaced with $d \cdot 2^{d-1}$ using a more careful analysis.

▶ **Lemma 30.** *Let M be a vector matroid and (T, f) a depth-decomposition of M with depth d such that every branch is at capacity. There exists a mapping g such that (T, f, g) is an extended depth-decomposition of M and every element of $\mathrm{Im}(g)$ is a linear combination of at most 2^{2d-1} elements of M.*

▶ **Lemma 31.** *Let A be an integer matrix of branch-depth (over $\mathbb{Q}$) at most d such that all its entries are between $-K$ and $+K$. Further, let q be a prime larger than $(K2^{2d})^{2^{2d}}$. The following holds for any subset X of the columns of A: the vectors contained in X are linearly independent over $\mathbb{Q}$ if and only they are independent over the q-element field.*

We derive the following using Theorem 28, Corollary 29, and Lemmas 30 and 31.

▶ **Theorem 32.** *For the parameterization by positive integers d and K, there exists a fixed parameter algorithm that for a vector matroid M over $\mathbb{Q}$ such that the entries of all vectors in M have complexity at most K either outputs that $\mathrm{bd}(M)$ is larger than d, or computes $\mathrm{bd}(M)$ and outputs an extended depth-decomposition (T, f, g) of M with depth $\mathrm{bd}(M)$. Moreover, the entry complexity of the vectors in $\mathrm{Im}(g)$ is bounded by a function of d and K.*

Theorem 32 yields Theorem 3.

References

1 Matthias Aschenbrenner and Raymond Hemmecke. Finiteness theorems in stochastic integer programming. *Foundations of Computational Mathematics*, 7(2):183–227, 2007.

2 Lin Chen and Dániel Marx. Covering a tree with rooted subtrees–parameterized and approximation algorithms. In *29th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 2801–2820. SIAM, 2018.

3 William H. Cunningham and Jim Geelen. On integer programming and the branch-width of the constraint matrix. In *Integer Programming and Combinatorial Optimization, 12th International IPCO Conference*, pages 158–166, 2007.

4 Matt DeVos, O-joung Kwon, and Sang-il Oum. Branch-depth: Generalizing tree-depth of graphs. *preprint*, 2019. `arXiv:1903.11988`.

5 Pavel Dvořák, Eduard Eiben, Robert Ganian, Dušan Knop, and Sebastian Ordyniak. Solving integer linear programs with a small number of global variables and constraints. In *26th International Joint Conference on Artificial Intelligence*, pages 607–613. AAAI Press, 2017.

6 Friedrich Eisenbrand, Christoph Hunkenschröder, and Kim-Manuel Klein. Faster Algorithms for Integer Programs with Block Structure. In *45th International Colloquium on Automata, Languages, and Programming, ICALP*, pages 49:1–49:13, 2018.

7 Friedrich Eisenbrand, Christoph Hunkenschröder, Kim-Manuel Klein, Martin Koutecký, Asaf Levin, and Shmuel Onn. An algorithmic theory of integer programming. *preprint*, 2019. `arXiv:1904.01361`.

8 Fedor V. Fomin, Fahad Panolan, M. S. Ramanujan, and Saket Saurabh. On the optimality of pseudo-polynomial algorithms for integer programming. In *26th Annual European Symposium on Algorithms, ESA*, pages 31:1–31:13, 2018.

9 Robert Ganian and Sebastian Ordyniak. The complexity landscape of decompositional parameters for ILP. *Artificial Intelligence*, 2018.

10 Robert Ganian, Sebastian Ordyniak, and M. S. Ramanujan. Going beyond primal treewidth for (M)ILP. In *31st AAAI Conference on Artificial Intelligence*, pages 815–821, 2017.

11 Tomas Gavenčiak, Daniel Král', and Sang-il Oum. Deciding first order properties of matroids. In *39th International Colloquium Automata, Languages, and Programming, ICALP*, volume 7392 of *Lecture Notes in Computer Science*, pages 239–250, 2012.

12 Jim Geelen, Albertus Gerards, Neil Robertson, and Geoff Whittle. On the excluded minors for the matroids of branch-width k. *Journal of Combinatorial Theory, Series B*, 88:261–265, 2003.

13 Paul Halmos. *Finite-Dimensional Vector Spaces*. Undergraduate Texts in Mathematics. Springer, 1993.

14 Raymond Hemmecke, Matthias Köppe, and Robert Weismantel. Graver basis and proximity techniques for block-structured separable convex integer minimization problems. *Mathematical Programming*, 145:1–18, 2014.

15 Raymond Hemmecke, Shmuel Onn, and Lyubov Romanchuk. N-fold integer programming in cubic time. *Mathematical Programming*, 137:325–341, 2013.

16 Petr Hliněný. Branch-width, parse trees, and monadic second-order logic for matroids. In Helmut Alt and Michel Habib, editors, *20th Annual Symposium on Theoretical Aspects of Computer Science, STACS*, volume 2607 of *LNCS*, pages 319–330, 2003.

17 Petr Hliněný. On matroid properties definable in the MSO logic. In Branislav Rovan and Peter Vojtáš, editors, *27th International Symposium on Mathematical Foundations of Computer Science, MFCS*, volume 2747 of *LNCS*, pages 470–479, 2003.

18 Petr Hliněný. Branch-width, parse trees, and monadic second-order logic for matroids. *Journal of Combinatorial Theory, Series B*, 96(3):325–351, 2006.

19 Petr Hliněný and Sang-il Oum. Finding branch-decompositions and rank-decompositions. In Lars Arge, Michael Hoffmann, and Emo Welzl, editors, *15th Annual European Symposium, ESA*, volume 4698 of *LNCS*, pages 163–174, 2007.

20 Petr Hliněný and Sang-il Oum. Finding branch-decompositions and rank-decompositions. *SIAM Journal on Computing*, 38(3):1012–1032, 2008.

21 Ravi Kannan. Minkowski's convex body theorem and integer programming. *Mathematics of Operations Research*, 12(3):415–440, August 1987.

22 František Kardoš, Daniel Král', Anita Liebenau, and Lukáš Mach. First order convergence of matroids. *preprint*, 2015. `arXiv:1501.06518`.

23 František Kardoš, Daniel Král', Anita Liebenau, and Lukáš Mach. First order convergence of matroids. *Eur. J. Comb*, 59:150–168, 2017.

24 Dušan Knop, Martin Koutecký, and Matthias Mnich. Voting and bribing in single-exponential time. In *34th Symposium on Theoretical Aspects of Computer Science, STACS*, volume 66, pages 46:1–46:14, 2017.

25 Martin Koutecký, Asaf Levin, and Shmuel Onn. A parameterized strongly polynomial algorithm for block structured integer programs. In *45th International Colloquium on Automata, Languages, and Programming, ICALP*, volume 107, pages 85:1–85:14, 2018.

26 Hendrik W. Lenstra, Jr. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8(4):538–548, 1983.

27 Susan Margulies, Jing Ma, and Illya V. Hicks. The Cunningham-Geelen method in practice: Branch-decompositions and integer programming. *INFORMS Journal on Computing*, 25(4):599–610, 2013.

Dynamic Longest Common Substring in Polylogarithmic Time

Panagiotis Charalampopoulos
Department of Informatics, King's College London, UK
Institute of Informatics, University of Warsaw, Poland
panagiotis.charalampopoulos@kcl.ac.uk

Paweł Gawrychowski
Institute of Computer Science, University of Wrocław, Poland
gawry@cs.uni.wroc.pl

Karol Pokorski
Institute of Computer Science, University of Wrocław, Poland
pokorski@cs.uni.wroc.pl

Abstract

The longest common substring problem consists in finding a longest string that appears as a (contiguous) substring of two input strings. We consider the dynamic variant of this problem, in which we are to maintain two dynamic strings S and T, each of length at most n, that undergo substitutions of letters, in order to be able to return a longest common substring after each substitution. Recently, Amir et al. [ESA 2019] presented a solution for this problem that needs only $\tilde{\mathcal{O}}(n^{2/3})$ time per update. This brought the challenge of determining whether there exists a faster solution with polylogarithmic update time, or (as is the case for other dynamic problems), we should expect a polynomial (conditional) lower bound. We answer this question by designing a significantly faster algorithm that processes each substitution in amortized $\log^{\mathcal{O}(1)} n$ time with high probability. Our solution relies on exploiting the local consistency of the parsing of a collection of dynamic strings due to Gawrychowski et al. [SODA 2018], and on maintaining two dynamic trees with labeled bicolored leaves, so that after each update we can report a pair of nodes, one from each tree, of maximum combined weight, which have at least one common leaf-descendant of each color. We complement this with a lower bound of $\Omega(\log n / \log \log n)$ for the update time of any polynomial-size data structure that maintains the LCS of two dynamic strings, even allowing amortization and randomization.

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases string algorithms, dynamic algorithms, longest common substring

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.27

Category Track A: Algorithms, Complexity and Games

Related Version A full version of the paper is available at `https://arxiv.org/abs/2006.02408`.

Funding *Panagiotis Charalampopoulos*: Partially supported by ERC grant TOTAL under the European Union's Horizon 2020 Research and Innovation Programme (agreement no. 677651).

1 Introduction

The well-known longest common substring (LCS) problem, formally stated below, was conjectured by Knuth to require $\Omega(n \log n)$ time. However, in his seminal paper that introduced suffix trees, Weiner showed how to solve it in linear time (for constant alphabets) [29]. Since then, this classical question was considered in many different versions, such as obtaining a tradeoff between the time and the working space [21, 27], or computing an approximate LCS under either the Hamming or the edit distance (see [9, 20, 28] and references therein), to name a few.

Problem: Longest Common Substring
Input: Two strings S and T of length at most n over an alphabet Σ.
Output: A longest substring X of S that is a substring of T.

We consider the dynamic version of this problem where the strings are updated and we are to report an LCS after each update. That is, we return the length of an LCS and a pair of starting positions of its occurrences in the strings. The allowed update operations are substitutions of single letters in either S or T. In fact, with due care, our algorithms can be adapted to handle all edit operations, i.e. insertions and deletions as well, but we only allow substitutions for the sake of a clearer exposition of the main ideas.

Dynamic problems on strings are of wide interest. Maybe the most basic question in this direction is that of maintaining a dynamic text while enabling efficient pattern matching queries. This is clearly motivated by, say, the possible application in a text editor. The first structure achieving polylogarithmic update time and optimal query time for this problem was designed by Sahinalp and Vishkin [26]. Later, the update time was improved to $\mathcal{O}(\log^2 n \log\log n \log^* n)$ at the cost of $\mathcal{O}(\log n \log\log n)$ additional time per query by Alstrup et al. [2]. Recently, Gawrychowski et al. [15] presented a data structure that requires $\mathcal{O}(\log^2 n)$ time per update and allows for time-optimal queries. Other problems on strings that have been studied in the dynamic setting include maintaining repetitions, such as the set of square substrings [5] or a longest palindromic substring [4, 7].

As for the LCS problem itself, Amir et al. [6] initiated the study of this question in the dynamic setting by considering the problem of constructing a data structure over two strings that returns the LCS after a single edit operation in one of the strings. However, in their solution, after each edit operation, the string is immediately reverted to its original version. Abedin et al. [1] improved the tradeoffs for this problem by designing a more efficient solution for the so-called heaviest induced ancestors problem. Amir and Boneh [3] investigated some special cases of the *partially dynamic LCS* problem (in which one of the strings is assumed to be static); namely, the case where the static string is periodic and the case where the substitutions in the dynamic string are substitutions with some letter $\# \notin \Sigma$. Finally, Amir et al. [7] presented the first algorithm for the *fully dynamic LCS* problem (in which both strings are subject to updates) that needs only sublinear time per edit operation (insertion or deletion of a letter) in either string, namely $\tilde{\mathcal{O}}(n^{2/3})$. As a stepping stone towards this result, they designed an algorithm for the partially dynamic LCS problem that takes $\tilde{\mathcal{O}}(\sqrt{n})$ time per edit operation.

For some natural dynamic problems, the best known bounds on the query and the update time are of the form $\mathcal{O}(n^\alpha)$, where n is the size of the input and α is some constant. Henzinger et al. [16] introduced the online Boolean matrix-vector multiplication conjecture that can be used to provide some justification for the polynomial time hardness of many such dynamic problems in a unified manner. This brings the question of determining if the bound on the update time in the dynamic LCS problem should be polynomial or subpolynomial.

We answer this question by significantly improving on the bounds presented by Amir et al. [7] and presenting a solution for the fully dynamic LCS problem that handles each update in amortized polylogarithmic time with high probability. As a warm-up, we present a (relatively simple) deterministic solution for the partially dynamic LCS problem that handles each update in amortized $\mathcal{O}(\log^2 n)$ time.

After having determined that the complexity of fully dynamic LCS is polylogarithmic, the next natural question is whether we can further improve the bound to polyloglogarithmic. By now we have techniques that can be used to not only distinguish between these two situations but (in some cases) also provide tight bounds. As a prime example, static predecessor for a set of n numbers from $[n^2]$ requires $\Omega(\log\log n)$ time for structures of size $\tilde{\mathcal{O}}(n)$ [25], and dynamic connectivity for forests requires $\Omega(\log n)$ time [24], with both bounds being asymptotically tight. In some cases, seemingly similar problems might have different complexities, as in the orthogonal range emptiness problem: Nekrich [22] showed a data structure of size $\mathcal{O}(n\log^4 n)$ with $\mathcal{O}(\log^2\log n)$ query time for 3 dimensions, while for the same problem in 4 dimensions Pătraşcu showed that any polynomial-size data structure requires $\Omega(\log n/\log\log n)$ query time [23]. In the full version of this work, we show the following results, each obtained through a series of reductions, starting from the problem of answering reachability queries in butterfly graphs that was considered in the seminal paper of Pătraşcu [23].

▶ **Theorem 1.** *Any structure of $\tilde{\mathcal{O}}(n)$ size for maintaining an LCS of a dynamic string S and a static string T, each of length at most n, requires $\Omega(\log n/\log\log n)$ time per update operation.*

▶ **Theorem 2.** *Any polynomial-size structure for maintaining the LCS of two dynamic strings of length n requires $\Omega(\log n/\log\log n)$ time per update operation.*

Finally, we demonstrate that the difference in the allowed space in the above two lower bounds is indeed needed. To this end, we show that partially dynamic LCS admits an $\mathcal{O}(n^{1+\epsilon})$-space, $\mathcal{O}(\log\log n)$-update time solution, for any $\epsilon > 0$.

Techniques and roadmap. We first consider the partially dynamic version of the problem where updates are only allowed in one of the strings, say S, in Section 3. This problem is easier as we can use the static string T as a reference point. We maintain a partition of S into blocks (i.e. substrings of S whose concatenation equals S), such that each block is a substring of T, but the concatenation of any two consecutive blocks is not. This is similar to the approach of [8] and other works that consider one dynamic and one static string. The improvement upon the $\tilde{\mathcal{O}}(\sqrt{n})$-time algorithm presented in [7] comes exactly from imposing the aforementioned maximality property, which guarantees that the sought LCS is a substring of the concatenation of at most three consecutive blocks and contains the first letter of one of these blocks. The latter property allows us to anchor the LCS in S. Upon an update, we can maintain the block decomposition, by updating a constant number of blocks. It then suffices to show how to efficiently compute the longest substring of T that contains the first letter of a given block. We reduce this problem to answering a *heaviest induced ancestors* (HIA) query. This reduction was also presented in [1, 6], but we describe the details to make following the more involved solution of fully dynamic LCS easier.

In Section 4 we move to the fully dynamic LCS problem. We try to anchor the LCS in both strings as follows. For each of the strings S and T we show how to maintain, in $\log^{\mathcal{O}(1)} n$ time, a collection of pairs of adjacent fragments (e.g. $(S[i\mathinner{.\,.} j-1], S[j\mathinner{.\,.} k])$), denoted by J_S for S and J_T for T, with the following property. For any common substring X of S and T there exists a partition $X = X_\ell X_r$ for which there exists a pair $(U_\ell, U_r) \in J_S$ and a pair $(V_\ell, V_r) \in J_T$ such that X_ℓ is a suffix of both U_ℓ and V_ℓ, while X_r is a prefix of both U_r and V_r. We can maintain this collection by exploiting the properties of the locally consistent parsing previously used for maintaining a dynamic collection of strings [15]. We maintain tries for fragments in the collections and reduce the dynamic LCS problem to a problem on dynamic bicolored trees, which we solve by using dynamic heavy-light decompositions and 2D range trees.

2 Preliminaries

We use $[n]$ to denote the set $\{1, 2, \ldots, n\}$. Let $S = S[1]S[2] \cdots S[n]$ be a *string* of length $|S| = n$ over an integer alphabet Σ. For two positions i and j on S, we denote by $S[i\,..\,j] = S[i] \cdots S[j]$ the *fragment* of S that starts at position i and ends at position j (it is the empty string ε if $j < i$). A string Y, of length m with $0 < m \leq n$, is a *substring* of S if there exists a position i in S such that $Y = S[i\,..\,i+m-1]$. The prefix of S ending at the i-th letter of S is denoted by $S[..\,i]$ and the suffix of S starting at the i-th letter of S is denoted by $S[i\,..]$. The reverse string of S is denoted by S^R. The concatenation of strings S and T is denoted by ST, and the concatenation of k copies of string S is denoted by S^k. By $\mathsf{lcp}(S, T)$ we denote the length of the longest common prefix of strings S and T.

We define the trie of a collection of strings $C = \{S_1, S_2, \ldots, S_k\}$ as follows. It is a rooted tree with edges labeled by single letters. Every string S that is a prefix of some string in C is represented by exactly one path from the root to some node v of the tree, such that the concatenation of the labels of the edges of the path, the *path-label* of v, is equal to S. The compacted trie of C is obtained by contracting maximal paths consisting of nodes with one child to an edge labeled by the concatenation of the labels of the edges of the path. Usually, the label of the new edge is stored as the start/end indices of the corresponding fragment of some S_i. The *suffix tree* of a string T is the compacted trie of all suffixes of $T\$$ where \$ is a letter smaller than all letters of the alphabet Σ. It can be constructed in $\mathcal{O}(|T|)$ time for linear-time sortable alphabets [11]. For a node u in a (compacted) trie, we define its *depth* as the number of edges on the path from the root to u. Analogously, we define the *string-depth* of u as the total length of labels along the path from the root to u.

We say that a tree is weighted if there is a weight $w(u)$ associated with each node u of the tree, such that weights along the root-to-leaf paths are increasing, i.e. for any node u other than the root, $w(u) > w(\text{parent}(u))$. Further, we say that a tree is labeled if each of its leaves is given a distinct label.

▶ **Definition 3.** *For rooted, weighted, labeled trees $\mathcal{T}_1$ and $\mathcal{T}_2$, two nodes $u \in \mathcal{T}_1$ and $v \in \mathcal{T}_2$, are* induced *(by ℓ) if and only if there are leaves x and y with the same label ℓ, such that x is a descendant of u and y is a descendant of v.*

Problem: Heaviest Induced Ancestors
Input: Two rooted, weighted, labeled trees $\mathcal{T}_1$ and $\mathcal{T}_2$ of total size n.
Query: Given a pair of nodes $u \in \mathcal{T}_1$ and $v \in \mathcal{T}_2$, return a pair of nodes u', v' such that u' is ancestor of u, v' is ancestor of v, u' and v' are induced and they have the largest total combined weight $w(u') + w(v')$.

This problem was introduced in [14], with the last advances made in [1]. The next lemma encapsulates one of the known trade-offs.

▶ **Lemma 4** ([14]). *There is a data structure for the Heaviest Induced Ancestors problem, that can be built in $\mathcal{O}(n \log^2 n)$ time and answers queries in $\mathcal{O}(\log^2 n)$ time.*

3 Partially Dynamic LCS

In this section, we describe an algorithm for solving the partially dynamic variant of the LCS problem, where updates are only allowed on one of the strings, say S, while T is given in advance and is not subject to change.

Let us assume for now that all the letters of S throughout the execution of the algorithm occur at least once in T; we will waive this assumption later. Also, for simplicity, we assume that S is initially equal to $\$^{|S|}$, for $\$ \notin \Sigma$. We can always obtain any other initial S by performing an appropriate sequence of updates in the beginning.

▶ **Definition 5.** *A* block decomposition *of string S with respect to string T is a sequence of strings $(s_1, s_2, \dots, s_k)$ such that $S = s_1 s_2 \dots s_k$ and every s_i is a fragment of T. An element of the sequence is called a* block *of the decomposition. A decomposition is* maximal *if and only if $s_i s_{i+1}$ is not a substring of T for every $i \in [k-1]$.*

Maximal block decompositions are not necessarily unique and may have different lengths, but all admit the following useful property.

▶ **Lemma 6.** *For any maximal block decomposition of S with respect to T, any fragment of S that occurs in T is contained in at most three consecutive blocks. Furthermore, any occurrence of an LCS of S and T in S must contain the first letter of some block.*

Proof. We prove the first claim by contradiction. If $(s_1, s_2, \dots, s_k)$ is a maximal block decomposition of S with respect to T and a fragment of S that occurs in T spans at least four consecutive blocks $s_i, s_{i+1}, s_{i+2}, \dots, s_j$, then $s_{i+1}s_{i+2}$ is a substring of T, a contradiction.

As for the second claim, it is enough to observe, that if an occurrence of an LCS in S starts in some other than the first position of a block b, then it must contain the first letter of the next block, as otherwise its length would be smaller than the length of block b, which is a common substring of S and T. ◀

We will show that an update in S can be processed by considering a constant number of blocks in a maximal block decomposition of S with respect to T. We first summarize the basic building block needed for efficiently maintaining such a maximal block decomposition.

▶ **Lemma 7.** *Let T be a string of length at most n. After $\mathcal{O}(n \log^2 n)$-time and $\mathcal{O}(n)$-space preprocessing, given two fragments U and V of T, one can compute a longest fragment of T that is equal to a prefix of UV in $\mathcal{O}(\log\log n)$ time.*

Proof. We build a weighted ancestor queries structure over the suffix tree of T. A weighted ancestor query (ℓ, u) on a (weighted) tree $\mathcal{T}$, asks for the deepest ancestor of u with weight at most ℓ. Such queries can be answered in $\mathcal{O}(\log\log n)$ time after an $\mathcal{O}(n)$-time preprocessing of $\mathcal{T}$ if all weights are polynomial in n [12], as is the case for suffix trees with the weight of each node being its string-depth. We also build a data structure for answering unrooted LCP queries over the suffix tree of T. In our setting, such queries can be defined as follows: given nodes u and v of the suffix tree of T, we want to compute the (implicit or explicit) node where the search for the path-label of v starting from node u ends. Cole et al. [10] showed how to construct in $\mathcal{O}(n \log^2 n)$ time a data structure of size $\mathcal{O}(n \log n)$ that answers unrooted LCP queries in $\mathcal{O}(\log\log n)$ time. With these data structures at hand, the longest prefix of UV that is a fragment of T can be computed as follows. First, we retrieve the nodes of the suffix tree of T corresponding to U and V using weighted ancestor queries in $\mathcal{O}(\log\log n)$ time. In more detail, if $U = T[i\mathinner{.\,.} j]$ then we access the leaf of the suffix tree corresponding to $T[i\mathinner{.\,.}]$ and access its ancestor at string-depth $|U|$, and similarly for V. Second, we ask an unrooted LCP query to obtain the node corresponding to the sought prefix of UV. ◀

▶ **Lemma 8.** *A maximal block decomposition of a dynamic string S, with respect to a static string T, can be maintained in $\mathcal{O}(\log\log n)$ time per substitution operation with a data structure of size $\mathcal{O}(n \log n)$ that can be constructed in $\mathcal{O}(n \log^2 n)$ time.*

Proof. We keep the blocks on a doubly-linked list and we store the starting positions of blocks in an $\mathcal{O}(n)$-size predecessor/successor data structure over $[n]$ that supports $\mathcal{O}(\log\log n)$-time queries and updates [30]. This allows us to navigate in the structure of blocks, and in particular to be able to compute the block in which the edit occurred and its neighbors.

Suppose that we have a maximal block decomposition $B = (s_1, \ldots, s_k)$ of S with respect to T. Consider an operation which updates the letter x located in block s_i to y, so that $s_i = s_i^{\mathrm{l}} x s_i^{\mathrm{r}}$. Consider a block decomposition $B' = (s_1, s_2, \ldots, s_{i-1}, s_i^{\mathrm{l}}, y, s_i^{\mathrm{r}}, s_{i+1}, \ldots, s_k)$ of string S' after the update. Note that both s_i^{l} and s_i^{r} may be empty. This block decomposition does not need to be maximal. However, since B is a maximal block decomposition of S, none of the strings s_1s_2, s_2s_3, $\ldots$, $s_{i-2}s_{i-1}$, $s_{i+1}s_{i+2}$, $s_{i+2}s_{i+3}$, $\ldots$, $s_{k-1}s_k$ occurs in T. Thus, given B', we repeatedly merge any two consecutive blocks from $(s_{i-1}, s_i^{\mathrm{l}}, y, s_i^{\mathrm{r}}, s_{i+1})$ whose concatenation is a substring of T into one, until this is no longer possible. We have at most four merges before obtaining a maximal block decomposition B' of string S'. Each merge is implemented with Lemma 7 in $\mathcal{O}(\log\log n)$ time. ◀

As for allowing substitutions of letters that do not occur in T, we simply allow blocks of length 1 that are not substrings of T in block decompositions, corresponding to such letters. It is readily verified that all the statements above still hold.

Due to Lemma 6, for a maximal block decomposition $(s_1, s_2, \ldots, s_k)$ of S with respect to T, we know that any occurrence of an LCS of S and T in S must contain the first letter of some block of the decomposition and cannot span more than three blocks. In other words, it is the concatenation of a potentially empty suffix of $s_{i-1}s_i$ and a potentially empty prefix of $s_{i+1}s_{i+2}$ for some $i \in [k]$ (for convenience we consider the non-existent s_is to be equal to ε). We call an LCS that can be decomposed in such way a candidate of s_i. Our goal is to maintain the candidate proposed by each s_i in a max-heap with the length as the key. We also store a pointer to it from block s_i. The max-heap is implemented with an $\mathcal{O}(n)$-size predecessor/successor data structure over $[n]$ that supports $\mathcal{O}(\log\log n)$-time queries and updates [30]. We assume that each block s_i stores a pointer to its candidate in the max-heap.

After an update, the candidate of each block b that satisfies the following two conditions remains unchanged: (a) b did not change and (b) neither of b's neighbors at distance at most 2 changed. For the $\mathcal{O}(1)$ blocks that changed, we proceed as follows. First, in $\mathcal{O}(\log\log n)$ time, we remove from the max-heap any candidates proposed by the deleted blocks or blocks whose neighbors at distance at most 2 have changed. Then, for each new block and for each block whose neighbors at distance at most 2 have changed, we compute its candidate and insert it to the max-heap. To compute the candidate of a block s_i, we proceed as follows. We first compute the longest suffix U of $s_{i-1}s_i$ and the longest prefix V of $s_{i+1}s_{i+2}$ that occur in T in $\mathcal{O}(\log\log n)$ time using Lemma 7. Then, the problem in scope can be restated as follows: given two fragments U and V of T compute the longest fragment of UV that occurs in T. This problem can be reduced to a single HIA query over the suffix trees of T and T^R as shown in [1, 6] and we provide a brief overview at the end of this section. Combining the above discussion with Lemmas 4 and 8 we obtain that an LCS can be maintained after an $\mathcal{O}(n\log^2 n)$ time preprocessing in $\mathcal{O}(\log^2 n)$ time per update. In fact, the bottleneck in the update time in this approach is in Lemma 4, that is, the HIA structure, as the additional time in the update is only $\mathcal{O}(\log\log n)$. We can thus obtain a faster data structure at the expense of slower preprocessing using the following lemma.

▶ **Lemma 9.** *For any $\varepsilon > 0$, there is a structure for the* Heaviest Induced Ancestors *problem, that can be built in $\mathcal{O}(n^{1+\varepsilon})$ time and answers queries in constant time.*

Proof. Consider an instance of HIA on two trees $\mathcal{T}_1$ and $\mathcal{T}_2$ of total size m containing at most ℓ leaves, and let b be a parameter to be chosen later. We will show how to construct a structure of size $\mathcal{O}(b^2 m)$ that allows us to reduce in constant time a query concerning two nodes $u \in \mathcal{T}_1$ and $v \in \mathcal{T}_2$ to two queries to smaller instances of HIA. In each of the smaller instances the number of leaves will shrink by a factor of b, and the total size of all smaller instances will be $\mathcal{O}(m)$. Let $b = n^{\delta}$, where n is the total size of the original trees. We recursively repeat the construction always choosing b according to the formula. Because the depth of the recursion is at most $\log_b n = \mathcal{O}(1)$, this results in a structure of total size $\mathcal{O}(n^{1+\varepsilon})$ for $\varepsilon = 2\delta$ and allows us to answer any query in a constant number of steps, each taking constant time.

We select b evenly-spaced (in the order of in-order traversal) leaves of $\mathcal{T}_1$ and $\mathcal{T}_2$ and call them marked. Consider a query concerning a pair of nodes $u \in \mathcal{T}_1$ and $v \in \mathcal{T}_2$. Let u'', v'' be the nearest ancestors of u and v, respectively, that contain at least one marked leaf in their subtrees. u'' and v'' can be preprocessed in $\mathcal{O}(m)$ space and accessed in constant time. We have three possibilities concerning the sought ancestors u', v':

1. u' is an ancestor of u'' and v' is an ancestor of v'' (not necessarily proper),
2. u' is a descendant of u'',
3. v' is a descendant of v''.

To check the first possibility, we preprocess every pair of marked leaves x, y. Both u'' and v'' store pointers to some marked leaves in their subtrees, so it is enough to consider a query concerning two ancestors of marked leaves x, y. This can be solved similarly as preprocessing two heavy paths for HIA queries in $\mathcal{O}(\log^2 n)$ time [14], except that now we can afford to preprocess the predecessor for every possible depth on both paths in $\mathcal{O}(m)$ space, which decreases the query time to constant. The overall space is $\mathcal{O}(b^2 m)$.

The second and the third possibility are symmetric, so we focus on the second. By removing all marked leaves and their ancestors from $\mathcal{T}_1$ we obtain a collection of smaller trees, each containing less than n/b leaves. Because u' is below u'', u and u' belong to the same smaller tree. For technical reasons, we want to work with $\mathcal{O}(b)$ smaller trees, so we merge all smaller trees between two consecutive marked leaves by adding the subtree induced by their roots in $\mathcal{T}_1$. Now consider the smaller tree $\mathcal{T}_1^i$ containing u (and, by assumption, also u''). We extract the subtree of $\mathcal{T}_2$ induced by the leaves of $\mathcal{T}_1^i$, call it $\mathcal{T}_2^i$, and build a smaller instance of HIA for $\mathcal{T}_1^i$ and $\mathcal{T}_2^i$. To query the smaller instance, we need to replace v by its nearest ancestor that belong to $\mathcal{T}_2^i$. This can be preprocessed for each i and v in $\mathcal{O}(bm)$ space. By construction, $\mathcal{T}_1^i$ and $\mathcal{T}_2^i$ contain less than n/b leaves, and each node of $\mathcal{T}_1$ shows up in at most two trees $\mathcal{T}_1^i$. Each node of $\mathcal{T}_2$ might appear in multiple trees $\mathcal{T}_2^i$, but the number of non-leaf nodes in $\mathcal{T}_2^i$ is smaller than its number of leaves, so the overall number of non-leaf nodes is smaller than m, and consequently the overall number of nodes is smaller than $2m$.

The construction time can be verified to be at most the size of the structure. ◀

▶ **Theorem 10.** *It is possible to maintain an LCS of a dynamic string S and a static string T, each of length at most n, (i) after an $\mathcal{O}(n \log^2 n)$-time preprocessing in $\mathcal{O}(\log^2 n)$ time per substitution operation, or (ii) after an $\mathcal{O}(n^{1+\varepsilon})$-time preprocessing in $\mathcal{O}(\log\log n)$ time per substitution operation.*

We now briefly explain the reduction to HIA in the interests of self-containment and developing intuition in a relatively easier setting before we move on to the harder problem of maintaining an LCS of two dynamic strings.

Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be the suffix trees of $T\$$ and $T^R\#$, respectively, where \$ and # are sentinel letters not in the alphabet and lexicographically smaller than all other letters. Note that each suffix of $T\$$ corresponds to a leaf in $\mathcal{T}_1$; similarly for $\mathcal{T}_2$. We label a leaf v of $\mathcal{T}_1$ with the starting position of the suffix of $T\$$ that it represents. For $\mathcal{T}_2$, however, we label the leaf corresponding to $T^R[i\,..]\#$ with $n-i+2$. Intuitively, if we consider a split $T = T[..\,i-1]T[i\,..]$, the leaves corresponding to $T[i\,..]\$$ in $\mathcal{T}_1$ and $T[..\,i-1]^R\#$ in $\mathcal{T}_2$ get the same label. Further, let the weight of each node in $\mathcal{T}_1$ and $\mathcal{T}_2$ be its string-depth. Upon query, we first compute the node p corresponding to V in $\mathcal{T}_1$ and the node q corresponding to U^R in $\mathcal{T}_2$ using weighted ancestor queries in $\mathcal{O}(\log\log n)$ time. Then the length of the longest substring of UV is exactly the sum of the weights of the nodes returned by a HIA query for p and q. (Some technicalities arise when p or q are implicit nodes, which can be overcome straightforwardly.)

4 Fully Dynamic LCS

In this section, we prove our main result.

▶ **Theorem 11.** *We can maintain an LCS of two dynamic strings, each of length at most n, in $\log^{\mathcal{O}(1)} n$ time per substitution operation.*

We start with some intuition. Let us suppose that we can maintain a decomposition of each string in blocks of length roughly 2^k for each level $k = 0, 1, \dots, \log n$ with the following property: any two equal fragments $U = S[i\,..\,j]$ and $V = T[i'\,..\,j']$ are "aligned" by a pair of equal blocks B_1 in S and B_2 in T at some level k such that $2^k = \Theta(|U|)$. In other words, the decomposition of U (resp. V) at level k consists of a constant number of blocks, where the first and last blocks are potentially trimmed, including B_1 (resp. B_2), and the distance of the starting position of B_1 from position i in S equals the distance of the starting position of B_2 from position i' in T. The idea is that we can use such blocks as anchors for the LCS. For each level, for each string B appearing as a block in this level, we would like to design a data structure that:

a) supports insertions/deletions of strings corresponding to sequences of a constant number of level-k blocks, each containing a specified block equal to B and a boolean variable indicating the string this sequence originates from (S or T), and

b) can return the longest common substring among pairs of elements originating from different strings that is aligned by a pair of specified blocks (that are equal to B).

For each substitution in either of the strings, we would only need to update $\mathcal{O}(\log n)$ entries in our data structures – a constant number of them per level.

Unfortunately, it is not clear how to maintain a decomposition with these properties. We resort to the dynamic maintenance of a *locally consistent parsing* of the two strings, due to Gawrychowski et al. [15]. We exploit the structure of this parsing in order to apply the high-level idea outlined above in a much more technically demanding setting.

4.1 Locally Consistent Parsing

The authors of [15] settled the time complexity of maintaining a collection of strings $\mathcal{W}$ under the following operations: $\mathsf{makestring}(W)$ (insert a non-empty string W), $\mathsf{concat}(W_1, W_2)$ (insert W_1W_2 to $\mathcal{W}$, for $W_1, W_2 \in \mathcal{W}$), $\mathsf{split}(W, i)$ (split the string W at position i and insert both resulting strings to $\mathcal{W}$, for $W \in \mathcal{W}$), $\mathsf{lcp}(W_1, W_2)$ (return the length of the longest common prefix of W_1 and W_2, for $W_1, W_2 \in \mathcal{W}$). Let us note that operations concat and split do not remove their arguments from $\mathcal{W}$. A substitution can be implemented with a constant number of calls to such operations.

▶ **Theorem 12** (Gawrychowski et al. [15]). *A collection $\mathcal{W}$ of strings of total length n can be dynamically maintained under operations $\mathsf{makestring}(W)$, $\mathsf{concat}(W_1, W_2)$, $\mathsf{split}(W, i)$, and $\mathsf{lcp}(W_1, W_2)$ with the operations requiring time $\mathcal{O}(\log n + |W|)$, $\mathcal{O}(\log n)$, $\mathcal{O}(\log n)$ worst-case time with high probability and $\mathcal{O}(1)$ worst-case time, respectively.*

At the heart of Theorem 12 lies a locally consistent parsing of the strings in the collection that can be maintained efficiently while the strings undergo updates. It can be interpreted as a dynamic version of the recompression method of Jeż [18, 19] (see also [17]) for a static string T. As such, we first describe the parsing of Theorem 12 for a static string T and then extend the description to the dynamic variant for a collection of strings.

A *run-length straight line program* (*RLSLP*) is a context-free grammar which generates exactly one string and contains two kinds of non-terminals: *concatenations* with production rule of the form $A \rightarrow BC$ (for symbols B, C) and *powers* with production rule of the form $A \rightarrow B^k$ (for a symbol B and an integer $k \geq 2$), where a *symbol* can be a non-terminal or a letter in Σ. Every symbol A generates a unique string denoted by $\mathsf{gen}(A)$.

Let $T = T_0$. We can compute strings $T_1, \ldots, T_H$, where $H = \mathcal{O}(\log n)$ and $|T_H| = 1$ in $\mathcal{O}(n)$ time using interleaved calls to the following two auxiliary procedures:

RunCompress applied if h is even: for each B^r, $r > 1$, replace all occurrences of B^r as a run by a new letter A. There are no runs after an application of this procedure.[1]

HalfCompress applied if h is odd: first partition Σ into Σ_ℓ and Σ_r; then, for each pair of letters $B \in \Sigma_\ell$, $C \in \Sigma_r$ such that BC occurs in T_h replace all occurrences of BC by a new letter A.

We can interpret strings $T = T_0, T_1, \ldots, T_H$ as an *uncompressed parse tree* $\mathsf{PT}(T)$, by considering their letters as nodes, so that the parent of $T_h[i]$ is the letter of T_{h+1} that either (a) corresponds to $T_h[i]$ or (b) replaced a fragment of T_h containing $T_h[i]$. We say that the node representing $T_h[i]$ is the node left (resp. right) of the node representing $T_h[i+1]$ (resp. $T_h[i-1]$). Every node v of $\mathsf{PT}(T)$ is labeled with the symbol it represents, denoted by $\mathcal{L}(v)$. For a node v corresponding to a letter of T_h, we say that the level of v, denoted by $\mathsf{lev}(v)$, is h. The *value* $\mathsf{val}(v)$ of a node v is defined as the fragment of T corresponding to the leaf descendants of v and it is an occurrence of $\mathsf{gen}(A)$ for $A = \mathcal{L}(v)$.

We define a *layer* to be any sequence of nodes $v_1 v_2 \cdots v_r$ in $\mathsf{PT}(T)$ whose values are consecutive fragments of T, i.e. $\mathsf{val}(v_j) = T[r_{j-1}+1 \mathinner{.\,.} r_j]$ for some increasing sequence of r_i's. The value of a layer C is the concatenation of the values of its elements and is denoted by $\mathsf{val}(C)$. We similarly use $\mathsf{gen}(\cdot)$ for sequences of symbols, to denote the concatenation of the strings generated by them. We call a layer $v_1 v_2 \cdots v_r$ an *up-layer* when $\mathsf{lev}(v_i) \leq \mathsf{lev}(v_{i+1})$ for all i, and a *down-layer* when $\mathsf{lev}(v_i) \geq \mathsf{lev}(v_{i+1})$ for all i.

In [15], the authors show how to maintain an RLSLP for each string in the collection, each with at most $c \log n$ levels for some global constant c with high probability. Let T be a string in the collection. For each fragment $U = T[a \mathinner{.\,.} b]$ of T, one can compute in $\mathcal{O}(\log n)$ time a *context insensitive decomposition* that consists in a layer $\mathsf{C}(U)$ of nodes in $\mathsf{PT}(T)$ with value $T[a \mathinner{.\,.} b]$ and has the following property. It can be decomposed into an up-layer $\mathsf{C}_{\mathsf{up}}(U)$ and a down-layer $\mathsf{C}_{\mathsf{down}}(U)$ such that:

- The sequence of the labels of the nodes in $\mathsf{C}_{\mathsf{up}}(U)$ can be expressed as a sequence of at most $c \log n$ symbols and powers of symbols $\mathsf{d}_{\mathsf{up}}(U) = A_0^{r_0} A_1^{r_1} \cdots A_m^{r_m}$ such that, for all i, $A_i^{r_i}$ corresponds to r_i consecutive nodes at level i of $\mathsf{PT}(T)$; r_i can be 0 for $i < m$.

[1] A fragment $T[i \mathinner{.\,.} j] = B^r$ is a run if it is a maximal fragment consisting of Bs.

- Similarly, the sequence of the labels of the nodes in $\mathsf{C}_{\mathsf{down}}(U)$ can be expressed as a sequence of at most $c \log n$ symbols and powers of symbols $\mathsf{d}_{\mathsf{down}}(U) = B_m^{t_m} B_{m-1}^{t_{m-1}} \cdots B_0^{t_0}$ such that, for all i, $B_i^{t_i}$ corresponds to t_i consecutive nodes at level i of $\mathsf{PT}(T)$; t_i can be equal to 0.

We denote by $\mathsf{d}(U)$ the concatenation of $\mathsf{d}_{\mathsf{up}}(U)$ and $\mathsf{d}_{\mathsf{down}}(U)$. Note that $U = \mathsf{gen}(\mathsf{d}(U)) = \mathsf{gen}(A_0)^{r_0} \cdots \mathsf{gen}(A_m)^{r_m} \mathsf{gen}(B_m)^{t_m} \cdots \mathsf{gen}(B_0)^{t_0}$. See Figure 1 for a visualization. The parsing of the strings enjoys local consistency in the following way: $\mathsf{d}(U) = \mathsf{d}(V)$ for any fragment V of any string in the collection such that $U = V$. We will slightly abuse notation and use the term "context insensitive decomposition" to refer to both $\mathsf{d}(U)$ and $\mathsf{C}(U)$. In addition, we also use $\mathsf{d}(\cdot)$ for substrings and not just for fragments.

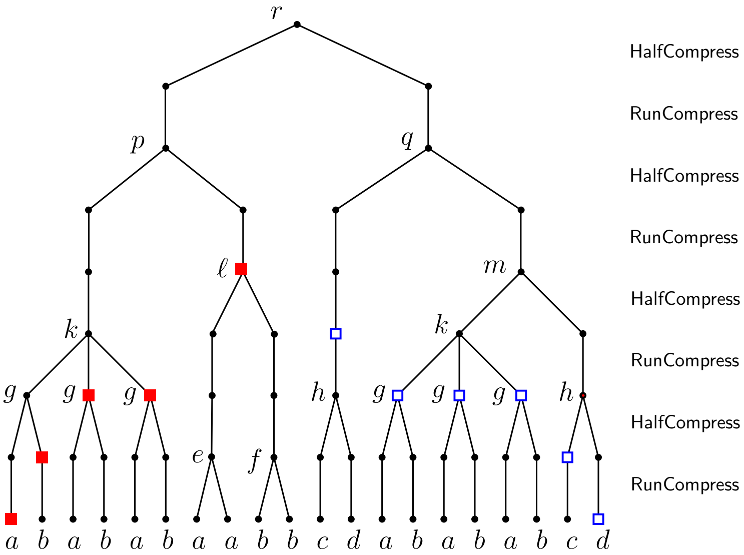


Figure 1 An example $\mathsf{PT}(T)$ for $T = T_0 = abababaabbcdababababcd$. We omit the label of each node v with a single child u; $\mathcal{L}(v) = \mathcal{L}(u)$. $T_3 = kefhkh$ and $T_6 = pq$. We denote the nodes $\mathsf{C}_{\mathsf{up}}(T)$ by red (filled) squares and the nodes of $\mathsf{C}_{\mathsf{down}}(T)$ with blue (unfilled) squares. $\mathsf{d}_{\mathsf{up}}(T) = abg^2\ell$, $\mathsf{d}_{\mathsf{down}}(T) = hg^3cd$ and hence $\mathsf{d}(T) = abg^2\ell hg^3cd$.

Let us consider any sequence of nodes corresponding, for some $j < m$, to $A_j^{r_j}$ with $r_j > 1$ or $B_j^{t_j}$ with $t_j > 1$. We note that T_j must have been obtained from T_{j-1} by an application of $\mathsf{HalfCompress}$, since there are no runs after an application of procedure $\mathsf{RunCompress}$. Thus, at level $j+1$ in $\mathsf{PT}(T)$, i.e. the one corresponding to T_{j+1}, all of these nodes collapse to a single one: their parent in $\mathsf{PT}(T)$. Hence, we have the following lemma.

▶ **Lemma 13.** *Let U be a fragment of T with $\mathsf{d}_{\mathsf{up}}(U) = A_0^{r_0} A_1^{r_1} \cdots A_m^{r_m}$ and $\mathsf{d}_{\mathsf{down}}(U) = B_m^{t_m} B_{m-1}^{t_{m-1}} \cdots B_0^{t_0}$. Then we have the following:*

- *The value of $\mathsf{C}_{\mathsf{up}}(U)$ is a suffix of the value of a layer L_{up} of (at most) $c \log n + r_m - 1$ level-m nodes, such that the two layers have the same rightmost node. The last r_m nodes are consecutive siblings with label A_m.*
- *The value of $\mathsf{C}_{\mathsf{down}}(U)$ is a prefix of the value of the layer L_{down} consisting of the subsequent (at most) $c \log n + \max(t_m - 1, 0)$ level-m nodes. If $t_m \neq 0$, then the first t_m nodes of L_{down} are consecutive siblings with label $B_m \neq A_m$.*

The parse trees of the strings in the collection are not maintained explicitly. However, we have access to the following pointers and functions, among others, which allow us to efficiently navigate through them. First, we can get a pointer to the root of $\mathsf{PT}(T)$ for any string T in the collection. Given a pointer P to some node v in $\mathsf{PT}(T)$ we can get $\deg(v)$ and pointers to the parent of v, the k-th child of v and the nodes to the left/right of v.

Let us now briefly explain how the dynamic data structure of [15] processes a substitution in T at some position i, that yields a string T'. First, the context insensitive decompositions of $T[\,..\,i-1]$ and $T[i+1\,..\,]$ are retrieved. These, together with the new letter at position i form a layer of $\mathsf{PT}(T')$. The sequence of the labels of the nodes of this layer can be expressed as a sequence of $\mathcal{O}(\log n)$ symbols and powers of symbols. Then, only the portion of $\mathsf{PT}(T)$ that lies above this layer needs to be (implicitly) computed, and the authors of [15] show how to do this in $\mathcal{O}(\log n)$ time. In total, we get $\mathsf{PT}(T')$ from $\mathsf{PT}(T)$ through $\mathcal{O}(\log^2 n)$ insertions and deletions of nodes and layers that consist of consecutive siblings.

4.2 Anchoring the LCS

We will rely on Lemma 13 in order to identify an LCS $S[i\,..\,j] = T[i'\,..\,j']$ at a pair of topmost nodes of the context insensitive decompositions of $S[i\,..\,j]$ and $T[i'\,..\,j']$ in $\mathsf{PT}(S)$ and $\mathsf{PT}(T)$, respectively. In order to develop some intuition, let us first sketch a solution for the case that $\mathsf{PT}(S)$ and $\mathsf{PT}(T)$ do not contain any power symbols throughout the execution of our algorithm. For each node v in one of the parse trees, let $\mathsf{Z}_\ell(v)$ be the value of the layer consisting of the (at most) $c\log n$ level-$\mathsf{lev}(v)$ nodes, with v being the layer's rightmost node, and $\mathsf{Z}_\mathsf{r}(v)$ be the value of the layer consisting of the (at most) $c\log n$ subsequent level-$\mathsf{lev}(v)$ nodes. Now, consider a common substring X of S and T and partition it into the prefix $X_\ell = \mathsf{gen}(\mathsf{d}_\mathsf{up}(X))$ and the suffix $X_r = \mathsf{gen}(\mathsf{d}_\mathsf{down}(X))$. For any fragment U of S that equals X, $\mathsf{C}_\mathsf{up}(U)$ is an up-layer of the form $v_1 \cdots v_m$. Hence, by Lemma 13, X_ℓ is a suffix of $\mathsf{Z}_\ell(v_m)$. Similarly, X_r is a prefix of $\mathsf{Z}_\mathsf{r}(v_m)$. Thus, it suffices to maintain pairs $(\mathsf{Z}_\ell(v), \mathsf{Z}_\mathsf{r}(v))$ for all nodes v in $\mathsf{PT}(S)$ and $\mathsf{PT}(T)$, and, in particular, a pair of nodes $u \in \mathsf{PT}(S)$ and $v \in \mathsf{PT}(T)$ that maximizes $\mathsf{lcp}(\mathsf{Z}_\ell(u)^R, \mathsf{Z}_\ell(v)^R) + \mathsf{lcp}(\mathsf{Z}_\mathsf{r}(u), \mathsf{Z}_\mathsf{r}(v))$. The existence of power symbols poses some technical challenges which we overcome below.

For each node of $\mathsf{PT}(T)$, we consider at most one pair consisting of an up-layer and a down-layer. The treatment of nodes differs, based on their parent. We have two cases.

1. For each node z with $\deg(z) = 2$ and $\mathcal{L}(z)$ being a concatenation symbol, for each child v of z, we consider the following layers:
   - The up-layer $\mathsf{J}_\mathsf{up}(v)$ of the (at most) $c\log n$ level-$\mathsf{lev}(v)$ consecutive nodes of $\mathsf{PT}(T)$ with v a rightmost node.
   - The down-layer $\mathsf{J}_\mathsf{down}(v)$ of the (at most) p level-$\mathsf{lev}(v)$ subsequent level-$\mathsf{lev}(v)$ nodes of $\mathsf{PT}(T)$. If the node to the right of v is a child of a node w with more than two children, then $p = c\log n + \deg(w)$. Otherwise $p = c\log n$.
2. For each node z of $\mathsf{PT}(T)$ whose label is a power symbol and has more than one child, we will consider $\mathcal{O}(\log n)$ pairs of layers. In particular, for each v, being one of the $c\log n + 1$ leftmost or $c\log n + 1$ rightmost children of z, we consider the following layers:
   - The up-layer $\mathsf{J}_\mathsf{up}(v)$ defined as the concatenation of (a) the (at most) $c\log n$ level-$\mathsf{lev}(v)$ consecutive nodes of $\mathsf{PT}(T)$ preceding the leftmost child of z and (b) all the children of z that lie weakly to the left of v, i.e. including v.
   - The down-layer $\mathsf{J}_\mathsf{down}(v)$ of the (at most) $c\log n$ subsequent level-$\mathsf{lev}(v)$ nodes of $\mathsf{PT}(T)$ – with one exception. If v is the rightmost child of z and the node to its right is a child of a node w with more than two children, then $\mathsf{J}_\mathsf{down}(v)$ consists of the $c\log n + \deg(w)$ subsequent level-$\mathsf{lev}(v)$ nodes.

In particular, we create at most one pair $(\mathsf{J}_{\mathsf{up}}(v), \mathsf{J}_{\mathsf{down}}(v))$ of layers for each node v of $\mathsf{PT}(T)$. Let $\mathsf{Y}_\ell(v) = \mathsf{val}(\mathsf{J}_{\mathsf{up}}(v))$ and $\mathsf{Y}_\mathsf{r}(v) = \mathsf{val}(\mathsf{J}_{\mathsf{down}}(v))$. Given a pointer to a node z in $\mathsf{PT}(T)$, we can compute the indices of the fragments corresponding to those layers with straightforward use of the pointers at hand in $\mathcal{O}(\log n)$ time. With a constant number of split operations, we can then add the string $\mathsf{Y}_\mathsf{r}(v)$ to our collection within $\mathcal{O}(\log n)$ time. Similarly, if we also maintain T^R in our collection of strings, we can add the reverse of $\mathsf{Y}_\ell(v)$ to the collection within $\mathcal{O}(\log n)$ time. We maintain pointers between v and these strings. Note that each node of $\mathsf{PT}(T)$ takes part in $\mathcal{O}(\log n)$ pairs of layers and these pairs can be retrieved in $\mathcal{O}(\log n)$ time. Similarly, for each node whose label is a power symbol, subsets of its children appear in $\mathcal{O}(\log n)$ pairs of layers; these can also be retrieved in $\mathcal{O}(\log n)$ time. Thus, throughout the updates on T, which delete/insert $\mathcal{O}(\log^2 n)$ nodes and layers of consecutive siblings, we can maintain the pairs of layers in $\tilde{\mathcal{O}}(1)$ time. These pairs of layers (or rather the pairs of their corresponding strings maintained in a dynamic collection) will be stored in an abstract structure presented in the next section. In order to keep the space occupied by our data structure $\tilde{\mathcal{O}}(n)$, after every n updates to the collection we delete our data structure, and initialize a new instance of it for an empty collection, on which we call $\mathsf{makestring}(S)$ and $\mathsf{makestring}(T)$. The cost of this reinitialization can be deamortized using standard techniques. We summarize the above discussion in the following lemma.

▶ **Lemma 14.** *We can maintain pairs* $(\mathsf{Y}_\ell(v)^R, \mathsf{Y}_\mathsf{r}(v))$ *for all* v *in* $\mathsf{PT}(T)$ *and* $\mathsf{PT}(S)$*, with each string given as a handle from the dynamic collection, in* $\tilde{\mathcal{O}}(1)$ *time per substitution, using* $\tilde{\mathcal{O}}(n)$ *space.*

▶ Remark 15. Note that the above lemma holds in the case that insertions and deletions are also allowed in S and T, as each such update operation is processed similarly to substitution and affects $\tilde{\mathcal{O}}(1)$ pairs $(\mathsf{Y}_\ell(v)^R, \mathsf{Y}_\mathsf{r}(v))$. Everything that follows in this section is oblivious to the kind of operations allowed in S and T.

The following lemma gives us an anchoring property, which is crucial for our approach.

▶ **Lemma 16.** *For any common substring* X *of* S *and* T*, there exists a partition* $X = X_\ell X_r$ *for which there exist nodes* $u \in \mathsf{PT}(S)$ *and* $v \in \mathsf{PT}(T)$ *such that:*

1. X_ℓ *is a suffix of* $\mathsf{Y}_\ell(u)$ *and* $\mathsf{Y}_\ell(v)$*, and*
2. X_r *is a prefix of* $\mathsf{Y}_\mathsf{r}(u)$ *and* $\mathsf{Y}_\mathsf{r}(v)$*.*

Proof. Let $\mathsf{d}_{\mathsf{up}}(X) = A_0^{r_0} A_1^{r_1} \cdots A_m^{r_m}$ and $\mathsf{d}_{\mathsf{down}}(X) = B_m^{t_m} B_{m-1}^{t_{m-1}} \cdots B_0^{t_0}$.

▷ Claim 17. Either $r_m > 1$, $t_m = 0$ and $\mathsf{gen}(\mathsf{d}_{\mathsf{up}}(X))$ is not a suffix of $A_m^{c \log n + r_m}$ or there exists a node $v \in \mathsf{PT}(T)$ such that:

1. $\mathsf{gen}(\mathsf{d}_{\mathsf{up}}(X))$ is a suffix of $\mathsf{Y}_\ell(v)$, and
2. $\mathsf{gen}(\mathsf{d}_{\mathsf{down}}(X))$ is a prefix of $\mathsf{Y}_\mathsf{r}(v)$.

Proof. We assume that $r_m = 1$ or $\mathsf{gen}(\mathsf{d}_{\mathsf{up}}(X))$ is a suffix of $A_m^{c \log n + r_m}$ or $t_m \neq 0$ and distinguish between the following cases.

Case 1. There exists an occurrence Y of X in T, where the label of the parent of the rightmost node u of $\mathsf{C}_{\mathsf{up}}(Y)$ is not a power symbol. (In this case $r_m = 1$.) Recall here, that we did not construct any pairs of layers for nodes whose parent has a single child. Let v be the highest ancestor of u with label A_m. If $u \neq v$ then all nodes that are descendants of v and strict ancestors of u have a single child, while the parent of v does not. In addition, the label of the parent of v must be a concatenation symbol, since only new letters are introduced at each level and thus we cannot have new nodes with label A_m appearing to the left/right of any strict ancestor of u. Finally, note that a layer of k level-$\mathsf{lev}(v)$ nodes with v a leftmost (resp. rightmost) node contains an ancestor of each of the nodes in a layer of k level-$\mathsf{lev}(u)$ nodes with u a leftmost (resp. rightmost) node. Thus, an application of Lemma 13 for u straightforwardly implies our claim for v.

Case 2. There exists an occurrence Y of X in T, where the label of the parent z of the rightmost node u of $\mathsf{C}_{\mathsf{up}}(Y)$ is a power symbol. Let W be the rightmost occurrence of X in T such that the rightmost node w of $\mathsf{C}_{\mathsf{up}}(W)$ is a child of z. We have three subcases.

a) We first consider the case $r_m = 1$. Let us assume towards a contradiction that u is not one of the $c\log n+1$ leftmost or the $c\log n+1$ rightmost children of z. Then, by Lemma 13 we have that $\mathsf{gen}(\mathsf{d}_{\mathsf{up}}(X))$ is a suffix of $A_m^{c\log n}$ and $\mathsf{gen}(\mathsf{d}_{\mathsf{down}}(X))$ is a prefix of $A_m^{c\log n}$. Hence, there is another occurrence of X $|\mathsf{gen}(A_m)|$ positions to the right of Y, contradicting our assumption that Y is a rightmost occurrence.

b) In the case that $t_m \neq 0$, u must be the rightmost child of z since $A_m \neq B_m$.

c) In the remaining case that $\mathsf{gen}(\mathsf{d}_{\mathsf{up}}(X))$ is a suffix of $A_m^{c\log n+r_m}$, either $t_m > 0$ and we are done, or $\mathsf{gen}(\mathsf{C}_{\mathsf{down}}(Y))$ is a prefix of the value of the (at most) $c\log n$ level-m nodes to the right of u. In the latter case, either u is already among the rightmost $c\log n+1$ children of z or there is another occurrence of X $|\mathsf{gen}(A_m)|$ positions to the right of Y, contradicting our assumptions on Y. ◁

We have to treat a final case.

▷ Claim 18. If $r_m > 1$, $t_m = 0$ and $\mathsf{gen}(\mathsf{d}_{\mathsf{up}}(X))$ is not a suffix of $A_m^{c\log n+r_m}$ then there exists a node $v \in \mathsf{PT}(T)$ such that:

1. $\mathsf{gen}(A_0^{r_0}A_1^{r_1}\cdots A_{m-1}^{r_{m-1}}A_m)$ is a suffix of $\mathsf{Y}_\ell(v)$, and
2. $\mathsf{gen}(A_m)^{r_m-1}\mathsf{gen}(\mathsf{d}_{\mathsf{down}}(X))$ is a prefix of $\mathsf{Y}_\mathsf{r}(v)$.

Proof. In any occurrence of X in T, the label of the parent z of the rightmost node of $\mathsf{C}_{\mathsf{up}}(Y)$ is a power symbol. Let u be the r_m-th rightmost node of $\mathsf{C}_{\mathsf{up}}(Y)$. By the assumption that $\mathsf{gen}(\mathsf{d}_{\mathsf{up}}(X))$ is not a suffix of $A_m^{c\log n+r_m}$ and Lemma 13, u must be one of the $c\log n$ leftmost children of z. ◁

The combination of the two claims applied to both S and T yields the lemma. ◀

4.3 A Problem on Dynamic Bicolored Trees

Due to Lemmas 14 and 16, our task reduces to solving the problem defined below in polylogarithmic time per update, as we can directly apply it to $\mathcal{R} = \{(\mathsf{Y}_\ell(u)^R, \mathsf{Y}_\mathsf{r}(u)) : u \in \mathsf{PT}(S)\}$ and $\mathcal{B} = \{(\mathsf{Y}_\ell(v)^R, \mathsf{Y}_\mathsf{r}(v)) : v \in \mathsf{PT}(T)\}$. Note that $|\mathcal{R}| + |\mathcal{B}| = \tilde{\mathcal{O}}(n)$ throughout the execution of our algorithm.

Problem: LCP for Two Families of Pairs of Strings
Input: Two families $\mathcal{R}$ and $\mathcal{B}$, each consisting of pairs of strings, where each string is given as a handle from a dynamic collection.
Update: Insertion or deletion of an element in $\mathcal{R}$ or $\mathcal{B}$.
Query: Return $(P,Q) \in \mathcal{R}$ and $(P',Q') \in \mathcal{B}$ that maximize $\mathsf{lcp}(P,P') + \mathsf{lcp}(Q,Q')$.

Each element of $\mathcal{B}$ and $\mathcal{R}$ is given a unique identifier. We maintain two compacted tries $\mathcal{T}_P$ and $\mathcal{T}_Q$. By appending unique letters, we can assume that no string is a prefix of another string. $\mathcal{T}_P$ (resp. $\mathcal{T}_Q$) stores the string P (resp. Q) for every $(P,Q) \in \mathcal{R}$, with the corresponding leaf colored red and labeled by the identifier of the pair and the string P' (resp. Q') for every $(P',Q') \in \mathcal{B}$, with the corresponding leaf colored blue and labeled by the identifier of the pair. Then, the sought result corresponds to a pair of nodes $u \in \mathcal{T}_P$ and $v \in \mathcal{T}_Q$ returned by a query to a data structure for the Dynamic Bicolored Trees Problem defined below for $\mathcal{T}_1 = \mathcal{T}_P$ and $\mathcal{T}_2 = \mathcal{T}_Q$, with node weights being their string-depths.

Problem: Dynamic Bicolored Trees Problem
Input: Two weighted trees $\mathcal{T}_1$ and $\mathcal{T}_2$ of total size at most m, whose leaves are bicolored and labeled, so that each label corresponds to exactly one leaf of each tree.
Update: Split an edge into two / attach a new leaf to a node / delete a leaf.
Query: Return a pair of nodes $u \in \mathcal{T}_1$ and $v \in \mathcal{T}_2$ with the maximum combined weight that have at least one red descendant with the same label, and at least one blue descendant with the same label.

To complete the reduction, we have to show how to translate an update in $\mathcal{R}$ or $\mathcal{B}$ into updates in $\mathcal{T}_P$ and $\mathcal{T}_Q$. Let us first explain how to represent $\mathcal{T}_P$ and $\mathcal{T}_Q$. For each edge, we store a handle to a string from the dynamic collection, and indices for a fragment of this string which represents the edge's label. For each explicit node, we store edges leading to its children in a dictionary structure indexed by the first letters of the edges' labels. For every leaf, we store its label and color. An insert operation receives a string (given as a handle from a dynamic collection), together with its label and color, and should create its corresponding leaf. A delete operation does not actually remove a leaf, but simply removes its label. However, in order to not increase the space complexity, we rebuild the whole data structure from scratch after every m updates. This rebuilding does not incur any extra cost asymptotically; the time required for it can be deamortized using standard techniques.

▶ **Lemma 19.** *Each update in $\mathcal{R}$ or $\mathcal{B}$ implies $\mathcal{O}(1)$ updates in $\mathcal{T}_P$ and $\mathcal{T}_Q$ that can be computed in $\mathcal{O}(\log n)$ time.*

Proof. Inserting a new leaf, corresponding to string U, to $\mathcal{T}_P$ requires possibly splitting an edge into two by creating a new explicit node, and then attaching a new leaf to an explicit node. To implement this efficiently, we maintain the set C of path-labels of explicit nodes of $\mathcal{T}_P$ in a balanced search tree, sorted in lexicographic order. Using lcp queries (cf. Theorem 12), we binary search for the longest prefix U' of U that equals the path-label of some implicit or explicit node of $\mathcal{T}_P$. If this node is explicit, then we attach a leaf to it. Otherwise, let the successor of U' in C be the path-label of node v. We split the edge $(\mathsf{parent}(v), v)$ appropriately and attach a leaf to the newly created node. This allows us to maintain $\mathcal{T}_P$ after each insert operation in $\mathcal{O}(\log n)$ time.

For a delete operation, we can access the leaf corresponding to the deleted string in $\mathcal{O}(\log n)$ time using the balanced search tree. ◀

It thus suffices to show a solution for the Dynamic Bicolored Trees Problem that processes each update in polylogarithmic time.

We will maintain a heavy-light decomposition of both $\mathcal{T}_1$ and $\mathcal{T}_2$. This can be done by using a standard method of rebuilding as used by Gabow [13]. Let $L(u)$ be the number of leaves in the subtree of u, including the leaves without labels, when the subtree was last rebuilt. Each internal node u of a tree selects at most one child v and the edge (u, v) is *heavy*. All other edges are *light*. Maximal sequences of consecutive heavy edges are called *heavy paths*. The node $r(p)$ closest to the root of the tree is called the *root* of the heavy path p and the node $e(p)$ furthest from the root of the tree is called the *end* of the heavy path. The following procedure receives a node u of the tree and recursively rebuilds its subtree.

```
function DECOMPOSE($u$, $r$)                    ▷ $r$ is the root of the heavy path containing $u$.
    $S \leftarrow \text{children}(u)$
    $v \leftarrow \text{argmax}_{v \in S} L(v)$
    if $L(v) \geq \frac{5}{6} \cdot L(u)$ then
        edge $(u, v)$ is heavy
        DECOMPOSE($v$, $r$)
        $S \leftarrow S \setminus \{v\}$
    for $v \in S$ do
        DECOMPOSE($v$, $v$)
```

Every root u of a heavy path maintains the number of insertions $I(u)$ in its subtree since it was last rebuilt. When $I(u) \geq \frac{1}{6} \cdot L(u)$, we recalculate the values of $L(v)$ for nodes v in the subtree of u and call DECOMPOSE(u, u). This maintains the property that $L(e(p)) \geq \frac{2}{3} L(r(p))$ for each heavy path p and leads to the following.

▶ **Proposition 20.** *There are $\mathcal{O}(\log m)$ heavy paths above any node.*

As rebuilding a subtree of size s takes $\mathcal{O}(s)$ time, by a standard potential argument, we get the following.

▶ **Lemma 21.** *The heavy-light decompositions of $\mathcal{T}_1$ and $\mathcal{T}_2$ can be maintained in $\mathcal{O}(\log m)$ amortized time per update.*

The main ingredient of our structure is a collection of additional structures, each storing a dynamic set of points. Each such point structure sends its current result to a max-heap, and after each update we return the largest element stored in the heap. The problem each of these point structures are designed for is the following.

Problem: Dynamic Best Bichromatic Point
Input: A multiset of at most m bicolored points from $[m] \times [m]$.
Update: Insertions and deletions of points from $[m] \times [m]$.
Query: Return a pair of points $R = (x, y)$ and $B = (x', y')$ such that R is red, B is blue, and $\min(x, x') + \min(y, y')$ is as large as possible.

We call the pair of points sought in this problem the *best bichromatic pair of points*. In Section 4.4 we explain how to modify range trees in order to obtain the following result.

▶ **Lemma 22.** *There is a data structure for Dynamic Best Bichromatic Point that processes each update in $\mathcal{O}(\log^2 m)$ amortized time.*

Conceptually, we maintain a point structure for every pair of heavy paths from $\mathcal{T}_P$ and $\mathcal{T}_Q$. However, the total number of points stored in all structures at any moment is only $\mathcal{O}(m \log^2 m)$ and the empty structures are not actually created. Consider heavy paths p of $\mathcal{T}_1$ and q of $\mathcal{T}_2$. Let ℓ be a label such that there are leaves u in the subtree of $r(p)$ in $\mathcal{T}_1$ and v in the subtree of $r(q)$ in $\mathcal{T}_2$ with the same color and both labeled by ℓ. Then, the point structure should contain a point (x, y) with this color, where x and y are the string-depths of the nodes of p and q containing u and v in their light subtrees, respectively. It can be verified that then the answer extracted from the point structure is equal to the sought result, assuming that the corresponding pair of nodes belongs to p and q, respectively. It remains to explain how to maintain this invariant when both trees undergo modifications.

Splitting an edge does not require any changes to the point structures. Each label appears only once in $\mathcal{T}_1$ and $\mathcal{T}_2$, and hence by Proposition 20 contributes to only $\mathcal{O}(\log^2 m)$ point structures. Furthermore, by navigating the heavy path decompositions we can access these structures efficiently. This allows us to implement each deletion in $\mathcal{O}(\log^4 m)$ amortized time, employing Lemma 22. To implement the insertions, we need to additionally explain what to do after rebuilding a subtree of u. In this case, we first remove all points corresponding to leaves in the subtree of u, then rebuild the subtree, and then proceed to insert points to existing and potentially new point structures. This can be amortized by the same standard potential argument if we add another factor of $\mathcal{O}(\log^2 n)$ in the analysis to account for the fact that we add a point in $\mathcal{O}(\log^2 n)$ point structures for each leaf in the subtree of u. Thus, insertions require $\mathcal{O}(\log^5 n)$ amortized time as well.

Wrap-up. Lemma 16 reduces our problem to the LCP for Two Families of Pairs of Strings problem for sets $\mathcal{R}$ and $\mathcal{B}$ of size $\tilde{\mathcal{O}}(n)$, so that each substitution in S or T yields $\tilde{\mathcal{O}}(1)$ updates to $\mathcal{R}$ and $\mathcal{B}$, which can be computed in $\tilde{\mathcal{O}}(1)$ time due to Lemma 14. The LCP for Two Families of Pairs of Strings problem is then reduced to the Dynamic Bicolored Trees Problem for trees $\mathcal{T}_1$ and $\mathcal{T}_2$ of size $\tilde{\mathcal{O}}(n)$, so that each update in $\mathcal{R}$ or $\mathcal{B}$ yields $\mathcal{O}(1)$ updates to the trees, which can be computed in $\mathcal{O}(\log n)$ time (Lemma 19). We solve the latter problem by maintaining a heavy-light decomposition of each of the trees in $\mathcal{O}(\log n)$ amortized time per update (Lemma 21), and an instance of a data structure for the Dynamic Best Bichromatic Point problem for each pair of heavy paths. For each update to the trees, we spend $\mathcal{O}(\log^5 n)$ amortized time to update the point structures.

4.4 Dynamic Best Bichromatic Point

In this section we prove Lemma 22, i.e. design an efficient data structure for the Dynamic Best Bichromatic Point problem.

With standard perturbation, we can guarantee that all x and y coordinates of points are distinct. We maintain an augmented dynamic 2D range tree [31] over the multiset of points. This is a balanced search tree $\mathcal{T}$ (called primary) over the x coordinates of all points in the multiset in which every x coordinate corresponds to a leaf and, more generally, every node $u \in \mathcal{T}$ corresponds to a range of x coordinates denoted by $x(u)$. Additionally, every $u \in \mathcal{T}$ stores another balanced search tree $\mathcal{T}_u$ (called secondary) over the y coordinates of all points $(x, y) \in S$ such that $x \in x(u)$. Thus, the leaves of $\mathcal{T}_u$ correspond to y coordinates of such points, and every $v \in \mathcal{T}_u$ corresponds to a range of y coordinates denoted by $y(v)$. We interpret every $v \in \mathcal{T}_u$ as the rectangular region of the plane $x(u) \times y(v)$, and, in particular, each leaf $v \in \mathcal{T}_u$ corresponds to a single point in the multiset. Each node $v \in \mathcal{T}_u$ will be augmented with some extra information that can be computed in constant time from the extra information stored in its children. Similarly, each node $u \in \mathcal{T}$ will be augmented with some extra information that can be computed in constant time from the extra information stored in its children together with the extra information stored in the root of the secondary tree $\mathcal{T}_u$. Irrespectively of what this extra information is, as explained by Willard and Lueker [31], if we implement the primary tree as a $BB(\alpha)$ tree and each secondary tree as a balanced search tree, each insertion and deletion can be implemented in $\mathcal{O}(\log^2 m)$ amortized time.

Before we explain what is the extra information, we need the following notion. Consider a non-leaf node $u \in \mathcal{T}$ and let $u_\ell, u_r \in \mathcal{T}$ be its children. Let $v \in \mathcal{T}_u$ be a non-leaf node with children $v_\ell, v_r \in \mathcal{T}_u$. The regions $A = x(u_\ell) \times y(v_\ell)$, $B = x(u_\ell) \times y(v_r)$, $C = x(u_r) \times y(v_\ell)$ and $D = x(u_r) \times y(v_r)$ partition $x(u) \times y(v)$ into four parts. We say that two points $p = (x, y)$

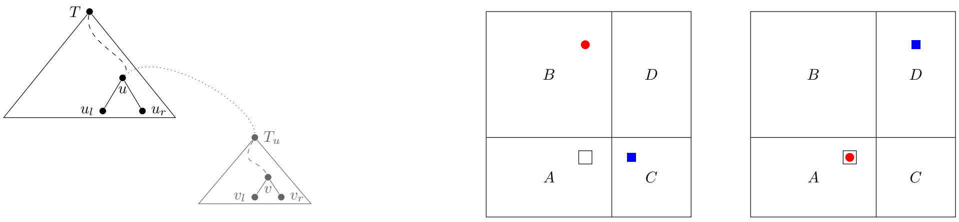


Figure 2 Left: A 2D range tree. Right: Node representing regions A, B, C, D. The best pair for each case is denoted by a small square.

and $q = (x', y')$ with $x < x'$ are shattered by $v \in \mathcal{T}_u$ if and only if $p \in A$ and $q \in D$ or $p \in B$ and $q \in C$ (note that the former is only possible when $y < y'$ while the latter can only hold when $y > y'$).

▶ **Proposition 23.** *Any pair of points in the multiset is shattered by a unique $v \in \mathcal{T}_u$ (for a unique u).*

Now we are ready to describe the extra information. Each node $u \in \mathcal{T}$ stores the best bichromatic pair with x coordinates from $x(u)$. Each node $v \in \mathcal{T}_u$ stores the best bichromatic pair shattered by one of its descendants $v' \in \mathcal{T}_u$ (possibly v itself). Additionally, each node $v \in \mathcal{T}_u$ stores the following information about points of each color in its region:

1. the point with the maximum x,
2. the point with the maximum y,
3. a point with the maximum $x + y$.

We need to verify that such extra information can be indeed computed in constant time from the extra information stored in the children.

▶ **Lemma 24.** *Let $v \in \mathcal{T}_u$ be a non-leaf node, and v_ℓ, v_r be its children. The extra information of v can be computed in constant time given the extra information stored in v_ℓ and v_r.*

Proof. This is clear for the maximum x, y and $x + y$ of each color, as we can take the maximum of the corresponding values stored in the children. For the best bichromatic pair shattered by a descendant v' of v, we start with considering the best bichromatic pair shattered by a descendant v'_ℓ of v_ℓ and v'_r of v_r. The remaining case is that the best bichromatic pair is shattered by v itself. Let A, B, C, D be as in the definition of shattering. Without losing generality we assume that the sought pair is $p = (x, y)$ and $q = (x', y')$ with $x < x'$, red p and blue q. We consider two cases:

1. $p \in A$ and $q \in D$: the best such pair is obtained by taking p with the maximum $x + y$ and any q,
2. $p \in B$ and $q \in C$: the best such pair is obtained by taking p with the maximum x and q with the maximum y.

In both cases, we are able to compute the best bichromatic pair shattered by v using the extra information stored at the children of v. See Figure 2. ◀

▶ **Lemma 25.** *Let $u \in \mathcal{T}$ be a non-leaf node, and u_ℓ, u_r be its children. The extra information of v can be computed in constant time given the extra information stored in v_ℓ, v_r and the root of $\mathcal{T}_u$.*

Proof. We seek the best bichromatic pair with x coordinates from $x(u)$. If the x coordinates are in fact from $x(u_\ell)$ or $x(u_r)$, we obtain the pair from the children of u. Otherwise, the pair must be shattered by some $v \in \mathcal{T}_u$ that is a descendant of the root of $\mathcal{T}_u$, so we obtain the pair from the root of $\mathcal{T}_u$. ◀

References

1 Paniz Abedin, Sahar Hooshmand, Arnab Ganguly, and Sharma V. Thankachan. The heaviest induced ancestors problem revisited. In *29th CPM*, pages 20:1–20:13, 2018. doi:10.4230/LIPIcs.CPM.2018.20.

2 Stephen Alstrup, Gerth Stølting Brodal, and Theis Rauhe. Pattern matching in dynamic texts. In *11th SODA*, pages 819–828, 2000. URL: http://dl.acm.org/citation.cfm?id=338219.338645.

3 Amihood Amir and Itai Boneh. Locally maximal common factors as a tool for efficient dynamic string algorithms. In *29th CPM*, pages 11:1–11:13, 2018. doi:10.4230/LIPIcs.CPM.2018.11.

4 Amihood Amir and Itai Boneh. Dynamic palindrome detection. *CoRR*, abs/1906.09732, 2019. arXiv:1906.09732.

5 Amihood Amir, Itai Boneh, Panagiotis Charalampopoulos, and Eitan Kondratovsky. Repetition Detection in a Dynamic String. In *27th ESA*, pages 5:1–5:18, 2019. doi:10.4230/LIPIcs.ESA.2019.5.

6 Amihood Amir, Panagiotis Charalampopoulos, Costas S. Iliopoulos, Solon P. Pissis, and Jakub Radoszewski. Longest common factor after one edit operation. In *24th SPIRE*, pages 14–26, 2017. doi:10.1007/978-3-319-67428-5_2.

7 Amihood Amir, Panagiotis Charalampopoulos, Solon P. Pissis, and Jakub Radoszewski. Longest common substring made fully dynamic. In *27th ESA*, pages 6:1–6:17, 2019. doi:10.4230/LIPIcs.ESA.2019.6.

8 Amihood Amir, Gad M. Landau, Moshe Lewenstein, and Dina Sokol. Dynamic text and static pattern matching. *ACM Trans. Algorithms*, 3(2):19, 2007. doi:10.1145/1240233.1240242.

9 Panagiotis Charalampopoulos, Maxime Crochemore, Costas S. Iliopoulos, Tomasz Kociumaka, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Linear-time algorithm for long LCF with k mismatches. In *29th CPM*, pages 23:1–23:16, 2018. doi:10.4230/LIPIcs.CPM.2018.23.

10 Richard Cole, Lee-Ad Gottlieb, and Moshe Lewenstein. Dictionary matching and indexing with errors and don't cares. In *36th STOC*, pages 91–100, 2004. doi:10.1145/1007352.1007374.

11 Martin Farach. Optimal suffix tree construction with large alphabets. In *38th FOCS*, pages 137–143, 1997. doi:10.1109/SFCS.1997.646102.

12 Martin Farach and S. Muthukrishnan. Perfect hashing for strings: Formalization and algorithms. In *7th CPM*, pages 130–140, 1996. doi:10.1007/3-540-61258-0_11.

13 Harold N. Gabow. Data structures for weighted matching and nearest common ancestors with linking. In *1st SODA*, pages 434–443, 1990. URL: http://dl.acm.org/citation.cfm?id=320176.320229.

14 Travis Gagie, Paweł Gawrychowski, and Yakov Nekrich. Heaviest induced ancestors and longest common substrings. In *25th CCCG*, 2013. URL: http://cccg.ca/proceedings/2013/papers/paper_29.pdf.

15 Paweł Gawrychowski, Adam Karczmarz, Tomasz Kociumaka, Jakub Lacki, and Piotr Sankowski. Optimal dynamic strings. In *29th SODA*, pages 1509–1528, 2018. doi:10.1137/1.9781611975031.99.

16 Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *47th STOC*, pages 21–30, 2015. doi:10.1145/2746539.2746609.

17 Tomohiro I. Longest common extensions with recompression. In *28th CPM*, pages 18:1–18:15, 2017. doi:10.4230/LIPIcs.CPM.2017.18.

18 Artur Jeż. Faster fully compressed pattern matching by recompression. *ACM Transactions on Algorithms*, 11(3):20:1–20:43, 2015. doi:10.1145/2631920.

19 Artur Jeż. Recompression: A simple and powerful technique for word equations. *J. ACM*, 63(1):4:1–4:51, 2016. doi:10.1145/2743014.

20 Tomasz Kociumaka, Jakub Radoszewski, and Tatiana A. Starikovskaya. Longest common substring with approximately k mismatches. *Algorithmica*, 81(6):2633–2652, 2019. doi:10.1007/s00453-019-00548-x.

21 Tomasz Kociumaka, Tatiana A. Starikovskaya, and Hjalte Wedel Vildhøj. Sublinear space algorithms for the longest common substring problem. In *22nd ESA*, pages 605–617, 2014. doi:10.1007/978-3-662-44777-2_50.

22 Yakov Nekrich. A data structure for multi-dimensional range reporting. In *23rd SOCG*, pages 344–353, 2007. doi:10.1145/1247069.1247130.

23 Mihai Patrascu. Unifying the landscape of cell-probe lower bounds. *SIAM J. Comput.*, 40(3):827–847, 2011. doi:10.1137/09075336X.

24 Mihai Pǎtraşcu and Erik D. Demaine. Logarithmic lower bounds in the cell-probe model. *SIAM J. Comput.*, 35(4):932–963, 2006. doi:10.1137/S0097539705447256.

25 Mihai Pǎtraşcu and Mikkel Thorup. Time-space trade-offs for predecessor search. In *38th STOC*, pages 232–240, 2006. doi:10.1145/1132516.1132551.

26 S. Sahinalp and U. Vishkin. Efficient approximate and dynamic matching of patterns using a labeling paradigm. In *54th FOCS*, pages 320–328, 1996. doi:10.1109/SFCS.1996.548491.

27 Tatiana A. Starikovskaya and Hjalte Wedel Vildhøj. Time-space trade-offs for the longest common substring problem. In *24th CPM*, pages 223–234, 2013. doi:10.1007/978-3-642-38905-4_22.

28 Sharma V. Thankachan, Alberto Apostolico, and Srinivas Aluru. A provably efficient algorithm for the k-mismatch average common substring problem. *Journal of Computational Biology*, 23(6):472–482, 2016. doi:10.1089/cmb.2015.0235.

29 Peter Weiner. Linear pattern matching algorithms. In *14th FOCS*, pages 1–11, 1973. doi:10.1109/SWAT.1973.13.

30 Dan E. Willard. Log-logarithmic worst-case range queries are possible in space $\Theta(n)$. *Information Processing Letters*, 17(2):81–84, 1983. doi:10.1016/0020-0190(83)90075-3.

31 Dan E. Willard and George S. Lueker. Adding range restriction capability to dynamic data structures. *J. ACM*, 32(3):597–617, 1985. doi:10.1145/3828.3839.

Improved Black-Box Constructions of Composable Secure Computation

Rohit Chatterjee
Stony Brook University, NY, USA
rochatterjee@cs.stonybrook.edu

Xiao Liang
Stony Brook University, NY, USA
liang1@cs.stonybrook.edu

Omkant Pandey
Stony Brook University, NY, USA
omkant@cs.stonybrook.edu

Abstract

We close the gap between black-box and non-black-box constructions of *composable* secure multiparty computation in the plain model under the *minimal assumption* of semi-honest oblivious transfer. The notion of protocol composition we target is *angel-based* security, or more precisely, security with super-polynomial helpers. In this notion, both the simulator and the adversary are given access to an oracle called an *angel* that can perform some predefined super-polynomial time task. Angel-based security maintains the attractive properties of the universal composition framework while providing meaningful security guarantees in complex environments without having to trust anyone.

Angel-based security can be achieved using non-black-box constructions in $\max(R_{\mathsf{OT}}, \widetilde{O}(\log n))$ rounds where R_{OT} is the round-complexity of semi-honest oblivious transfer. However, current best known *black-box* constructions under the same assumption require $\max(R_{\mathsf{OT}}, \widetilde{O}(\log^2 n))$ rounds. If R_{OT} is a constant, the gap between non-black-box and black-box constructions can be a multiplicative factor $\log n$. We close this gap by presenting a $\max(R_{\mathsf{OT}}, \widetilde{O}(\log n))$ round black-box construction. We achieve this result by constructing constant-round 1-1 CCA-secure commitments assuming only black-box access to one-way functions.

2012 ACM Subject Classification Security and privacy → Mathematical foundations of cryptography

Keywords and phrases Secure Multi-Party Computation, Black-Box, Composable, Non-Malleable

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.28

Category Track A: Algorithms, Complexity and Games

Related Version A full version of the paper is available at `https://eprint.iacr.org/2020/494`.

Funding This material is based upon work supported in part by DARPA SIEVE Award HR00112020026, NSF grant 1907908, and a Cisco Research Award. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Government, DARPA, NSF, or Cisco.

1 Introduction

Secure multiparty computation (MPC) [79, 24] enables two or more mutually distrustful parties to compute any functionality without compromising the privacy of their inputs. These early results [79, 24], along with a rich body of followup work that refined and developed the concept [25, 5, 65, 7, 73, 8], demonstrated the feasibility of general secure computation and its significance to secure protocol design. The existence of semi-honest oblivious transfer (OT) was established by Kilian [49] as the minimal, i.e., necessary and sufficient, assumption for general secure computation. The focus of this work is on *black-box constructions* of *composable* MPC protocols. We discuss these two aspects.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 28; pp. 28:1–28:20
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Black-Box constructions. A construction is black-box if it does not refer to the code of any cryptographic primitive it uses, and only depends on their input/output behavior. Such constructions are usually preferable since their efficiency is not affected by the implementation details of the underlying cryptographic primitives; moreover, they remain valid and applicable if the code of the underlying primitives is simply not available, e.g., in case of constructions based on hardware tokens [64, 23, 46, 28, 38].

Early constructions of general-purpose MPC were non-black-box in nature particularly due to NP-reductions required by underlying zero-knowledge proofs [24]. Ishai et al. [41] presented the first black-box construction of general purpose MPC based on enhanced trapdoor permutations or homomorphic public-key encryption schemes. Together with the subsequent work of Haitner [36], this provided a black-box construction of a general MPC protocol under minimal assumptions (i.e., semi-honest OT). The round complexity of black-box MPC was improved to $O(\log^* n)$ rounds by Wee [78], and to constant rounds by Goyal [26]. In the two party setting, a constant round construction was first obtained by Pass and Wee [72], and subsequently a 5-round construction was given by Ostrovsky, Richelson, and Scafuro [68], which is known to be optimal by the results of Katz and Ostrovsky [47].

Composable security. The notion of security considered in early MPC works is called *standalone security* since it only considers a single execution of the protocol. Stronger notions of security are required for complex environments such as the Internet where several MPC protocols may run *concurrently.* This setting is often referred to as the *concurrent* setting, and unfortunately, as shown by Feige and Shamir [19], stand-alone security does not necessarily imply security in the concurrent setting.

To address this issue, Canetti [8] proposed the notion of *universally composable* (UC) security which has two important properties: *concurrent security* and *modular analysis.* The former means that UC secure protocols maintain their security in the presence of other *concurrent* protocols and the latter means that the security of a larger protocol in the UC framework can be derived from the UC security of its component protocols. This latter property is stated as a composition theorem which, roughly speaking, states that UC is closed under composition [8]. Unfortunately, UC security turns out to be impossible in the plain model for most tasks [8, 9, 11]. Relaxations of UC that consider composing the same protocol were also ruled out by Lindell [59, 60].

These strong negative results motivated the search for alternative notions of concurrent security in the plain model by endowing more power to the simulator such as super-polynomial resources [69, 75, 6], ability to receive multiple outputs [30, 29], or resorting to weaker notions such as bounded concurrency [1, 70], input indistinguishability [63], or a combination thereof [27]. While all of these notions were (eventually) achieved under polynomial hardness assumptions [75, 4, 62, 12, 21, 55, 71, 52, 50, 32, 6, 22], only angel-based security by Prabhakaran and Sahai [75] (including its extension to interactive angels by Canetti, Lin, and Pass [12]) and shielded-oracle security by Broadnax et al. [6] are known to have the modular analysis property, i.e., admitting a composition theorem along the lines of UC. We focus on angel-based security in this work since it arguably has somewhat better composition properties than shielded oracles.[1]

[1] As noted in [6], shielded oracle security does not technically have the modular analysis property and is actually strictly weaker than angel-based. Nevertheless, it is still "compatible" with the UC framework – the security of a composed protocol can be derived from that of its components.

Angel based security is similar to UC except that it allows the simulator as well as the adversary access to a super-polynomial resource called an "angel" which can perform a pre-defined task such as inverting a one-way function. Early constructions of angel-based security were based on non-standard assumptions [75, 4, 62]. The beautiful work of Canetti et al. [12] presented the first construction under polynomial hardness assumptions, and the subsequent work of Goyal et al. [32] improved the round complexity to $\widetilde{O}(\log \lambda)$ under general assumptions.

The first *black-box construction* of angel-based security was obtained by Lin and Pass [55], under the minimal assumption of semi-honest OT. The main drawback of [55] is that it requires *polynomially many* rounds even if the underlying OT protocol has constant rounds. To address this situation, Kiyoshima [50] presented a $\widetilde{O}(\log^2 \lambda)$-round construction assuming constant-round semi-honest OT (or alternatively, $\max(\widetilde{O}(\log^2 \lambda), O(\mathrm{R}_{\mathsf{OT}}))$ rounds where R_{OT} is OT's round-complexity). We remark that Broadnax et al. [6] present a constant-round black-box construction for (the weaker but still composable) shielded-oracle security (utilizing prior work by Hazay and Venkitasubramaniam [39] who provide a constant-round protocol in the CRS-hybrid model); however, they require stronger assumptions, specifically, homomorphic commitments and public-key encryption with oblivious public-key generation.

State of the art. To summarize our discussion above, under the *minimal assumption* of polynomially secure semi-honest OT, the best known round complexity of *black-box constructions* for angel-based security, and in fact any composable notion with modular analysis property, is due to Kiyoshima [50] which requires $\max(\widetilde{O}(\log^2 \lambda), O(\mathrm{R}_{\mathsf{OT}}))$ rounds. This is in contrast to the non-black-box construction of Goyal et al. [32] which requires only $\max(\widetilde{O}(\log \lambda), O(\mathrm{R}_{\mathsf{OT}}))$ rounds. Therefore, there is a multiplicative gap of $\widetilde{O}(\log \lambda)$ between the round-complexities of state-of-the-art black-box and non-black-box constructions of angel-based MPC if, e.g., semi-honest OT has at most logarithmic rounds.

1.1 Our Results

In this work, we prove the following theorem, thus closing the gap between the round complexity of black-box and non-black-box constructions of angel-based MPC under minimal assumptions:

▶ **Theorem 1** (Main). *Assume the existence of* R_{OT}*-round semi-honest oblivious transfer protocols. Then, there exists a* $\max(\widetilde{O}(\log \lambda), O(\mathrm{R}_{\mathsf{OT}}))$*-round black-box construction of a general MPC protocol that satisfies angel-based UC security in the plain model.*

Note that this yields a $\widetilde{O}(\log \lambda)$-round construction under the general assumption of enhanced trapdoor permutations since they imply constant-round semi-honest OT.

We follow the framework of [12] and its extensions in [55, 50]. The main building block [12] is a special commitment scheme called a CCA-Secure Commitment. Roughly speaking, a CCA-secure commitment is a tag-based commitment scheme that maintains hiding even in the presence of a decommitment oracle $\mathcal{O}$. More specifically, the adversary receives one commitment from an honest committer and may simultaneously make concurrently many commitments to $\mathcal{O}$ (similar to non-malleable commitments [17]). The oracle immediately extracts and sends back any value adversary commits successfully provided that it used a tag that is different from the one used by the honest committer. Lin and Pass [55] show that $O(\max(\mathrm{R}_{\mathsf{CCA}}, \mathrm{R}_{\mathsf{OT}}))$-round black-box angel-based MPC can be obtained from a $\mathrm{R}_{\mathsf{CCA}}$-round CCA commitment and a R_{OT}-round semi-honest OT protocol. Kiyoshima [50] demonstrated that $\widetilde{O}(k \cdot \log \lambda)$-round CCA-secure commitments can be obtained in a black-box manner from a k-round commitment scheme with slightly weaker security called "one-one CCA" where the adversary can open only one session each with the committer as well as the oracle; they further construct a $O(\log \lambda)$-round one-one CCA scheme from one-way functions in a black-box manner.

We instead present a *constant round* construction of one-one CCA, which implies $\widetilde{O}(\log \lambda)$-round (full) CCA commitments using [50] (and Theorem 1 using [55]):

▶ **Theorem 2** (CCA Secure Commitments). *Assume the existence of one-way functions. Then, there exists a $\widetilde{O}(\log \lambda)$-round black-box construction of a CCA-secure commitment scheme.*

1.2 Overview of Techniques

Current approaches. Let us briefly review the current approaches for constructing CCA secure commitments. The main difficulty in constructing CCA secure commitments under polynomial hardness is to move from the real world – which contains the exponential time decommitment oracle $\mathcal{O}$– to a hybrid where $\mathcal{O}$'s responses can be efficiently simulated. A standard way to do this is to use a argument-of-knowledge (AoK): the protocol should require the (man-in-the-middle) adversary, say $\mathcal{A}$, to give a AoK of the value it commits. The main difficulty in employing this is that $\mathcal{A}$ may open concurrently many sessions with $\mathcal{O}$ (referred here to as "right" side sessions), interleaved in an arbitrary manner; furthermore, these values have to be extracted *immediately* within each session irrespective of what happens in other sessions. This is precisely the issue in constructing (black-box simulatable) concurrent zero-knowledge (CZK) protocols [18] as well, and ideas from there are applied in this setting too. A second difficulty is that these extractions must happen without rewinding the commitment $\mathcal{A}$ *receives* (referred to as "left" side session).

It is worthwhile to quickly recall the (tag based) non-malleable commitment construction in the original work of [17]. In this construction, $\mathcal{A}$ has only one right session; to prove that the value on the right is (computationally) independent from that on the left, the value on the right is extracted without rewinding the sensitive parts of the left side commitments. This is done by creating two types of AoK– one each for two possible values of a bit. These AoK create rewinding "slots" for extraction such that if $\mathcal{A}$ uses a different bit in the tag, it risks the possibility of having to perform a AoK on its own – i.e., without any "dangerous" rewinding on the left – in one of the slots (called a "free" slot). These special AoK are performed for each bit of the tag *sequentially* so that at least one free slot is guaranteed since the left and right tags are different by definition. While this requires n rounds n-bit tags, it is possible to split the tag into n smaller tags of $\log n$ bits and run the protocol for each of them in parallel [17, 57]. Referred to as "LOG trick," this yields a $O(\log n)$-round protocol.

The key idea for CCA commitments in [12], at a high level, is to ensure that in the *concurrent* setting, many free slots exist for each session so that extraction succeeds before the end of that session. This is achieved by creating a polynomial round protocol consisting of sequential repetition of special AoK as above and then relying on an analysis that is, at a high level, similar to early rewinding techniques from CZK literature [76, 10]. Once the issue of concurrent extraction is handled, the additional ideas in [55] are (again, at a high level) to enforce this approach using cut-and-choose protocols to obtain a black-box construction. The work of Goyal et al. [32] shows how to separate the tasks of "concurrent extraction" and"non-malleability" in this approach by proving a "robust extraction lemma." This allows them to follow a structure similar to that of concurrent non-malleable zero-knowledge (CNMZK) from [3] which matches the round complexity of CZK, i.e., $\widetilde{O}(\log n)$. However, their approach requires non-black usage of one-way functions. Kiyoshima [50] shows that the robust-extraction lemma can actually be applied to the previous black-box protocol of [55] to get $\widetilde{O}(k\cdot\log n)$ rounds if one has a slightly stronger primitive than non-malleable commitments: namely k-round 1-1 CCA commitments. To build such commitments, Kiyoshima builds non-malleability "from scratch" by combining the DDN "LOG trick" with cut-and-choose components of [55] so that the extraction on right in the standalone setting, can be done without any dangerous rewinding on left. This however results in $O(\log n)$ rounds for 1-1 CCA and $\widetilde{O}(\log^2 n)$ for full CCA.

Our approach. We significantly deviate from current approaches for constructing 1-1 CCA commitments. Instead of attempting to build non-malleability *from scratch*, our goal is to have a generic construction built around existing non-malleable commitments. The resulting protocol will not only have a simpler and more modular proof of security, but will also benefit from the efficiency and assumptions of the underlying non-malleable commitment (NMCom). Towards this goal, we return to investigate the structure of CNMZK protocols even for the simpler case of 1-1 CCA.

Setting aside the issue of round-complexity for the moment, a key idea in the construction of CNMZK protocols [3, 56, 67, 54] is to have the prover give a non-malleable commitment (NMCom) which can later be switched to a "trapdoor value" set by the verifier; the non-malleability of NMCom ensures that $\mathcal{A}$ cannot switch his value to a trapdoor on the right (unless he did so in the real world, which can be shown to be impossible through other means). The prover later proves that either the statement is true or it committed the trapdoor. The main problem with this approach is that it requires us to prove a predicate over the value committed in NMCom which requires non-black-box use of cryptographic primitives.

Non-malleable commit-and-prove. One potential idea to avoid non-black-box techniques is to turn to black-box commit-and-prove protocols in the literature and try to re-develop them in the context of non-malleability. Commit-and-prove protocols allow a committer to commit to a value v so that later, it can prove a predicate ϕ over the committed value in zero-knowledge. These protocols can be constructed in constant rounds using the powerful "MPC-in-the-head" approach introduced by Ishai et al. [42]. The approach allows committing multiple values $v_1, \ldots, v_n$ and then proving a joint predicate ϕ over them. One such construction is implicit in the work of Goyal et al. [31]. Such commitments were also used extensively by Goyal et al. to build size-hiding commit-and-prove [33] and an optimal four round construction was obtained by Khurana, Ostrovsky, and Srinivasan [48]. As noted above, if we can develop an appropriate non-malleable version of such protocols, it is conceivable that they can yield constant-round 1-1 CCA commitment. However, that non-malleable commitments are not usually equipped to handle proofs. Thus, such an approach will necessarily have to "open up" the construction of non-malleable commitments. In particular, like previous constructions, this approach cannot be based on non-malleable commitments in a black-box manner.

Changing the direction of NMCcom. In order to rely on non-malleable commitments directly, it is essential that we do not prove anything about the values committed inside the NMCom. Instead, we should restrict all proofs to be performed only over standard commitments since for them we can use standard black-box commit-and-prove protocols. Towards building this property, what if we change the direction of NMCom and ask the receiver of 1-1 CCA to send non-malleable commitments, which, for example, can be opened later? More specifically, in our 1-1 CCA protocol, the receiver will send a NMCom to a random value σ which it will open subsequently. The committer will send a "trapdoor commitment" t before it sees σ opened. Later, the committer will commit to the desired value v and give a AoK that either it knows v or t is a commitment to σ (the "trapdoor"). Observe that this structure completely avoids any proof directly over non-malleable commitments; all proofs only need to be performed over ordinary commitments. Therefore, if we use the commit phase of black-box commit-and-prove protocols to commit to σ and v we can easily complete the AoK in a black-box manner: the predicate ϕ in the proof phase will simply test for the presence of trapdoor σ. Some standard soundness issues arise in this approach but they can be handled by ensuring that the commit phase is extractable.

Although this approach yields a black-box construction directly from NMCom, it is hard to prove the 1-1 CCA property. At a high level, this is because of the following: if in the 1-1 CCA game, $\mathcal{A}$ schedules the completion of the left NMCom *before* the right one[2], the simulator in the security proof must extract σ from this NMCom while the right NMCom is still in play (so that it can generate t to be a commitment to σ). This involves rewinding the left NMCom (assuming it is extractable) which in turn rewinds the right session.[3] A similar issue arises in the work of Jain and Pandey [44] on black-box non-malleable zero-knowledge where it is resolved by using a NMCom that is already 1-1 CCA secure. We do not have this flexibility in our setting.

A possible fix for this issue is to rely on some kind of "delayed input" property: i.e., the commitment to t will be an extractable commitment that does not require the message m to be committed until the last round. This property can be obtained by committing to a key k in an extractable manner and then in the last round committing to m by simply encrypting with k. This however will no longer be compatible with the black-box commit-and-prove strategy since we will now have to take encryption into account.

We overcome this issue by making extensive use of extractable commitments. More specifically, we first *prepend* the NMCom with a standard "slot-based" extractable commitment which commits to the same value σ as the NMCom. If the NMCom also has a slot like extractable structure (e.g., the *three round* scheme of [34]), we can argue that non-synchronous adversaries must always leave a free slot either on top or at the bottom of NMCom. For example, in the troublesome scheduling discussed above, $\mathcal{A}$ can be easily rewound in the last two messages of NMCom (if we use [34]) *without* rewinding the right NMCom. In other non-synchronous schedules it will have a free slot in the top extractable commitment on the left. On the other hand, synchronous adversaries will fail in the NMCom step (and synchronous non-malleability suffices for our purposes). In summary, this will suffice for us to show that even if our simulator sets up the trapdoor statement on the left (by committing σ in t), $\mathcal{A}$ cannot do the same on the right. Other NMCom, particularly public-coin extractable NMCom also seem sufficient.

A second issue here is the intertwining of the left AoK[4] with "extractable" components on the right, e.g., the right AoK (or extractable commitment steps). In order to prove that $\mathcal{A}$ cannot setup the trapdoor, extraction from right AoK will be necessary and this will be troublesome when changing the witness in the left AoK during hybrids. This issue can be handled using the sequential repetition technique from [53]: we use $k+1$ AoKs where k is the (constant) rounds in a single AoK. Also note that other common methods for handling this issue do not work: e.g., we cannot rely on *statistical* WI since it requires stronger assumptions

[2] Note that NMCom's direction is opposite to that of 1-1 CCA: the receiver of 1-1 CCA is the sender of right NMCom.

[3] This is not an issue in the synchronous schedule since in that case, the value $\mathcal{A}$ commits to in NMCom is provided to the distinguisher along with the joint view.

[4] Observe that the AoK will just be the proof part of appropriate black-box commit-and-prove with right parameters to ensure black-box property; they will also satisfy witness-indistinguishability [19].

for constant rounds; we also cannot use proofs that are secure against a fixed number of rewinds since they usually allow a noticeable probability of extraction which is insufficient for 1-1 CCA commitments, where extraction must succeed with overwhelming probability.

1.3 Other Related Works

The focus of our work is constructions in the plain model. Hazay and Venkitasubramaniam [40] gave a black-box construction of an MPC protocol without any setup assumptions that achieves composable security against an *adaptive* adversary. UC security can be achieved by moving to other trusted setup models such as the common reference string model [14, 9, 35], assuming an honest majority of parties [11], trusted hardware [64, 23, 46, 15], timing assumptions on the network [45], registered public-key model [2], setups that may be expressed as a hybrid of two or more of these setups [20], and so on. Lin, Pass, and Venkitasubramaniam [58, 71] show that a large number of these setup models could be treated in a unified manner, and black-box analogues of these results were obtained by Kiyoshima, Lin, and Venkitasubramaniam [51].

2 Preliminaries

Notation. We use λ for the security parameter. We use $\overset{\mathrm{c}}{\approx}$ to denote computational indistinguishability between two distributions. For a set S, we use $x \overset{\$}{\leftarrow} \mathsf{S}$ to mean x is sampled uniformly at random from S. PPT denotes probabilistic polynomial time and $\mathsf{negl}(\cdot)$ denotes negligible function.

We assume familiarity with standard concepts such as commitment schemes, witness indistinguishability. In the following, we recall the definitions for extractable commitments, non-malleable commitments and CCA commitments. Definitions for the more basic primitives and other constructs (such as MPC related definitions) can be found in the full version of the paper [16].

2.1 Extractable Commitments

▶ **Definition 3** (Extractable Commitment Schemes). *A commitment scheme* $\mathsf{ExtCom} = (S, R)$ *is extractable if there exists an expected polynomial-time probabilistic oracle machine (the extractor)* Ext *that given oracle access to any PPT cheating sender* S^* *outputs a pair* (τ, σ^*) *such that:*

- ***Simulation:*** *τ is identically distributed to the view of S^* at the end of interacting with an honest receiver R in commitment phase.*
- ***Extraction:*** *the probability that τ is accepting and $\sigma^* = \bot$ is negligible.*
- ***Binding:*** *if $\sigma^* \neq \bot$, then it is statistically impossible to open τ to any value other than σ^*.*

The above construction of ExtCom (Figure 1) is standard [17, 74, 77, 70]. We will refer to it as the *standard* ExtCom.

▶ Remark 4 (Regarding Over-Extraction). Intuitively, Definition 3 stipulates that if the committer indeed commits to some value, the extractor must be able to extract it. We remark that it does not rule out what is called the "over-extraction" issue – namely, the extractor may extract a valid looking value even though none actually exists.

However, this definition suffices for most ZK and MPC applications (e.g. [75, 70, 72]), including ours. In our arguments specifically, we will employ hybrids where we extract the value that the adversary commits to using these commitment schemes. Jumping ahead, in the security proof for our protocol, we will need successful extraction only if the adversary actually commits to some *valid* value; otherwise, completing the hybrids is trivial.

The extractable commitment scheme, based on any commitment scheme Com, works in the following way. The scheme has 3 rounds if Com is non-interactive.

Input:

- Both S and R get security parameter 1^λ as the common input.
- S gets a string σ as his private input.

Commitmment Phase:

- The sender (committer) S commits using Com to λ pairs of strings $\{(v_i^0, v_i^1)\}_{i=1}^{\lambda}$ where $(v_i^0, v_i^1) = (\eta_i, \sigma \oplus \eta_i)$ and η_i are random strings in $\{0,1\}^{\ell(\lambda)}$ for $1 \leq i \leq \lambda$.[a]
- Upon receiving a challenge $\overline{c} = (c_1, \dots, c_\lambda)$ from the receiver R, S opens the commitments to $(v_1^{c_1}, \dots, v_\lambda^{c_\lambda})$.
- R checks that the openings are valid.

Decommitment Phase:

- S sends σ and opens the commitments to all λ pairs of strings.
- R checks that all the openings are valid, and also that $\sigma = v_1^0 \oplus v_1^1 = \cdots = v_\lambda^0 \oplus v_\lambda^1$.

[a] The scheme supports extraction as long as $k = \omega(\log \lambda)$ pairs are used.

Figure 1 Extractable Commitment Scheme $\langle S, R \rangle$.

2.2 Non-Malleable Commitments

We follow the definition of non-malleability from [57, 34]. This definition is based on the comparison between a real execution with an ideal one. In the *real* interaction, we consider a man-in-the-middle adversary $\mathcal{A}$ interacting with a committer C in the *left* session, and a receiver R in the *right*. We denote the relevant entities used in the right interaction as "tilde'd" version of the corresponding entities on the left. In particular, suppose that C commits to v in the left interaction, and $\mathcal{A}$ commits to $\widetilde{v}$ on the right. Let MIM_v denote the random variable that is the pair $(\mathsf{view}, \widetilde{v})$, consisting of the adversary's entire view of the man-in-the-middle execution as well as the value committed to by $\mathcal{A}$ on the right (assuming C commits to v on the left). The *ideal* interaction is similar, except that C commits to some arbitrary fixed value (say 0) on the left. Let MIM_0 denote the pair $(\mathsf{view}, \widetilde{v})$ in the ideal interaction. We use a *tag-based* (or "identity-based") specification, and ensure that $\mathcal{A}$ uses a distinct tag $\widetilde{\mathsf{id}}$ on the right from the tag id it uses on the left. This is done by stipulating that MIM_v and MIM_0 both output a special value $\perp_{\mathsf{id}}$ when $\mathcal{A}$ uses the same tag in both the left and right executions. The reasoning is that this corresponds to the uninteresting case when $\mathcal{A}$ is simply acting as a channel, forwarding messages from C on the left to R on the right and vice versa. We let $\mathsf{MIM}_v(z)$ and $\mathsf{MIM}_0(z)$ denote the real and ideal interactions respectively when the adversary receives auxiliary input z.

▶ **Definition 5** (Non-Malleable Commitment Schemes). *A (tag-based) commitment scheme $\langle C, R \rangle$ is non-malleable if for every* PPT *man-in-the-middle adversary $\mathcal{A}$, and for all values v, we have $\{\mathsf{MIM}_v(z)\}_{z \in \{0,1\}^*} \overset{c}{\approx} \{\mathsf{MIM}_0(z)\}_{z \in \{0,1\}^*}$.*

Synchronizing Adversaries. This notion refers to man-in-the-middle adversaries who upon receiving a message in one session, immediately respond with the corresponding message in the other session. An adversary is said to be *non-synchronizing* if it is not synchronizing.

2.3 CCA Commitments

We define the notion of CCA-secure commitments (and 1-1 CCA security in particular). These definitions rely on the notion of a *decommitment oracle*, which provide decommitments given valid transcripts to a particular (tag based) commitment protocol. Specifically, a decommitment oracle $\mathcal{O}$ for a given commitment protocol acts as follows:

- $\mathcal{O}$ acts as an honest reciever against some committer C, participating faithfully according to the specified commitment scheme. C is allowed to pick a tag for this interaction adaptively.
- At the end of this interaction, if the honest reciever were to accept the transcript as containing a valid commitment with respect to the given tag, $\mathcal{O}$ returns the value v committed by C to it. Otherwise, it returns $\perp$.

We denote an adversary with access to the decommitent oracle as $\mathcal{A}^{\mathcal{O}}$. CCA security then essentially constitutes preservation of the hiding property even against adversaries enjoying such oracle access. More formally, we define the following game $\mathsf{IND}_b(\langle C, R\rangle, \mathcal{A}, \mathcal{O}, n, z)$ ($b \in \{0, 1\}$) as follows: given the public parameter 1^n and auxillary input z, the adversary $\mathcal{A}^{\mathcal{O}}$ adaptively generates two challenge values v_0, v_1 of length n, and a tag $\mathsf{tag} \in \{0, 1\}^n$. Then, $\mathcal{A}^{\mathcal{O}}$ receives a commitment to v_b with tag tag from the challenger. Let y be the output of $\mathcal{A}$ in this game. The output of the game is $\perp$ if during the game, $\mathcal{A}$ sends $\mathcal{O}$ any commitment using tag tag. Otherwise, the output of the game is y. We abuse notation to denote the output of the game $\mathsf{IND}_b(\langle C, R\rangle, \mathcal{A}, \mathcal{O}, n, z)$ by the same symbol $\mathsf{IND}_b(\langle C, R\rangle, \mathcal{A}, \mathcal{O}, n, z)$.

▶ **Definition 6** (CCA Commitment). *Let $\langle C, R\rangle$ be a tag-based commitment scheme, and $\mathcal{O}$ be an associated decommitment oracle. Then $\langle C, R\rangle$ is said to be* ***CCA secure w.r.t. $\mathcal{O}$****, if for every nonuniform P.P.T. machine $\mathcal{A}$, the following ensembles are computationally indistinguishable:*

- $\{\mathsf{IND}_0(\langle C, R\rangle, \mathcal{A}, \mathcal{O}, n, z)\}_{n\in\mathbb{N}, z\in\{0,1\}^*}$
- $\{\mathsf{IND}_1(\langle C, R\rangle, \mathcal{A}, \mathcal{O}, n, z)\}_{n\in\mathbb{N}, z\in\{0,1\}^*}$

It is customary to call any commitment scheme that is CCA secure with respect to some decommitment oracle as just CCA secure (but in general the oracle is usually also described, and is of course necessary to prove such security). It is also customary to call the interaction between the challenger and adversary as the *left* interaction, and that between adversary and oracle as the *right* interaction, in the fashion of non-malleable commitments, where the security property chiefly considers man in the middle attacks.

Finally, a scheme is *1-1 CCA secure* (denoted as $\mathrm{CCA}^{1:1}$) if the corresponding adversary is only allowed one interaction with the oracle.

3 A New $\mathrm{CCA}^{1:1}$ Commitment Scheme

We will require the following ingredients for our $\mathrm{CCA}^{1:1}$ protocol:

- A statistically-binding commitment Com. Naor's commitment works.
- A 3-round slot-based extractable commitment scheme ExtCom; for concreteness we will use the standard 3-round scheme, shown in Figure 1. based on Naor's commitment (the first message ρ of Naor's commitment is not counted in rounds and assumed to be available from other parts of the protocol).
- An (extractable) commitment scheme ENMC that is *non-malleable* against *synchronizing* adversaries. We will need this protocol to be "compatible with slots" of the ExtCom defined above. For concreteness, we assume that ENMC is the 3-round commitment scheme of [34] which satisfies all our requirements.
- A k round witness indistinguishable argument of knowledge WIAoK.

We stress that all of these ingredients have constant rounds, and can be constructed from standard OWFs in a black-box manner.

Our Protocol. We now describe our first protocol for $\mathrm{CCA}^{1:1}$ commitments. This protocol does not specifically try to achieve the black-box usage of cryptographic primitives. This allows us to focus on proving CCA security. However, it achieves two important properties: it is based on minimal assumptions, and it has a constant number of rounds. Moreover, the structure of this protocol is chosen in such a way that later, it will be possible to convert into a fully black-box construction. We remark that we also directly use identities of length λ directly (this is in keeping with the [34] construction which does the same).

The formal description of the protocol appears in Figure 2. At a high level, the protocol proceeds as follows. First, it requires the receiver to commit to a trapdoor string α using two extractable primitives: ExtCom as well as ENMC. Next, the committer will commit to an all zero-string β using ExtCom. Jumping ahead, in the security proof a "simulator machine" on left will set $\beta = \alpha$ and use it as a "fake witness" in a WIAoK; later we shall instantiate ExtCom with, roughly speaking, a "black-box commit-and-prove" to obtain a black-box construction. The receiver simply opens α in the next step, and the committer commits to the desired value, say v, followed by a proof of knowledge of v or that $\beta = \alpha$. A crucial observation here is that *the proofs are not required to deal with values inside* ENMC – by ensuring that ENMC values opened in the protocol execution.

▶ **Theorem 7.** *The protocol $\langle C, R\rangle_{\mathsf{CCA}}$ (described in Figure 2) is a 1-1 CCA commitment scheme for all polynomial time adversaries.*

The statistical-binding property of protocol $\langle C, R\rangle_{\mathsf{CCA}}$ is straightforward. The computational hiding property is implied by the 1-1 CCA security as per Definition 6. Due to lack of space, we present an outline of the proof for non-malleability below. The complete proof is given in the full version of our paper [16].

Proof for Non-Malleability (Sketch.) We start with a man-in-the-middle adversary $\mathcal{A}$ that participates in the CCA challenge outlined above. Consider any two arbitrary values v_0 and v_1 in the message space. We will now show indistinguishability between the games $\{\mathsf{IND}_0(\langle C, R\rangle_{\mathsf{CCA}}, \mathcal{A}, \mathcal{O}, n, z)\}_{n\in\mathbb{N}, z\in\{0,1\}^*}$ and $\{\mathsf{IND}_1(\langle C, R\rangle_{\mathsf{CCA}}, \mathcal{A}, \mathcal{O}, n, z)\}_{n\in\mathbb{N}, z\in\{0,1\}^*}$. To this end, we will use a hybrid argument:

Our proof proceeds as follows. We start with the honest committer on the left committing to some arbitrary value v_0. The overall idea is to move to an intermediate hybrid where the left challenger is able to "set the trapdoor" and go through the WIAoKs without using the commitment. This will allow us to then replace the initial commitment to v_0 to one to v_1 (and move back to doing everything on the left "honestly"). Further, we will also maintain the following invariant across all the hybrids:

▶ **Definition 8** (Invariant Condition (informal)). *In the right session, the adversary* MIM *cannot set $\tilde{\beta} = \tilde{\alpha}$ except with negligible probability.*

We outline the necessary hybrids to get to this stage below:

Hybrid H_0^0. This is just the original experiment with the honest committer on the left. In other words, this is the experiment $\{\mathsf{IND}_0(\langle C, R\rangle_{\mathsf{CCA}}, \mathcal{A}, \mathcal{O}, n, z)\}_{n\in\mathbb{N}, z\in\{0,1\}^*}$. It is straightforward to see that the invariant holds in this hybrid: if it does not, then MIM must break the hiding property of the commitments in Stage 1 or 2 in the right execution to learn $\tilde{\alpha}$.

We let $\lambda \in \mathbb{N}$ denote the security parameter. All primitives used in the protocol by default have 1^λ as part of their input. We omit this detail in the following. Further, we assume that the execution involves a tag or identity $\mathsf{id} \in \{0,1\}^\lambda$.
Input: The committer C and reciever R have common input as the security parameter 1^λ. Additionally, C has as private input a value v which it wishes to commit to.

Commit Phase: This proceeds as follows:

Stage 0: C commits to the value v using Com and sends the identity id to R.

Stage 1: This consists of the following steps:
(a) R picks a value $\alpha \xleftarrow{\$} \{0,1\}^\lambda$.
(b) R commits to $\alpha_1 = \alpha$ using ExtCom.

Stage 2: R commits to $\alpha_2 = \alpha$ using ENMC, using identity id.
For future reference, we denote by $\mathsf{CombinedCom}$ the joint execution of Stage 1 and 2 up to this point. Observe that $\mathsf{CombinedCom}$ is a statistically binding commitment scheme.

Stage 3: C now commits to $\beta = 0^\lambda$ using ExtCom.

Stage 4: This goes as follows:
1. R decommits to both its commitments so far, revealing α_1 and α_2.
2. C checks these decommitments, aborting if $\alpha_1 \neq \alpha_2$.

Stage 5: C and R engage in $k+1$ WIAoK protocols *sequentially*. We denote these WIAoK executions as WIAoK_i for $i = 1, \ldots, k+1$. In all these WIAoKs, C proves the *same* (compound) statement which is true if and only if:
(a) there exists randomness η s.t. $c = \mathsf{Com}(v;\eta)$; **or**
(b) $\beta = \alpha_1 = \alpha_2$, where β is the unique string committed in the transcript of Stage-3.

Note that an honest prover will always use the witness for part-(a) of the above compound statement, which we refer as the "original witness". We will refer the witness for part-(b) of the compound statement. Looking ahead, some hybrids will use the trapdoor witness to go through the WIAoKs.

Decommit Phase: The committer C decommits to v and β. R checks if these decommitments are valid, and accepts if so.

Figure 2 Protocol $\langle C, R\rangle_{\mathsf{CCA}}$: $\mathrm{CCA}^{1:1}$ Commitment Scheme.

Hybrid H_1^0. In this hybrid, the decommitment oracle on the right is removed. All messages are generated as an honest receiver, and the value $\tilde{v}$ committed by the adversary is obtained by extracting the witness from the final WIAoK on the right.

Hybrid H_2^0. This hybrids proceeds as the previous one except that on the left, we also extract the value α_2 from the stage 2 ENMC.

In these two hybrids, the adversary's view up to Stage 4 is identical to that in H_0. Therefore, the invariant must hold in these hybrids (by the same reasoning as in H_0^0). By the knowledge soundness of WIAoK, the extracted witness should be the same as the one returned by the oracle in H_0. The entire view of the adversary is therefore unaffected by these changes, and so is identical to that in H_0^0.

Hybrid $\boldsymbol{H_3^0}$. This hybrid also proceeds as the previous one except that the value β is now set to be the extracted value α_2.

While it is easy to argue that the adversary cannot detect this change, it may still be able to use the change in the left ExtCom to change its own ExtCom on the right (note that the ExtCom scheme is malleable)! This is the primary reason we need the invariant condition, as it explicitly prevents this exact occurrence. We argue that the invariant holds by considering separate cases for synchronous and nonsynchronous adversaries. For synchronous adversaries, we show roughly that if the adversary could identify any change in the left stage 3 ExtCom, then it must have been influencing the earlier two commitments it made in a malleable fashion - this is ruled out by the non-malleability of ENMC. *As mentioned in the Technical Overview, this is the most difficult case to deal with, and where our main contribution lies.* For nonsynchronous adversaries, we show instead that there are "extraction opportunities" on the left where no messages on the right are exchanged for the corresponding duration (and extraction on the left can be performed unhindered). This relies on carefully setting the appropriate round complexities for ExtCom and ENMC.

Hybrid $\boldsymbol{H_4^0}$. In this hybrid, we ask the left execution to use the *trapdoor witness* in the Stage 5 WIAoKs (the actual changes happen by constructing a sequence of intermediate hybrids where the witness is replaced one by one in each WIAoK execution on the left, in order of occurrence).

In this hybrid, we ensure that any change of witness on the left does not occur during the same time as extraction of the witness on the right, since the latter involves rewinding and that can interfere with the witness indistinguishability of the left proof. In the synchronizing case, it is easy to see the invariant holds in this hybrid since the corresponding changes all occur *after* the right stage 3 ExtCom is concluded. For nonsynchronous adversaries, this may not be the case, but we can use an argument very similar to that used to argue the invariant in H_3^0. We can also argue indistinguishability of (the view in) this hybrid using the witness indistinguishability of our WIAoK scheme and the fact that the change in witness on the left, and extraction of witness on the right, occur at different times.

Finally, we define H_3^1, H_2^1, H_1^1 to H_0^1, where for H_i^1 is just the analogue of H_i^0, but replacing the initial commitment to v_0 with one to v_1 on the left. Using the same arguments above, we can show that both the invariant condition and indistinguishability views holds among H_4^1, H_3^1 down to H_0^1. Note that H_0^1 is just the experiment $\{\mathsf{IND}_1(\langle C, R\rangle_{\mathsf{CCA}}, \mathcal{A}, \mathcal{O}, n, z)\}_{n\in\mathbb{N}, z\in\{0,1\}^*}$, and hence we show that this is computationally indistinguishable from $\{\mathsf{IND}_0(\langle C, R\rangle_{\mathsf{CCA}}, \mathcal{A}, \mathcal{O}, n, z)\}_{n\in\mathbb{N}, z\in\{0,1\}^*}$ to $\mathcal{A}$. ◀

4 Our Black-Box CCA Commitment

Our starting point is to determine how we can make our protocol (in Figure 2) fully black-box. In fact, we note that the only component that is not already so is our argument system. Note that we use the arguments to prove statements about the Com in **Stage 0** and the ExtCom in **Stage 3**. Thus, we look to change these components with a suitable "commit-and-prove" protocol that is fully black-box. Further, we will require the following properties from the protocol so that it works with our template:

1. It should provide us a Com, and an ExtCom scheme. Importantly, this "first part" of the protocol should alone serve as a valid commitment with desired properties, regardless of whether we perform a subsequent WIAoK or not. This is in contrast to the definition (and construction) in [48].
2. Later we should be able to perform WIAoK on a compound statement regarding the values committed in the above Com and ExtCom;
3. The WIAoK part should make use of Com and ExtCom only in a black-box manner.

We provide a definition of Commit-and-Prove WIAoK (actually ZKAoK) formally in the full version [16] that captures all the required properties.

We note that there are not many approaches in the literature that achieve what we want: the work of [48] constructed a round-optimal version of such a primitive, but their protocol does not fit our needs because we require some properties that are not immediately avaiable from their construction. We discuss the issues briefly. While their commit stage is a statistically binding commitment, it cannot be modified to be *extractable*, which we crucially need. Further, we will require a *multi-commitment* property for our proof, namely that the predicate to be proved can support values used in *multiple* commit stages; while it is possible that the [48] protocol can be modified to achieve this property, the modification is unclear. Yet another issue is that we will use multiple proof modules (for the same proof) in our final protocol. Again it is not clear how to modify their protocol so that we ensure consistency of openings while also ensuring extractability from every proof module (note that their protocol has a challenge-response format that can allow extraction from just two challenges even across different sessions). We remark however that their protocol does support the *delayed predicate* property, which we also rely on.

We therefore build a commit-and-prove protocol suitable to our purposes. Our starting point is the "MPC-in-the-head" technique from [42]. This approach was originally used to construct black-box zero-knowledge arguments, by having the prover run a virtual MPC execution "in its head" and committing to the views of the virtual parties. The verifier then asks for some of these views to be opened and checks that the opened views are consistent. This construction achieves honest-verifier ZKAoK.[5] To meet our requirements, we want to turn this construction into commit-and-prove form, and bolster the security to tolerate malicious verifiers.

There are a few previous works that already take this approach to create commit and prove protocols. There is the recent work of Hazay, Ishai and Venkitasubramaniam [37], who make use of a commit and prove style protocol constant round secure 2PC protocols against malicious adversaries. Their overall design of the construction utilizes the MPC-in-the-head idea, but by way of using *server watchlists* which is a slightly different implementation of this concept first used in [43]. Being a part of their overall 2PC compiler, their protocol is in the OT-hybrid model, which makes it difficult to adapt to our usage, which is in the plain model. It is also unclear how to modify their construction to have the *multi-commitment* property, as well as make it argument of knowledge, as there is no immediate extraction algorithm to extract the sender's committed value from the proof stage.

We instead follow the approach used in the older work of Goyal et al [31], who use two virtual MPC executions. The first execution is simply a verifiable secret sharing of the value to be committed (the commitment to all the views serves also to commit to this value), which

[5] The authors of [42] also showed how to make it secure against dishonest verifiers. But their technique results in a construction with polynomially-many rounds, because they employ sequential repetition.

is sent on to the verifier (we will call this the *commit phase*). The second execution continues on from the first, where the virtual parties use their shares to compute some predicate on the shared value that represents the statement to be proved over the commitment (we will call this the *proof phase*).

To obtain security against malicious verifiers, one idea is to have the verifier *commit to* its challenge *before* the prover sends its first message. However, we cannot resort to this since this would excise the argument of knowledge property from this construction. We therefore employ a different approach, by building in a *coin-tossing* protocol into the argument system, which only uses an extractable commitment scheme. This is similar to the construction used in [61] to convert ZK arguments to ZKAoKs. It allows the PPT simulator to bias the coin-tossing result, thus allowing for simulation against dishonest verifiers; meanwhile, the knowledge extraction strategy in [42] still works.

The remaining task is to build an extractable commitment scheme that is compatible with the above. To obtain this, we observe that if the predicate ϕ to be simply the identity predicate, then we can still extract the committed value (i.e., the “witness” in the proof phase) as outlined above. Therefore, we view an execution of the above commit-and-prove protocol (with the identity predicate) as an extractable commitment scheme: hiding follows from the hiding of the commit phase as well as zero knowledge of the proof phase, and extractability follows just as mentioned above (i.e., by the argument of knowledge property). We claim further that this commitment scheme is actually *compatible* with our commit-and-prove scheme; this is due to the property of *multiple proofs* mentioned earlier, wherein *separate* proof modules can be performed on the *same* commitments (by adjusting the parameters of verifiable secret sharing). In particular, this allows the prover to use this new extractable commitment scheme, then later perform (black-box) arguments of knowledge for some statement about the value committed to in this scheme. This suffices for our purposes. We provide more details and the formal construction in the full version of our paper [16].

Now we can simply integrate these components into our original 1-1 CCA commitment scheme to obtain a fully black-box instantiation. We present our final protocol and security proof in the full version [16]. Note that our commit-and-prove scheme is constant round, and therefore our final protocol is still constant rounds.

5 Angel-Based MPC in $\widetilde{O}(\log\lambda)$ Rounds

Kiyoshima [50] presents a black-box construction of a CCA-secure commitment scheme with the following ingredients: (a) a two-round statistically-binding commitment scheme, and a constant round “strongly extractable” commitment, both of which are known from (black-box) one-way functions, (b) a concurrently-extractable commitment (due to Micciancio et al., [66]), with a “robustness parameter” ℓ, and (c) an R-round 1-1 CCA-secure commitment provided that $\ell = O(R \cdot \log\lambda \cdot \log\log\lambda)$. The round-complexity of the resulting protocol is $O(\ell)$. If R is a constant, this yields a $\widetilde{O}(\log\lambda)$-round construction for CCA secure commitments.[6] This yields Theorem 2.

As mentioned in the introduction, the security model that we consider is *angel-based* security, or UC *security with superpolynomial helpers*. Very briefly, this is essentially the same as the UC model used in [8], except that the adversary (in the real world) and the

[6] More precisely, Kiyoshima states his results with a specific value of ℓ, namely, $O(\log^2\lambda \cdot \log\log\lambda)$, since $R = O(\log\lambda)$ in his case. However, his construction and proof work for any value of R if ℓ is as described above.

environment (in the ideal world) both have access to a superpolynomial time functionality that acts as an oracle or a *helper*. Formal definitions for this security model can be found in [12] and [55]. If there is a protocol Π that emulates a functionality $\mathcal{H}$ with helper $\mathcal{H}$ in this setting, we say that Π $\mathcal{H}$-EUC-realizes $\mathcal{F}$.

Now, as in [50], we combine Theorem 2 with the following two results due to Canetti et al. [12, 13] and Lin and Pass [55] respectively to obtain Theorem 1.

▶ **Theorem 9** ([55]). *Assume the existence of an R_{CCA}-round robust CCA-secure commitment scheme $\langle C, R\rangle$ and the existence of an R_{OT}-round semi-honest oblivious transfer protocol $\langle S, R\rangle$. Then, there is an $O(\max(R_{\mathsf{CCA}}, R_{\mathsf{OT}}))$-round protocol that $\mathcal{H}$-EUC-realizes $\mathcal{F}_{\mathsf{OT}}$. Furthermore, this protocol uses $\langle C, R\rangle$ and $\langle S, R\rangle$ only in a black-box way.*

▶ **Theorem 10** ([12, 13]). *For every well-formed functionality $\mathcal{F}$, there exists a constant-round $\mathcal{F}_{\mathsf{OT}}$-hybrid protocol that $\mathcal{H}$-EUC-realizes $\mathcal{F}_{OT}$.*

References

1 Boaz Barak. Constant-round coin-tossing with a man in the middle or realizing the shared random string model. In *43rd FOCS*, pages 345–355. IEEE Computer Society Press, November 2002. doi:10.1109/SFCS.2002.1181957.

2 Boaz Barak, Ran Canetti, Jesper Buus Nielsen, and Rafael Pass. Universally composable protocols with relaxed set-up assumptions. In *45th FOCS*, pages 186–195. IEEE Computer Society Press, October 2004. doi:10.1109/FOCS.2004.71.

3 Boaz Barak, Manoj Prabhakaran, and Amit Sahai. Concurrent non-malleable zero knowledge. In *47th FOCS*, pages 345–354. IEEE Computer Society Press, October 2006. doi:10.1109/FOCS.2006.21.

4 Boaz Barak and Amit Sahai. How to play almost any mental game over the net – concurrent composition via super-polynomial simulation. In *46th FOCS*, pages 543–552. IEEE Computer Society Press, October 2005. doi:10.1109/SFCS.2005.43.

5 Donald Beaver. Foundations of secure interactive computing. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 377–391. Springer, Heidelberg, August 1992. doi:10.1007/3-540-46766-1_31.

6 Brandon Broadnax, Nico Döttling, Gunnar Hartung, Jörn Müller-Quade, and Matthias Nagel. Concurrently composable security with shielded super-polynomial simulators. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 351–381. Springer, Heidelberg, 2017. doi:10.1007/978-3-319-56620-7_13.

7 Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, January 2000. doi:10.1007/s001459910006.

8 Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001. doi:10.1109/SFCS.2001.959888.

9 Ran Canetti and Marc Fischlin. Universally composable commitments. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 19–40. Springer, Heidelberg, August 2001. doi:10.1007/3-540-44647-8_2.

10 Ran Canetti, Oded Goldreich, Shafi Goldwasser, and Silvio Micali. Resettable zero-knowledge (extended abstract). In *32nd ACM STOC*, pages 235–244. ACM Press, May 2000. doi:10.1145/335305.335334.

11 Ran Canetti, Eyal Kushilevitz, and Yehuda Lindell. On the limitations of universally composable two-party computation without set-up assumptions. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 68–86. Springer, Heidelberg, May 2003. doi:10.1007/3-540-39200-9_5.

12 Ran Canetti, Huijia Lin, and Rafael Pass. Adaptive hardness and composable security in the plain model from standard assumptions. In *51st FOCS*, pages 541–550. IEEE Computer Society Press, October 2010. doi:10.1109/FOCS.2010.86.

13 Ran Canetti, Huijia Lin, and Rafael Pass. Adaptive hardness and composable security in the plain model from standard assumptions. *SIAM J. Comput.*, 45(5):1793–1834, 2016.

14 Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *34th ACM STOC*, pages 494–503. ACM Press, May 2002. doi:10.1145/509907.509980.

15 Nishanth Chandran, Wutichai Chongchitmate, Rafail Ostrovsky, and Ivan Visconti. Universally composable secure computation with corrupted tokens. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 432–461. Springer, Heidelberg, August 2019. doi:10.1007/978-3-030-26954-8_14.

16 Rohit Chatterjee, Xiao Liang, and Omkant Pandey. Improved black-box constructions of composable secure computation. Cryptology ePrint Archive, Report 2020/494, 2020. URL: https://eprint.iacr.org/2020/494.

17 Danny Dolev, Cynthia Dwork, and Moni Naor. Non-malleable cryptography (extended abstract). In *23rd ACM STOC*, pages 542–552. ACM Press, May 1991. doi:10.1145/103418.103474.

18 Cynthia Dwork, Moni Naor, and Amit Sahai. Concurrent zero-knowledge. In *30th ACM STOC*, pages 409–418. ACM Press, May 1998. doi:10.1145/276698.276853.

19 Uriel Feige and Adi Shamir. Witness indistinguishable and witness hiding protocols. In *22nd ACM STOC*, pages 416–426. ACM Press, May 1990. doi:10.1145/100216.100272.

20 Sanjam Garg, Vipul Goyal, Abhishek Jain, and Amit Sahai. Bringing people of different beliefs together to do UC. In Yuval Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 311–328. Springer, Heidelberg, March 2011. doi:10.1007/978-3-642-19571-6_19.

21 Sanjam Garg, Vipul Goyal, Abhishek Jain, and Amit Sahai. Concurrently secure computation in constant rounds. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 99–116. Springer, Heidelberg, April 2012. doi:10.1007/978-3-642-29011-4_8.

22 Sanjam Garg, Susumu Kiyoshima, and Omkant Pandey. A new approach to black-box concurrent secure computation. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 566–599. Springer, Heidelberg, 2018. doi:10.1007/978-3-319-78375-8_19.

23 Rosario Gennaro, Anna Lysyanskaya, Tal Malkin, Silvio Micali, and Tal Rabin. Algorithmic tamper-proof (ATP) security: Theoretical foundations for security against hardware tampering. In Moni Naor, editor, *TCC 2004*, volume 2951 of *LNCS*, pages 258–277. Springer, Heidelberg, February 2004. doi:10.1007/978-3-540-24638-1_15.

24 Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987. doi:10.1145/28395.28420.

25 Shafi Goldwasser and Leonid A. Levin. Fair computation of general functions in presence of immoral majority. In Alfred J. Menezes and Scott A. Vanstone, editors, *CRYPTO'90*, volume 537 of *LNCS*, pages 77–93. Springer, Heidelberg, August 1991. doi:10.1007/3-540-38424-3_6.

26 Vipul Goyal. Constant round non-malleable protocols using one way functions. In Lance Fortnow and Salil P. Vadhan, editors, *43rd ACM STOC*, pages 695–704. ACM Press, June 2011. doi:10.1145/1993636.1993729.

27 Vipul Goyal, Divya Gupta, and Abhishek Jain. What information is leaked under concurrent composition? In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 220–238. Springer, Heidelberg, August 2013. doi:10.1007/978-3-642-40084-1_13.

28 Vipul Goyal, Yuval Ishai, Amit Sahai, Ramarathnam Venkatesan, and Akshay Wadia. Founding cryptography on tamper-proof hardware tokens. In Daniele Micciancio, editor, *TCC 2010*, volume 5978 of *LNCS*, pages 308–326. Springer, Heidelberg, February 2010. doi:10.1007/978-3-642-11799-2_19.

29 Vipul Goyal and Abhishek Jain. On concurrently secure computation in the multiple ideal query model. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 684–701. Springer, Heidelberg, May 2013. doi:10.1007/978-3-642-38348-9_40.

30 Vipul Goyal, Abhishek Jain, and Rafail Ostrovsky. Password-authenticated session-key generation on the internet in the plain model. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 277–294. Springer, Heidelberg, August 2010. doi:10.1007/978-3-642-14623-7_15.

31 Vipul Goyal, Chen-Kuei Lee, Rafail Ostrovsky, and Ivan Visconti. Constructing non-malleable commitments: A black-box approach. In *53rd FOCS*, pages 51–60. IEEE Computer Society Press, October 2012. doi:10.1109/FOCS.2012.47.

32 Vipul Goyal, Huijia Lin, Omkant Pandey, Rafael Pass, and Amit Sahai. Round-efficient concurrently composable secure computation via a robust extraction lemma. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part I*, volume 9014 of *LNCS*, pages 260–289. Springer, Heidelberg, March 2015. doi:10.1007/978-3-662-46494-6_12.

33 Vipul Goyal, Rafail Ostrovsky, Alessandra Scafuro, and Ivan Visconti. Black-box non-black-box zero knowledge. In David B. Shmoys, editor, *46th ACM STOC*, pages 515–524. ACM Press, 2014. doi:10.1145/2591796.2591879.

34 Vipul Goyal, Omkant Pandey, and Silas Richelson. Textbook non-malleable commitments. In Daniel Wichs and Yishay Mansour, editors, *48th ACM STOC*, pages 1128–1141. ACM Press, June 2016. doi:10.1145/2897518.2897657.

35 Jens Groth and Rafail Ostrovsky. Cryptography in the multi-string model. In Alfred Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 323–341. Springer, Heidelberg, August 2007. doi:10.1007/978-3-540-74143-5_18.

36 Iftach Haitner. Semi-honest to malicious oblivious transfer – the black-box way. In Ran Canetti, editor, *TCC 2008*, volume 4948 of *LNCS*, pages 412–426. Springer, Heidelberg, March 2008. doi:10.1007/978-3-540-78524-8_23.

37 Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Actively secure garbled circuits with constant communication overhead in the plain model. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part II*, volume 10678 of *LNCS*, pages 3–39. Springer, Heidelberg, November 2017. doi:10.1007/978-3-319-70503-3_1.

38 Carmit Hazay, Antigoni Polychroniadou, and Muthuramakrishnan Venkitasubramaniam. Composable security in the tamper-proof hardware model under minimal complexity. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part I*, volume 9985 of *LNCS*, pages 367–399. Springer, Heidelberg, 2016. doi:10.1007/978-3-662-53641-4_15.

39 Carmit Hazay and Muthuramakrishnan Venkitasubramaniam. On black-box complexity of universally composable security in the CRS model. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part II*, volume 9453 of *LNCS*, pages 183–209. Springer, Heidelberg, 2015. doi:10.1007/978-3-662-48800-3_8.

40 Carmit Hazay and Muthuramakrishnan Venkitasubramaniam. Composable adaptive secure protocols without setup under polytime assumptions. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part I*, volume 9985 of *LNCS*, pages 400–432. Springer, Heidelberg, 2016. doi:10.1007/978-3-662-53641-4_16.

41 Yuval Ishai, Eyal Kushilevitz, Yehuda Lindell, and Erez Petrank. Black-box constructions for secure computation. In Jon M. Kleinberg, editor, *38th ACM STOC*, pages 99–108. ACM Press, May 2006. doi:10.1145/1132516.1132531.

42 Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In David S. Johnson and Uriel Feige, editors, *39th ACM STOC*, pages 21–30. ACM Press, June 2007. doi:10.1145/1250790.1250794.

43 Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer – efficiently. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 572–591. Springer, Heidelberg, August 2008. doi:10.1007/978-3-540-85174-5_32.

44 Abhishek Jain and Omkant Pandey. Non-malleable zero knowledge: Black-box constructions and definitional relationships. In Michel Abdalla and Roberto De Prisco, editors, *SCN 14*, volume 8642 of *LNCS*, pages 435–454. Springer, Heidelberg, September 2014. doi:10.1007/978-3-319-10879-7_25.

45 Yael Tauman Kalai, Yehuda Lindell, and Manoj Prabhakaran. Concurrent general composition of secure protocols in the timing model. In *STOC*, pages 644–653, 2005.

46 Jonathan Katz. Universally composable multi-party computation using tamper-proof hardware. In Moni Naor, editor, *EUROCRYPT 2007*, volume 4515 of *LNCS*, pages 115–128. Springer, Heidelberg, May 2007. doi:10.1007/978-3-540-72540-4_7.

47 Jonathan Katz and Rafail Ostrovsky. Round-optimal secure two-party computation. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 335–354. Springer, Heidelberg, August 2004. doi:10.1007/978-3-540-28628-8_21.

48 Dakshita Khurana, Rafail Ostrovsky, and Akshayaram Srinivasan. Round optimal black-box "commit-and-prove". In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018, Part I*, volume 11239 of *LNCS*, pages 286–313. Springer, Heidelberg, November 2018. doi:10.1007/978-3-030-03807-6_11.

49 Joe Kilian. Founding cryptography on oblivious transfer. In *20th ACM STOC*, pages 20–31. ACM Press, May 1988. doi:10.1145/62212.62215.

50 Susumu Kiyoshima. Round-efficient black-box construction of composable multi-party computation. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 351–368. Springer, Heidelberg, August 2014. doi:10.1007/978-3-662-44381-1_20.

51 Susumu Kiyoshima, Huijia Lin, and Muthuramakrishnan Venkitasubramaniam. A unified approach to constructing black-box UC protocols in trusted setup models. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 776–809. Springer, Heidelberg, November 2017. doi:10.1007/978-3-319-70500-2_26.

52 Susumu Kiyoshima, Yoshifumi Manabe, and Tatsuaki Okamoto. Constant-round black-box construction of composable multi-party computation protocol. In Yehuda Lindell, editor, *TCC 2014*, volume 8349 of *LNCS*, pages 343–367. Springer, Heidelberg, February 2014. doi:10.1007/978-3-642-54242-8_15.

53 Huijia Lin and Rafael Pass. Non-malleability amplification. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 189–198. ACM Press, 2009. doi:10.1145/1536414.1536442.

54 Huijia Lin and Rafael Pass. Concurrent non-malleable zero knowledge with adaptive inputs. In Yuval Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 274–292. Springer, Heidelberg, March 2011. doi:10.1007/978-3-642-19571-6_17.

55 Huijia Lin and Rafael Pass. Black-box constructions of composable protocols without set-up. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 461–478. Springer, Heidelberg, August 2012. doi:10.1007/978-3-642-32009-5_27.

56 Huijia Lin, Rafael Pass, Wei-Lung Dustin Tseng, and Muthuramakrishnan Venkitasubramaniam. Concurrent non-malleable zero knowledge proofs. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 429–446. Springer, Heidelberg, August 2010. doi:10.1007/978-3-642-14623-7_23.

57 Huijia Lin, Rafael Pass, and Muthuramakrishnan Venkitasubramaniam. Concurrent non-malleable commitments from any one-way function. In Ran Canetti, editor, *TCC 2008*, volume 4948 of *LNCS*, pages 571–588. Springer, Heidelberg, March 2008. doi:10.1007/978-3-540-78524-8_31.

58 Huijia Lin, Rafael Pass, and Muthuramakrishnan Venkitasubramaniam. A unified framework for concurrent security: universal composability from stand-alone non-malleability. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 179–188. ACM Press, 2009. doi:10.1145/1536414.1536441.

59 Yehuda Lindell. Bounded-concurrent secure two-party computation without setup assumptions. In *35th ACM STOC*, pages 683–692. ACM Press, June 2003. doi:10.1145/780542.780641.

60 Yehuda Lindell. Lower bounds for concurrent self composition. In Moni Naor, editor, *TCC 2004*, volume 2951 of *LNCS*, pages 203–222. Springer, Heidelberg, February 2004. doi:10.1007/978-3-540-24638-1_12.

61 Yehuda Lindell. A note on constant-round zero-knowledge proofs of knowledge. *Journal of Cryptology*, 26(4):638–654, October 2013. doi:10.1007/s00145-012-9132-7.

62 Tal Malkin, Ryan Moriarty, and Nikolai Yakovenko. Generalized environmental security from number theoretic assumptions. In Shai Halevi and Tal Rabin, editors, *TCC 2006*, volume 3876 of *LNCS*, pages 343–359. Springer, Heidelberg, March 2006. doi:10.1007/11681878_18.

63 Silvio Micali, Rafael Pass, and Alon Rosen. Input-indistinguishable computation. In *47th FOCS*, pages 367–378. IEEE Computer Society Press, October 2006. doi:10.1109/FOCS.2006.43.

64 Silvio Micali and Leonid Reyzin. Physically observable cryptography (extended abstract). In Moni Naor, editor, *TCC 2004*, volume 2951 of *LNCS*, pages 278–296. Springer, Heidelberg, February 2004. doi:10.1007/978-3-540-24638-1_16.

65 Silvio Micali and Phillip Rogaway. Secure computation (abstract). In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 392–404. Springer, Heidelberg, August 1992. doi:10.1007/3-540-46766-1_32.

66 Daniele Micciancio, Shien Jin Ong, Amit Sahai, and Salil P. Vadhan. Concurrent zero knowledge without complexity assumptions. In Shai Halevi and Tal Rabin, editors, *TCC 2006*, volume 3876 of *LNCS*, pages 1–20. Springer, Heidelberg, March 2006. doi:10.1007/11681878_1.

67 Rafail Ostrovsky, Omkant Pandey, and Ivan Visconti. Efficiency preserving transformations for concurrent non-malleable zero knowledge. In Daniele Micciancio, editor, *TCC 2010*, volume 5978 of *LNCS*, pages 535–552. Springer, Heidelberg, February 2010. doi:10.1007/978-3-642-11799-2_32.

68 Rafail Ostrovsky, Silas Richelson, and Alessandra Scafuro. Round-optimal black-box two-party computation. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 339–358. Springer, Heidelberg, August 2015. doi:10.1007/978-3-662-48000-7_17.

69 Rafael Pass. Simulation in quasi-polynomial time, and its application to protocol composition. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 160–176. Springer, Heidelberg, May 2003. doi:10.1007/3-540-39200-9_10.

70 Rafael Pass. Bounded-concurrent secure multi-party computation with a dishonest majority. In László Babai, editor, *36th ACM STOC*, pages 232–241. ACM Press, June 2004. doi:10.1145/1007352.1007393.

71 Rafael Pass, Huijia Lin, and Muthuramakrishnan Venkitasubramaniam. A unified framework for UC from only OT. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 699–717. Springer, Heidelberg, December 2012. doi:10.1007/978-3-642-34961-4_42.

72 Rafael Pass and Hoeteck Wee. Black-box constructions of two-party protocols from one-way functions. In Omer Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 403–418. Springer, Heidelberg, March 2009. doi:10.1007/978-3-642-00457-5_24.

73 Birgit Pfitzmann and Michael Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *2001 IEEE Symposium on Security and Privacy*, pages 184–200. IEEE Computer Society Press, May 2001. doi:10.1109/SECPRI.2001.924298.

74 Manoj Prabhakaran, Alon Rosen, and Amit Sahai. Concurrent zero knowledge with logarithmic round-complexity. In *43rd FOCS*, pages 366–375. IEEE Computer Society Press, November 2002. doi:10.1109/SFCS.2002.1181961.

75 Manoj Prabhakaran and Amit Sahai. New notions of security: Achieving universal composability without trusted setup. In László Babai, editor, *36th ACM STOC*, pages 242–251. ACM Press, June 2004. doi:10.1145/1007352.1007394.

76 Ransom Richardson and Joe Kilian. On the concurrent composition of zero-knowledge proofs. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 415–431. Springer, Heidelberg, May 1999. doi:10.1007/3-540-48910-X_29.

77 Alon Rosen. A note on constant-round zero-knowledge proofs for NP. In Moni Naor, editor, *TCC 2004*, volume 2951 of *LNCS*, pages 191–202. Springer, Heidelberg, February 2004. doi:10.1007/978-3-540-24638-1_11.

78 Hoeteck Wee. Black-box, round-efficient secure computation via non-malleability amplification. In *51st FOCS*, pages 531–540. IEEE Computer Society Press, October 2010. doi:10.1109/FOCS.2010.87.

79 Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986. doi:10.1109/SFCS.1986.25.

Simplifying and Unifying Replacement Paths Algorithms in Weighted Directed Graphs

Shiri Chechik
Blavatnik School of Computer Science, Tel Aviv University, Israel
shiri.chechik@gmail.com

Moran Nechushtan
Blavatnik School of Computer Science, Tel Aviv University, Israel
morann@mail.tau.ac.il

Abstract

In the replacement paths (RP) problem we are given a graph G and a shortest path P between two nodes s and t [1]. The goal is to find for every edge $e \in P$, a shortest path from s to t that avoids e. The first result of this paper is a simple reduction from the RP problem to the problem of computing shortest cycles for all nodes on a shortest path.

Using this simple reduction we unify and extremely simplify two state of the art solutions for two different well-studied variants of the RP problem.

In the first variant (*algebraic*) we show that by using at most n queries to the Yuster-Zwick distance oracle [FOCS 2005], one can solve the the RP problem for a given directed graph with integer edge weights in the range $[-M, M]$ in $\tilde{O}\left(Mn^{\omega}\right)$ time [2] [3] . This improves the running time of the state of the art algorithm of Vassilevska Williams [SODA 2011] by a factor of $\log^6 n$.

In the second variant (*planar*) we show that by using the algorithm of Klein for the multiple-source shortest paths problem ($MSSP$) [SODA 2005] one can solve the RP problem for directed planar graph with non negative edge weights in $O\left(n \log n\right)$ time. This matches the state of the art algorithm of Wulff-Nilsen [SODA 2010], but with arguably much simpler algorithm and analysis.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Theory of computation → Shortest paths

Keywords and phrases Fault tolerance, Distance oracle, Planar graph

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.29

Category Track A: Algorithms, Complexity and Games

Funding This publication is part of a project that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 803118 UncertainENV).

Acknowledgements This publication is part of a project that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 803118 UncertainENV)

1 Introduction

The replacement paths problem is defined as follows. We are given a graph G and a shortest path P connecting two nodes s and t. The problem is to find for every edge $e \in P$ a shortest path from s to t in the graph $(V, E \backslash e)$. As in previous works, we focus on computing the lengths of the replacement paths instead of the paths itself. Our results can be modified to return the paths as well.

[1] For the rest of this paper we assume $G = (V, E)$, $n = |V|$, $m = |E|$.
[2] ω denotes the matrix multiplication exponent ($\omega < 2.373$).
[3] The $\tilde{O}$ notation suppresses polylogarithmic factors.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 29; pp. 29:1–29:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The motivation for studying replacement paths is rooted in the fact that real-world graphs are vulnerable and are subject to nodes and links failures and having backup paths is a desirable property. Replacement paths are also well motivated by other important applications. One application stems from auction theory, where we use replacement paths to compute the Vickrey pricing of edges owned by selfish agents [15, 7]. In addition, Replacement paths are also used to compute the k shortest simple paths between two given vertices, as one can compute the k shortest simple paths by k calls to the replacement paths algorithm. The k shortest simple paths has many applications in and of itself [4].

1.1 Upper bounds

For undirected graphs with arbitrary (non negative) edge weights, Malik et al. [12] gave an $O(m+n\log n)$ algorithm; Nardelli et al. [14] later improve the running time to $O(m\alpha(m,n))$[4] in the Word RAM model for positive edge weights. For directed graphs with arbitrary edge weights (possibly negative), Gotthilf and Lewenstein [5] gave an $O(mn+n^2\log\log n)$ algorithm. For unweighted directed graphs, Roditty and Zwick [16] gave a randomized combinatorial algorithm that solves the problem in $\tilde{O}(m\sqrt{n})$ time. For directed graphs with integer edge weights in the range $[-M,M]$, Weimann and Yuster [17] obtain a randomized algorithm that runs in $\tilde{O}(Mn^{1+\frac{2}{3}\omega})$ time. Vassilevska Williams [18] improved the latter by giving a randomized algorithm that runs in $\tilde{O}(Mn^{\omega})$ time for $w>2$ and in $O(Mn^{\omega+\epsilon})$ time for any $\epsilon>0$ for $\omega=2$. The replacement paths problem was also studied for special family of graphs. For planar directed graphs with non negative edge weights, Emek, Peleg, and Roditty [3] obtain a recursive algorithm that runs in $O(n\log^3 n)$ time. Klein, Mozes, and Weimann [10] improve the running time to $O(n\log^2 n)$. At last, Wulff-Nilsen [20] improved the latter by giving an $O(n\log n)$ time algorithm.

1.2 Lower bounds

Hershberger et al. [8] showed a $\Omega(m\sqrt{n})$ time lower bound for the replacement paths problem for directed graphs with non negative edge weights, in the $path-comparison$ model of Karger et al [9]. Vassilevska-Williams and Williams [19] showed that the replacement paths problem in directed graphs with arbitrary edge weights is equivalent to the all pairs shortest paths problem (APSP), under subcubic reductions. Agarwal and Ramachandran [1] showed that the All-Nodes Shortest Cycles problem (ANSC) for directed graph with arbitrary edge weights, in which we are required to find for every node a shortest cycle containing it, is at least as hard as computing the replacement paths problem in directed graphs. This reduction can be used to solve the replacement paths problem using an oracle to the ANSC problem. However, the reduction does not preserve the range of the weights. That is, the reduction they present increases the weights of the graph by a factor of n. This means that the reduction is not applicable in the algebraic variant, as it would lead to a $\tilde{O}\left((Mn)\,n^{\omega}\right)$ solution. Moreover, the reduction does not preserve planarity, and therefore is not applicable in the planar variant as well.

1.3 Our result

The first result of this paper is a simple linear time reduction from the RP problem to the problem of computing shortest cycles for all nodes on a shortest path. The reduction maps the graph G into a new graph G^r with the following properties; The mapping from

[4] $\alpha(m,n)$ is the functional inverse of the Ackermann function.

G to G^r is invertible and is the inverse of itself. i.e. $(G^r)^r = G$. The reduction preserves the range of the edge weights i.e. if G has integer weights in the range $[-M, M]$ then so does G^r. The reduction preserves planarity i.e. if G is planar then so does G^r. The first property implies that the problem of computing shortest cycles for all nodes on a shortest path is equivalent to the RP problem. The second property enable us to solve the RP problem for directed graphs with integer edge weights in the range $[-M, M]$ in $O(Preprocess_{\mathcal{A}}(n, m, M) + n \cdot Query_{\mathcal{A}}(n, m, M))$ time, where $\mathcal{A}$ is a distance oracle. In particular, using Yuster-Zwick distance oracle, we solve the problem in deterministic $\tilde{O}(Mn^{\omega})$ time. Vassilevska Williams solved the problem using few techniques such as; Node sampling, graph compression and recursion. The running time of Vassilevska Williams's algorithm is at least $\log^{\frac{\omega}{\omega-2}}(n) \cdot Preprocess_{\mathcal{A}}(n, m, M)$. Therefore, when considering polylogarithmic factors, our algorithm improves the running time of Vassilevska Williams's algorithm by a factor of $\log^{\frac{\omega}{\omega-2}}(n)$. Hence, for $\omega < 2.373$ we improve the running time by at least a factor of $\log^6(n)$. The third property enable us to solve the RP problem for directed planar graphs with non negative edge weights using an oracle to the $MSSP$ problem. In particular, using Klein algorithm for the $MSSP$ problem, we solve the problem in $O(n \log n)$ time. The $MSSP$ problem is as follows; Given a planar graph G, and a list of distance queries from a source to a target, such that all sources share the same face, answer all distance queries. In fact, our reduction is to a simpler problem than the $MSSP$ problem, that is, computing at most $O(n)$ distance queries where both the source and the target are on the given face. Moreover, the boundary of the face consists of two shortest paths. The fact that our reduction is to a simpler problem than the $MSSP$ problem gives additional insights on the RP problem that might lead to further improvements.

2 Preliminaries

2.1 General notations

Let $G = (V, E)$ be a directed graph with n vertices and m edges, and let w be a weight function over E.

Let P be a path in G. We denote by $w(P)$ the length of the path P which is defined as the sum of the weights of the edges along P, and by $|P|$ the number of edges in P. We assume G does not contain negative cycles, hence the distance between every two nodes $u, v \in V$ is well defined and denoted by $d(u, v)$.

Let $e \in E$, we denote by $G \backslash e$ the graph $(V, E \backslash \{e\})$, and by $d_e(u, v)$ the distance from u to v in the graph $G \backslash e$.

Path concatenation: Given two paths $\Omega_1 = \langle u, ..., v\rangle$ $\Omega_2 = \langle v, ..., w\rangle$ we denote by $\Omega_1 \cdot \Omega_2$ the concatenation of Ω_1 and Ω_2.

Path slicing: Given a path $\Omega = \langle x_0, x_1, ..., x_k\rangle$, we denote by $\Omega[x_i, ..., x_j]$ for $i \leq j$ the subpath of Ω connecting x_i to x_j.

We denote the set $\{0, 1, ..., N\}$ by $[N]$.

2.2 Distance oracle

▶ **Definition 1.** Distance product - Let A be an $m \times n$ matrix and B an $n \times p$ matrix, where A and B have entries from $\mathbb{Z} \cup \{\infty\}$. Then their distance product $A \star B$ is an $m \times p$ matrix defined as

$$(A \star B)_{ij} = \underset{k \in [n]}{min} A_{ik} + B_{kj}$$

▶ **Theorem 2** (Alon, Galil and Margalit [2]). *One can compute the distance product of two $n \times n$ matrices with entries in $[-M, M]$ in $\tilde{O}(Mn^\omega)$ time.*

▶ **Theorem 3** (Yuster-Zwick [21]). *Given a directed graph G with edge weights in $[-M, ..., M]$, one can compute in $\tilde{O}(Mn^\omega)$ time a $n \times n$ matrix A, so that the (i, j) entry of the distance product $A \star A$ is the distance between nodes i and j in G.*

By Theorem 3, there exists a distance oracle $\mathcal{A}$ with the following preprocessing and query time; $Preprocess_{\mathcal{A}}(n, m, M) = \tilde{O}(Mn^\omega)$ and $Query_{\mathcal{A}}(n, m, M) = O(n)$. At the preprocess step, we compute the matrix A, and we answer a distance query between two nodes, i and j, by computing the (i, j) entry of $A \star A$.

2.3 Replacement path

Let G be a weighted directed graph, and let $P = \langle v_0, ..., v_k\rangle$ be a shortest path in G from $s = v_0$ to $t = v_k$.

We think of the path $P = \langle v_0, .., v_k\rangle$ as going from left to right, so for $v_i, v_j \in P$ whenever we say v_i is to the left (respectively right) of v_j we mean $i \leq j$ (respectively $i \geq j$).

▶ **Definition 4.** Let D be simple path connecting v_i to v_j for $i < j$. We call D a detour from v_i to v_j for the path P, if $D \cap P = \{v_i, v_j\}$ and $E(D) \cap E(P) = \emptyset$ (note that we need the second requirement for the case that $j = i + 1$).

Let D be a detour for the path P, connecting v_i to v_j. We denote by $Prefix(P, D)$ the path $P[v_0, ..., v_i]$, by $Suffix(P, D)$ the path $P[v_j, ..., v_k]$, and by $S(P, D)$ (S for segment) the path $P[v_i, ..., v_j]$. We say the detour D is skipping $e = (u, v)$ if v_i is to the left of u and v_j is to the right of v.

When P is clear from the context we abbreviate $Prefix(P, D)$ and simply write $Prefix(D)$ instead, and similarly for $Suffix(D)$ and $S(D)$.

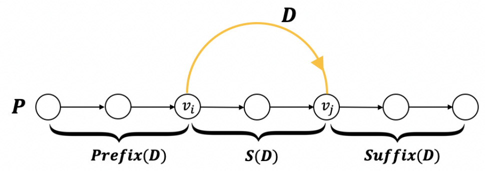


Figure 1 Partition of the path P, with respect to a detour D, into $Prefix(D)$, $S(D)$ and $Suffix(D)$.

The following Lemma is folklore.

▶ **Lemma 5.** *Let P be a shortest path in G connecting two nodes $s, t \in V$, and let $e \in P$. Suppose t is reachable from s in $G \backslash e$, then there exists a detour D skipping e s.t. $Prefix(D) \cdot D \cdot Suffix(D)$ is a shortest path from s to t in $G \backslash e$.*

In the next section we present the idea of reducing the RP problem to the problem of computing shortest cycles for all nodes on a shortest path. The results of this section are applicable for both the algebraic variant and the planar variant as well. In Section 4 we focus on directed graphs with integer weights in the range $[-M, M]$, and in section 5 we focus on directed planar graphs with non negative weights.

3 Replacement paths in weighted directed graphs

Let $s, t \in V$, and $P = \langle v_0 = s, v_1, ..., v_k = t\rangle$ be a shortest path connecting s to t in G.

We first start with two simple observations.

Observation 1. Let D be a detour of P. If we flip the orientation of P, i.e. replace every edge $e = (u, v)$ in P, with a new edge $e^r = (v, u)$, then in the new graph, the concatenation of D with $S(D)$ forms a directed cycle.

Observation 2. Let $e \in P$, and let D be a detour skipping e. D minimizes $w(Prefix(D)) + w(D) + w(Suffix(D))$ if and only if it minimizes $w(D) - w(S(D))$. This holds, as $P = Prefix(D) \cdot S(D) \cdot Suffix(D)$.

These two simple observations lead us to consider a new graph where we flip the orientation of P, and flip the sign of the weights of P. As we prove later, the minimal cycle containing e in the new graph, corresponds to some detour D skipping e, that minimize the expression $w(D) - w(S(D))$. The fact that the weight of the minimal cycle containing e is not smaller than what we are actually looking for $(\underset{D\,skip\,e}{min}\ w(D) - w(S(D)))$, requires some details.

▶ **Definition 6.** Let $Q = \langle u_0, u_1, .., u_N\rangle$ be a path in G, we denote by Q^r the path $\langle u_N, ..., u_1, u_0\rangle$ with new weights $w^r(u_{i+1}, u_i) = -w(u_i, u_{i+1})$.

In other words, we flip the orientation of Q and flip the sign of it's weights.

▶ **Definition 7.** (See Figure 2 for illustration) Denote by $G^r = (V^r, E^r)$ the following graph:

$V^r = V$

$E^r = E \cup E(P^r) \backslash E(P)$

and denote by $G^+ = (V^+, E^+)$ the following graph:

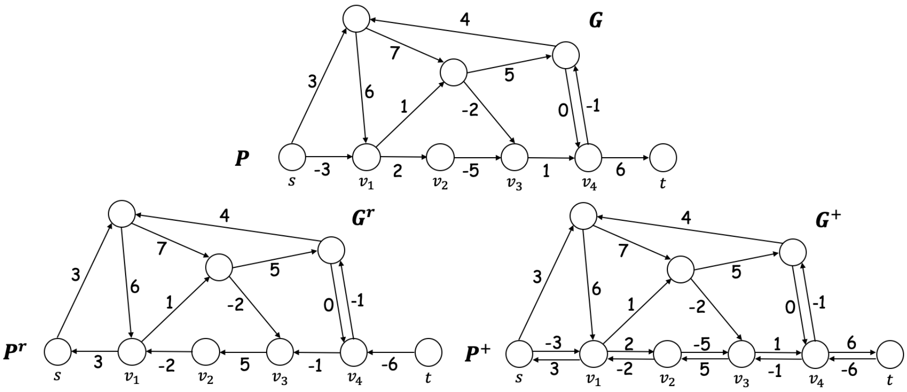
$V^+ = V$

$E^+ = E \cup E(P^r)$

Finally, let $P^+ = P \cup P^r$

▶ Remark 8. If there were already edges in the opposite direction of P then we disregard them. Notice $w(v, u) \geq -w(u, v)$ (as otherwise we would have a negative cycle), hence after adding an edge (v, u) with weight of $-w(u, v)$ there is no reason to keep the original weight of (v, u) when considering shortest paths.

In the following we show that G^+ does not contain negative cycles.



■ **Figure 2** An example of transformation of a graph G into G^r and G^+.

▶ **Lemma 9.** *Consider the graph G^+, and let $u, v \in P^+$. The simple path from u to v that lies in P^+ is a shortest path (notice u can be to the left of v, to the right of v, or even equal to v).*

Proof. We first show that there is a shortest path (not necessarily simple) from u to v that is fully contained in P^+. Let $P' = \langle x_0 = u, ..., x_N = v\rangle$ be a shortest path connecting u to v in G^+. The proof is by induction on the number of edges of P'. The base case, $|P'| = 0$ is clear, so we assume $|P'| \geq 1$. If the first edge of P' belongs to P^+, we can easily continue by induction over $P'[x_1, .., x_N]$ so we assume otherwise. Let $x_i \in P'$ be the first vertex except x_0 itself that belongs to P (x_i might be equal to x_N). We partition P' into two paths, $P'_1 = P'[x_1, .., x_i]$ $P'_2 = P'[x_i, .., x_N]$. Notice that the edges of P'_1 are not in P^+ (hence the edges of P'_1 belong to E). We separate to two cases:

1. x_i is to the right of x_0. In that case we replace P'_1 with the path $P_1 \coloneqq P[x_0, ..., x_i]$.

   $w(P'_1) \geq d(x_0, x_i) = w(P[x_0, ..., x_i])$, where the first inequality follows as P'_1 is edge disjoint from P^+ and therefore is contained in G.

2. x_i is to the left of x_0. In that case we replace P'_1 with the path $P_1 \coloneqq P^r[x_0, ..., x_i]$.

   Notice $P[x_i, ..., x_0] \cdot P'_1$ is a cycle in G hence its weight is non negative, so we have the following:
   $w(P'_1) + w(P[x_i, ..., x_0]) \geq 0 \implies w(P'_1) \geq -w(P[x_i, ..., x_0]) = w^r(P^r[x_0, ..., x_i])$.

Therefore we can replace P'_1 with P_1 without increasing the total weight of P'. By the induction hypothesis we can replace P'_2 with a path P_2 that lies in P^+ without increasing the total weight of P'. The concatenation of the paths P_1 and P_2 is a path from u to v that lies entirely in P^+ and it's length is at most $w(P')$.

Finally, notice we can assume the shortest path from u to v that lies in P^+, lies entirely in P or entirely in P^r, depending if u is to the left or to the right of v (there is no reason to go left and right over the same edge as the weights will cancel each other). ◀

▶ **Corollary 10.** *The graph G^+ does not contain negative cycles (hence G^r as well).*

Proof. Let C be a cycle in G^+. If C does not contain any vertex in P we are done (since the cycle C exists in G, and G does not contain negative cycles), so we assume otherwise. Let $v \in C \cap P$. Note that C is a path from v to itself. By Lemma 9 we can replace C with a simple path in P^+ from v to itself without increasing the cycle weight, that path is simply $\langle v\rangle$ with zero weight. ◀

▶ **Corollary 11.** *The path P^r is a shortest path in the graph G^r.*

Proof. By Lemma 9, P^r is a shortest path in G^+, hence it's also a shortest path in G^r (G^r is a subgraph of G^+ that contains P^r) ◀

▶ **Lemma 12.** *Let $e = (u, v) \in P$. Suppose that v is reachable from u in G^r, then there exists a shortest cycle in G^r containing e^r of the form $D \cdot S(D)^r$ where D is a detour skipping e.*

Proof. Let $C = \langle x_0, ..., x_N\rangle$ be a minimal cycle in G^r containing e^r s.t. $x_0 = v, x_1 = u$ and $x_N = v$.

Let $l \in [N]$ be the largest index s.t. $x_l \in P$ and x_l is to the left of u in P.

Let $r \in \{l+1, ..., N\}$ (i.e. r is greater than l) be the smallest index s.t. $x_r \in P$ (notice it implies that x_r is to the right of v in P). By Corollary 11 we can replace $C[x_0, ..., x_l]$ with $P^r[v, ..., x_l]$ and $C[x_r, ..., v]$ with $P^r[x_r, ..., v]$ without increasing the weight of the cycle. The middle part $D \coloneqq C[x_l, ..., x_r]$ is a detour skipping e, and $P^r[x_r, ..., x_l] = S(D)^r$. Hence $D \cdot S(D)^r$ is a cycle containing e^r with weight of at most $w^r(C)$. ◀

Notice the similarity of Lemma 5 and Lemma 12.

▶ **Definition 13.** Let $e \in P$. We denote by C_e the shortest cycle in G^r containing e^r. Moreover, by Lemma 12 we may assume that C_e is of the form $D \cdot S(D)^r$ for some detour D skipping e. If e^r is not contained in any cycle in G^r, then C_e is defined as having weight infinity (we define this to ease notation).

We denote by d^r the distance function over G^r.

▶ Remark 14. For any edge $e = (u, v) \in P$, $w^r(C_e) = d^r(u, v) - w(u, v)$.

▶ **Theorem 15.** *For any edge $e = (u, v) \in P$, $d_e(s, t) = w(P) + w^r(C_e)$ $(= w(P) - w(u, v) + d^r(u, v))$.*

Proof. From Lemma 5 it follows that $d_e(s,t) = \underset{D\,skip\,e}{min}\ w(Prefix(D)) + w(D) + w(Suffix(D))$ (where the right expression equals to ∞ if there is no such detour D).

From Lemma 12 it follows that $\underset{D\,skip\,e}{min}\ w(D) - w(S(D)) = \underset{C\,contain\,e^r}{min}\ w^r(C)$.

Hence in total we have:

$d_e(s,t) = \underset{D\,skip\,e}{min}\ w(Prefix(D)) + w(D) + w(Suffix(D)) = w(P) + \underset{D\,skip\,e}{min}\ w(D) - w(S(D)) = w(P) + \underset{C\,contain\,e^r}{min}\ w^r(C) = w(P) + w^r(C_e)$ ◀

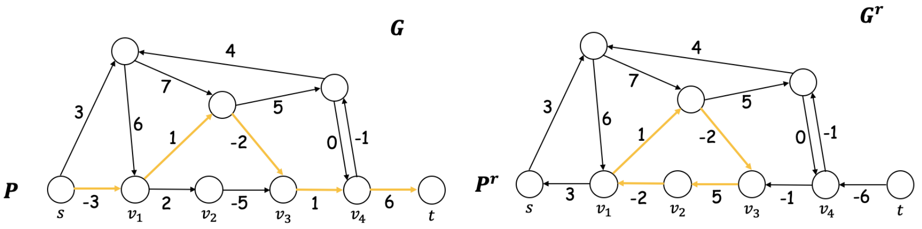


Figure 3 An example of a shortest path in $G \backslash (v_1, v_2)$ and the corresponding minimal cycle in G^r.

4 Replacement paths in graphs with integer weights

In this section, we aim to solve the replacement paths problem for directed graphs with integer weights in $[-M, M]$. Let $\mathcal{A}$ be a distance oracle for directed graphs with integer weights in $[-M, M]$. We let $Preprocess_{\mathcal{A}}(n, m, M)$ be the computation time of $\mathcal{A}$ to preprocess the graph and $Query_{\mathcal{A}}(n, m, M)$ be the computation time of $\mathcal{A}$ to answer a distance query. Finally, we denote by $d_{\mathcal{A}}(u, v)$ the answer of the oracle $\mathcal{A}$ to the query of the distance from u to v.

▶ **Theorem 16.** *Let $\mathcal{A}$ be a distance oracle for weighted directed graph with integer weights in $[-M, M]$. Then, there is an algorithm for the replacement paths problem that runs in $O(Preprocess_{\mathcal{A}}(n, m, M) + n \cdot Query_{\mathcal{A}}(n, m, M))$ time.*

Proof. We first compute G^r in linear time. Then we preprocess G^r to compute a distance oracle $\mathcal{A}$ in $Preprocess_{\mathcal{A}}(n, m, M)$ time, and finally for every edge $e = (u, v) \in P$ we compute $d^r(u, v)$ using the oracle $\mathcal{A}$ in $Query_{\mathcal{A}}(n, m, M)$ time, and store $d_e(s, t) \leftarrow d^r(u, v) - w(e) + w(P)$ $(= w^r(C_e) + w(P))$. The number of calls to the distance query is equal to the number of edges in P which is bounded by n. The correctness of the algorithm follows directly from Theorem 15. ◀

▶ **Theorem 17.** *The replacement paths problem for weighted directed graph, with integer weights in $[-M, M]$ can be solved in deterministic $\tilde{O}(Mn^{\omega})$ time.*

Proof. We denote by $\mathcal{A}$ the distance oracle of $Yuster-Zwick$ (recall Theorem 3). $Preprocess_{\mathcal{A}}(n, m, M) = \tilde{O}(Mn^{\omega})$ and $Query_{\mathcal{A}}(n, m, M) = O(n)$.

Hence, by Theorem 16 we can solve the replacement paths problem in total time of $\tilde{O}(Mn^{\omega}) + O(n^2) = \tilde{O}(Mn^{\omega})$. ◀

Algorithm 1 ReplacementPaths.

```
1. Input: $[G, P, s, t]$
2. Compute $G^r$ (for every edge in $P$, flip its orientation and its sign)
3. Compute a distance oracle $\mathcal{A}$ for the graph $G^r$
4. $d_e(s,t) \leftarrow d_{\mathcal{A}}(u,v) - w(e) + w(P)\ \ for\, e = (u,v) \in P$
5. Output: $[d_e(s,t)\ \ for\, e \in P]$
```

5 Replacement paths in planar graphs

In this section we aim to solve the replacement paths problem for directed planar graphs with non negative edge weights.

▶ **Definition 18** ($MSSP$ **problem).** Let G be a directed planar graph with arbitrary weights (possibly negative). Let $U = \{u_1, u_2, .., u_\ell\}$ be a set of nodes sharing the same face, T be a shortest path tree rooted at u_1, and $L \subseteq U \times V$ be a list of k pairs. The $MSSP$ (Multiple-source shortest paths) problem is as follows; Given G, T, L, output $d\,(u, v)$ for all $(u, v) \in L$.

Let $\mathcal{A}$ be an algorithm that solves the $MSSP$ problem. We denote by $T_{\mathcal{A}}(n, k)$ the running time of $\mathcal{A}$ to solve the $MSSP$ problem for a graph with n nodes, and a list of k pairs.

▶ **Theorem 19** (Klein [11]). *There exists an algorithm $\mathcal{A}$ that solves the $MSSP$ problem in time $T_{\mathcal{A}}(n, k) = O\,(n \log n + k \log n)$.*

Let $e = (u, v) \in P$. By Theorem 15, $d_e\,(s, t) = w\,(P) + d^r\,(u, v) - w\,(u, v)$. That is, in order to solve the RP problem we need to compute $d^r\,(u, v)$ for every $(u, v) \in P$. We will do so by invoking the $MSSP$ algorithm. However, there are two slight obstacles with this approach. The first obstacle is that the nodes of P^r are not necessarily belong to the same face. The second obstacle is that even though the $MSSP$ algorithm can handle negative weights, it requires a computation of a shortest path tree. The graph G^r contains negative weights, thus a naive computation of a shortest path tree would be too time consuming, as the best running time for computing $SSSP$ (single source shortest path) with negative weights in planar graphs is $O\left(n \log^2 n / \log\log n\right)$ [13]. We will face the first obstacle using a standard approach of creating a new planar graph, denoted as H_1, such that all nodes of P^r (more precisely a copy of them) belong to the same face in H_1. For tackling the second obstacle, notice that the only edges in G^r with negative weights belong to $E\,(P^r)$. We will see how we can utilize this to compute a shortest path tree rooted at t in time $O\,(n)$ by creating a new planar graph, denoted by H_2.

▶ **Definition 20.** Given three edges $e_1, e_2, e_3 \in E\,(G^r)$ incident with v, we say that e_2 is between e_1 and e_3 if a counterclockwise traversal of the edges incident with v that begins at e_1 reaches e_2 before it reaches e_3.

Let $v_i \in P^r \backslash \{s,t\}$, and let e be an edge incident with v_i such that $e \notin E(P^r)$. We say that e is to the right of P^r at v_i, if e is between the edge (v_{i+1}, v_i) and the edge (v_i, v_{i-1}). Otherwise, we say that e is to the left of P^r.

▶ **Definition 21** (See Figure 4 for illustration)**.** We define a new graph, denoted as H_1, obtained from G^r, with the following modifications;

1. For every node $v_i \in \{v_1, v_2, ..., v_{k-1}\}$, replace v_i with two new nodes v_i^l and v_i^r (l for left and r for right).
   Notice that we leave s and t as they are. To ease notations, whenever we write v_0^l or v_0^r we actually mean v_0 $(= s)$. Similarly whenever we write v_k^l or v_k^r we actually mean v_k $(= t)$.
2. For every $i \in [k-1], x \in \{l, r\}$, add an edge $\left(v_{i+1}^x, v_i^x\right)$ with weight $w^r(v_{i+1}, v_i)$.
3. Let $v \in \{v_1, v_2, ..., v_{k-1}\}$, and let e be an edge incident with v in G^r such that $e \notin E(P^r)$. If e is to the left of P^r at v, we modify e by substitute v with v^l . Otherwise, we modify e by substitute v with v^r.

▶ **Definition 22** (See Figure 4 for illustration)**.** We define a new graph, denoted as H_2, obtained from H_1, with the following modifications;

For every $i \in [k-1], x \in \{l, r\}$, remove the edge $\left(v_{i+1}^x, v_i^x\right)$ and add an edge (t, v_i^x) with weight $d^r(t, v_i)$.

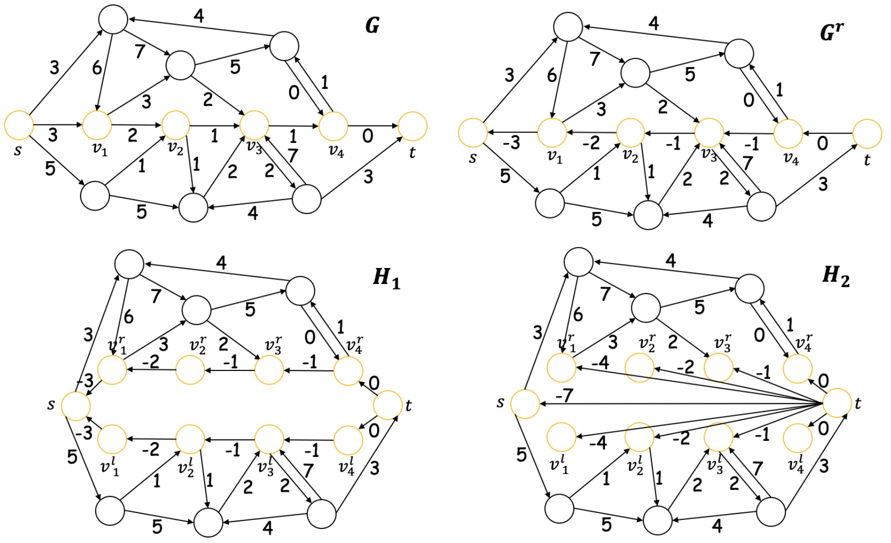


Figure 4 An example of transformation of a planar graph G into G^r,H_1 and H_2.

▶ Remark 23. H_1 and H_2 are planar graphs of size $O(n)$. Moreover, all nodes in the set $\left\{v_0^l, .., v_k^l, v_0^r, .., v_k^r\right\}$ share the same face in H_1.

Let us use the following notation; For every $i \in [k-1], x, y \in \{l, r\}$, $\rho_i^{xy} = d_{H_1}\left(v_i^x, v_{i+1}^y\right)$.

▶ **Lemma 24.** *For every $i \in [k-1]$ we have $d^r(v_i, v_{i+1}) = \min\left\{\rho_i^{ll}, \rho_i^{lr}, \rho_i^{rl}, \rho_i^{rr}\right\}$.*

Proof. Let Q be a path in H^1. We can simulate the path Q in the graph G^r, by replacing every instance of v_i^l and v_i^r with v_i. The resulting path has the same length of Q. In particular this holds for a path from v_i^x to v_{i+1}^y for any choice of $x, y \in \{l, r\}$. Therefore $d^r(v_i, v_{i+1}) \leq \min\left\{\rho_i^{ll}, \rho_i^{lr}, \rho_i^{rl}, \rho_i^{rr}\right\}$. Let P_i be a shortest path from v_i to v_{i+1} in G^r. By Lemma 12 we can assume P_i is of the form $P^r[v_i, ..., u] \cdot D_{uv} \cdot P^r[v, ..., v_{i+1}]$, where

$u \in P^r$ is to the left of v_i, $v \in P^r$ is to the right of v_{i+1}, and D_{uv} is a detour from u to v skipping the edge (v_i, v_{i+1}). The first and last edges of D_{uv} can either be to the left or to the right of P^r. Therefore there are four possible cases for the departure and entrance orientation of D_{uv} with respect to P^r. Let $x \in \{l, r\}$ be the orientation of the first edge of D_{uv}, and $y \in \{l, r\}$ be the orientation of the last edge of D_{uv}. We can simulate P_i in the graph H_1 by replacing all nodes $v_j \in P^r[v_i, ..., u]$ with v_j^x, and all nodes $v_j \in P^r[v, ..., v_{i+1}]$ with v_j^y. The resulting path is a path from v_i^x to v_{i+1}^y, and has the same length of P_i. Therefore $d^r(v_i, v_{i+1}) \geq \rho_i^{xy} \geq \min\left\{\rho_i^{ll}, \rho_i^{lr}, \rho_i^{rl}, \rho_i^{rr}\right\}$. It follows that $d^r(v_i, v_{i+1}) = \min\left\{\rho_i^{ll}, \rho_i^{lr}, \rho_i^{rl}, \rho_i^{rr}\right\}$. ◀

Our current goal is to compute a shortest path tree rooted at t in the graph H_1 in $O(n)$ time.

▶ **Lemma 25.** *For all $v \in V(H_1) (= V(H_2))$, $d_{H_1}(t, v) = d_{H_2}(t, v)$.*

Proof. For the first direction, let us prove $d_{H_1}(t, v) \leq d_{H_2}(t, v)$. Let Q_2 be a shortest path from t to v in H_2. We can simulate the path Q_2 in the graph H_1 as follows; If the first edge of the path Q_2 is of the form (t, v_i^x), then replace that edge with a path from t to v_i^x of length $d^r(t, v_i^x)$. The rest of the simulated path is identical to Q_2. The resulting path has the same length as Q_2. Therefore $d_{H_1}(t, v) \leq w(Q_2) = d_{H_2}(t, v)$. For the second direction, let us prove that $d_{H_2}(t, v) \leq d_{H_1}(t, v)$. Let Q_1 be a shortest path from t to v in H_1. The path $\langle t = v_k^x, ..., v_1^x, v_0^x = s\rangle$ is a shortest path in H_1 for $x \in \{l, r\}$. Let v_i^x be the last node in Q_1 that belongs to the set $\left\{v_k^l, ..., v_1^l, v_0^l, v_k^r, ..., v_1^r, v_0^r\right\}$. We can simulate the path Q_1 in the graph H_2 as follows; Replace the path $Q_1[t, ..., v_i^x]$ with a direct edge (t, v_i^x), and the rest of the simulated path is identical to Q_1. The resulting path is of the same length as Q_1. Therefore $d_{H_2}(t, v) \leq w(Q_1) = d_{H_1}(t, v)$. It follows that $d_{H_1}(t, v) = d_{H_2}(t, v)$. ◀

▶ **Theorem 26** (Henzinger, Klein, Rao and Subramanian [6]). *Let G be a directed planar graph with non negative weights, and let $s \in V(G)$. One can compute a shortest path tree rooted at s in $O(n)$ time.*

Notice that the only edges with negative weights in H_2 are incident with t. Let $c_0 = \min\{d^r(t, v_i) \mid i \in [k-1]\}$. We modify H_2 by increasing the weights of all edges going out from t by $-c_0$. The modified graph is a planar graph with non negative weights. Therefore by Theorem 26, we can compute a shortest path tree rooted at t, denoted as T_2, in the modified graph in $O(n)$ time. T_2 is a shortest path tree for the graph H_2.

Next we show how to modify T_2 to a shortest path tree in the graph H_1.

▶ **Definition 27.** Let $A = \left\{\left(v_{i+1}^x, v_i^x\right) \mid i \in [k-1], x \in \{l, r\}\right\}$. We define a new tree, denoted as T_1, obtained from T_2, with the following modifications;

For every edge $e = (u, v) \in A$, remove the edge $(p(v), v)$ from T_2, and insert the edge (u, v) with weight $d^r(u, v)$ to T_2, where $p(v)$ is the parent of v in T_2.

Note that for every node v_i^x for $x \in \{l, r\}$, the shortest path from t to v_i^x in T_1 is of the same length as in T_2. By Lemma 25, we conclude that T_1 is a shortest path tree in H_1.

▶ **Corollary 28.** *One can compute a shortest path tree T_1 rooted at t in H_1 in $O(n)$ time.*

▶ **Theorem 29.** *Let G be a directed planar graph with non negative weights. Let $\mathcal{A}$ be an algorithm that solves the MSSP problem on a graph with n nodes, and a list of k pairs, in time $T_{\mathcal{A}}(n, k)$. Then, there is an algorithm that solves the replacement paths problem on G, that runs in $O(T_{\mathcal{A}}(2n, 4n))$ time.*

Proof. By Corollary 28, we can compute H_1 and T_1 in $O(n)$ time.

Let $L_1 = \left\langle \left(v_i^x, v_{i+1}^y\right) | i \in [k-1], x \in \{l, r\} \right\rangle$ be a list of $4k$ pairs. By invoking $\mathcal{A}$ with (H_1, T_1, L_1) as an input, we have computed $\rho_i^{xy} = d_{H_1}\left(v_i^x, v_{i+1}^y\right)$ for all $i \in [k-1], x \in \{l, r\}$, in $T_{\mathcal{A}}(2n, 4n)$ time. Let $e = (v_i, v_{i+1}) \in P$. By Theorem 15, $d_e(s,t) = w(P) - w(e) + d^r(v_i, v_{i+1})$. By Lemma 24, $d^r(v_i, v_{i+1}) = \min\left\{\rho_i^{ll}, \rho_i^{lr}, \rho_i^{rl}, \rho_i^{rr}\right\}$. Thus for every edge $e = (v_i, v_{i+1}) \in P$, we store $d_e(s,t) \leftarrow w(P) - w(e) + \min\left\{\rho_i^{ll}, \rho_i^{lr}, \rho_i^{rl}, \rho_i^{rr}\right\}$. The total running time to solve the replacement paths problem is $O(n) + T_{\mathcal{A}}(2n, 4n) = O(T_{\mathcal{A}}(2n, 4n))$. ◀

▶ **Theorem 30.** *Let G be a directed planar graph with non negative weights. Then, there is an algorithm for the replacement paths problem over G that runs in $O(n \log n)$ time.*

Proof. We denote by $\mathcal{A}$ the algorithm of Klein for the $MSSP$ problem [11]. By Theorem 19, $T_{\mathcal{A}}(n, n) = O(n \log n)$. Hence by Theorem 29, we can solve the replacement paths problem in $O(2n \log(4n)) = O(n \log n)$ time. ◀

References

1 Udit Agarwal and Vijaya Ramachandran. Fine-grained complexity for sparse graphs. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2018, pages 239–252, New York, NY, USA, 2018. ACM. doi:10.1145/3188745.3188888.

2 Noga Alon, Zvi Galil, and Oded Margalit. On the exponent of the all pairs shortest path problem. *Journal of Computer and System Sciences*, 54(2):255–262, 1997.

3 Yuval Emek, David Peleg, and Liam Roditty. A near-linear-time algorithm for computing replacement paths in planar directed graphs. *ACM Transactions on Algorithms (TALG)*, 6(4):64, 2010.

4 David Eppstein. Finding the k shortest paths. *SIAM Journal on computing*, 28(2):652–673, 1998.

5 Zvi Gotthilf and Moshe Lewenstein. Improved algorithms for the k simple shortest paths and the replacement paths problems. *Information Processing Letters*, 109(7):352–355, 2009.

6 Monika R Henzinger, Philip Klein, Satish Rao, and Sairam Subramanian. Faster shortest-path algorithms for planar graphs. *journal of computer and system sciences*, 55(1):3–23, 1997.

7 John Hershberger and Subhash Suri. Vickrey prices and shortest paths: What is an edge worth? In *Proceedings 2001 IEEE International Conference on Cluster Computing*, pages 252–259. IEEE, 2001.

8 John Hershberger, Subhash Suri, and Amit Bhosle. On the difficulty of some shortest path problems. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 343–354. Springer, 2003.

9 David R Karger, Daphne Koller, and Steven J Phillips. Finding the hidden path: Time bounds for all-pairs shortest paths. *SIAM Journal on Computing*, 22(6):1199–1217, 1993.

10 Philip Klein, Shay Mozes, and Oren Weimann. Shortest paths in directed planar graphs with negative lengths: A linear-space o (n log2 n)-time algorithm. In *Proceedings of the twentieth annual ACM-SIAM symposium on Discrete algorithms*, pages 236–245. SIAM, 2009.

11 Philip N Klein. Multiple-source shortest paths in planar graphs. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 146–155. Society for Industrial and Applied Mathematics, 2005.

12 Kavindra Malik, Ashok K Mittal, and Santosh K Gupta. The k most vital arcs in the shortest path problem. *Operations Research Letters*, 8(4):223–227, 1989.

13 Shay Mozes and Christian Wulff-Nilsen. Shortest paths in planar graphs with real lengths in o (nlog 2 n/loglogn) time. In *European Symposium on Algorithms*, pages 206–217. Springer, 2010.

14 Enrico Nardelli, Guido Proietti, and Peter Widmayer. A faster computation of the most vital edge of a shortest path? *Information Processing Letters*, 79(2):81–85, 2001.

15 Noam Nisan and Amir Ronen. Algorithmic mechanism design. *Games and Economic behavior*, 35(1-2):166–196, 2001.

16 Liam Roditty and Uri Zwick. Replacement paths and k simple shortest paths in unweighted directed graphs. In *International Colloquium on Automata, Languages, and Programming*, pages 249–260. Springer, 2005.

17 Oren Weimann and Raphael Yuster. Replacement paths via fast matrix multiplication. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 655–662. IEEE, 2010.

18 Virginia Vassilevska Williams. Faster replacement paths. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, pages 1337–1346. Society for Industrial and Applied Mathematics, 2011.

19 Virginia Vassilevska Williams and R Ryan Williams. Subcubic equivalences between path, matrix, and triangle problems. *Journal of the ACM (JACM)*, 65(5):27, 2018.

20 Christian Wulff-Nilsen. Solving the replacement paths problem for planar directed graphs in o (n log n) time. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 756–765. Society for Industrial and Applied Mathematics, 2010.

21 Raphael Yuster and Uri Zwick. Answering distance queries in directed graphs using fast matrix multiplication. In *Foundations of Computer Science, 2005. FOCS 2005. 46th Annual IEEE Symposium on*, pages 389–396. IEEE, 2005.

Sublinear Algorithms and Lower Bounds for Metric TSP Cost Estimation

Yu Chen
Department of Computer and Information Science,
University of Pennsylvania, Philadelphia, PA, USA
chenyu2@cis.upenn.edu

Sampath Kannan
Department of Computer and Information Science,
University of Pennsylvania, Philadelphia, PA, USA
kannan@cis.upenn.edu

Sanjeev Khanna
Department of Computer and Information Science,
University of Pennsylvania, Philadelphia, PA, USA
sanjeev@cis.upenn.edu

Abstract

We consider the problem of designing sublinear time algorithms for estimating the cost of minimum metric traveling salesman (TSP) tour. Specifically, given access to a $n \times n$ distance matrix D that specifies pairwise distances between n points, the goal is to estimate the TSP cost by performing only sublinear (in the size of D) queries. For the closely related problem of estimating the weight of a metric minimum spanning tree (MST), it is known that for any $\varepsilon > 0$, there exists an $\tilde{O}(n/\varepsilon^{O(1)})$ time algorithm that returns a $(1+\varepsilon)$-approximate estimate of the MST cost. This result immediately implies an $\tilde{O}(n/\varepsilon^{O(1)})$ time algorithm to estimate the TSP cost to within a $(2+\varepsilon)$ factor for any $\varepsilon > 0$. However, no $o(n^2)$ time algorithms are known to approximate metric TSP to a factor that is strictly better than 2. On the other hand, there were also no known barriers that rule out existence of $(1+\varepsilon)$-approximate estimation algorithms for metric TSP with $\tilde{O}(n)$ time for any fixed $\varepsilon > 0$. In this paper, we make progress on both algorithms and lower bounds for estimating metric TSP cost.

On the algorithmic side, we first consider the graphic TSP problem where the metric D corresponds to shortest path distances in a connected unweighted undirected graph. We show that there exists an $\tilde{O}(n)$ time algorithm that estimates the cost of graphic TSP to within a factor of $(2-\varepsilon_0)$ for some $\varepsilon_0 > 0$. This is the first sublinear cost estimation algorithm for graphic TSP that achieves an approximation factor less than 2. We also consider another well-studied special case of metric TSP, namely, $(1,2)$-TSP where all distances are either 1 or 2, and give an $\tilde{O}(n^{1.5})$ time algorithm to estimate optimal cost to within a factor of 1.625. Our estimation algorithms for graphic TSP as well as for $(1,2)$-TSP naturally lend themselves to $\tilde{O}(n)$ space streaming algorithms that give an 11/6-approximation for graphic TSP and a 1.625-approximation for $(1,2)$-TSP. These results motivate the natural question if analogously to metric MST, for any $\varepsilon > 0$, $(1+\varepsilon)$-approximate estimates can be obtained for graphic TSP and $(1,2)$-TSP using $\tilde{O}(n)$ queries. We answer this question in the negative – there exists an $\varepsilon_0 > 0$, such that any algorithm that estimates the cost of graphic TSP ($(1,2)$-TSP) to within a $(1+\varepsilon_0)$-factor, necessarily requires $\Omega(n^2)$ queries. This lower bound result highlights a sharp separation between the metric MST and metric TSP problems.

Similarly to many classical approximation algorithms for TSP, our sublinear time estimation algorithms utilize subroutines for estimating the size of a maximum matching in the underlying graph. We show that this is not merely an artifact of our approach, and that for any $\varepsilon > 0$, any algorithm that estimates the cost of graphic TSP or $(1,2)$-TSP to within a $(1+\varepsilon)$-factor, can also be used to estimate the size of a maximum matching in a bipartite graph to within an εn additive error. This connection allows us to translate known lower bounds for matching size estimation in various models to similar lower bounds for metric TSP cost estimation.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases sublinear algorithms, TSP, streaming algorithms, query complexity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.30

Category Track A: Algorithms, Complexity and Games

Related Version https://arxiv.org/abs/2006.05490

Acknowledgements We thank the anonymous reviewers for their valuable feedback. This work was supported in part by NSF awards CCF-1617851, CCF-1733794, CCF-1763307, CCF-1763514, and CCF-1934876.

1 Introduction

In the metric traveling salesman problem (TSP), we are given n points in an arbitrary metric space with an $n \times n$ matrix D specifying pairwise distances between them. The goal is to find a simple cycle (a TSP tour) of minimum cost that visits all n points. An equivalent view of the problem is that we are given a complete weighted undirected graph $G(V, E)$ where the weights satisfy triangle inequality, and the goal is to find a Hamiltonian cycle of minimum weight. The study of metric TSP is intimately connected to many algorithmic developments, and the poly-time approximability of metric TSP and its many natural variants are a subject of extensive ongoing research (see, for instance, [3, 11, 14, 15, 17, 24–28] and references within for some relatively recent developments). In this paper, we consider the following question: can one design sublinear algorithms that can be used to obtain good estimates of the cost of an optimal TSP tour? Since the complete description of the input metric is of size $\Theta(n^2)$, the phrase sublinear here refers to algorithms that run in $o(n^2)$ time.

A standard approach to estimating the metric TSP cost is to compute the cost of a minimum spanning tree, and output two times this cost as the estimate of the TSP cost (since any spanning tree can be used to create a spanning simple cycle by at most doubling the cost). The problem of approximating the cost of the minimum spanning tree in sublinear time was first studied in the graph adjacency-list model by Chazelle, Rubinfeld, and Trevisan [7]. The authors gave an $\tilde{O}(dW/\varepsilon^2)$-time algorithm to estimate the MST cost to within a $(1+\varepsilon)$-factor in a graph where average degree is d, and all edge costs are integers in $[1..W]$. For certain parameter regimes this gives a sublinear time algorithm for estimating the MST cost but in general, this run-time need not be sublinear. Subsequently, in an identical setting as ours, Czumaj and Sohler [10] showed that for any $\varepsilon > 0$, there exists an $\tilde{O}(n/\varepsilon^{O(1)})$ time algorithm that returns a $(1 + \varepsilon)$-approximate estimate of the MST cost when the input is an n-point metric. This result immediately implies an $\tilde{O}(n/\varepsilon^{O(1)})$ time algorithm to estimate the TSP cost to within a $(2 + \varepsilon)$ factor for any $\varepsilon > 0$. However, no $o(n^2)$ query algorithms are known to approximate metric TSP to a factor that is strictly better than 2. On the other hand, there are also no known barriers that rule out existence of $(1 + \varepsilon)$-approximate estimation algorithms for metric TSP with $\tilde{O}(n)$ queries for any fixed $\varepsilon > 0$. In this paper, we make progress on both algorithms and lower bounds for estimating metric TSP cost.

On the algorithmic side, we first consider the *graphic TSP* problem, an important case of metric TSP that has been extensively studied in the classical setting – the metric D corresponds to the shortest path distances in a connected unweighted undirected graph [17, 18, 25]. We give the first $\tilde{O}(n)$ time algorithm for graphic TSP that achieves an approximation factor *strictly better* than 2.

▶ **Theorem 1.1.** *There is an $\tilde{O}(n)$ time randomized algorithm that estimates the cost of graphic TSP to within a factor of $2 - \varepsilon_0$ for some constant $\varepsilon_0 > 0$.*

On the other hand, if we are willing to allow a higher sublinear time, we can get a better approximation ratio.

▶ **Theorem 1.2.** *There is an* $\tilde{O}(n^{1.5})$ *time randomized algorithm that estimates the cost of graphic TSP to within a factor of* $(27/14)$.

At a high-level, our algorithm is based on showing the following: if a graph G either lacks a matching of size $\Omega(n)$ or has $\Omega(n)$ biconnected components (blocks), then the optimal TSP cost is not too much better than $2n$. Conversely, if the graph G has both a large matching and not too many blocks, then we can show that the optimal TSP cost is distinctly better than $2n$. Since we do not know an efficient sublinear algorithm to estimate the number of blocks in a graph G, we work with another quantity that serves as a proxy for this and can be estimated in $\tilde{O}(n)$ time. The main remaining algorithmic challenge then is to estimate sufficiently well the size of a largest matching. This problem is very important by itself, and has received much attention [13,19,20,23,30]; please see a detailed discussion of this problem, and relevant recent developments towards the end of this section. Our $\tilde{O}(n)$ query results utilize the recent result of Kapralov et al. [13] who give an algorithm to approximate the size of maximum matching to within a constant factor (for some very large constant) in $\tilde{O}(n)$ time in the *pair query* model (is there an edge between a given pair of vertices?). We also show that matching size can be estimated to within a factor of 2 in $\tilde{O}(n^{1.5})$ time, crucial to obtaining the approximation guarantee in Theorem 1.2.

Our approach for estimating graphic TSP cost in sublinear time also lends itself to an $\tilde{O}(n)$ space streaming algorithm that can obtain an even better estimate of the cost. To our knowledge, no estimate better than a 2-approximation was known previously. In the streaming model, we assume that the input to graphic TSP is presented as a sequence of edges of the underlying graph G. Any algorithm for this model, clearly also works if instead the entries of the distance matrix are presented in the stream – an entry that is 1 corresponds to an edge of G, and it can be ignored otherwise as a non-edge.

▶ **Theorem 1.3.** *There is an* $O(n)$ *space randomized streaming algorithm that estimates the cost of graphic TSP to within a factor of* $(11/6)$ *in insertion-only streams.*

We next consider another well-studied special case of metric TSP, namely, $(1,2)$-TSP where all distances are either 1 or 2 [2,5,22], and obtain the following result.

▶ **Theorem 1.4.** *There is an* $\tilde{O}(n^{1.5})$ *time randomized algorithm that estimates the cost of* $(1,2)$*-TSP to within a factor of* 1.625.

Throughout the paper, whenever we refer to a graph associated with a $(1,2)$-TSP instance, it refers to the graph G induced by edges of distance 1 in our $\{1,2\}$-metric. At a high-level, the idea underlying our algorithm is to analyze the structure of the graph G induced by edges of distance 1. We design an algorithm to estimate the size of a maximal "matching pair" of G which is defined to be a pair of edge-disjoint matchings. We show that whenever the size of a matching pair is large in a graph G, the TSP cost is distinctly smaller than $2n$, and conversely, if this quantity is not large, the TSP cost is close to $2n$. The main remaining algorithm challenge then is to estimate sufficiently well the size of a maximal matching pair, and we show that this can be done in $\tilde{O}(n^{1.5})$ time.

For $(1,2)$-TSP, an $\tilde{O}(n)$ query algorithm that estimates the cost of $(1,2)$-TSP to within a factor of 1.75 was claimed in [1] but this result is based on the matching size estimation results of [20]. Unfortunately, as confirmed by the authors [21], there is a problem with the proof of one of the statements in the paper – Observation 3.9 – which is crucial for the correctness of the main result. As a result, the $\tilde{O}(d)$ time result in the neighbor query model as well as the $\tilde{O}(n)$ time result in the adjacency matrix, claimed in [20] can no longer be relied upon, and we have chosen to make this paper independent of these results [20]. It is worth mentioning that if the $\tilde{O}(n)$-time matching estimation result of [20] can be shown to hold, then the run-time of both Theorems 1.2 and 1.4 can be improved to $\tilde{O}(n)$ time.

We note that it is easy to show that randomization is crucial to getting better than a 2-approximation in sublinear time for both graphic TSP and $(1, 2)$-TSP (we include the proof of these results in the full version of this paper). The algorithms underlying Theorems 1.2 and 1.4, lend themselves to $\tilde{O}(n)$ space single-pass streaming algorithms with identical approximation guarantees. These sublinear time algorithms motivate the natural question if analogously to metric MST, there exist sublinear time algorithms that for any $\varepsilon > 0$, output a $(1 + \varepsilon)$-approximate estimate of TSP cost for graphic TSP and $(1, 2)$-TSP in $\tilde{O}(n)$ time. We rule out this possibility in a strong sense for both graphic TSP and $(1, 2)$-TSP.

▶ **Theorem 1.5.** *There exists an $\varepsilon_0 > 0$, such that any algorithm that estimates the cost of graphic TSP ($(1, 2)$-TSP) to within a $(1 + \varepsilon_0)$-factor, necessarily requires $\Omega(n^2)$ queries.*

This lower bound result highlights a sharp separation between the behavior of metric MST and metric TSP problems. At a high-level, our lower bound is inspired by the work of Bogdanov et al. [6] who showed that any query algorithm that for any $\varepsilon > 0$ distinguishes between instances of parity equations (mod 2) that are either satisfiable (Yes) or at most $(1/2 + \varepsilon)$-satisfiable (No), requires $\Omega(n)$ queries where n denotes the number of variables. However, the query model analyzed in [6] is different from ours (see more details in Section 4). We first show that the lower bound of [6] can be adapted to an $\Omega(n^2)$ lower bound in our model, and then show that instances of parity equations can be converted into instances of graphic TSP (resp. $(1, 2)$-TSP) such that for some $\varepsilon_0 > 0$, any $(1 + \varepsilon_0)$-approximation algorithm for graphic TSP (resp. $(1, 2)$-TSP), can distinguish between the Yes and No instances of the parity equations, giving us the desired result.

Finally, similar to many classical approximation algorithms for TSP, our sublinear time estimation algorithms utilize subroutines for estimating the size of a maximum matching in the underlying graph. We show that this is not merely an artifact of our approach.

▶ **Theorem 1.6.** *For any $\varepsilon \in [0, 1/5)$, any algorithm that estimates the cost of an n-vertex instance of graphic TSP or $(1, 2)$-TSP to within a $(1 + \varepsilon)$-factor, can also be used to estimate the size of a maximum matching in an n-vertex bipartite graph to within an εn additive error, with an identical query complexity, running time, and space usage.*

This connection allows us to translate known lower bounds for matching size estimation in various models to similar lower bounds for metric TSP cost estimation. In particular, using the results of [4], we can show that there exists an ε_0 such that any randomized single-pass dynamic streaming algorithm for either graphic TSP or $(1, 2)$-TSP that estimates the cost to within a factor of $(1 + \varepsilon_0)$, necessarily requires $\Omega(n^2)$ space.

We conclude by establishing several additional lower bound results that further clarify the query complexity of approximating TSP cost. For instance, we show that if an algorithm can access an instance of graphic TSP by only querying the edges of the graph (via neighbor and pair queries), then any algorithm that approximates the graphic TSP cost to a factor better than 2, necessarily requires $\Omega(n^2)$ queries. This is in sharp contrast to Theorem 1.1, and shows that working with the distance matrix is crucial to obtaining sublinear time algorithms for graphic TSP. We also show that even in the distance matrix representation, the task of *finding a tour* that is $(2 - \varepsilon)$-approximate for any $\varepsilon > 0$, requires $\Omega(n^2)$ queries for both graphic TSP and $(1, 2)$-TSP.

Matching Size Estimation. As the problem of matching size estimation is intimately connected to metric TSP cost estimation, we briefly review some relevant work here. This line of research primarily assumes that we are given a graph $G(V, E)$ with maximum degree d, that can be accessed via *neighbor queries* [12]: (a) for any vertex v, we can query its degree, and (b) for any vertex v and an integer i, we can learn the i_{th} neighbor of v.

Parnas and Ron [23] initiated the study of matching size estimation in sublinear time and gave an $d^{O(\log(d/\varepsilon)}$ time algorithm that estimates the matching size to within a constant factor plus an additive εn error for any $\varepsilon > 0$. Nguyen and Onak [19] presented a new estimation algorithm and showed that it can estimate the matching size to within a factor of 2 plus an additive εn error in $2^{O(d)}/\varepsilon^2$ time. We will refer to this approximation guarantee as a $(2, \varepsilon)$-*approximation* of matching size. Yoshida et al. [30] strongly improved upon the performance guarantee obtained in [19], and showed that a $(2, \varepsilon)$-approximation to matching size can be accomplished in $O(d^4/\varepsilon^2)$ time (in fact, they obtain the stronger $(2 \pm \varepsilon)$-approximation guarantee). The analysis of [30] was further improved by Onak et al. [20] who showed that the state of the art for $(2, \varepsilon)$-approximation of matching size. We note that it is known that any $(O(1), \varepsilon)$-approximate estimate of matching size necessarily requires $\Omega(d)$ queries [23], so the result of [20] is essentially best possible. Unfortunately, as mentioned above, we recently discovered a subtle mistake in the analysis of Onak et al. [21]. Consequently, the best known time complexity for obtaining a $(2, \varepsilon)$-approximate estimate is $\tilde{O}(d^2/\varepsilon^2)$); this weaker result also follows from the work of [20], but does not rely on the incorrect observation in [20].

The difference between a linear dependence versus a quadratic dependence on degree d is however huge in the sublinear time applications when the graph is not very sparse. In particular, while an $\tilde{O}(d)$ query result translates into an $\tilde{O}(n)$ time algorithm in the adjacency matrix model, an $\tilde{O}(d^2)$ query result gives only an $\tilde{O}(n^2)$ time algorithm, which is clearly not useful. Very recently, Kapralov et al. [13] gave an alternate approach based on a vertex "peeling" strategy (originally proposed in [23]) that yields an $(O(1), \varepsilon)$-approximation of matching size in $\tilde{O}(d/\varepsilon^2)$ time. Unfortunately, the constant hidden in the $O(1)$ notation is very large, and efficiently obtaining a $(2, \varepsilon)$-approximation to matching size remains an important open problem. Meanwhile, by directly building on the work of [30], we obtain an $\tilde{O}(n^{1.5})$ time algorithm for a $(2, \varepsilon)$-approximation to matching size in the adjacency matrix model, and it is this algorithm that is used in the results of Theorem 1.2 and Theorem 1.4.

Other Related Work. We note here that there is an orthogonal line of research that focuses on computing an approximate solution in near-linear time when the input is presented as a weighted undirected graph, and the metric is defined by shortest path distances on this weighted graph. It is known that in this model, for any $\varepsilon > 0$, there is an $\tilde{O}(m/\varepsilon^2 + n^{1.5}/\varepsilon^3)$ time algorithm that computes a $(3/2 + \varepsilon)$-approximate solution; here n denotes the number of vertices and m denotes the number of edges [9], and that a $(3/2 + \varepsilon)$-approximate estimate of the solution cost can be computed in $\tilde{O}(m/\varepsilon^2)$ time [8]. It is not difficult to show that in this access model, even when the input graph is unweighted (i.e. a graphic TSP instance), any algorithm that outputs better than a 2-approximate estimate of the TSP cost, requires $\Omega(n + m)$ time even when $m = \Omega(n^2)$. Hence this access model does not admit sublinear time algorithms that beat the trivial 2-approximate estimate.

Organization. In Section 2, we present our algorithms for graphic TSP (Theorem 1.1 and Theorem 1.2). In section 3, we present the 1.625-approximation algorithm of (1, 2)-TSP (Theorem 1.4). In Section 4, we present our lower bound result that rules out possibility of a sublinear-time approximation scheme for both graphic TSP and (1, 2)-TSP (Theorem 1.5).

In Section 5, we present a strong connection between approximating metric TSP cost and estimating matching size (Theorem 1.6). We defer to the full version the proofs of several additional lower bound results on the complexity of approximating graphic TSP and $(1, 2)$-TSP cost.

2 Approximation for Graphic TSP Cost

In this section, we exploit well-known properties of biconnected graphs and biconnected components in graphs to give an algorithm that achieves a $(2 - \frac{1}{7c_0})$-approximation for graphic TSP if we have an efficient algorithm that approximates the maximum matching size within a factor of c_0. We first relate the cost of the TSP tour in a graph to the costs of the TSP tours in the biconnected components of the graph. Next we show that if the graph does not have a sufficiently big matching, it does not have a TSP tour whose length is much better than $2n$. We also show that if a graph has too many degree 1 vertices, or vertices of degree 2, both whose incident edges are bridges, then it does not have a TSP tour of cost much better than $2n$. We then establish the converse - a graph that has a good matching and not too many bad vertices (namely, vertices of degree 1 or articulation points of degree 2), then it necessarily has a TSP tour of cost much better than $2n$. We design $\tilde{O}(n)$ time test for the second condition, allowing us to approximate the cost of an optimal graphic TSP tour in sublinear time together with some known techniques for testing the first condition. In what follows, we first present some basic concepts and develop some tools that will play a central role in our algorithms.

2.1 Preliminaries

An unweighted graph $G = (V, E)$, defines a *graphic metric* in V, where the distance between any two vertices u and v is given by the length of the shortest path between u and v. The *graphic TSP* is the Traveling Salesman Problem defined on such a graphic metric. In this paper our goal is to find a non-trivial approximation to the length of the traveling salesman tour in sublinear time in a model where we are allowed to make *distance queries*. In the distance query model, the algorithm can make a query on a pair of vertices (u, v) and get back the answer $d(u, v)$, the distance between u and v in G.

In a connected graph G, an edge e is a *bridge* if the deletion of e would increase the number of connected components of G. A connected graph with no bridge is called a *2-edge-connected graph*. A maximal 2-edge-connected subgraph of G is called a *2-edge-connected component*. The *bridge-block tree* of a graph is a tree such that the vertex set contains the 2-edge-connected components and the edge set contains the bridges in the graph.

A connected graph G is called *2-vertex-connected* or *biconnected* if when any one vertex is removed, the resulting graph remains connected. In a graph which is not biconnected, a vertex v whose removal increases the number of components is called an *articulation point*. It is easy to prove that any biconnected graph with at least 3 vertices does not have degree 1 vertices. A well-known alternate characterization of biconnectedness is that, a graph G is biconnected if and only if for any two distinct edges, there is a simple cycle that contains them.

A *biconnected component* or *block* in a graph is a maximal biconnected subgraph. Any graph G can be decomposed into blocks such that the intersection of any two blocks is either empty, or a single articulation point. Each articulation point belongs to at least two blocks. If a block is a single edge, then we call this block a *trival block*; otherwise it is a *non-trivial* *block*. A trival block is also a bridge in the graph. The size of a block is the number of vertices in the block. The following lemma shows the relationship between the number of blocks and the sum of the sizes of the blocks.

▶ **Lemma 2.1.** *If a connected graph G has n vertices and k blocks, then the sum of the sizes of the blocks is equal to $n + k - 1$.*

Proof. We prove the lemma by induction on the number k of blocks. The base case is when $k = 1$. In this case, G itself is a block of size n.

For the induction step, we have $k > 1$ and thus the graph has at least one articulation point. Suppose v is an arbitrary articulation point in G. Let $V_1, V_2, \dots, V_j$ be the set of vertices in the connected components of $G \setminus \{v\}$. We have $\sum_{i=1}^{j} |V_i| = n - 1$. Let $G_1, G_2, \dots, G_j$ be the subgraphs of G induced by $V_1 \cup \{v\}, V_2 \cup \{v\}, \dots, V_j \cup \{v\}$. For any G_i, let k_i be the number of blocks in G_i, we have $\sum_{i=1}^{j} k_i = k$. By induction hypothesis, the sum of the sizes of blocks in G_i is $|V_i| + 1 + k_i - 1 = |V_i| + k_i$. So the sum of the sizes of blocks in G is $\sum_{i=1}^{j} |V_i| + k_i = n - 1 + k$. ◀

The block decomposition of a graph has a close relationship with the cost of graphic TSP of the graph.

▶ **Lemma 2.2** (Lemma 2.1 of [16]). *The cost of the graphic TSP of a connected graph $G = (V, E)$ is equal to the sum of the costs of the graphic TSP of all blocks in the graph.*

Together these two lemmas give us a simple lower bound on the cost of the graphic TSP of a graph G (using the fact that the cost of graphic TSP is at least the number of vertices in the graph).

▶ **Lemma 2.3.** *If a graph G has n vertices and k blocks, then the cost of graphic TSP of G is at least $n + k - 1$.*

An *ear* in a graph is a simple cycle or a simple path. An ear which is a path is also called an *open ear* and it has two endpoints, whereas for a cycle, one vertex is designated as the endpoint. An *ear decomposition* of a graph is a partition of a graph into a sequence of ears such the endpoint(s) of each ear (except for the first) appear on previous ears and the internal points (the points that are not endpoints) are not on previous ears. A graph G is biconnected if and only if G has an ear decomposition such that each ear but the first one is an open ear [29]. An ear is *nontrivial* if it has at least one internal point. The following lemma upper bounds the cost of graphic TSP of a biconnected graph.

▶ **Lemma 2.4** (Lemma 5.3 of [25], also a corollary of Lemma 3.2 of [17]). *Given a 2-vertex-connected graph $G = (V, E)$ and an ear-decomposition of G in which all ears are nontrivial, a graphic TSP tour of cost at most $\frac{4}{3}(|V| - 1) + \frac{2}{3}\pi$ can be found in polynomial-time, where π denotes the number of ears.*

We now prove an important lemma that gives an upper bound on the cost of graphic TSP in a biconnected graph in terms of the size of a matching in the graph.

▶ **Lemma 2.5.** *Suppose G is a biconnected graph with at least $n \geq 3$ vertices. If G has a matching M, then the cost of graphic TSP of G is at most $2n - 2 - \frac{2|M|}{3}$.*

Proof. We first find a spanning biconnected subgraph of G that only contains $2n - 2 - M$ edges, then use Lemma 2.4 to bound the cost of graphic TSP.

We construct a spanning biconnected subgraph $G^\star = P_0 \cup P_1 \cup \dots$ recursively: P_0 contains a single edge in M. If $G_{i-1} = P_0 \cup P_1 \cup \cdots \cup P_{i-1}$ is a spanning subgraph of G, let $G^\star = G_{i-1}$ and finish the construction. Otherwise we construct P_i as follows. Let e be an edge in M both whose endpoints are not in G_{i-1}. If there is no such edge, then let e be an arbitrary edge such that at least one of its endpoints is not in G_{i-1}. Let e' be an arbitrary edge in G_{i-1}. By the alternate characterization of biconnectedness, there is a simple cycle C_i that contains both e and e'. Let P_i be the path in C_i that contains e and exactly two vertices in G_{i-1}, which are the endpoints of P_i.

Since P_i contains at least one vertex not in G_{i-1}, the construction always terminates. Note that $P_0 \cup P_1$ is a cycle, and each P_i $(i > 1)$ is an open ear of $G^\star$. So, $(P_0 \cup P_1, P_2, \dots)$ is an open ear decomposition of $G^\star$, which means $G^\star$ is biconnected.

Now we prove that the number of edges in $G^\star$ is at most $2n - 2 - M$. Let n_i be the number of vertices in $G_i \backslash G_{i-1}$. Let G_{-1} be the empty graph, so that n_0=2. Let p_i be the number of edges in P_i and m_i be the number of edges e in M such that $e \cap G_i \neq \emptyset$ and $e \cap G_{i-1} = \emptyset$. (Here we view an edge as a 2-vertex set.) Note that $m_0 = 1$. Suppose $G^\star = G_k$. Then $\sum_{i=1}^k n_i = n$, $\sum_{i=1}^k p_i$ is the number of edges in $G^\star$ and $\sum_{i=1}^k m_i = |M|$. For any $i > 0$, P_i is an open ear whose internal points are not in G_{i-1}. So $n_i = p_i - 1$. If there is an edge $e \in M$ such that $e \cap G_{i-1} = \emptyset$, then P_i contains both endpoints of an edge in M, which means $m_i \leq n_i - 1$. If all edges in M already have an endpoint in G_{i-1}, $m_i = 0 \leq n_i - 1$. So in both cases, $p_i = n_i + 1 = 2n_i - (n_i - 1) \leq 2n_i - m_i$. Also, $p_0 = 1 = 2n_0 - 2 - m_0$. So the number of edges in $G^\star$ is $\sum_{i=0}^k p_i \leq 2n_0 - 2 - m_0 + \sum_{i=1}^k (2n_i - m_i) = 2n - 2 - |M|$.

As $(P_0 \cup P_1, P_2, P_3, \dots, P_k)$ is an open ear decomposition of $G^\star$, the number of ears in G is k. On the other hand, $\sum_{i=0}^k p_i = 1 + \sum_{i=1}^k (n_i + 1) = n - 1 + k$, we have $n - 1 + k \leq 2n - 2 - |M|$, which means $k \leq n - 1 - |M|$. By Lemma 2.4, the cost of graphic TSP of $G^\star$ is at most $\frac{4}{3}(n-1) + \frac{2}{3}k \leq 2(n-1) - \frac{2}{3}|M|$.

Since $G^\star$ is a subgraph of G that contains all the vertices in G, the cost of graphic TSP of G is at most the cost of graphic TSP of $G^\star$, which is at most $2n - 2 - \frac{2}{3}|M|$. ◀

2.2 Approximation Algorithm for Graphic TSP

We now give our sublinear-time algorithm for approximating the cost of graphic TSP of a graph G to within a factor strictly smaller than 2.

We call a vertex v a *bad* vertex if v has degree 1 or is an articulation point with degree 2.

For any given $\delta > 0$, the graphic TSP algorithm performs the following two steps.

1. Obtain an estimate $\hat{\alpha} n$ of the size of maximum matching αn.
2. Obtain an estimate $\hat{\beta} n$ of the number of bad vertices βn.

The algorithm then output $\min\{2n, (2 - \frac{2}{7}(\hat{\alpha} - 2\hat{\beta}))n\}$.

To perform the second step in $\tilde{O}(n)$ distance queries and time, we randomly sample $O(\frac{1}{\delta^2})$ vertices. For each sampled vertex, we can obtain the degree with n queries. The following lemma shows that we can also check whether a degree 2 vertex is an articulation point using distance queries in $O(n)$ time. Then by the Chernoff bound, we can approximate the number of bad vertices with additive error $O(\delta n)$ with a high constant probability.

▶ **Lemma 2.6.** *Suppose a vertex v in a connected graph G has only two neighbors u and w. The following three conditions are equivalent:*

1. *v is an articulation point.*
2. *The edges (u, v) and (v, w) are both bridges.*
3. *For any vertex $v' \neq v$, $|d(u, v') - d(w, v')| = 2$.*

Proof. We first prove the first two conditions are equivalent. If v is an articulation point, then v is in two different blocks. So edge (u, v) and (v, w) are in different blocks, which means v has degree 1 in both blocks. So both blocks are trivial, which means (u, v) and (v, w) are both bridges. If (u, v) and (v, w) are both bridges, then deleting either (u, v) or (v, w) will disconnect u and w, which means deleting v will also disconnect u and w.

Next we prove that the third condition is equivalent to the first two. Suppose v is an articulation point. Since v has degree 2, the graph $G \setminus \{v\}$ has only two components, one containing u and the other containing w. For any vertex $v' \neq v$, without loss of generality, suppose v' is in the same component as u in $G \setminus \{v\}$. Since (u, v) and (v, w) are both bridges in G, any path between v' and w contains u and v. So $d(v', w) = d(v', u) + 2$.

If v is not an articulation point, then u and w are connected in $G \setminus \{v\}$. Let $(u = v_0, v_1, v_2, \dots, v_k = w)$ be the shortest path between u and w in $G \setminus \{v\}$. For any vertex v_i on the path, the distance between v_i and u (resp. w) in $G \setminus \{v\}$ is i (resp. $k - i$). Consider the shortest path between u and v_i in G. If this path does not contain v, then it is the same as the path in $G \setminus \{v\}$. In this case, $d(u, v_i) = i$. If the shortest path contains v, then v must be the second last vertex on the path and w be the third last one. In this case, $d(u, v_i) = k - i + 2$. So $d(u, v_i) = \min\{i, k - i + 2\}$. Similarly, we also have $d(v_i, w) = \min\{i + 2, k - i\}$. Let $v' = v_{\lfloor k/2 \rfloor}$. Since $|i - (k - i)| \leq 1$, we have $i < k - i + 2$ and $k - i < i + 2$, which means $|d(u, v') - d(w, v')| = |i - (k - i)| \leq 2$. ◀

Next, we prove that if α is small or β is large, the cost of graphic TSP is bounded away from n. The following lemma shows that if the size of maximum matching of a graph is small, then the cost of the graphic TSP is large.

▶ **Lemma 2.7.** *For any $\varepsilon > 0$, if the maximum matching of a graph G has size at most $\frac{(1-\varepsilon)n}{2}$, then the cost of graphic TSP of G is at least $(1 + \varepsilon)n$.*

Proof. Suppose the optimal TSP tour is $(v_0, v_1, \dots, v_{n-1}, v_n = v_0)$. Since the size of maximum matching in G is at most $\frac{(1-\varepsilon)n}{2}$, there are at most $\frac{(1-\varepsilon)n}{2}$ edges between pairs (v_i, v_{i+1}) where i is even (resp. odd). So there are at least εn pairs of (v_i, v_{i+1}) that have distance at least 2, which means that the optimal cost of TSP tour of G is $\sum_{i=1}^{n-1} d(v_i, v_{i+1}) \geq n + \varepsilon n = (1 + \varepsilon)n$. ◀

The following lemma shows that if β is large, the cost of graphic TSP is large.

▶ **Lemma 2.8.** *For any $\varepsilon > 0$, if a connected graph G has εn bad vertices, then the cost of graph-TSP of G is at least $(1 + \varepsilon)n - 2$.*

Proof. We first prove by induction on the number of vertices that a graph with k bad vertices has $k - 1$ bridges. The base case is when $n = 2$, the graph has $k = 2$ bad vertices and $1 = k - 1$ bridge.

For the induction step, the graph has n vertices with $n \geq 3$. If G has no degree 1 vertices, then the graph has k articulation points with degree 2. By Lemma 2.6, any edge incident on a degree 2 articulation point is a bridge. So each bad vertex is incident on 2 bridges. On the other hand, a bridge is incident on at most 2 vertices. So there are at least $\frac{2k}{2} = k$ bridges in G. Next, suppose G has degree 1 vertices. Let v be an arbitrary such vertex and let u be its neighbor. Since G is connected and $n \geq 3$, u must has degree at least 2, since otherwise u and v are not connected to other vertices in G. Consider the graph $G \setminus \{v\}$, if u is a bad vertex in G, u has degree 1 in $G \setminus \{v\}$ and is still a bad vertex. So the number of bad vertices in $G \setminus \{v\}$ is $k - 1$. By induction hypothesis, $G \setminus \{v\}$ has at least $k - 2$ bridges. G has at least $k - 1$ bridges since (u, v) is also a bridge.

So G has at least $\varepsilon n - 1$ bridges, and the number of blocks in G is at least $\varepsilon n - 1$. By Lemma 2.3, the cost of graph-TSP of G is at least $n + \varepsilon n - 2 = (1 + \varepsilon)n - 2$. ◀

Finally, the following lemma shows that the cost of graphic TSP is at most $(2-\frac{2}{7}(\hat{\alpha}-2\beta))n$.

▶ **Lemma 2.9.** *If a graph has a matching M of size $\alpha' n$ and the graph has βn bad vertices, the cost of graphic TSP of G is at most $(2-\frac{2}{7}(\alpha'-2\beta))n$.*

Proof. Let $G_1, G_2, \dots, G_k$ be the block decomposition of G. Let n_i be the size of G_i. If $|n_i| \geq 3$, by Lemma 2.5, the cost of the graphic TSP of G_i is at most $2n_i - 3$ since any non-empty graph has a matching of size at least 1. If $|n_i| = 2$, then the graphic TSP of G_i is exactly $2 = 2n_i - 2$. Suppose G has ℓ non-trivial blocks. Then by Lemma 2.2 the cost of graphic TSP of G is at most $\sum_{i=1}^{k}(2n_i - 2) - \ell$, which equals to $2n - 2 - \ell$ by Lemma 2.1.

Let m_i be the size of maximum matching in G_i if G_i is a non-trivial block, and let $m_i = 0$ if G_i is a trivial block. By Lemma 2.5, the cost of the graphic TSP of G_i is at most $2n_i - 2 - \frac{2m_i}{3}$. For any non-trivial block G_i, $M \cap G_i$ is a matching in G_i. So the size of maximum matching in G_i is at least the number of edges in $M \cap G_i$. So by Lemma 2.2 and Lemma 2.1, the cost of graphic TSP of G is at most $\sum_{i=1}^{k}(2n_i - 2 - \frac{2}{3}m_i) = 2n - 2 - \frac{2}{3}|M'|$, where M' is the set of edges in M that are not bridges in G. Let B be the number of bridges in G. We have $2n - 2 - \frac{2}{3}|M'| \leq 2n - 2 - \frac{2}{3}(|M| - B)$.

So there are two upper bounds of the graphic TSP of G – $2n-2-\ell$ and $2n-2-\frac{2}{3}(|M|-B)$. Which bound is better depends on the number of bridges B.

If $B \leq (\frac{4}{7}\alpha' + \frac{6}{7}\beta)n$, the cost of graphic TSP of G is at most

$$2n - 2 - \frac{2}{3}(|M| - B) \leq 2n - \frac{2}{3}(\frac{3}{7}\alpha' - \frac{6}{7}\beta)n = (2 - \frac{2}{7}(\alpha' - 2\beta))n$$

If $B > (\frac{4}{7}\alpha' + \frac{6}{7}\beta)n$, consider the bridge-block tree T of G. T has at least B edges and at least $B + 1$ vertices. Since T is a tree, there are at least $\frac{B}{2}$ vertices of degree at most 2. For any vertex v_T of degree at most 2 in T, if the vertex v_T represents a single vertex v in G, then v is either a degree 1 vertex or a degree 2 articulation point in G, otherwise v_T represents a 2-edge-connected component of size at least 2 in G. So There are at least $\frac{B}{2} - \beta n \geq (\frac{2}{7}\alpha' - \frac{4}{7}\beta)n$ 2-edge-connected components of size at least 2. Since any 2-edge-connected component of size at least 2 has no bridge, each such component of G contains at least 1 non-trivial block in G, implying that $\ell \geq \frac{2}{7}(\alpha' - 2\beta)n$. So the cost of graphic TSP of G is at most $2n - 2 - \ell \leq (2 - \frac{2}{7}(\alpha' - 2\beta))n$. ◀

We summarize the ideas in this section and prove the following lemma.

▶ **Lemma 2.10.** *For any $c_0 > 1$ and $\delta > 0$, suppose $\hat{\alpha} \leq \alpha \leq c_0\hat{\alpha} + \delta$ and $\hat{\beta} - \delta \leq \beta \leq \hat{\beta}$. Then $(2 - \frac{2}{7}(\hat{\alpha} - 2\hat{\beta}))n$ is an approximation of the size of graphic TSP within a factor of $2 - \frac{1}{7c_0} + \delta$.*

Proof. Let $\hat{T} = (2 - \frac{2}{7}(\hat{\alpha} - 2\hat{\beta}))n$. Since $\hat{\beta} \geq \beta$ and $\hat{\alpha} \leq \alpha$, by Lemma 2.9, $T \leq \hat{T}$.

Then we prove that $\hat{T} \leq (2 - \frac{1}{7c_0} + \delta)T$. By Lemma 2.7 and Lemma 2.8, $T \geq \max\{(2 - 2\alpha)n, (1 + \beta)n - 2\}$, which means

$$(2 - \frac{1}{7c_0} + \delta)T \geq (2 - \frac{1}{7c_0})\max\{(2 - 2\alpha)n, (1 + \beta)n\} - 4 + \delta n$$

On the other hand, $\hat{T} \leq (2-\frac{2}{7}(\frac{\alpha}{c_0}-2\beta))n+\frac{6}{7}\delta n$ since $c_0\hat{\alpha}+\delta \leq \alpha$ and $\hat{\beta} \leq \beta+\delta$. For sufficient large n, we have $\delta n - 4 \geq \frac{6}{7}\delta n$, so it is sufficient to prove that $\frac{2-\frac{2}{7}(\frac{\alpha}{c_0}-2\beta)}{\max\{2-2\alpha,1+\beta\}} \leq 2 - \frac{1}{7c_0}$ for any $0 \leq \alpha, \beta \leq 1$ and $c_0 \geq 1$.

Let $\gamma = \frac{\alpha}{c_0} - 2\beta$, $1+\beta = 1 + (\frac{\alpha}{c_0} - \gamma)/2$, so if we fix γ, $\max\{2-2\alpha, 1+\beta\}$ is minimized when $2 - 2\alpha = 1 + (\frac{\alpha}{c_0} - \gamma)/2$. In this case $\alpha = \frac{(2+\gamma)c_0}{4c_0+1}$ and $\max\{2-2\alpha, 1+\beta\} = \frac{4c_0+2}{4c_0+1} - \frac{2c_0}{4c_0+1}\gamma$. If $\gamma \le \frac{1}{2c_0}$,

$$\begin{aligned}\frac{2-\frac{2}{7}(\alpha-2\beta)}{\max\{2-2\alpha,1+\beta\}} &\le \frac{2-\frac{2}{7}\gamma}{\frac{4c_0+2}{4c_0+1}-\frac{2c_0}{4c_0+1}\gamma} = \frac{4c_0+1}{7c_0} - \frac{2-\frac{4c_0+2}{7c_0}}{\frac{4c_0+2}{4c_0+1}-\frac{2c_0}{4c_0+1}\gamma} \\ &\le \frac{4c_0+1}{7c_0} + 2 - \frac{4c_0+2}{7c_0} = 2 - \frac{1}{7c_0}\end{aligned}$$

If $\gamma > \frac{1}{2c_0}$, $\frac{2-\frac{2}{7}(\alpha-2\beta)}{\max\{2-2\alpha,1+\beta\}} < \frac{2-\frac{1}{7c_0}}{1} = 2 - \frac{1}{7c_0}$ since $\beta \ge 0$. So $\hat{T} \le (2 - \frac{1}{7c_0} + \delta)T$. ◀

By Lemma 2.10, we immediately have the following theorem.

▶ **Theorem 2.11.** *For any $\delta > 0$ and $c_0 \ge 1$. Given a graph G with maximum matching size αn, suppose there is an algorithm that uses pair queries, runs in t time, and with probability at least $2/3$, outputs an estimate of the maximum matching size $\hat{\alpha} n$ such that $\hat{\alpha} \le \alpha \le c_0\hat{\alpha} + \delta$. Then there is an algorithm that approximates the cost of graphic TSP of G to within a factor of $2 - \frac{1}{7c_0} + \delta$, using distance queries, in $t + \tilde{O}(n/\delta^2)$ time with probability at least $3/5$.*

Proof. We first use the algorithm in the assumption to obtain an estimate $\hat{\alpha} n$ of the size of maximum matching αn. The following analysis is based on the event that this algorithm is run successfully, which has probability $2/3$.

We then sample $N = \frac{100}{\delta^2}$ vertices. For each sampled vertex v, we first query the distance between v and every vertex in G to obtain the degree of v. If v has degree 2, suppose u and w are the neighbors of v. We query the distance from u and w to every vertex in G. By Lemma 2.6, v is an articulation point if and only if there is no vertex v' such that $|d(u,v') - d(w,v')| \le 1$. So we can check if v is a bad vertex with $O(n)$ distance queries and time. Suppose there are βn bad vertices in G and $(\hat{\beta} - \delta/2)N$ sampled vertices are bad. By Chernoff bound, the probability that $\left|\beta - \hat{\beta} + \delta/2\right| > \delta/2$ is at most $2e^{\frac{\delta^2 N^2}{16}} < 1/15$. We analyze the performance based on the event that $\beta \le \hat{\beta} \le \beta + \delta$.

By Lemma 2.10, $(2 - \frac{2}{7}(\hat{\alpha} - 2\hat{\beta}))$ is a $(2 - \frac{1}{7c_0} + \delta)$ approximation of the size of graphic TSP of G. The probability of failure is at most $1/3 + 1/15 = 2/5$. ◀

Proof of Theorem 1.2. The following theorem whose proof appears in the full version, gives an algorithm for matching size estimation that only uses *pair queries* – given a pair of vertices, is there an edge between them? Note that any pair query can be simulated by a single query to the distance matrix in a graphic TSP instance. ◀

▶ **Theorem 2.12.** *For any $\varepsilon > 0$, there is an algorithm that uses pair queries, runs in $\tilde{O}(n^{1.5}/\varepsilon^2)$ time, and with probability $2/3$, outputs an estimate of the size of a maximal matching within an additive error εn.*

Substituting the above result in Theorem 2.11 and using the fact that a maximum matching has size at most twice the size of a maximal matching (setting $c_0 = 2$, and $\delta = \varepsilon$), we obtain Theorem 1.2.

Proof of Theorem 1.1. Kapralov et al. [13] give an algorithm that uses $\tilde{O}(d)$ queries to approximate the size of maximum matching in a graph with average degree d in the neighbor query model (the approximation ratio is a very large constant). Together with a reduction in [20], this implies a pair query algorithm that uses $\tilde{O}(n)$ queries to estimate matching size to a constant factor. Combined with Theorem 2.11, this implies Theorem 1.1. ◀

2.3 An $O(n)$ Space $(\frac{11}{6})$-Approximate Streaming Algorithm

We show here that our approach above can be extended to the insertion-only streaming model to obtain for any $\varepsilon > 0$, an $(\frac{11}{6} + \varepsilon)$-approximate estimate of the graphic TSP cost using $O(n/\varepsilon^2)$ space. Given a stream containing edges of a graph $G(V, E)$, our algorithm performs the following two tasks simultaneously:

- Find a maximal matching M in G – let αn denote its size.
- Estimate the number of bridges in the maximal matching M, say βn, to within an additive error of εn.

The algorithm outputs $(2 - \frac{2}{3}(\alpha - \beta))n$ as the estimated cost of graphic TSP of G.

In an insertion-only stream, it is easy to compute a maximal stream with $O(n)$ space, we start with M initialized as an empty set, and add a new edge (u, v) into the matching M iff neither u nor v are already in M. It is also easy to check if the edge is a bridge in insertion-only stream with $O(n)$ space. We can maintain a disjoint-set data structure. Whenever an edge arrives (other than e), we merge the connected components of its endpoints. If there is only one component remaining at the end of the stream, then e is not a bridge. Otherwise, e is a bridge.

To estimate the number of bridges in the maximal matching, we sample $N = 100/\varepsilon^2$ edges in the matching, and run in parallel N tests where each test determines whether or not the sampled edge is a bridge. We use $O(n/\varepsilon^2)$ space in total since we sample $N = O(1/\varepsilon^2)$ edges. Suppose there are $\bar{\beta}$ sampled edges are bridges, then by Chernoff bound, $\hat{\beta}n = \frac{\bar{\beta}|M|}{N}$ is an approximation of βn to within additive error εn with probability at least $9/10$.

As stated, this gives us a two-pass algorithm: the first pass for computing the matching M, and the second pass for estimating the number of bridges in M. However, we can do both these tasks simultaneously in a single pass as follows: at the beginning of the stream, we start the process of finding connected components of graph G. Whenever an edge e is added to M, if $|M| < N$, then we create a new instance I_e of the connectivity problem that ignores the edge e. This clearly allows us to test whether or not e is a bridge. Once $|M| > N$, then whenever an edge e is added to M, with probability $\frac{N}{|M|}$, we randomly drop an existing instance, say $I_{e'}$ of connectivity, and create a new instance I_e of connectivity that only ignores edge e (we insert back the edge e' into I_e). Since there are at most N instances of connectivity instance that are running in parallel, the algorithm uses $O(nN) = O(n/\varepsilon^2)$ space.

We defer to the full version the details of the analysis showing that the cost estimate output above is a $(\frac{11}{6} + \varepsilon)$-approximate estimate.

3 (1.625)-Approximation for (1, 2)-TSP Cost in $\tilde{O}(n^{1.5})$ Time

In this section, we give an algorithm that for any $\delta > 0$, approximates the cost of the minimum $(1, 2)$-TSP to within a factor of $1.625 + \delta$ with $\tilde{O}(n^{1.5}/\delta^2)$ queries. The idea of the algorithm is to approximate the size of a maximal "matching pair" of G. In a graph G, a *matching pair* (M_1, M_2) is a pair of edge-disjoint matchings. A *maximal* matching pair is a matching pair (M_1, M_2) such that for any edge $e \notin M_1 \cup M_2$, neither $M_1 \cup \{e\}$ nor $M_2 \cup \{e\}$ is a matching. The size of a matching pair (M_1, M_2) is the sum of the sizes of M_1 and M_2. The following lemma shows that the size of any maximal matching pair is lower bounded by the size of maximum matching in the graph.

▶ **Lemma 3.1.** *Suppose M is a matching in a graph G. Then any maximal matching pair (M_1, M_2) in G has size at least $|M|$.*

Proof. Let X_1 be the set of vertices matched in both M and M_1, and X_2 be the set of vertices matched in both M and M_2. We have $|X_1| + |X_2| \leq 2\,|M_1| + 2\,|M_2|$ since M_1 and M_2 are both matchings. On the other hand, for any edge $e \in M$, if e is either in M_1 or M_2, then both of its endpoints are in X_1 or X_2. If e is neither in M_1 or M_2, then there are edges $e_1 \in M_1$ and $e_2 \in M_2$ that share an endpoint with e since (M_1, M_2) is a maximal matching pair. So both X_1 and X_2 contain at least one endpoint of e. In both case e's endpoints appear twice in X_1 and X_2. So $|X_1| + |X_2| \geq 2\,|M|$, which means $|M_1| + |M_2| \geq |M|$. ◀

If a graph has a large sized matching pair, then the cost of $(1, 2)$-TSP is not very large.

▶ **Lemma 3.2.** *If a graph G with n vertices contains a matching pair (M_1, M_2) of size X, then the cost of $(1, 2)$-TSP of G is at most $2n - \frac{3}{4}X$.*

Proof. Since M_1 and M_2 are both matchings, $M_1 \cup M_2$ only contains paths and cycles of even length. We delete one edge from each cycle in $M_1 \cup M_2$, resulting in a graph that only contains paths. Since the cycles in $M_1 \cup M_2$ are of even length, the size of any cycle is at least 4. We deleted at most $\frac{1}{4}X$ edges, so G contains a set of vertex disjoint paths (including some of length 0, corresponding to isolated vertices), with total size at least $\frac{3}{4}X$. Construct a TSP tour by ordering the paths arbitrarily, orienting each one, and connecting the end of one path with the start of the next, cyclically. The tour contains at least $\frac{3}{4}X$ edges of weight 1, while the remaining edges are of weight 2. So the cost of the tour is at most $2n - \frac{3}{4}X$. ◀

By Lemma 3.1, the maximum matching size is upper bounded by the size of any maximal matching pair. It follows that if the maximum matching size is small, the cost of $(1, 2)$-TSP is large.

▶ **Lemma 3.3.** *For any $\varepsilon > 0$, if the maximum matching of a graph G has size at most $\frac{(1-\varepsilon)n}{2}$, then the cost of $(1, 2)$-TSP of G is at least $(1 + \varepsilon)n$.*

The proof of Lemma 3.3 is similar to the proof of Lemma 2.7 and we omit it here. By Lemma 3.2 and Lemma 3.3, if we can approximate the size of an arbitrary maximal matching pair, then we will get a good approximation of the cost of the $(1, 2)$-TSP.

▶ **Theorem 3.4.** *There is an algorithm that uses pair queries, with probability at least $2/3$, approximates the size of a maximal matching pair with additive error εn using $\tilde{O}(n^{1.5}/\varepsilon^2)$ time.*

The algorithm in Theorem 3.4 is given in the full verstion of this paper. With Theorem 3.4, we can approximate the cost of $(1, 2)$-TSP in a graph G by the size of maximal matching pair.

▶ **Theorem 3.5.** *For any $\delta > 0$, there is an algorithm that estimates the cost of $(1, 2)$-TSP of a graph G within a factor of $1.625 + \delta$ using $\tilde{O}(n^{1.5}/\delta^2)$ queries with probability at least $2/3$.*

Proof. Let $\varepsilon = \delta/2$. We use the algorithm in Theorem 3.4 that approximates the size of a maximal matching pair. Suppose the output of the algorithm is $\bar{X}$. Then, by Theorem 3.4, there is a maximal matching pair of size X such that $\left|X - \bar{X}\right| \leq \varepsilon n$. We output the cost of the $(1, 2)$-TSP of G to be $\bar{T} = 2n - \frac{3}{4}(\bar{X} - \varepsilon n)$. Suppose the optimal $(1, 2)$-TSP has cost T. By Lemma 3.2, $T \leq 2n - \frac{3}{4}X \leq 2n - \frac{3}{4}(\bar{X} - \varepsilon n) = \bar{T}$. On the other hand, by Lemma 3.3, the size of maximum matching in G is at least $(2n - T)/2$. So by Lemma 3.1, $X \geq (2n - T)/2$, which means $\bar{X} \geq (2n - T)/2 - \varepsilon n$. So $\bar{T} \leq 2n - \frac{3}{4}(n - T/2 - 2\varepsilon n) < 1.25n + 0.375T + \delta n$. Since T is the cost of $(1, 2)$-TSP of G, which is at least n, we have $\bar{T} \leq (1.625 + \delta)T$. ◀

▶ Remark 3.6. The algorithm can be generalized to insertion-only streaming model. In insertion-only streaming model, we can compute a maximal matching pair as follows: we set M_1 and M_2 as empty set before the stream. Whenever an edge e comes, we first check if there is an edge in M_1 that shares an endpoint with e. If not, then we add e into M_1. Otherwise, we check if there is an edge in M_2 that shares and endpoint with e. If not, then we add e into M_2. So we get an algorithm that only uses $O(n)$ space to compute a maximal matching pair. We have the following corollary.

▶ **Corollary 3.7.** *There is an insertion-only streaming algorithm that estimates the cost of $(1,2)$-TSP of a graph G within a factor of 1.625 using $O(n)$ space.*

4 An $\Omega(n^2)$ Query Lower Bound for Approximation Schemes

In this section, we prove that there exists an $\varepsilon_0 > 0$, such that any query algorithm for graphic or $(1,2)$-TSP that returns a $(1+\varepsilon_0)$-approximate estimate of optimal cost, requires $\Omega(n^2)$ queries. In order to prove this, we design a new query model for the 3SAT problem and show an $\Omega(n^2)$ query lower bound for 3SAT in this model. We then use a reduction from 3SAT to $(1,2)$-TSP in [22] to prove the lower bound for $(1,2)$-TSP; with some additional changes, we also get an identical lower bound for graphic TSP.

The idea of proving query lower bound for APX-hard problems by reduction from 3SAT is similar to the idea used in [6], and we follow their general approach. However, in [6], the authors study lower bounds for problems in sparse graphs and hence the query model uses only neighbor queries. So in their query model, the lower bound for 3SAT is $\Omega(n)$. In order to prove an $\Omega(n^2)$ query lower bound in the pair query model, we need to design a new query model for 3SAT.

In the 3SAT problem, we are given a 3CNF instance on n variables, and the goal is to estimate the largest fraction of clauses that can be satisfied by any assignment. The algorithm is allowed to perform only one kind of query: is a variable x present in a clause c? If the answer is yes, then the algorithm is given the full information about all variables that appear in the clause c. The proof of the next theorem is deferred to the full version of this paper.

▶ **Theorem 4.1.** *For any $\varepsilon > 0$, any algorithm that with probability at least $2/3$ distinguishes between satisfiable 3CNF instances and 3CNF instances where at most $(7/8+\varepsilon)$ fraction of clauses can be satisfied, needs $\Omega(n^2)$ queries.*

4.1 Reduction from 3SAT to $(1,2)$-TSP

We will utilize an additional property of the hard instances of 3SAT in Theorem 4.1, namely, each variable occurs the same constant number of times where the constant only depends on ε. We denote the number of variables by n, the number of clauses by m, and the number of occurrences of each variable by k; thus $m = kn/3$.

We use the reduction in [22] to reduce a 3SAT instance to a $(1,2)$-TSP instance. In this reduction, there is a gadget for each variable and for each clause. Each of these gadgets has size at most $L = \Theta(k^2)$. Thus the $(1,2)$-TSP contains N vertices where $N \leq L(n+m) = \frac{L(k+3)n}{3}$. Let G_{x_j} be the gadget of variable x_j and G_{c_i} be the gadget of clause c_i. There is a ground graph which is the same for each 3SAT instance. Each variable gadget is connected with the gadgets for clauses that contain that variable. The reduction satisfies the following property. If the 3SAT instance is satisfiable, then the $(1,2)$-TSP instance contains a Hamilton cycle supported only on the weight 1 edges. On the other hand, if at most $m-\ell$ clauses can be satisfied in the 3SAT instance, the $(1,2)$-TSP cost is at least $N + \lceil \ell/2 \rceil$. Thus there is a constant factor separation between the optimal $(1,2)$-TSP cost in the two cases. However, what remains to be shown is that any query algorithm for $(1,2)$-TSP can also be directly simulated on the underlying 3SAT instance with a similar number of queries. The theorem below now follows by establishing this simulation.

▶ **Theorem 4.2.** *There is a constant ε_0 such that any algorithm that approximates the $(1,2)$-TSP cost to within a factor of $(1+\varepsilon_0)$ needs $\Omega(n^2)$ queries.*

Proof. We consider the following stronger queries for $(1,2)$-TSP: for any query (u,v), if u is in a vertex gadget G_{x_j} and v is in a clause gadget G_{c_i} (or vice versa) and x_j occurs in c_i in the 3SAT instance, then the algorithm is given all the edges incident on G_{c_i}. Otherwise the algorithm just learns if the there is an edge between u and v.

Let $\varepsilon = 1/16$, and let the values of k, L and N correspond to this choice for ε according to the redution in Section 4.1. Let $\varepsilon_0 = \frac{k}{32(k+3)L}$. Consider the $(1,2)$-TSP instance reduced from the 3SAT instance generated by the hard distribution in Theorem 4.1 with $\varepsilon = 1/16$. If the 3SAT instance is perfectly satisfiable, then the $(1,2)$-TSP instance has a Hamilton cycle of cost N. If the 3SAT instance satisfies at most $(15/16)$-fraction of clauses, then each Hamilton cycle in the $(1,2)$-TSP instance has cost at least

$$N + (1/8 - \varepsilon)m/2 = N + (1/8 - \varepsilon)kn/6 \geq (1 + \frac{(1/8-\varepsilon)k}{2(k+3)L})N = (1+\varepsilon_0)N$$

For any query (u,v) in the $(1,2)$-TSP instance, we can simulate it by at most one query in the corresponding 3SAT instance as follows: if u is in a vertex gadget G_{x_j} and v is in a clause gadget G_{c_i} (or vice versa), then we make a query of x_j and c_i in the 3SAT instance. If the 3SAT query returns YES and the full information of c_i, then we return all the edges incident on G_{c_i} according to the reduction rule and the full information of c_i. If the 3SAT query returns NO or (u,v) are not in a vertex gadget and a clause gadget respectively, we return YES if (u,v) is an edge in the ground graph and NO otherwise.

By Theorem 4.1, any algorithm that distinguishes a perfectly satisfiable 3SAT instance from an instance where at most $(15/16)$-fraction of the clauses can be satisfied needs $\Omega(n^2)$ queries. So any algorithm that distinguishes a $(1,2)$-TSP instance containing a Hamilton cycle of length N from an instance that has minimum Hamilton cycle of cost $(1+\varepsilon_0)N$ needs $\Omega(n^2)$ queries. ◀

4.2 $\Omega(n^2)$ Lower Bound for Graphic TSP

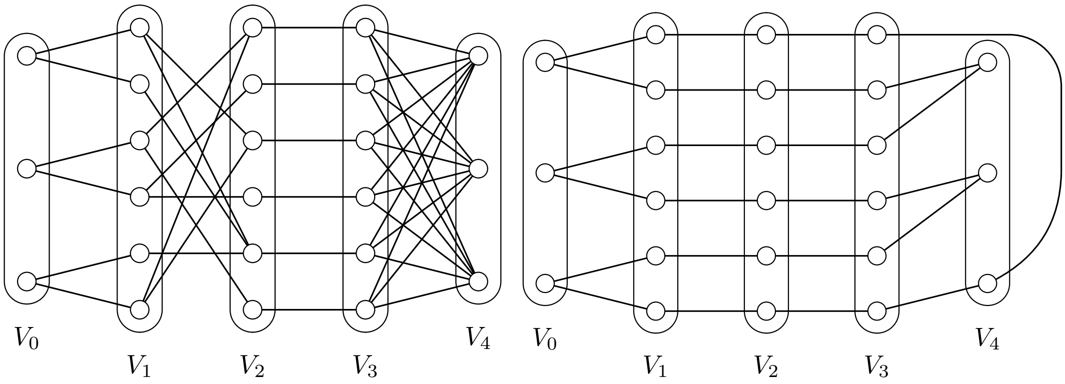
We can reduce an instance of $(1,2)$-TSP to an instance of graphic TSP by adding a new vertex that is adjacent to all other vertices. By doing so, any pair of vertices in the new graph has a distance at most 2. On the other hand, the cost of graphic TSP in the new graph differs by at most 1 from the cost of $(1,2)$-TSP in the old graph. So the $\Omega(n^2)$ query lower bound for $(1,2)$-TSP also holds for the graphic TSP problem.

5 A Reduction from Matching Size to TSP Cost Estimation

In this section, we give a reduction from the problem of estimating the maximum matching size in a bipartite graph to the problem of estimating the optimal $(1,2)$-TSP cost. An essentially identical reduction works for graphic TSP cost using the idea described in Section 4.2.

We will denote the size of the largest matching in a graph G by $\alpha(G)$. Given a bipartite graph $G(V, E)$ with n vertices on each side, we construct an instance $G'(V', E')$ of the $(1,2)$-TSP problem on $4n$ vertices such that the optimal TSP cost on G' is $5n - \alpha(G)$. Thus for any $\varepsilon \in [0, 1/5)$, any algorithm that can estimate $(1,2)$-TSP cost to within a $(1+\varepsilon)$-factor, also gives us an estimate of the matching size in G to within an additive error of $5\varepsilon n$.

We will now describe our construction of the graph G'. For clarity of exposition, we will describe G' as the graph that contains edges of cost 1 – all other edges have cost 2. Suppose the vertex set V of G consists of the bipartition $V_1 = \{v_1^1, v_2^1, \dots, v_n^1\}$ and $V_2 = \{v_1^2, v_2^2, \dots, v_n^2\}$. We construct the graph G' as follows: we start with the graph G, then add three sets of vertices V_0, V_3 and V_4, such that $V_0 = \{v_1^0, v_2^0, \dots, v_{n/2}^0\}$ with $n/2$ vertices, $V_3 = \{v_1^3, v_2^3, \dots, v_n^3\}$ with n vertices, and $V_4 = \{v_1^4, v_2^4, \dots, v_{n/2}^4\}$ with $n/2$ vertices. The graph G' will only have edges between V_j and V_{j+1} ($j = \{0, 1, 2, 3\}$). We will denote the set of edges between V_j and V_{j+1} as $E_{j,j+1}$. For any vertex $v_i^0 \in V_0$, it connects to v_{2i-1}^1 and v_{2i}^1 in V_1. $E_{1,2}$ has the same edges as the edges in G. Each vertex $v_i^2 \in V_2$ is connected to vertex v_i^3 in V_3, that is, vertices in V_2 and V_3 induce a perfect matching (identity matching). Finally, each vertex in V_3 is connected to all the vertices in V_4. See Figure 1(a) for an illustration.



(a) The illustration of G'.

(b) The illustration of tour T, where V_2 and V_3 are arranged with order $(v_{f(1)}^2, \dots, v_{f(6)}^2)$ and $(v_{f(1)}^3, \dots, v_{f(6)}^3)$.

Figure 1 An illustration of the reduction for $n = 6$.

The lemmas below relate matching size in G to $(1,2)$-TSP cost in G'.

▶ **Lemma 5.1.** *Let M be any matching in G. Then there is a $(1,2)$-TSP tour T in G' of cost at most $5n - |M|$.*

Proof. Let $f : [n] \to [n]$ be any bijection from $[n]$ to $[n]$ such that whenever a vertex v_i^1 is matched to a vertex v_j^2 in M, then $f(i) = j$. Consider the following $(1,2)$-TSP tour T: each vertex $v_i^0 \in V_0$ connects to v_{2i-1}^1 and v_{2i}^1 in T; each vertex $v_i^1 \in V_1$ connects to $v_{\lceil (i+1)/2 \rceil}^0$ and $v_{f(i)}^2$ in T. For any $v_{f(i)}^2 \in V_2$, it connects to v_i^1 and $v_{f(i)}^3$ in T. For any vertex $v_{f(i)}^3 \in V_3$, if $i > 1$, it connects to $v_{f(i)}^2$ and $v_{\lceil i/2 \rceil}^4$ in T; if $i = 1$, it connects to $v_{f(i)}^2$ and $v_{n/2}^4$ in T. See Figure 1(b) as an illustration. T is clearly a TSP-tour.

All edges in T are also edges in G' except for possibly some edges between V_1 and V_2. If v_i^1 is matched in M, then $(v_i^1, v_{f(i)}^2)$ is an edge in G', otherwise it is not in G' and thus has weight 2. So T only has $n - |M|$ weight 2 edges, which means T has cost at most $4n + n - |M| = 5n - |M|$. ◀

▶ **Lemma 5.2.** *For any $(1,2)$-TSP tour T in G, T has cost at least $5n - \alpha(G)$.*

To prove Lemma 5.2, we first prove an auxiliary claim.

▷ Claim 5.3. Suppose $G = (V_1, V_2, E)$ is a bipartite graph which has maximum size $\alpha(G)$. For any 2-degree subgraph H of G, if there are at most X vertices in V_1 has degree 2 in H, then there are at most $\alpha(G) + X$ vertices in V_2 which have degree at least 1 in H. Similarly, if there are at most X vertices in V_2 has degree 2 in H, then there are at most $\alpha(G) + X$ vertices in V_1 which have degree at least 1 in H.

Proof. If there are at most X vertices in V_1 has degree 2 in H. We construct H' by deleting an arbitrary edge on each degree 2 vertex in V_1, then construct H'' by deleing an arbitrary edge on each degree 2 vertex in V_2. Since H'' does not have degree two vertex, it is a matching of G. So the number of degree 1 vertices in V_2 in H'' is at most $\alpha(G)$. On the other hand, any vertex in V_2 which has degree at least 1 in H' also has degree 1 in H''. So there are at most $\alpha(G)$ vertices of degree at least 1 in V_2 in H'. Furthermore, since there are only X vertices of degree 2 in V_1 in H, we delete at most X edges in H when constructing H'. So H' has at most X more isolate vertices in V_2 than in H, which means H has at most $\alpha(G) + X$ vertices with degree at least 1 in V_2.

The second part of the claim follows via a similar argument as the first part of the claim. ◁

Proof of Lemma 5.2. Let a_{01} be the number of edges in $T \cap E_{0,1}$, $a_{2,3}$ be the number of edges in $T \cap E_{3,4}$. Let G^X be the intersection graph of G and T. Since the vertices in V_0 only connect to the vertices in V_1 in G', and any vertex in T has degree 2, there are at least $n - a_{01}$ edges incident on V_0 in T are not an edge in G'. On the other hand, since any vertex in V_1 is incident on at at most 1 edge in $E_{0,1}$, there are at least a_{01} vertices in V_1 is connected to a vertex in V_0 in T, which means there are at most $n - a_{01}$ vertices in V_1 has degree 2 in G^X. By Claim 5.3, there are at most $n - a_{01} + \alpha(G)$ vertices in V_2 has edge in G^X. For any isolate vertex in V_2 in T, it has only one edge in G' connecting to V_3, so this vertex must incident on an edge in T which is not in G'. So there are at least $n - (n - a_{01} + \alpha(G)) = \alpha(G) - a_{01}$ edges incident on V_2 in T which is not in G'.

There are $2n$ edges incident on V_3 in T, but among them, there are only a_{23} edges between V_2 and V_3 which is also in G', and there are at most n edges between V_3 and V_4 in T since each vertex has degree only 2. So there are at least $2n - n - a_{23} = n - a_{23}$ edges incident on V_3 which is not in G'. On the other hand, since any vertex in V_2 is incident on at at most 1 edge in $E_{2,3}$, there are at least a_{23} vertices in V_2 is connected to a vertex in V_3 in T, which means there are at most $n - a_{23}$ vertices in V_2 has degree 2 in G^X. By Claim 5.3, there are at most $n - a_{23} + \alpha(G)$ vertices in V_1 has edge in G^X. For any isolate vertex in V_1 in T, it has only one edge in G' connecting to V_0, so this vertex must incident on an edge in T which is not in G'. So there are at least $n - (n - a_{23} + \alpha(G)) = a_{23} - \alpha(G)$ edges incident on V_1 in T which is not in G'.

Since any edge has two endpoints, the number of edges in T but not in G' is at least $((n - a_{01}) + (a_{01} - \alpha(G)) + (n - a_{23}) + (a_{23} - \alpha(G)))/2 = n - \alpha(G)$, which means T has cost at least $4n + n - \alpha(G) = 5n - \alpha(G)$. ◀

▶ **Corollary 5.4.** *For any $\varepsilon \in [0, 1/5)$, any algorithm that can estimate $(1, 2)$-TSP cost to within a $(1 + \varepsilon)$-factor, can be used to estimate the size of a largest matching in a bipartite graph G on $2n$ vertices to within an additive error of $5\varepsilon n$.*

Proof. We use the reduction above to construct a $(1, 2)$-TSP instance G' on $4n$ vertices. By Lemmas 5.1 and 5.2, the optimal TSP cost for G' is $5n - \alpha(G)$. We now run the $(1 + \varepsilon)$-approximation algorithm for $(1, 2)$-TSP on graph G' (note that the reduction can be simulated in each of neighbor query model, pair query model, and the streaming model without altering the asymptotic number of queries used). Suppose the output is C which satisfies $(1-\varepsilon)(5n-\alpha(G)) \leq C \leq (1+\varepsilon)(5n-\alpha(G))$, which means $5n-\alpha(G)-5\varepsilon n < C < 5n-\alpha(G)+5\varepsilon n$. Let $\hat{\alpha} = 5n - C$, we have $\alpha(G) - 5\varepsilon n < \hat{\alpha} < \alpha(G) + 5\varepsilon n$. ◀

References

1 https://sublinear.info/71.

2 Anna Adamaszek, Matthias Mnich, and Katarzyna Paluch. New approximation algorithms for (1, 2)-tsp. In *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

3 Hyung-Chan An, Robert D. Kleinberg, and David B. Shmoys. Improving christofides' algorithm for the s-t path TSP. *J. ACM*, 62(5):34:1–34:28, 2015.

4 Sepehr Assadi, Sanjeev Khanna, and Yang Li. On estimating maximum matching size in graph streams. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1723–1742. SIAM, 2017.

5 Piotr Berman and Marek Karpinski. 8/7-approximation algorithm for (1, 2)-tsp. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 641–648. Society for Industrial and Applied Mathematics, 2006.

6 Andrej Bogdanov, Kenji Obata, and Luca Trevisan. A lower bound for testing 3-colorability in bounded-degree graphs. In *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.*, pages 93–102. IEEE, 2002.

7 Bernard Chazelle, Ronitt Rubinfeld, and Luca Trevisan. Approximating the minimum spanning tree weight in sublinear time. *SIAM J. Comput.*, 34(6):1370–1379, 2005.

8 Chandra Chekuri and Kent Quanrud. Approximating the held-karp bound for metric TSP in nearly-linear time. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 789–800, 2017.

9 Chandra Chekuri and Kent Quanrud. Fast approximations for metric-TSP via linear programming. *CoRR*, abs/1802.01242, 2018. arXiv:1802.01242.

10 Artur Czumaj and Christian Sohler. Estimating the weight of metric minimum spanning trees in sublinear time. *SIAM Journal on Computing*, 39(3):904–922, 2009.

11 Zhihan Gao. On the metric s-t path traveling salesman problem. *SIAM Review*, 60(2):409–426, 2018.

12 Oded Goldreich and Dana Ron. Property testing in bounded degree graphs. *Algorithmica*, 32(2):302–343, 2002.

13 Michael Kapralov, Slobodan Mitrović, Ashkan Norouzi-Fard, and Jakab Tardos. Space efficient approximation to maximum matching size from uniform edge samples. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1753–1772. SIAM, 2020.

14 Marek Karpinski, Michael Lampis, and Richard Schmied. New inapproximability bounds for TSP. *J. Comput. Syst. Sci.*, 81(8):1665–1677, 2015.

15 Matthias Mnich and Tobias Mömke. Improved integrality gap upper bounds for traveling salesperson problems with distances one and two. *European Journal of Operational Research*, 266(2):436–457, 2018.

16 Tobias Mömke and Ola Svensson. Approximating graphic TSP by matchings. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 560–569, 2011. doi:10.1109/FOCS.2011.56.

17 Tobias Mömke and Ola Svensson. Removing and adding edges for the traveling salesman problem. *Journal of the ACM (JACM)*, 63(1):2, 2016.

18 Marcin Mucha. $\frac{13}{9}$-approximation for graphic tsp. *Theory of computing systems*, 55(4):640–657, 2014.

19 Huy N. Nguyen and Krzysztof Onak. Constant-time approximation algorithms via local improvements. In *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*, pages 327–336, 2008.

20 Krzysztof Onak, Dana Ron, Michal Rosen, and Ronitt Rubinfeld. A near-optimal sublinear-time algorithm for approximating the minimum vertex cover size. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 1123–1131. Society for Industrial and Applied Mathematics, 2012.

21 Krzysztof Onak, Dana Ron, Michal Rosen, and Ronitt Rubinfeld. Personal communication, 2019.

22 Christos H Papadimitriou and Mihalis Yannakakis. The traveling salesman problem with distances one and two. *Mathematics of Operations Research*, 18(1):1–11, 1993.

23 Michal Parnas and Dana Ron. Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms. *Theor. Comput. Sci.*, 381(1-3):183–196, 2007.

24 András Sebö and Anke van Zuylen. The salesman's improved paths: A 3/2+1/34 approximation. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 118–127, 2016.

25 András Sebö and Jens Vygen. Shorter tours by nicer ears: 7/5-approximation for the graph-tsp, 3/2 for the path version, and 4/3 for two-edge-connected subgraphs. *Combinatorica*, 34(5):597–629, 2014.

26 Vera Traub and Jens Vygen. Beating the integrality ratio for s-t-tours in graphs. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 766–777, 2018.

27 Vera Traub and Jens Vygen. Approaching 3/2 for the s-t-path TSP. *J. ACM*, 66(2):14:1–14:17, 2019. `doi:10.1145/3309715`.

28 Jens Vygen. New approximation algorithms for the tsp, 2012.

29 Douglas B. West. *Introduction to graph theory.* Prentice-Hall Inc., 1996.

30 Yuichi Yoshida, Masaki Yamamoto, and Hiro Ito. Improved constant-time approximation algorithms for maximum matchings and other optimization problems. *SIAM Journal on Computing*, 41(4):1074–1093, 2012.

Computational Complexity of the α-Ham-Sandwich Problem

Man-Kwun Chiu
Institut für Informatik, Freie Universität Berlin, Germany
chiumk@inf.fu-berlin.de

Aruni Choudhary
Institut für Informatik, Freie Universität Berlin, Germany
arunich@inf.fu-berlin.de

Wolfgang Mulzer
Institut für Informatik, Freie Universität Berlin, Germany
mulzer@inf.fu-berlin.de

Abstract

The classic Ham-Sandwich theorem states that for any d measurable sets in $\mathbb{R}^d$, there is a hyperplane that bisects them simultaneously. An extension by Bárány, Hubard, and Jerónimo [DCG 2008] states that if the sets are convex and *well-separated*, then for any given $\alpha_1, \dots, \alpha_d \in [0, 1]$, there is a unique oriented hyperplane that cuts off a respective fraction $\alpha_1, \dots, \alpha_d$ from each set. Steiger and Zhao [DCG 2010] proved a discrete analogue of this theorem, which we call the *α-Ham-Sandwich theorem.* They gave an algorithm to find the hyperplane in time $O(n(\log n)^{d-3})$, where n is the total number of input points. The computational complexity of this search problem in high dimensions is open, quite unlike the complexity of the Ham-Sandwich problem, which is now known to be PPA-complete (Filos-Ratsikas and Goldberg [STOC 2019]).

Recently, Fearnley, Gordon, Mehta, and Savani [ICALP 2019] introduced a new sub-class of CLS (Continuous Local Search) called *Unique End-of-Potential Line* (UEOPL). This class captures problems in CLS that have unique solutions. We show that for the α-Ham-Sandwich theorem, the search problem of finding the dividing hyperplane lies in UEOPL. This gives the first non-trivial containment of the problem in a complexity class and places it in the company of classic search problems such as finding the fixed point of a contraction map, the unique sink orientation problem and the P-matrix linear complementarity problem.

2012 ACM Subject Classification Theory of computation → Computational geometry; Theory of computation → Complexity classes; Theory of computation → Problems, reductions and completeness

Keywords and phrases Ham-Sandwich Theorem, Computational Complexity, Continuous Local Search

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.31

Category Track A: Algorithms, Complexity and Games

Related Version A full version of the paper is available at `https://arxiv.org/abs/2003.09266`.

Funding Supported in part by ERC StG 757609.

1 Introduction

The Ham-Sandwich Theorem [41] is a classic result about partitioning sets in high dimensions: for any d measurable sets $S_1, \dots, S_d \subset \mathbb{R}^d$ in d dimensions, there is an oriented hyperplane H that simultaneously *bisects* $S_1, \dots, S_d$. More precisely, if H^+, H^- are the closed half-spaces bounded by H, then for $i = 1, \dots, d$, the measure of $S_i \cap H^+$ equals the measure of $S_i \cap H^-$. The traditional proof goes through the Borsuk-Ulam Theorem [30]. The Ham-Sandwich Theorem is a cornerstone of geometry and topology, and it has found applications in other areas of mathematics.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 31; pp. 31:1–31:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Let $[n] = \{1, \dots, n\}$. The *discrete* Ham-Sandwich Theorem [28,30] states that for any d finite point sets $P_1, \dots, P_d \subset \mathbb{R}^d$ in d dimensions, there is an oriented hyperplane H such that H bisects each P_i, i.e., for $i \in [d]$, we have $\min\{|P_i \cap H^+|, |P_i \cap H^-|\} \geq \lceil |P_i|/2 \rceil$. We denote the associated search problem as Ham-Sandwich. Lo, Matoušek, and Steiger [28] gave an $n^{O(d)}$-time algorithm for Ham-Sandwich. They also provided a linear-time algorithm for points in $\mathbb{R}^3$, under additional constraints.

There are many alternative and more general variants of both the continuous and the discrete Ham-Sandwich Theorem. For example, Bárány and Matoušek [5] derived a version where measures in the plane can be divided into any (possibly different) ratios by *fans* instead of hyperplanes (lines). A discrete variant of this result was given by Bereg [7]. Schnider [37] and Karasev [27] studied generalizations in higher dimensions. Recently Barba, Pilz, and Schnider [6] showed that four measures in the plane can be bisected with two lines. Higher dimensional generalizations of this result were presented in [9,25]. Zivaljević and Vrećica [44] and independently, Dol'nikov [19] proved a result called the Center Transversal Theorem that interpolates between the Ham-Sandwich Theorem and the Centerpoint Theorem [35]. There is also a no-dimensional version [14] for the Center Transversal Theorem. Schnider [38] presented a generalization based on this result among others.

Here, we focus on a version that allows for dividing the sets into arbitrary given ratios instead of simply bisecting them. The sets $S_1, \dots, S_d \subset \mathbb{R}^d$ are *well-separated* if every selection of them can be strictly separated from the others by a hyperplane. Bárány, Hubard, and Jerónimo [4] showed that if $S_1, \dots, S_d$ are well-separated and convex, then for any given reals $\alpha_1, \dots, \alpha_d \in [0, 1]$, there is a unique hyperplane that divides $S_1, \dots, S_d$ in the ratios $\alpha_1, \dots, \alpha_d$, respectively. Their proof goes through Brouwer's Fixed Point Theorem. Steiger and Zhao [40] formulated a discrete version. In this setup, $S_1, \dots, S_d$ are finite point sets. Again, we need that the (convex hulls of the) S_i are well-separated. Additionally, we require that the S_i follow a weak version of general position. Let $\alpha_1, \dots, \alpha_d \in \mathbb{N}$ be d integers with $1 \leq \alpha_i \leq |S_i|$, for $i \in [d]$. Then, there is a unique oriented hyperplane H that passes through one point from each S_i and has $|H^+ \cap S_i| = \alpha_i$, for $i \in [d]$ [40]. In other words, H simultaneously cuts off α_i points from S_i, for $i \in [d]$. This statement does not necessarily hold if the sets are not well-separated, see Figure 1 for an example.

Steiger and Zhao called their result the *Generalized Ham-Sandwich Theorem*, yet it is not a strict generalization of the classic Ham-Sandwich Theorem. Their result requires that the point sets obey well-separation and weak general position, while the classic theorem always holds without these assumptions. Therefore, we call this result the *α-Ham-Sandwich theorem*, for a clearer distinction. Set $n = \sum_{i \in [d]} |S_i|$. Steiger and Zhao gave an algorithm that computes the dividing hyperplane in $O\left(n(\log n)^{d-3}\right)$ time, which is exponential in d. Later, Bereg [8] improved this algorithm to achieve a running time of $n2^{O(d)}$, which is linear in n but still exponential in d. We denote the associated computational search problem of finding the dividing hyperplane as Alpha-HS.

No polynomial algorithms are known for Ham-Sandwich and for Alpha-HS if the dimension is not fixed, and the notion of approximation is also not well-explored. Despite their superficial similarity, it is not immediately apparent whether the two problems are comparable in terms of their complexity. Due to the additional requirements on an input for Alpha-HS, an instance of Ham-Sandwich may not be reducible to Alpha-HS in general.

A dividing hyperplane for Alpha-HS is guaranteed to exist if the sets satisfy the conditions of well-separation and (weak) general position. Therefore, the search problem Alpha-HS is total, that is, there is a solution for every valid instance. In general, such problems are modelled by the complexity class TFNP (Total Function Nondeterministic Polynomial) of

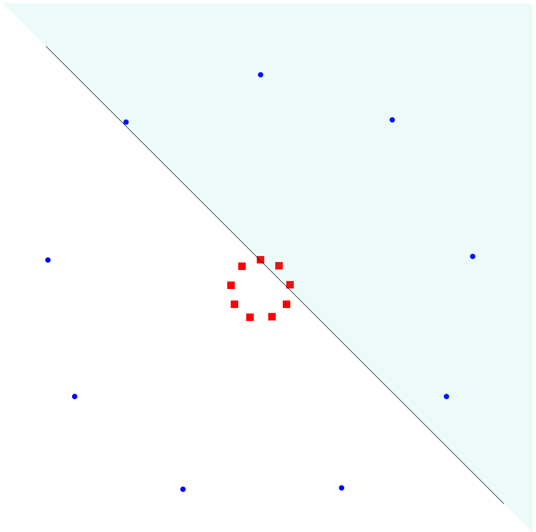


Figure 1 The red (square) and the blue (round) point sets are not well-separated. Every halfplane that contains three red points must contain at least five blue points. Thus, there is no halfplane that contains exactly three red and three blue points.

NP-search problems that always admit a solution. Two popular subclasses of TFNP, originally defined by Papadimitriou [34], are PPA (Polynomial Parity Argument) and its sub-class PPAD (Polynomial Parity Arguments on Directed graphs). These classes contain total search problems where the existence of a solution is based on a parity argument in an undirected or in a directed graph, respectively. Another sub-class of TFNP is PLS (polynomial local search). It models total search problems where the solutions can be obtained as minima in a local search process, while the number of steps in the local search may be exponential in the input size. The class PLS was introduced by Johnson, Papadimitriou, and Yannakakis [26]. A noteworthy sub-class of PPAD $\cap$ PLS is CLS (continuous local search) [18]. It models similar local search problems over a continuous domain using a continuous potential function.

Up to very recently, these complexity classes had mostly been studied in the context of algorithmic game theory. These classes have also found relevance in the study of fairness [33] and markets [10, 12]. However, there have been increasing efforts towards mapping the complexity landscape of existence theorems in high-dimensional discrete geometry. Computing an approximate solution for the search problem associated with the Borsuk-Ulam Theorem is in PPA. In fact, this problem is complete for this class. The discrete analogue of the Borsuk-Ulam Theorem, Tucker's Lemma [42], is also PPA-complete [1, 34]. Therefore, since the traditional proof of the Ham-Sandwich Theorem goes through the Borsuk-Ulam Theorem, it follows that Ham-Sandwich lies in PPA. In fact, Filos-Ratsikas and Goldberg [21] recently showed that Ham-Sandwich is complete for PPA. The (presumably smaller) class PPAD is associated with fixed-point type problems: computing an approximate Brouwer fixed point is a prototypical complete problem for PPAD. The discrete analogue of Brouwer's Fixed Point Theorem, Sperner's Lemma, is also complete for PPAD [34]. The computational version of the Hairy Ball Theorem has recently been shown to be PPAD-complete [24]. In a celebrated result, the relevance of PPAD for algorithmic game theory was made clear when it turned out that computing a Nash-equilibrium in a three player game is PPAD-complete [17]. Subsequently, this was also shown for the two player game [11]. In discrete geometry, finding a solution to the Colorful Carathéodory problem [3] was shown to lie in the intersection

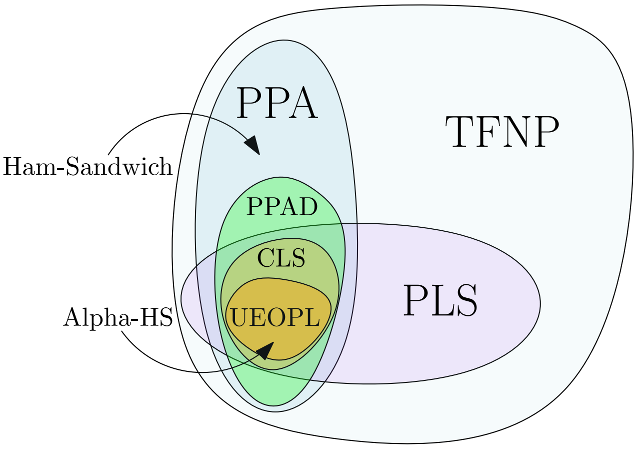


Figure 2 The hierarchy of complexity classes.

PPAD ∩ PLS [31,32]. This further implies that finding a *Tverberg* partition (and computing a centerpoint) also lies in the intersection [29,36,43]. The problem of computing the (unique) fixed point of a contraction map is known to lie in CLS [18].

Recently, at ICALP 2019, Fearnley, Gordon, Mehta, and Savani defined a sub-class of CLS that represents a family of total search problems with unique solutions [20]. They named the class *Unique End of Potential Line* (UEOPL) and defined it through the canonical complete problem UniqueEOPL. This problem is modelled as a directed graph. There are polynomially-sized Boolean circuits that compute the successor and predecessor of each node, and a potential value that always increases on a directed path. There is supposed to be only a single vertex with no predecessor (*start of line*). Under these conditions, there is a unique path in the graph that ends on a vertex (called *end of line*) with the highest potential along the path. This vertex is the solution to UniqueEOPL. Since the uniqueness of the solution is guaranteed only under certain assumptions, such a formulation is called a *promise* problem. Since there seems to be no efficient way to verify the assumptions, the authors allow two possible outcomes of the search algorithm: either report a correct solution, or provide any solution that was found to be in violation of the assumptions. This formulation turns UniqueEOPL into a *non-promise* problem and places it in TFNP, since a correct solution is bound to exist when there are no violations, and otherwise a violation can be reported as a solution. Fearnley et al. [20] also introduced the concept of a *promise-preserving* reduction between two problems A and B, such that if an instance of A has no violations, then the reduced instance of B is also free of violations. This notion is particularly meaningful for non-promise problems.

Contributions. We provide the first non-trivial containment in a complexity class for the α-Ham-Sandwich problem by locating it in UEOPL. More precisely, we formulate Alpha-HS as a non-promise problem in which we allow for both valid solutions representing the correct dividing hyperplane, as well as violations accounting for the lack of well-separation and/or (weak) general position of the input point sets. A precise formulation of the problem is given in Definition 4 in Section 2. We then show a promise-preserving reduction from Alpha-HS to UniqueEOPL. This implies that Alpha-HS lies in UEOPL, and hence in CLS $\subseteq$ PPAD ∩ PLS. See Figure 2 for a pictorial description.

It is not surprising to discover that Alpha-HS lies in PPAD, since the proof of the continuous version in [4] was based on Brouwer's Fixed Point Theorem. The observation that it also lies in PLS is new and noteworthy, putting Alpha-HS into the reach of local search algorithms. In contrast, given our current understanding of total search problems, it is unlikely that the problem Ham-Sandwich would be in PLS.

Since Alpha-HS lies in PPAD $\subseteq$ PPA, it is computationally easier than Ham-Sandwich, which is PPA-complete. This implies the existence of a polynomial-time reduction from Alpha-HS to Ham-Sandwich. A reduction in the other direction is unlikely. It thus turns out that well-separation brings down the complexity of the problem significantly.

Often, problems in TFNP come in the guise of a polynomial-size Boolean circuit with some property. In contrast, Alpha-HS is a purely geometric problem that has no circuit in its problem definition. Apart from the P-Matrix Linear complementarity problem, this is one of the few problems in UEOPL and hence in CLS that do not have a description in terms of circuits.

Our local-search formulation is based on the intuition of rotating a hyperplane until we reach the desired solution. We essentially start with a hyperplane that is tangent to the convex hull of each input set, and we deterministically rotate the hyperplane until it hits a new point. This rotation can be continued whenever the hyperplane hits a new point, until we reach the correct dividing hyperplane. In other words, we can follow a local-search argument to find the solution. We show that this sequence of rotations can be modelled as a canonical path in a grid graph, and we give a potential function that guides the rotation and always increases along this path. Every violation of well-separation and (weak) general position can destroy this path. Furthermore, no efficient methods to verify these two assumptions are known. This poses a major challenge in handling the violations. One of our main technical contributions is to handle the violation solutions concisely.

An alternative approach would have been to look at the dual space of points where we get an arrangement of hyperplanes. The dividing hyperplane could then be found by looking at the correct level sets of the arrangement. However, this approach has the problem that the orientations of the hyperplanes in the original space and the dual space are not consistent. This complicates the arguments on the level sets, so we found it more convenient to use our notion of rotating hyperplanes. We show that we can maintain a consistent orientation throughout the rotation, and an inconsistent rotation is detected as a violation of the promise.

Outline of the paper. We discuss the background about the α-Ham-sandwich Theorem and UniqueEOPL in Section 2. In Section 3, we describe our instance of Alpha-HS and give an overview of the reduction and violation-handling. We conclude in Section 4. The technical details of the reduction and some proofs can be found in the full version of the paper in [13].

2 Preliminaries

2.1 The α-Ham-Sandwich problem

For conciseness, we describe the discrete version of α-Ham-Sandwich Theorem [40] here. The continuous version [4] follows a similar formulation.

Let $P_1, \dots, P_d \subset \mathbb{R}^d$ be a collection of d finite point sets. Let $n_1, \dots, n_d$ denote the sizes of $P_1, \dots, P_d$, respectively. For each $i \in [d]$ we say that the point set P_i represents a unique color and let $P := P_1 \cup \cdots \cup P_d$ denote the union of all the points. A set of points $\{p_1, \dots, p_m\}$ is said to be *colorful* if there are no two points p_i, p_j both from the same color. Indeed a colorful point set can have size at most d.

Weak general position. We say that P has *very weak general position* [40], if for every choice of points $x_1 \in P_1, \ldots, x_d \in P_d$, the affine hull of the set $\{x_1, \ldots, x_d\}$ is a $(d-1)$-flat and does not contain any other point of P. This definition is sufficient for the result of Steiger and Zhao, where they simply call it as weak general position. Of course, this definition of weak general position has no restriction on sets $\{x_1, \ldots, x_d\}$ that contain multiple points from the same color. To simplify our proofs we need a slightly stronger form of general position. We discuss how to deal with very weak general position at the end of Section 3. We say that P has *weak general position* if the above restriction also applies to sets having exactly $d-1$ colors. That means, each color may contribute at most one point to the set, except perhaps one color which is allowed to contribute two points. A certificate for checking violations of weak general position is a set of $d+1$ points whose affine hull has dimension at most $d-1$, with at least $d-1$ colors in the set. Testing whether a point set is in general position can be shown to be NP-Hard, using the result in [23]. It is easy to see that when $d = 2$, weak general position is equivalent to general position.

Well-separation. The point set P is said to be *well-separated* [4, 40], if for every choice of points $y_1 \in \text{conv}(P_{i_1}), \ldots, y_k \in \text{conv}(P_{i_k})$, where $i_1, \ldots, i_k$ are distinct indices and $1 \leq k \leq d$, the affine hull of $\{y_1, \ldots, y_k\}$ is a $(k-1)$-flat. An equivalent definition is as follows: P is well-separated if and only if for every disjoint pair of index sets $I, J \subset [d]$, there is a hyperplane that separates the set $\{\cup_{i \in I} P_i\}$ from the set $\{\cup_{j \in J} P_j\}$ strictly. Formally:

▶ **Lemma 1.** *Let $y_1, \ldots, y_d$ be a colorful set of points in the corresponding $\text{conv}(P_i)$. The affine hull of $y_1, \ldots, y_d$ has dimension $d-2$ or less if and only if there is a partition of $[d]$ into index sets I, J such that $\text{conv}(\{\cup_{i \in I} P_i\}) \cap \text{conv}(\{\cup_{j \in J} P_j\}) \neq \emptyset$.*

Given such a colorful set, the partition of $[d]$ can be computed in $\text{poly}(n, d)$ time. Vice-versa, given such a partition, the colorful set can be computed in $\text{poly}(n, d)$ time.

A certificate for checking violations of well-separation is a colorful set $\{x_1, \ldots, x_d\}$ whose affine hull has dimension at most $d-2$. Another certificate is a partition $I, J \subset [d]$ such that the convex hulls of the indexed sets are not separable. Due to Lemma 1, both certificates are equivalent and either can be converted into the other in polynomial time. To the best of our knowledge, the complexity of testing well-separation is unknown.

Given any set of positive integers $\{\alpha_1, \ldots, \alpha_d\}$ satisfying $1 \leq \alpha_i \leq n_i$, $i \in [d]$, an $(\alpha_1, \ldots, \alpha_d)$*-cut* is an oriented hyperplane H that contains one point from each color and satisfies $|H^+ \cap P_i| = \alpha_i$ for $i \in [d]$, where H^+ is the closed positive half-space defined by H.

▶ **Theorem 2** (α-Ham-Sandwich Theorem [40]). *Let $P_1, \ldots, P_d$ be finite, well-separated point sets in $\mathbb{R}^d$. Let $\alpha = (\alpha_1, \ldots, \alpha_d)$ be a vector, where $\alpha_i \in [n_i]$ for $i \in [d]$.*

1. *If an α-cut exists, then it is unique.*
2. *If P has weak general position, then an α-cut exists for each choice of α.*

That means, every colorful d-tuple of P represents an oriented hyperplane that corresponds to exactly one α-vector. Steiger and Zhao [40] also presented an algorithm to compute the cut in $O(n(\log n)^{d-3})$ time, where $n = \sum_{i=1}^{d} n_i$. The algorithm proceeds inductively in dimension and employs a prune-and-search technique. Bereg [8] improved the pruning step to improve the runtime to $n2^{O(d)}$.

2.2 Unique End of Potential Line

We briefly explain the *Unique end of potential line* problem that was introduced in [20]. More details about the problem and the associated class can be found in the above reference.

▶ **Definition 3** (from [20]). *Let n, m be positive integers. The input consists of*

- *a pair of Boolean circuits $\mathbb{S}, \mathbb{P} : \{0,1\}^n \to \{0,1\}^n$ such that $\mathbb{P}(0^n) = 0^n \neq \mathbb{S}(0^n)$, and*
- *a Boolean circuit $\mathbb{V} : \{0,1\}^n \to \{0,1,\ldots,2^m - 1\}$ such that $\mathbb{V}(0^n) = 0$,*

each circuit having $\text{poly}(n,m)$ *size. The* UniqueEOPL *problem is to report one of the following:*

(U1). *A point $v \in \{0,1\}^n$ such that $\mathbb{P}(\mathbb{S}(v)) \neq v$.*

(UV1). *A point $v \in \{0,1\}^n$ such that $\mathbb{S}(v) \neq v$, $\mathbb{P}(\mathbb{S}(v)) = v$, and $\mathbb{V}(\mathbb{S}(v)) - \mathbb{V}(v) \leq 0$.*

(UV2). *A point $v \in \{0,1\}^n$ such that $\mathbb{S}(\mathbb{P}(v)) \neq v \neq 0^n$.*

(UV3). *Two points $v, u \in \{0,1\}^n$ such that $v \neq u$, $\mathbb{S}(v) \neq v$, $\mathbb{S}(u) \neq u$, and either $\mathbb{V}(v) = \mathbb{V}(u)$ or $\mathbb{V}(v) < \mathbb{V}(u) < \mathbb{V}(\mathbb{S}(v))$.*

The problem defines a graph G with up to 2^n vertices. Informally, $\mathbb{S}(\cdot), \mathbb{P}(\cdot), \mathbb{V}(\cdot)$ represent the *successor*, *predecessor* and *potential* functions that act on each vertex in G. The in-degree and out-degree of each vertex is at most one. There is an edge from vertex u to vertex v if and only if $\mathbb{S}(u) = v$, $\mathbb{P}(v) = u$ and $\mathbb{V}(u) < \mathbb{V}(v)$. Thus, G is a directed acyclic path graph (line) along which the potential strictly increases. The condition $\mathbb{S}(\mathbb{P}(x)) \neq x$ means that x is the start of the line, $\mathbb{P}(\mathbb{S}(x)) \neq x$ means that x is the end of the line, and $\mathbb{P}(\mathbb{S}(x)) = x$ occurs when x is neither. The vertex 0^n is a given start of the line in G.

(U1) is a solution representing the end of a line. **(UV1)**, **(UV2)** and **(UV3)** are violations. **(UV1)** gives a vertex v that is not the end of line, and the potential of $\mathbb{S}(v)$ is not strictly larger than that of v, which is a violation of our assumption that the potential increases strictly along the line. **(UV2)** gives a vertex that is the start of a line, but is not 0^n. **(UV3)** shows that G has more than one line, which is witnessed by the fact that v and u cannot lie on the same line if they have the same potential, or if the potential of u is sandwiched between that of v and the successor of v. Under the promise that there are no violations, G is a single line starting at 0^n and ending at a vertex that is the unique solution. UniqueEOPL is formulated in the non-promise setting, placing it in the class TFNP.

The complexity class UEOPL represents the class of problems that can be reduced in polynomial time to UniqueEOPL. This has been shown to lie in CLS and contains three classical problems in [20]: finding the fixed point of a piecewise-linear contraction map, solving the P-Matrix Linear complementarity problem, and finding the unique sink of a directed graph (with arbitrary edge orientations such that each face has a sink) on the 1-skeleton of a hypercube. Note that finding the fixed point of a contraction map is in CLS [18], but is not known to lie in UEOPL.

A notion of *promise-preserving* reductions is also defined in [20]. Let X and Y be two problems both having a formulation that allows for valid and violation solutions. A reduction from X to Y is said to be promise-preserving, if whenever it is promised that X has no violations, then the reduced instance of Y also has no violations. Thus a promise-preserving reduction to UniqueEOPL would mean that whenever the original problem is free of violations, then the reduced instance always has a single line that ends at a valid solution.

2.3 Formulating the search problem

We formalize the search problem for α-Ham-Sandwich in a non-promise setting:

▶ **Definition 4** (Alpha-HS). *Given d finite sets of points $P = P_1 \cup \ldots \cup P_d$ in $\mathbb{R}^d$ and a vector $(\alpha_1, \ldots, \alpha_d)$ of positive integers such that $\alpha_i \leq |P_i|$ for all $i \in [d]$, the* Alpha-HS *problem is to find one of the following:*

(G1). *An $(\alpha_1, \ldots, \alpha_d)$-cut.*

(GV1). *A subset of P of size $d+1$ and at least $d-1$ colors that lies on a hyperplane.*

(GV2). *A disjoint pair of sets $I, J \subset [d]$ such that $\text{conv}(\{\cup_{i\in I} P_i\}) \cap \text{conv}(\{\cup_{j\in J} P_j\}) \neq \emptyset$.*

Here a solution of type **(G1)** corresponds to a solution representing a valid cut, while solutions of type **(GV1)** and **(GV2)** refer to violations of weak general position and well-separation, respectively. From Theorem 2 we see that a valid solution is guaranteed if no violations are presented, which shows that Alpha-HS is a total search problem.

3 Alpha-HS is in UEOPL

In this section we describe our instance of Alpha-HS in more detail and briefly outline a reduction to UniqueEOPL.

Setup. The input consists of d finite point sets $P_1, \dots, P_d \subset \mathbb{R}^d$ each representing a unique color, of sizes $n_1, \dots, n_d$, respectively, and a vector of integers $\alpha = (\alpha_1, \dots, \alpha_d)$ such that $\alpha_i \in [n_i]$ for each $i \in [d]$. Let k denote the number of coordinates of α that are not equal to 1. Without loss of generality, we assume that $\{\alpha_1, \dots, \alpha_k\}$ are the non-unit entries in α. Let P denote the union $P_1 \cup \dots \cup P_d$. For each $i \in [d]$ we define an arbitrary order $\prec_i$ on P_i. Concatenating the orders $\prec_1, \prec_2, \dots, \prec_d$ in sequence gives a global order $\prec$ on P. That means, $p \prec q$ if $p \in P_i, q \in P_j$ and $i < j$ or $p, q \in P_j$ and $p \prec_j q$.

We follow the notation of [40] to define the orientation of a hyperplane in $\mathbb{R}^d$ that has a non-empty intersection with the convex hull of each P_i. For any hyperplane H passing via $\{x_1 \in \text{conv}(P_1), \dots, x_d \in \text{conv}(P_d)\}$, the normal is the unit vector $\hat{n} \in \mathbb{R}^d$ that satisfies $\langle x_i, \hat{n} \rangle = t$ for some fixed $t \in \mathbb{R}$ and each $i \in [d]$, and $\det \begin{vmatrix} x_1 & x_2 & \dots & x_d & \hat{n} \\ 1 & 1 & \dots & 1 & 0 \end{vmatrix} > 0$, where the columns of the matrix are determined using the order $\prec$. The positive and negative half-spaces of H are defined accordingly. In [4, Proposition 2], the authors show that the choice of $\hat{n}$ does not depend on the choice of $x_i \in \text{conv}(P_i)$ for any i, if the colors are well-separated. Notice that if the colors are not well-separated, then the dimension of the affine hull of $\{x_1, \dots, x_d\}$ may be less than $d - 1$. This makes the value of the determinant above to be zero, so the orientation is not well-defined.

We call a hyperplane *colorful* if it passes through a colorful set $\{p_1, \dots, p_d\} \subset P$. Otherwise, we call the hyperplane *non-colorful*. There is a natural orientation for colorful hyperplanes using the definition above. In order to define an orientation for non-colorful hyperplanes, one needs additional points from the convex hulls of unused colors on the hyperplane. Let H' denote a hyperplane that passes through points of $(d-1)$ colors. Let P_j denote the missing color in H'. To define an orientation for H', we choose a point from $\text{conv}(P_j)$ that lies on H' as follows. We collect the points of P_j on each side of H', and choose the highest ranked points under the order $\prec_j$. Let these points on opposite sides of H' be denoted by x and y. Let z denote the intersection of the line segment xy with H'. By convexity, z is a point in $\text{conv}(P_j)$, so we choose z to define the orientation of H'. The intersection point z does not change if x and y are interchanged, giving a valid definition of orientation for H'. We can also extend this construction to define orientations for hyperplanes containing points from fewer than $d - 1$ colors, but for our purpose this definition suffices. The *α-vector* of any oriented hyperplane H is a d-tuple $(\alpha_1, \dots, \alpha_d)$ of integers where α_i is the number of points of P_i in the closed halfspace H^+ for $i \in [d]$.

3.1 An overview of the reduction

We give a short overview of the ideas used in the reduction from Alpha-HS to UniqueEOPL. The details are technical and we encourage the interested reader to go through the details of our reduction in [13].

Our intuition is based on rotating a colorful hyperplane H to another colorful hyperplane H' through a sequence of local changes of the points on the hyperplanes such that the α-vector of H' increases in some coordinate by one from that of H. We next define the rotation operation in a little more detail. An *anchor* is a colorful $(d-1)$-tuple of P which spans a $(d-2)$-flat. The following procedure takes as input an anchor R and some point $p \in P \setminus R$ and determines the next hyperplane obtained by a rotation. The output is (R', p'), where R' is an anchor and $p' \in P \setminus R'$ is some point.

```
Procedure $(R', p') = NextRotate(R, p)$
1. Let $H$ denote the hyperplane defined by $R \cup \{p\}$ and $t_1$ be the missing color in $R$.
2. If the orientation of $H$ is not well-defined, report a violation of weak general position and well-separation.
3. Let $P_{t_1}^+$ be the subset of $P_{t_1}$ that lies in the closed halfspace $H^+$ and $P_{t_1}^-$ be the subset of $P_{t_1}$ that lies in the open halfspace $H^-$. Let $x \in P_{t_1}^+$ be the highest ranked point according to the order $\prec_{t_1}$ and $y \in P_{t_1}^-$ be the highest ranked point according to $\prec_{t_1}$.
4. If $p$ has color $t_1$ and $|P_{t_1}^+| = n_{t_1}$, report out of range.
5. We rotate $H$ around the anchor $R$ in a direction such that the hyperplane is moving away from $x$ along the segment $xy$ until it hits some point $q \in P$.
6. If the hyperplane hits multiple points at the same time, report a violation of weak general position.
7. If $q$ is not color $t_1$, set $R' := R \cup \{q\} \setminus \{r\}$ and $p' = r$, where $r$ is a point in $R$ with the same color as $q$. Otherwise, set $R' = R$ and $p' = q$.
8. Return $(R', p')$.
```

Figure 3 shows an application of this procedure, rotating H_0 to H_4 through H_1, H_2, H_3.

This rotation function can be interpreted as a function that assigns each hyperplane to the next hyperplane. The set of colorful hyperplanes can be interpreted as vertices in a graph with the rotation function determining the connectivity of the graph.

Canonical path. Each colorful hyperplane H is incident to a colorful set of d points. This set of points defines d possible anchors, and each anchor can be used to rotate H in a different fashion. To define a unique sequence of rotations, we pick a specific order as follows: first, we assume that the colorful hyperplane H whose α-vector is $(1, \ldots, 1)$ is given (we show later how this assumption can be removed). We start at H and pick the anchor that excludes the first color, then apply a sequence of rotations until we hit another colorful hyperplane with α-vector $(2, 1, \ldots, 1)$. Similarly, we move to a colorful hyperplane with α-vector $(3, 1, \ldots, 1)$ and so on until we reach $(\alpha_1, 1, \ldots, 1)$. Then, we repeat this for the other colors in order to reach $(\alpha_1, \alpha_2, 1, \ldots, 1)$ and so on until we reach the target α-vector. This pattern of α-vectors helps in defining a potential function that strictly increases along the path. We can encode this sequence of rotations as a unique path in the UniqueEOPL instance, and we call it *canonical path.*

A natural way to define the UniqueEOPL graph would be to consider hyperplanes as the vertices in the graph. However, this leads to complications. Figure 3 shows a rotation from H_0 to H_4, with α-vectors $(3, 2)$ and $(3, 3)$ respectively. During the rotation, we encounter a hyperplane H_2 for which its α-vector is $(4, 2)$, which differs from our desired sequence of $(3, 2), \ldots, (3, 2), (3, 3)$. This makes it difficult to define a potential function in the graph that strictly increases along the path $v_{H_0}, \ldots, v_{H_4}$ where v_{H_i} is the vertex representing hyperplane H_i. One way to alleviate this problem is to not use H_i as a vertex directly, but the *double-wedge* that is traced out by the rotation from H_i to H_{i+1}. If the α-vector is now measured using the hyperplane that bisects the double-wedge, then we get the desired sequence of $(3, 2), \ldots, (3, 2), (3, 3)$. See Figure 3 for an example.

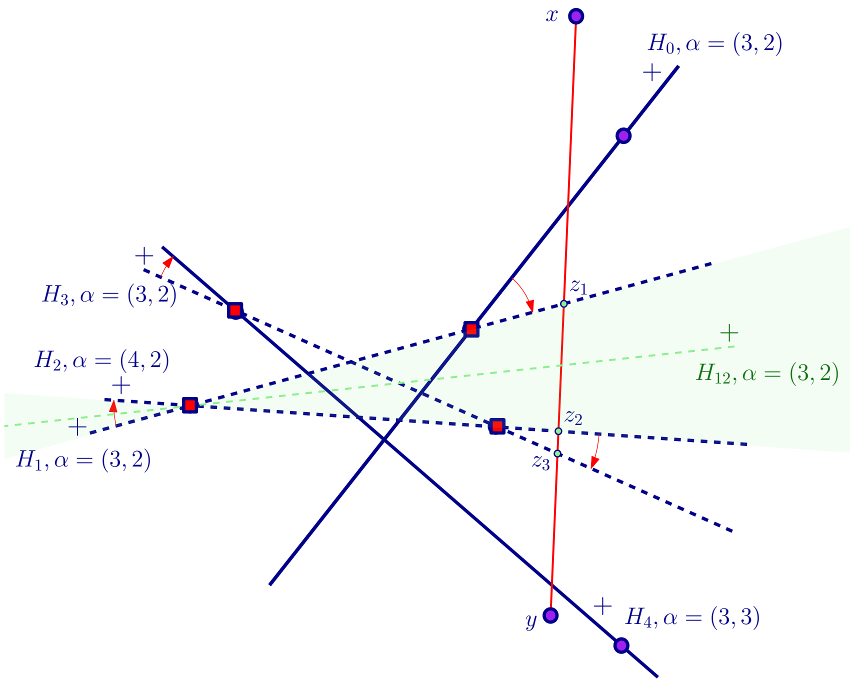


Figure 3 An example showing a sequence of rotations from H_0 to H_4 through H_1, H_2, H_3. Red (square) is the first color and purple (disk) is the second color. This sequence represents a path between two vertices in the UniqueEOPL graph that is generated in the reduction. The double-wedge is shaded and its angular bisector H_{12} has the desired α-vector.

With additional overhead, the rotation function can be extended to double-wedges. This in turn also leads to a neighborhood graph where the vertices are the double-wedges and the rotations can be used to define the edges. The graph is connected and has a grid-like structure that may be of independent interest. Due to lack of space, the description of double-wedges and the associated graph can be found in [13].

Distance parameter and potential function. The α-vector is not sufficient to define the potential function, since the sequence of rotations between two colorful hyperplanes may have the same α-vector. For instance, the bisectors of the rotations in $H_0, \ldots, H_3$ in Figure 3 all have the same α-vector. Hence, we need an additional measurement in order to determine the direction of rotation that increases the α-vector.

Similar to how we define the orientation for a non-colorful hyperplane, let H denote a hyperplane that passes through points of $(d-1)$ colors. Let P_j denote the missing color in H. Let $x, y \in P_j$ be the highest ranked points under $\prec_j$ in H^+ and H^- respectively. Let z denote the intersection of xy and H. We define a distance parameter called *dist-value* of H to be the distance $\|x - z\|$. In Figure 3, we can see that rotating from H_0 to H_4 sweeps the segment xy in one direction, with the dist-value of the hyperplanes increasing strictly. This is sufficient to break ties and hence determine the correct direction of rotation. The precise statement is given in Lemma 6. We can extend this definition to the domain of double-wedges. We define a potential value for each vertex on the canonical path in UniqueEOPL using the sum of weighed components of α-vector and dist-value for the tie-breaker.

Correctness. We show that if there are no violations, we can always apply **Procedure** *NextRotate* to increment the α-vector until we find the desired solution, which implies that the canonical path exists. If the input satisfies weak general position, we can see that the rotating hyperplane always hits a unique point in Step 5, which may be swapped to form a new anchor in Step 7.

The well-separation condition guarantees that the potential function always increases along the rotation. Let H_1, H_2 denote a pair of hyperplanes that are the input and output of **Procedure** *NextRotate* respectively. Let H denote any intermediate hyperplane during the rotation from H_1 to H_2 through the common anchor. Let P_j be the color missing from the anchor and x be the highest ranked point under $\prec_j$ in H_1^+. We say that the orientation of H_2 (resp. H) is *consistent* with that of H_1 if $x \in H_2^+$ (resp. $x \in H^+$). Lemma 5 shows that the orientations are always consistent when H_1 and H_2 are non-colorful hyperplanes even without the assumption of well-separation.

▶ **Lemma 5** (consistency of orientation). *Assume that weak general position holds. Let H_1, H_2 be the input and output of* ***Procedure*** *NextRotate respectively. Let H denote any intermediate hyperplane within the rotation. The orientations of H_1 (resp. H_2) and H are consistent when H_1 (resp. H_2) is a non-colorful hyperplane.*

Proof. Since H_1 is a non-colorful hyperplane, let P_j denote the color missing from H_1. H_1 and H give the same partition of P_j into two sets because the continuous rotation from H_1 to H does not hit any point in P_j. Let x and y be the highest ranked points under $\prec_j$ in each set. Since we have weak general position, the segment xy cannot pass through the anchor of the rotation so that the orientations of H_1 and H are well-defined by the $(d-1)$ colored points in the anchor and the intersections of the hyperplanes with the segment xy. Thus, the determinant defining the normal of the rotating hyperplane from H_1 to H for the orientation is always non-zero. Since the intersection of the rotating hyperplane from H_1 to H and the segment xy moves continuously along xy, by a continuity argument, the normal of the hyperplane does not flip during the rotation. Without loss of generality, assume that $x \in H_1^+$. This implies that x is always in the positive half-space of H and hence H has a consistent orientation as H_1. The same proof holds for H_2. ◀

Next, we show that the dist-value is strictly increasing for all the intermediate hyperplanes in the sequence of rotations from one colorful hyperplane to another colorful hyperplane.

▶ **Lemma 6.** *Assume that weak general position holds. Let H_0 be a colorful hyperplane and H_k be the first colorful hyperplane obtained by a sequence of rotations by* ***Procedure*** *NextRotate. We denote by $H_1, \dots, H_{k-1}$ the non-colorful hyperplanes obtained from the above sequence of rotations. The dist-values of $H_1, \dots, H_{k-1}$ are strictly increasing.*

Proof. Let P_j denote the color missing from H_1. Then, $H_2, \dots, H_{k-1}$ all miss the color P_j, otherwise H_k is not the first colorful hyperplane obtained by the rotations. Therefore, each H_i gives the same partition of P_j into two sets for $i = 1, \dots, k-1$ because the continuous rotations from H_1 to H_{k-1} does not hit any point in P_j. Let x and y be the highest ranked points under $\prec_j$ in each set. Without loss of generality, assume that $x \in H_1^+$. Since $H_1, \dots, H_{k-1}$ are non-colorful hyperplanes, by Lemma 5, the consistent of the orientation can carry from H_1 to H_2 and so on. Then we have $x \in H_1^+, \dots, x \in H_{k-1}^+$ and $y \in H_1^-, \dots, y \in H_{k-1}^-$. Let $z_1 = xy \cap H_1, \dots, z_{k-1} = xy \cap H_{k-1}$. According to Step 5 of **Procedure** *NextRotate*, each rotation is performed by moving away from x along the segment xy. Hence we have $\|x - z_1\| < \|x - z_2\| < \cdots < \|x - z_{k-1}\|$. ◀

The last step for proving that the potential function always increases along the canonical path is to show that the α-vector increases in some coordinate from one colorful hyperplane to another colorful hyperplane through **Procedure** *NextRotate*. This requires the assumption of well-separation. Lemma 7 shows that if the orientations of H_1, H_2 and H are inconsistent, then well-separation is violated. By the contrapositive, if well-separation is satisfied, then all hyperplanes in the rotation always give consistent orientations. Then, it implies that rotating from a colorful hyperplane H_0 to another colorful hyperplane H_k through a sequence of non-colorful hyperplanes that miss color P_j, we have $H_0^+ \cap P_j \subset H_k^+ \cap P_j$ and H_k contains one additional point in P_j that is hit by the last rotation. Therefore, α_j is increased by 1 and other α_is keep the same value because of the way we swap the point of repeated color with the one in the anchor and the direction of rotation.

▶ **Lemma 7.** *Assume that weak general position holds. Let H_1, H_2 be the input and output of* ***Procedure*** *NextRotate respectively. Let R denote the anchor of the rotation from H_1 to H_2, and P_j denote the color missing from R. Let H denote any intermediate hyperplane within the rotation. If the orientations of H_1 (resp. H_2) and H are inconsistent, then H_1 (resp. H_2) is a colorful hyperplane and we can find a colorful set $R \cup \{x'\}$ lying in a $(d-2)$-flat where $x' \in \operatorname{conv}(P_j)$, in $O(d^3)$ arithmetic operations. The set $R \cup \{x'\}$ witnesses the violation of well-separation.*

Proof. Since the orientations of H_1 and H are inconsistent, H_1 must be a colorful hyperplane by Lemma 5. Therefore, the point in H_1 that is not in the anchor is in P_j, denoted by p.

Let x and y be the points defined in Lemma 5 such that $x, y \in P_j$, and x and y are on different sides of H_1 and H. The $(d-2)$-flat containing R separates H_1 and H into two $(d-1)$-dimensional half-subspaces each. Let $H^+_{1,R}$ and H^+_R be the half-subspaces intersecting with xy on H_1 and H respectively, and let us denote the intersection points by z_p and z, respectively. The opposite half-subspaces are denoted by $H^-_{1,R}$ and H^-_R, respectively. By definition of the orientation for non-colorful hyperplanes, the orientation of H is defined by $R \cup \{z\}$. Although the orientation of H_1 is defined by $R \cup \{p\}$, if we consider the determinant defining the orientation using $R \cup \{z_p\}$, it gives an orientation consistent with that of H. Therefore, it must be that $p \in H^-_{1,R}$. Then, we can see that the line segment pz_p intersects the $(d-2)$-flat of R. We can compute z_p and also the intersection point x' of pz_p and the $(d-2)$-flat of R by solving systems of linear equations with d equations and d variables in $O(d^3)$ arithmetic operations. Since $x' \in \operatorname{conv}(P_j)$, $R \cup \{x'\}$ is a colorful set contained in the $(d-2)$-flat of R. ◀

In order to guarantee that there is no other path in UniqueEOPL apart from the canonical path, we introduce self-loops for vertices that are not on the canonical path. The detailed proof in [13] shows that if there are no violations, then the reduced instance of UniqueEOPL only gives a **(U1)** solution, which readily translates to a **(G1)** solution, so our reduction is promise-preserving, and this can be done in polynomial time.

Since we do not know the hyperplane with α-vector $(1, \ldots, 1)$ in advance, we split the problem into two sub-problems: in the first we start with any colorful hyperplane. We reverse the direction of the canonical path determined by the potential and construct an Alpha-HS instance for which the vertex with α-vector $(1, \ldots, 1)$ is the solution. In the second, we use this vertex as the input to the main Alpha-HS instance. If the input is free of violations, then both sub-problems give valid solutions and together they answer the original question. To merge the two sub-problems into one UniqueEOPL instance, we can make two layer copies of the vertices with an additional flag variable to indicate which copy is in the first layer. In the first layer, we build the canonical path from any colorful vertex to the colorful vertex with α-vector $(1,\ldots,1)$, which connects to the colorful vertex with α-vector $(1,\ldots,1)$ in the second layer. Similarly, in the second layer, we build the canonical path from the colorful vertex with α-vector $(1,\ldots,1)$ to the vertex with the target α-vector. Then, we can also easily modify the potential function accordingly.

An alternative approach is to define the canonical path directly from any colorful vertex to the target vertex. In this case, each coordinate of the current α-vector may increase or decrease depending on the signed distance to the target α-vector along the canonical path. However, the potential function can still be defined in a way that it is strictly increasing along the path.

Handling violations. The reduction maps violations of Alpha-HS to violations of the UniqueEOPL instance, and certificates for the violations can be recovered from additional processing. When a violation of weak general position is witnessed on a vertex that lies on the canonical path, a hyperplane incident to d colors may contain additional points. This in turn implies that some α-cut is missing, so that the correct solution for the target may not exist. For cuts that exist in spite of the violation, reporting either the correct solution or the violation are sufficient for Alpha-HS.

In addition, the (highest-ranked) points x, y from the missing color that we choose to define the orientation of a non-colorful hyperplane may form a segment xy that passes through the $(d-2)$-flat spanned by the anchor. In that case the orientation of the hyperplane is not well-defined. In the reduction, these problematic vertices are removed from the canonical path, thereby creating some additional starting points and end points in the reduced instance. These violations can be captured by **(U1)** with a wrong α-vector or **(UV2)**. Furthermore, the hyperplanes that contain the degenerate point sets could be represented by different choices of anchors and an additional point on the plane. Each such pair represents a vertex in the reduced instance. We join these vertices in the form of a cycle in the UniqueEOPL instance with all vertices having the same potential value, so that the violations can also be captured by **(UV1)** and **(UV3)**.

When a violation of well-separation is witnessed on a vertex on the canonical path, the orientations of the two hyperplanes paired by **Procedure** *NextRotate* may be inconsistent, which may not guarantee that the α-vector is incremented in one component by one (See Figure 4). Hence, the canonical path is split into two paths that can be captured by **(UV2)**. Furthermore, a violation of well-separation also creates multiple colorful hyperplanes with the same α-vector (See Figure 4, left). Two vertices in the UniqueEOPL graph with the same potential value, which could correspond to some colorful or non-colorful hyperplanes, can be reported by **(UV3)**. We show that this gives a certificate of violation of well-separation in the following lemmas, where m_0 is the number of bits used to represent each coordinate of points of P.

▶ **Lemma 8.** *Given two colorful hyperplanes H_p, H_q with the same α-vector, we can find a colorful set $\{x_1 \in \operatorname{conv}(P_1), \ldots, x_d \in \operatorname{conv}(P_d)\}$ that lies on a $(d-2)$-flat in $\operatorname{poly}(n, d, m_0)$ time.*

▶ **Lemma 9.** *Given two non-colorful hyperplanes that both contain $d-1$ points and have the same missing color, α-vector and dist-value, we can find a colorful set of points $\{x_1 \in \operatorname{conv}(P_1), \ldots, x_d \in \operatorname{conv}(P_d)\}$ that lies on a $(d-2)$-flat in $\operatorname{poly}(n, d, m_0)$ time.*

For the second output $(\mathbb{V}(v) < \mathbb{V}(u) < \mathbb{V}(\mathbb{S}(v)))$ of **(UV3)**, there are two cases to consider. In the first case, if both v and $\mathbb{S}(v)$ correspond to the same α-vector, then u also has the same α-vector and its dist-value is between that of v and $\mathbb{S}(v)$. Since rotating the hyperplane from

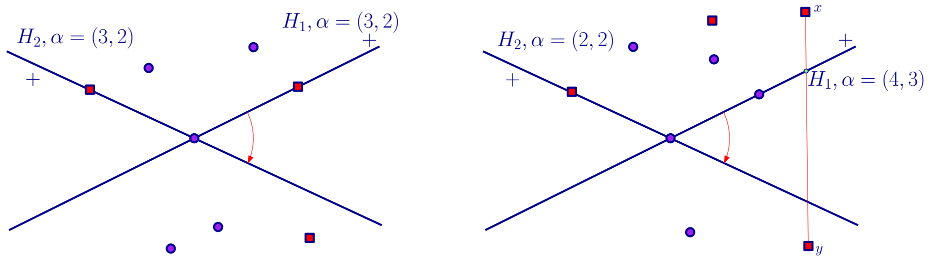


Figure 4 The examples show two sets of points that are not well-separated. Purple (circle) represents the first color and red (square) represents the second color. In both examples the rotation procedure does not increase the α-vector. Both examples show that the orientation of the hyperplane may be flipped after the rotation, so the resulting α-vector can go wrong.

v to $\mathbb{S}(v)$ does not pass through u, we can find a different hyperplane that is interpolated by v and $\mathbb{S}(v)$ and has the same dist-value as u. Hence, we apply Lemma 9 again to find a witness of the violation. For the second case that the α-vector of $\mathbb{S}(v)$ increases in one coordinate by one from that of v, since the role of dist-value is dominated by the role of α-vector in the potential function, the dist-value of u can be arbitrarily large. Therefore, we may not be able to apply the interpolation technique from the first again. We argue that we can transform P to a point set P' satisfying $\text{conv}(P'_i) \subseteq \text{conv}(P_i)$ for all $i \in [d]$, such that the hyperplanes of v and u become colorful. Then, we apply Lemma 8 to show that P' is not well-separated, which also implies that P is not well-separated. The precise statement and proof are given in [13]. We also show

- how to compute a **(GV1)** solution from a **(UV1)** solution,
- how to compute a **(GV1)** or **(GV2)** solution, given a **(UV2)** or **(UV3)** solution, and
- a **(GV1)** or **(GV2)** solution that can occur with a **(U1)** solution that has the incorrect α-vector.

We show that converting these solutions always takes $\text{poly}(n, d)$ time. The violations may be detected in either the first sub-problem or the second sub-problem. Our constructions thus culminate in the promised result:

▶ **Theorem 10.** *Alpha-HS* $\in$ UEOPL $\subseteq$ CLS.

Handling very weak general position. We have described our construction for the case when weak general position holds. If we only assume that very weak general position holds, then there may exist a hyperplane that passes through more than d points of at most $d-1$ colors. Therefore, in Step 5 of **Procedure** *NextRotate* the rotating hyperplane may hit more than one point so that it is not clear how to define the new anchor in Step 7. From the point of view of the reduction, there are many non-colorful vertices that represent the same hyperplane. We need a new approach to define a unique path to traverse these vertices with respect to this hyperplane. In other words, we charge the computational time of finding the new anchor to traversing these vertices on the path instead of considering it as one operation.

If we consider the space of all the points lying on the hyperplane, we have $d-1$ sets of points each representing a unique color in an affine subspace of $d-1$ dimensions. Thus, we can consider it as a new instance of Alpha-HS in one dimension lower. Let H be the rotating hyperplane that hits more than one point and contains $d-1$ colors. Without loss of generality, we assume that d is the missing color. We denote by $Q = Q_1 \cup Q_2 \cup \ldots \cup Q_{d-1}$ the $d-1$ sets of points in H such that $Q_i \subseteq P_i$ and denote by $\widehat{Q}_i$ the set of points represented in

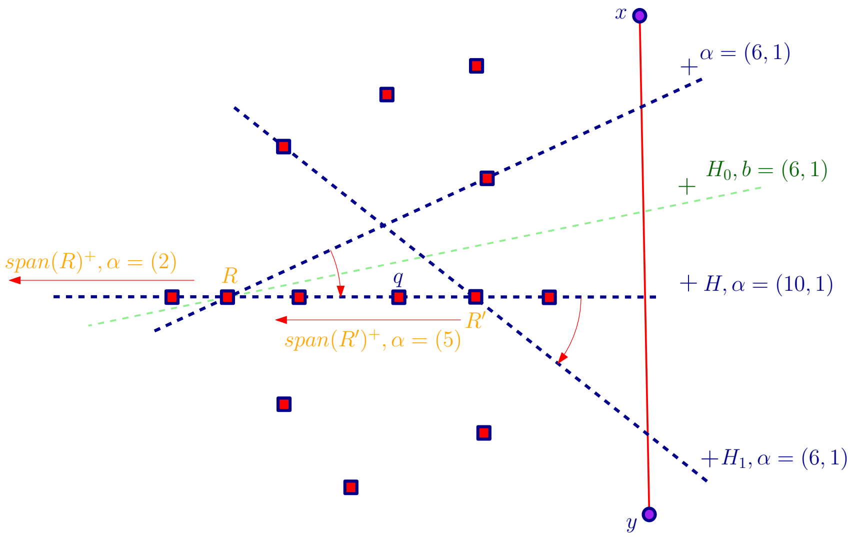


Figure 5 An example showing the relationship between the α-vector in a subproblem in $\mathbb{R}$ and the α-vector in the original problem in $\mathbb{R}^2$. Red (square) is the first color and purple (disk) is the second color. The orientation of $span(R)$ in H is defined such that it is consistent with H_0. $b = (6,1)$ is the α-vector of H_0. $k_1 = 4$ is the number of red points in $H^+ \setminus H$. The α-vector of the starting vertex (i.e., R) with respect to H is $(6-4=2)$. The α-vector of the end vertex is $(6+1-2=5)$. We can see that $q \in span(R')^+$ and q moves to the negative side of H_1 when rotating from H to H_1.

the new coordinate system in $\mathbb{R}^{d-1}$ for Q_i in H. First, we claim that if P is well-separated and in very weak general position, then $\widehat{Q}$ is also well-separated and in very weak general position. Since $Q \subset P$, it is clear that well-separation follows. Suppose that $\widehat{Q}$ violates very weak general position, then there exists a $(d-2)$-flat that contains more than $d-1$ points of $d-1$ colors in Q. In particular, any $(d-1)$-flat spanned by the $(d-2)$-flat and any point in P_d contains more than d points of d colors, which contradicts the fact that P is in very weak general position.

Suppose that P is well-separated and in very weak general position. Now we define what is the unique path with respect to $\widehat{Q}$. Let $b = (b_1, \dots, b_d)$ be the α-vector of the rotating hyperplane H_0 just before rotating to H at the anchor R. In the new instance of Alpha-HS, we would pick the orientation of $(d-2)$-flats in $\mathbb{R}^{d-1}$ such that every point $p \in Q$ lies in H_0^+ if and only if the corresponding point $\widehat{p} \in \widehat{Q}$ lies in $span(\widehat{R})^+$. Let $k_1, \dots k_{d-1}$ denote the number of points of $P_1, \dots, P_{d-1}$ in H^+, but not in Q_i. Then, we can see that the number of points in $\widehat{Q}_i$ lying in $span(\widehat{R})^+$ is equal to $b_i - k_i$. Thus, the α-vector of $span(\widehat{R})^+$ is $(b_1 - k_1, \dots, b_{d-1} - k_{d-1})$, which is the α-vector of the starting vertex of the path. On the other hand, the α-vector of the end vertex is $(|Q_1| + 1 - b_1 + k_1, \dots, |Q_{d-1}| + 1 - b_{d-1} + k_{d-1})$. It is because the points in $H_0^+ \setminus H_0$ become in the opposite side after the rotation passes through H. Therefore, if we rotate at the new anchor with α-vector $(|Q_1| + 1 - b_1 + k_1, \dots, |Q_{d-1}| + 1 - b_{d-1} + k_{d-1})$ in $\widehat{Q}$, then the α-vector of the new rotating hyperplane is still $(b_1, \dots, b_d)$. The next question is that if the vertex only stores any d points of H, we cannot recover b and H_0 so that the orientation cannot be defined consistently and the target α-vector for $\widehat{Q}$ is not known. To handle this problem, we need to redefine the double-wedge to be (R_1, p_1, R_2, p_2) instead of (R, p, q) in such a way that $R_1 = R_2$ if the double-wedge contains exactly $d+1$ points, otherwise $R_1 \subset span(R_2 \cup \{p_2\})$. For instance, if $(\widehat{R}_1, \widehat{q}_1) -> \dots -> (\widehat{R}_m, \widehat{q}_m)$ is the unique path in $\widehat{Q}$, where $\widehat{R}_i$ is an anchor of size $d-2$ so that $\widehat{R}_i$ and $\widehat{q}_i$ represent a $(d-2)$-flat in $\mathbb{R}^{d-1}$, then the corresponding path in the original problem is $(R, p, R_1 \cup \{q_1\}, p_1) -> (R, p, R_2 \cup \{q_2\}, p_2) -> \ldots -> (R, p, R_m \cup \{q_m\}, p_m)$, where p_i is some point in H that is picked under $\prec$ in a way that the tuple is uniquely defined in the path. Hence, b can be computed from the bisector of (R, p) and $(R_i \cup \{q_i\}, p_i)$, and the orientation of $(d-2)$-flats can also be defined by the bisector. There may exist some other double-wedge $(*, *, R_i \cup \{q_i\}, p_i)$ that is incident to H, but it will not have the same b.

In conclusion, the unique path in the reduction can be defined recursively as above in an Alpha-HS instance of one dimension lower. As a result, the representation of the double-wedges gets more complicated and the size is increased by a factor of $O(d)$. The potential function becomes a weighted sum of the potential function in each recursive level, but the number of bits is still in polynomial size. For handling violations, there are not many changes. Instead of reporting the violation of weak general position, we now report the violation of very weak general position when the rotating hyperplane in R^i contains more than i points of i colors. If any recursive subproblem violates very weak general position, it also implies that the original input P violates very weak general position.

4 Conclusion and future work

We gave a complexity-theoretic upper bound for Alpha-HS. No hardness results are known for this search problem, and the next question is determining if this is hard for UEOPL. One challenge is that UniqueEOPL is formulated as Boolean circuits, whereas Alpha-HS is purely geometric. Emulating circuits using purely geometric arguments is highly non-trivial. Filos-Ratsikas and Goldberg showed a reduction of this form in [21]. They reduced the PPA-complete 2D-Tucker circuit to Ham-Sandwich, going via the *Consensus-Halving* [39], and the *Necklace-splitting problems* [2]. A simplified argument was recently presented in [22]. It could be a worthwhile exercise to investigate if their techniques can provide insights for hardness of Alpha-HS.

Some related problems are determining the complexity of answering whether a point set is well-separated, whether it is in weak general position, or whether a given α-cut exists for the point set. A given α-cut may exist even when both assumptions are violated. On a related note, deciding whether the Linear Complementarity problem has a solution is NP-complete [15]. The solution is unique if the problem involves a P-matrix, but checking this condition is coNP-complete [16]. However, using witnesses to verify whether a matrix is P-matrix or not, a total search version is shown to be in UEOPL. Our result for Alpha-HS would go in a similar vein, if the complexities of the above problems were better determined.

Another line to work could be to determine the computational complexities of other extensions of the Ham-Sandwich theorem. For other geometric problems that are total and admit unique solutions, it could be worthwhile to explore their place in the class UEOPL. Faster algorithms for computing the α-cut can also be explored.

References

1 James Aisenberg, Maria Luisa Bonet, and Sam Buss. 2-D Tucker is PPA-complete. *J. Comput. System Sci.*, 108:92–103, 2020.

2 Noga Alon and Douglas B. West. The Borsuk-Ulam theorem and bisection of necklaces. *Proc. American Mathematical Society*, 98(4):623–628, 1986.

3 Imre Bárány. A generalization of Carathéodory's theorem. *Discrete Mathematics*, 40(2-3):141–152, 1982.

4 Imre Bárány, Alfredo Hubard, and Jesús Jerónimo. Slicing convex sets and measures by a hyperplane. *Discrete Comput. Geom.*, 39(1):67–75, 2008.

5 Imre Bárány and Jiří Matoušek. Simultaneous partitions of measures by k-fans. *Discrete Comput. Geom.*, 25(3):317–334, 2001.

6 Luis Barba, Alexander Pilz, and Patrick Schnider. Sharing a pizza: bisecting masses with two cuts. *CoRR*, abs/1904.02502, 2019.

7 Sergey Bereg. Equipartitions of measures by 2-fans. *Discrete Comput. Geom.*, 34(1):87–96, 2005.

8 Sergey Bereg. Computing generalized ham-sandwich cuts. *Inform. Process. Lett.*, 112(13):532–534, 2012.

9 Pavle V. M. Blagojević , Aleksandra Dimitrijević Blagojević , Roman Karasev, and Jonathan Kliem. More bisections by hyperplane arrangements, 2018. `arXiv:1809.05364`.

10 Xi Chen, Decheng Dai, Ye Du, and Shang-Hua Teng. Settling the complexity of arrow-debreu equilibria in markets with additively separable utilities. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009*, pages 273–282, 2009.

11 Xi Chen, Xiaotie Deng, and Shang-Hua Teng. Settling the complexity of computing two-player Nash equilibria. *J. ACM*, 56(3):14:1–14:57, 2009.

12 Xi Chen, Dimitris Paparas, and Mihalis Yannakakis. The complexity of non-monotone markets. *J. ACM*, 64(3):20:1–20:56, 2017.

13 Man-Kwun Chiu, Aruni Choudhary, and Wolfgang Mulzer. Computational complexity of the α-ham-sandwich problem, 2020. `arXiv:2003.09266`.

14 Aruni Choudhary and Wolfgang Mulzer. No-dimensional Tverberg theorems and algorithms. In *Proc. 36th Int. Sympos. Comput. Geom. (SoCG)*, page to appear, 2020.

15 S. J. Chung. NP-completeness of the Linear Complementarity Problem. *Journal of Optimization Theory and Applications*, 60(3):393–399, 1989.

16 Gregory E. Coxson. The P-matrix problem is co-NP-complete. *Mathematical Programming*, 64(1):173–178, 1994.

17 Constantinos Daskalakis, Paul W. Goldberg, and Christos H. Papadimitriou. The complexity of computing a Nash Equilibrium. *SIAM J. Comput.*, 39(1):195–259, 2009.

18 Constantinos Daskalakis and Christos H. Papadimitriou. Continuous Local Search. In *Proc. 22nd Annu. ACM-SIAM Sympos. Discrete Algorithms (SODA)*, pages 790–804, 2011.

19 Vladimir L. Dol'nikov. A generalization of the Ham-sandwich theorem. *Mathematical Notes*, 52(2):771–779, 1992.

20 John Fearnley, Spencer Gordon, Ruta Mehta, and Rahul Savani. Unique End of Potential Line. In *Proc. 46th Internat. Colloq. Automata Lang. Program. (ICALP)*, pages 56:1–56:15, 2019.

21 Aris Filos-Ratsikas and Paul W. Goldberg. The complexity of splitting Necklaces and bisecting Ham sandwiches. In *Proc. 51st Annu. ACM Sympos. Theory Comput. (STOC)*, pages 638–649, 2019.

22 Aris Filos-Ratsikas, Alexandros Hollender, Katerina Sotiraki, and Manolis Zampetakis. Consensus-halving: Does it ever get easier? *CoRR*, abs/2002.11437, 2020. `arXiv:2002.11437`.

23 Vincent Froese, Iyad A. Kanj, André Nichterlein, and Rolf Niedermeier. Finding points in general position. *Internat. J. Comput. Geom. Appl.*, 27(4):277–296, 2017.

24 Paul W. Goldberg and Alexandros Hollender. The Hairy Ball problem is PPAD-complete. In *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, pages 65:1–65:14, 2019.

25 Alfredo Hubard and Roman Karasev. Bisecting measures with hyperplane arrangements. *Mathematical Proceedings of the Cambridge Philosophical Society*, pages 1–9, 2019.

26 David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. How easy is local search? *J. Comput. System Sci.*, 37(1):79–100, 1988.

27 Roman N. Karasev. Topological methods in combinatorial geometry. *Russian Mathematical Surveys*, 63(6):1031–1078, 2008.

28 Chi-Yuan Lo, Jiří Matoušek, and William L. Steiger. Algorithms for Ham-sandwich cuts. *Discrete Comput. Geom.*, 11:433–452, 1994.

29 Jesús De Loera, Xavier Goaoc, Frédéric Meunier, and Nabil Mustafa. The discrete yet ubiquitous theorems of Carathéodory, Helly, Sperner, Tucker, and Tverberg. *Bulletin of the American Mathematical Society*, 56(3):415–511, 2019.

30 Jiří Matoušek. *Using the Borsuk-Ulam theorem.* Springer-Verlag Berlin Heidelberg, 2003.

31 Frédéric Meunier, Wolfgang Mulzer, Pauline Sarrabezolles, and Yannik Stein. The rainbow at the end of the line: A PPAD formulation of the Colorful Carathéodory theorem with applications. In *Proc. 28th Annu. ACM-SIAM Sympos. Discrete Algorithms (SODA)*, pages 1342–1351, 2017.

32 Wolfgang Mulzer and Yannik Stein. Computational aspects of the Colorful carathéodory theorem. *Discrete Comput. Geom.*, 60(3):720–755, 2018.

33 Abraham Othman, Christos H. Papadimitriou, and Aviad Rubinstein. The complexity of fairness through equilibrium. *ACM Trans. Economics and Comput.*, 4(4):20:1–20:19, 2016.

34 Christos H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *J. Comput. System Sci.*, 48(3):498–532, 1994.

35 Richard Rado. A theorem on general measure. *Journal of the London Mathematical Society*, s1-21(4):291–300, 1946.

36 Karanbir S. Sarkaria. Tverberg's theorem via number fields. *Israel Journal of Mathematics*, 79(2-3):317–320, 1992.

37 Patrick Schnider. Equipartitions with wedges and cones. *CoRR*, abs/1910.13352, 2019.

38 Patrick Schnider. Ham-sandwich cuts and center transversals in subspaces. In *Proc. 35th Int. Sympos. Comput. Geom. (SoCG)*, pages 56:1–56:15, 2019.

39 Forest W. Simmons and Francis Edward Su. Consensus-halving via theorems of Borsuk-Ulam and Tucker. *Mathematical Social Sciences*, 45(1):15–25, 2003.

40 William Steiger and Jihui Zhao. Generalized Ham-sandwich cuts. *Discrete Comput. Geom.*, 44(3):535–545, 2010.

41 Arthur H. Stone and John W. Tukey. Generalized "Sandwich" theorems. *Duke Mathematical Journal*, 9(2):356–359, June 1942.

42 Albert W. Tucker. Some topological properties of disk and sphere. In *Proc. First Canadian Math. Congress, Montreal, 1945*, pages 285–309. University of Toronto Press, Toronto, 1946.

43 Helge Tverberg. A generalization of Radon's theorem. *Journal of the London Mathematical Society*, s1-41(1):123–128, 1966.

44 Rade T. Zivaljević and Siniša T. Vrećica. An extension of the Ham sandwich theorem. *Bulletin of the London Mathematical Society*, 22(2):183–186, 1990.

Existence and Complexity of Approximate Equilibria in Weighted Congestion Games

George Christodoulou
Department of Computer Science, University of Liverpool, UK
g.christodoulou@liverpool.ac.uk

Martin Gairing
Department of Computer Science, University of Liverpool, UK
gairing@liverpool.ac.uk

Yiannis Giannakopoulos
Operations Research Group, TU Munich, Germany
yiannis.giannakopoulos@tum.de

Diogo Poças
Operations Research Group, TU Munich, Germany
diogo.pocas@tum.de

Clara Waldmann
Operations Research Group, TU Munich, Germany
clara.waldmann@tum.de

Abstract

We study the existence of approximate pure Nash equilibria (α-PNE) in weighted atomic congestion games with polynomial cost functions of maximum degree d. Previously it was known that d-approximate equilibria always exist, while nonexistence was established only for small constants, namely for 1.153-PNE. We improve significantly upon this gap, proving that such games in general do not have $\tilde{\Theta}(\sqrt{d})$-approximate PNE, which provides the first super-constant lower bound.

Furthermore, we provide a black-box gap-introducing method of combining such nonexistence results with a specific circuit gadget, in order to derive NP-completeness of the decision version of the problem. In particular, deploying this technique we are able to show that deciding whether a weighted congestion game has an $\tilde{O}(\sqrt{d})$-PNE is NP-complete. Previous hardness results were known only for the special case of *exact* equilibria and arbitrary cost functions.

The circuit gadget is of independent interest and it allows us to also prove hardness for a variety of problems related to the complexity of PNE in congestion games. For example, we demonstrate that the question of existence of α-PNE in which a certain set of players plays a specific strategy profile is NP-hard for any $\alpha < 3^{d/2}$, even for *unweighted* congestion games.

Finally, we study the existence of approximate equilibria in weighted congestion games with general (nondecreasing) costs, as a function of the number of players n. We show that n-PNE always exist, matched by an almost tight nonexistence bound of $\tilde{\Theta}(n)$ which we can again transform into an NP-completeness proof for the decision problem.

2012 ACM Subject Classification Theory of computation → Algorithmic game theory; Theory of computation → Exact and approximate computation of equilibria; Theory of computation → Representations of games and their complexity; Theory of computation → Problems, reductions and completeness

Keywords and phrases Atomic congestion games, existence of equilibria, pure Nash equilibria, approximate equilibria, hardness of equilibria

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.32

Category Track A: Algorithms, Complexity and Games

Related Version A full version of this paper is available at [8]: `https://arxiv.org/abs/2002.07466`.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 32; pp. 32:1–32:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Funding Supported by the Alexander von Humboldt Foundation with funds from the German Federal Ministry of Education and Research (BMBF).

Acknowledgements Y. Giannakopoulos is an associated researcher with the Research Training Group GRK 2201 "Advanced Optimization in a Networked Economy", funded by the German Research Foundation (DFG).

1 Introduction

Congestion games constitute the standard framework to study settings where selfish players compete over common resources. They are one of the most well-studied classes of games within the field of *algorithmic game theory* [32, 27], covering a wide range of applications, including, e.g., traffic routing and load balancing. In their most general form, each player has her own weight and the latency on each resource is a nondecreasing function of the total weight of players that occupy it. The cost of a player on a given outcome is just the total latency that she is experiencing, summed over all the resources she is using.

The canonical approach to analysing such systems and predicting the behaviour of the participants is the ubiquitous game-theoretic tool of equilibrium analysis. More specifically, we are interested in the *pure Nash equilibria (PNE)* of those games; these are stable configurations from which no player would benefit from unilaterally deviating. However, it is a well-known fact that such desirable outcomes might not always exist, even in very simple weighted congestion games. A natural response, especially from a computer science perspective, is to relax the solution notion itself by considering *approximate* pure Nash equilibria (α-PNE); these are states from which, even if a player could improve her cost by deviating, this improvement could not be by more than a (multiplicative) factor of $\alpha \geq 1$. Allowing the parameter α to grow sufficiently large, existence of α-PNE is restored. But how large does α really *need* to be? And, perhaps more importantly from a computational perspective, how hard is it to check whether a specific game has indeed an α-PNE?

1.1 Related Work

The origins of the systematic study of (atomic) congestion games can be traced back to the influential work of Rosenthal [30, 31]. Although Rosenthal showed the existence of congestion games without PNE, he also proved that *unweighted* congestion games always possess such equilibria. His proof is based on a simple but ingenious *potential function* argument, which up to this day is essentially still the only general tool for establishing existence of pure equilibria.

In follow-up work [20, 26, 17], the nonexistence of PNE was demonstrated even for special simple classes of (weighted) games, including network congestion games with quadratic cost functions and games where the player weights are either 1 or 2. On the other hand, we know that equilibria do exist for affine or exponential latencies [17, 28, 22], as well as for the class of singleton[1] games [16, 23]. Dunkel and Schulz [13] were able to extend the nonexistence instance of Fotakis et al. [17] to a gadget in order to show that deciding whether a congestion game with step cost functions has a PNE is a (strongly) NP-hard problem, via a reduction from 3-Partition.

Regarding approximate equilibria, Hansknecht et al. [21] gave instances of very simple, two-player polynomial congestion games that do not have α-PNE, for $\alpha \approx 1.153$. This lower bound is achieved by numerically solving an optimization program, using polynomial

[1] These are congestion games where the players can only occupy single resources.

latencies of maximum degree $d = 4$. On the positive side, Caragiannis et al. [4] proved that $d!$-PNE always exist; this upper bound on the existence of α-PNE was later improved to $\alpha = d + 1$ [21, 9] and $\alpha = d$ [3].

1.2 Our Results and Techniques

After formalizing our model in Section 2, in Section 3 we show the nonexistence of $\Theta(\frac{\sqrt{d}}{\ln d})$-approximate equilibria for polynomial congestion games of degree d. This is the first super-constant lower bound on the nonexistence of α-PNE, significantly improving upon the previous constant of $\alpha \approx 1.153$ and reducing the gap with the currently best upper bound of d. More specifically (Theorem 1), for any integer d we construct congestion games with polynomial cost functions of maximum degree d (and nonnegative coefficients) that do not have α-PNE, for any $\alpha < \alpha(d)$ where $\alpha(d)$ is a function that grows as $\alpha(d) = \Omega\left(\frac{\sqrt{d}}{\ln d}\right)$. To derive this bound, we had to use a novel construction with a number of players growing unboundedly as a function of d.

Next, in Section 4 we turn our attention to computational hardness constructions. Starting from a Boolean circuit, we create a gadget that transfers hard instances of the classic Circuit Satisfiability problem to (even unweighted) polynomial congestion games. Our construction is inspired by the work of Skopalik and Vöcking [34], who used a similar family of lockable circuit games in their PLS-hardness result. Using this gadget we can immediately establish computational hardness for various computational questions of interest involving congestion games (Theorem 3). For example, we show that deciding whether a d-degree polynomial congestion game has an α-PNE in which a specific set of players play a specific strategy profile is NP-hard, even up to exponentially-approximate equilibria; more specifically, the hardness holds for *any* $\alpha < 3^{d/2}$. Our investigation of the hardness questions presented in Theorem 3 (and later on in Corollary 7 as well) was inspired by some similar results presented before by Conitzer and Sandholm [11] (and even earlier in [19]) for *mixed* Nash equilibria in general (normal-form) games. To the best of our knowledge, our paper is the first to study these questions for *pure* equilibria in the context of congestion games. It is of interest to also note here that our hardness gadget is *gap-introducing*, in the sense that the α-PNE and exact PNE of the game coincide.

In Section 5 we demonstrate how one can combine the hardness gadget of Section 4, in a black-box way, with any nonexistence instance for α-PNE, in order to derive hardness for the decision version of the existence of α-PNE (Lemma 4, Theorem 5). As a consequence, using the previous $\Omega\left(\frac{\sqrt{d}}{\ln d}\right)$ lower bound construction of Section 3, we can show that deciding whether a (weighted) polynomial congestion has an α-PNE is NP-hard, for any $\alpha < \alpha(d)$, where $\alpha(d) = \Omega\left(\frac{\sqrt{d}}{\ln d}\right)$ (Corollary 6). Since our hardness is established via a rather transparent, "master" reduction from Circuit Satisfiability, which in particular is parsimonious, one can derive hardness for a family of related computation problems; for example, we show that computing the number of α-approximate equilibria of a weighted polynomial congestion game is #P-hard (Corollary 7).

In Section 6 we drop the assumption on polynomial cost functions, and study the existence of approximate equilibria under arbitrary (nondecreasing) latencies as a function of the number of players n. We prove that n-player congestion games always have n-approximate PNE (Theorem 8). As a consequence, one cannot hope to derive super-constant nonexistence lower bounds by using just simple instances with a fixed number of players (similar to, e.g., Hansknecht et al. [21]). In particular, this shows that the super-constant number of players in our construction in Theorem 1 is necessary. Furthermore, we pair this positive result with an almost matching lower bound (Theorem 9): we give examples of n-player congestion games (where latencies are simple step functions with a single breakpoint) that do not have α-PNE for all $\alpha < \alpha(n)$, where $\alpha(n)$ grows according to $\alpha(n) = \Omega\left(\frac{n}{\ln n}\right)$. Finally, inspired by our hardness construction for the polynomial case, we also give a new reduction that establishes NP-hardness for deciding whether an α-PNE exists, for any $\alpha < \alpha(n) = \Omega\left(\frac{n}{\ln n}\right)$. Notice that now the number of players n is part of the description of the game (i.e., part of the input) as opposed to the maximum degree d for the polynomial case (which was assumed to be fixed). On the other hand though, we have more flexibility on designing our gadget latencies, since they can be arbitrary functions.

Concluding, we would like to elaborate on a couple of points. First, the reader would have already noticed that in all our hardness results the (in)approximability parameter α ranges freely within an entire interval of the form $[1, \tilde{\alpha})$, where $\tilde{\alpha}$ is a function of the degree d (for polynomial congestion games) or of the number of players n; and that α, $\tilde{\alpha}$ are *not* part of the problem's input. It is easy to see that these features only make our results stronger, with respect to computational hardness, but also more robust. Secondly, although in this introductory section all our hardness results were presented in terms of NP-*hardness*, they immediately translate to NP-*completeness* under standard assumptions on the parameter α; e.g., if α is rational (for a more detailed discussion of this, see also the end of Section 2).

Due to space constraints we had to either fully omit, or just give very short sketches of, the proofs of our results. All proofs can be found in the full version of this paper [8].

2 Model and Notation

A (weighted, atomic) *congestion game* is defined by: a finite (nonempty) set of *resources* E, each $e \in E$ having a nondecreasing *cost (or latency) function* $c_e : \mathbb{R}_{>0} \longrightarrow \mathbb{R}_{\geq 0}$; and a finite (nonempty) set of *players* N, $|N| = n$, each $i \in N$ having a *weight* $w_i > 0$ and a set of *strategies* $S_i \subseteq 2^E$. If all players have the same weight, $w_i = 1$ for all $i \in N$, the game is called *unweighted.* A *polynomial congestion game* of degree d, for d a nonnegative integer, is a congestion game such that all its cost functions are polynomials of degree at most d with nonnegative coefficients.

A *strategy profile* (or *outcome*) $\mathbf{s} = (s_1, s_2, \dots, s_n)$ is a collection of strategies, one for each player, i.e. $\mathbf{s} \in \mathbf{S} = S_1 \times S_2 \times \cdots \times S_n$. Each strategy profile $\mathbf{s}$ induces a *cost* of $C_i(\mathbf{s}) = \sum_{e \in s_i} c_e(x_e(\mathbf{s}))$ to every player $i \in N$, where $x_e(\mathbf{s}) = \sum_{i:e \in s_i} w_i$ is the induced *load* on resource e. An outcome $\mathbf{s}$ will be called *α-approximate (pure Nash) equilibrium (α-PNE)*, where $\alpha \geq 1$, if no player can unilaterally improve her cost by more than a factor of α. Formally:

$$C_i(\mathbf{s}) \leq \alpha \cdot C_i(s_i', \mathbf{s}_{-i}) \qquad \text{for all } i \in N \text{ and all } s_i' \in S_i. \tag{1}$$

Here we have used the standard game-theoretic notation of $\mathbf{s}_{-i}$ to denote the vector of strategies resulting from $\mathbf{s}$ if we remove its i-th coordinate; in that way, one can write $\mathbf{s} = (s_i, \mathbf{s}_{-i})$. Notice that for the special case of $\alpha = 1$, (1) is equivalent to the classical definition of pure Nash equilibria; for emphasis, we will sometimes refer to such 1-PNE as *exact* equilibria.

If (1) does not hold, it means that player i could improve her cost by more than α by moving from s_i to some other strategy s_i'. We call such a move *α-improving.* Finally, strategy s_i is said to be *α-dominating* for player i (with respect to a fixed profile $\mathbf{s}_{-i}$) if

$$C_i(s_i', \mathbf{s}_{-i}) > \alpha \cdot C_i(\mathbf{s}) \qquad \text{for all } s_i' \neq s_i. \tag{2}$$

In other words, if a strategy s_i is α-dominating, every move from some other strategy s'_i to s_i is α-improving. Notice that each player i can have at most one α-dominating strategy (for $\mathbf{s}_{-i}$ fixed). In our proofs, we will employ a *gap-introducing* technique by constructing games with the property that, for any player i and any strategy profile $\mathbf{s}_{-i}$, there is always a (unique) α-dominating strategy for player i. As a consequence, the sets of α-PNE and exact PNE coincide.

Finally, for a positive integer n, we will use Φ_n to denote the unique positive solution of equation $(x+1)^n = x^{n+1}$. Then, Φ_n is strictly increasing with respect to n, with $\Phi_1 = \phi \approx 1.618$ (golden ratio) and asymptotically $\Phi_n \sim \frac{n}{\ln n}$ (see [9, Lemma A.3]).

Computational Complexity

Most of the results in this paper involve complexity questions, regarding the existence of (approximate) equilibria. Whenever we deal with such statements, we will implicitly assume that the congestion game instances given as inputs to our problems can be succinctly represented in the following way:

- all player have *rational* weights;
- the resource cost functions are "efficiently computable"; for polynomial latencies in particular, we will assume that the coefficients are *rationals*; and for step functions we assume that their values and breakpoints are *rationals*;
- the strategy sets are given *explicitly*.[2]

There are also computational considerations to be made about the number α appearing in the definition of α-PNE. For simplicity, throughout this paper we will assume that α is a rational number. However, all our hardness results are still valid for any real α, while for our completeness results one needs to assume that α is actually a *polynomial-time computable* real. For more details we refer to the full version of our paper [8].

3 The Nonexistence Gadget

In this section we give examples of polynomial congestion games of degree d, that do *not* have $\alpha(d)$-approximate equilibria; $\alpha(d)$ grows as $\Omega\left(\frac{\sqrt{d}}{\ln d}\right)$. Fixing a degree $d \geq 2$, we construct a family of games $\mathcal{G}^d_{(n,k,w,\beta)}$, specified by parameters $n \in \mathbb{N}, k \in \{1,\dots,d\}, w \in [0,1]$, and $\beta \in [0,1]$. In $\mathcal{G}^d_{(n,k,w,\beta)}$ there are $n+1$ players: a *heavy player* of weight 1 and n *light players* $1,\dots,n$ of equal weights w. There are $2(n+1)$ resources $a_0, a_1, \dots, a_n, b_0, b_1, \dots, b_n$ where a_0 and b_0 have the same cost function c_0 and all other resources $a_1,\dots,a_n,b_1,\dots,b_n$ have the same cost function c_1 given by

$$c_0(x) = x^k \quad \text{and} \quad c_1(x) = \beta x^d.$$

Each player has exactly two strategies, and the strategy sets are given by

$$S_0 = \{\{a_0,\dots,a_n\},\{b_0,\dots,b_n\}\} \quad \text{and} \quad S_i = \{\{a_0,b_i\},\{b_0,a_i\}\} \quad \text{for } i = 1,\dots,n.$$

The structure of the strategies is visualized in Figure 1.

[2] Alternatively, we could have simply assumed succinct representability of the strategies. A prominent such case is that of *network* congestion games, where each player's strategies are all feasible paths between two specific nodes of an underlying graph. Notice however that, since in this paper we are proving hardness results, insisting on explicit representation only makes our results even stronger.

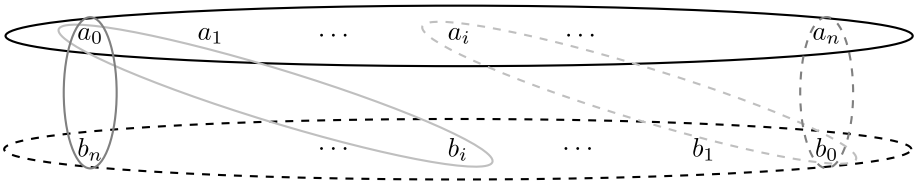


Figure 1 Strategies of the game $\mathcal{G}^d_{(n,k,w,\beta)}$. Resources contained in the two ellipses of the same colour correspond to the two strategies of a player. The strategies of the heavy player and light players n and i are depicted in black, grey and light grey, respectively.

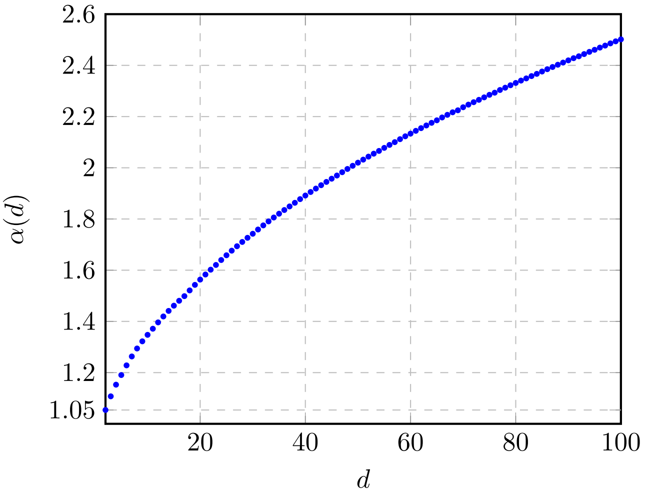


Figure 2 Nonexistence of $\alpha(d)$-PNE for weighted polynomial congestion games of degree d, as given by (3) in Theorem 1, for $d = 2, 3, \ldots, 100$. In particular, for small values of d, $\alpha(2) \approx 1.054$, $\alpha(3) \approx 1.107$ and $\alpha(4) \approx 1.153$.

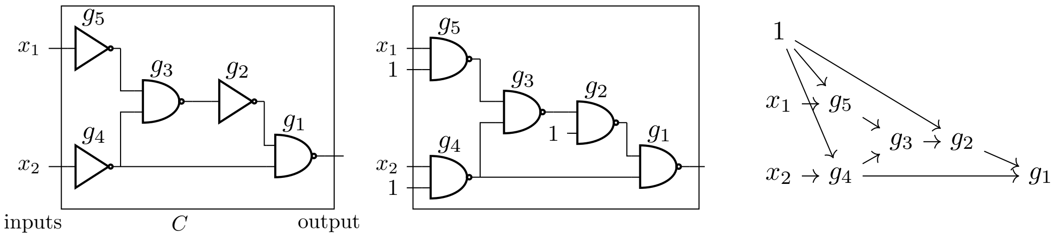
In the following theorem we give a lower bound on α, depending on parameters (n, k, w, β), such that games $\mathcal{G}^d_{(n,k,w,\beta)}$ do not admit an α-PNE. Maximizing this lower bound over all games in the family, we obtain a general lower bound $\alpha(d)$ on the inapproximability for polynomial congestion games of degree d (see (3) and its plot in Figure 2). Finally, choosing specific values for the parameters (n, k, w, β), we prove that $\alpha(d)$ is asymptotically lower bounded by $\Omega(\frac{\sqrt{d}}{\ln d})$.

▶ **Theorem 1.** *For any integer $d \geq 2$, there exist (weighted) polynomial congestion games of degree d that do not have α-approximate PNE for any $\alpha < \alpha(d)$, where*

$$\alpha(d) = \sup_{n,k,w,\beta} \min\left\{ \frac{1 + n\beta(1+w)^d}{(1+nw)^k + n\beta}, \frac{(1+w)^k + \beta w^d}{(nw)^k + \beta(1+w)^d} \right\} \qquad (3)$$
$$\text{s.t.} \quad n \in \mathbb{N}, k \in \{1, \ldots, d\}, w \in [0,1], \beta \in [0,1].$$

In particular, we have the asymptotics $\alpha(d) = \Omega\left(\frac{\sqrt{d}}{\ln d}\right)$ and the bound $\alpha(d) \geq \frac{\sqrt{d}}{2 \ln d}$, valid for large enough d. A plot of the exact values of $\alpha(d)$ (given by (3)*) for small degrees can be found in Figure 2.*

Interestingly, for the special case of $d = 2, 3, 4$, the values of $\alpha(d)$ (see Figure 2) yield *exactly* the same lower bounds with Hansknecht et al. [21]. This is a direct consequence of the fact that $n = 1$ turns out to be an optimal choice in (3) for $d \leq 4$, corresponding to an



(a) valid circuit C. **(b)** canonical form of C. **(c)** directed acyclic graph.

Figure 3 Example of a valid circuit C (having both NOT and NAND gates), its canonical form (having only NAND gates), and the directed acyclic graph corresponding to C.

instance with only $n+1 = 2$ players (which is the regime of the construction in [21]); however, this is not the case for larger values of d, where more players are now needed in order to derive the best possible value in (3). Furthermore, as we discussed also in Section 1.2, no construction with only 2 players can result in bounds larger than 2 (Theorem 8).

4 The Hardness Gadget

In this section we construct an unweighted polynomial congestion game from a Boolean circuit. In the α-PNE of this game the players emulate the computation of the circuit. This gadget will be used in reductions from CIRCUIT SATISFIABILITY to show NP-hardness of several problems related to the existence of approximate equilibria with some additional properties. For example, deciding whether a congestion game has an α-PNE where a certain set of players choose a specific strategy profile (Theorem 3).

Circuit Model

We consider Boolean circuits consisting of NOT gates and 2-input NAND gates only. We assume that the two inputs to every NAND gate are different. Otherwise we replace the NAND gate by a NOT gate, without changing the semantics of the circuit. We further assume that every input bit is connected to exactly one gate and this gate is a NOT gate. See Figure 3a for a *valid* circuit. In a valid circuit we replace every NOT gate by an equivalent NAND gate, where one of the inputs is fixed to 1. See the replacement of gates g_5, g_4 and g_2 in the example in Figure 3b. Thus, we look at circuits of 2-input NAND gates where both inputs to a NAND gate are different and every input bit of the circuit is connected to exactly one NAND gate where the other input is fixed to 1. A circuit of this form is said to be in *canonical form*. For a circuit C and a vector $x \in \{0,1\}^n$ we denote by $C(x)$ the output of the circuit on input x.

We model a circuit C in canonical form as a *directed acyclic graph*. The nodes of this graph correspond to the input bits $x_1, \dots, x_n$, the gates $g_1, \dots, g_K$ and a node 1 for all fixed inputs. There is an arc from a gate g to a gate g' if the output of g is input to gate g' and there are arcs from the fixed input and all input bits to the connected gates. We index the gates in reverse topological order, so that all successors of a gate g_k have a smaller index and the output of gate g_1 is the output of the circuit. Denote by $\delta^+(v)$ the set of the direct successors of node v. Then we have $|\delta^+(x_i)| = 1$ for all input bits x_i and $\delta^+(g_k) \subseteq \{g_{k'} \mid k' < k\}$ for every gate g_k. See Figure 3 for an example of a valid circuit, its canonical form and the corresponding directed acyclic graph.

Translation to Congestion Game

Fix some integer $d \geq 1$ and a parameter $\mu \geq 1 + 2 \cdot 3^{d+d/2}$. From a valid circuit in canonical form with input bits $x_1, \ldots, x_n$, gates $g_1, \ldots, g_K$ and the extra input fixed to 1, we construct a polynomial congestion game $\mathcal{G}_\mu^d$ of degree d. There are n *input players* $X_1, \ldots, X_n$ for every input bit, a *static player* P for the input fixed to 1, and K *gate players* $G_1, \ldots, G_K$ for the output bit of every gate. G_1 is sometimes called *output player* as g_1 corresponds to the output $C(x)$.

The idea is that every input and every gate player has a *zero* and a *one strategy*, corresponding to the respective bit being 0 or 1. In every α-PNE we want the players to emulate the computation of the circuit, i.e. the NAND semantics of the gates should be respected. For every gate g_k, we introduce two *resources* 0_k and 1_k. The zero (one) strategy of a player consists of the $0_{k'}$ ($1_{k'}$) resources of the direct successors in the directed acyclic graph corresponding to the circuit and its own 0_k (1_k) resource (for gate players). The static player has only one strategy playing all 1_k resources of the gates where one input is fixed to 1: $s_P = \{1_k \,|\, g_k \in \delta^+(1)\}$. Formally, we have

$$s_{X_i}^0 = \left\{0_k \,|\, g_k \in \delta^+(x_i)\right\} \text{ and } s_{X_i}^1 = \left\{1_k \,|\, g_k \in \delta^+(x_i)\right\}$$

for the zero and one strategy of an input player X_i. Recall that $\delta^+(x_i)$ is the set of direct successors of x_i, thus every strategy of an input player consists of exactly one resource. For a gate player G_k we have the two strategies

$$s_{G_k}^0 = \{0_k\} \cup \left\{0_{k'} \,|\, g_{k'} \in \delta^+(g_k)\right\} \text{ and } s_{G_k}^1 = \{1_k\} \cup \left\{1_{k'} \,|\, g_{k'} \in \delta^+(g_k)\right\}$$

consisting of at most k resources each. Notice that all 3 players related to a gate g_k (gate player G_k and the two players corresponding to the input bits) are different and observe that every resource 0_k and 1_k can be played by exactly those 3 players.

We define the cost functions of the resources using parameter μ. The cost functions for resources 1_k are given by c_{1_k} and for resources 0_k by c_{0_k}, where

$$c_{1_k}(x) = \mu^k x^d \qquad \text{and} \qquad c_{0_k}(x) = \lambda \mu^k x^d, \text{ with } \lambda = 3^{d/2}. \tag{4}$$

Our construction here is inspired by the lockable circuit games of Skopalik and Vöcking [34]. The key technical differences are that our gadgets use polynomial cost functions (instead of general cost functions) and only 2 resources per gate (instead of 3). Moreover, while in [34] these games are used as part of a PLS-reduction from Circuit/FLIP, we are also interested in constructing a gadget to be studied on its own, since this can give rise to additional results of independent interest (see Theorem 3).

Properties of the Gadget

For a valid circuit C in canonical form consider the game $\mathcal{G}_\mu^d$ as defined above. We interpret any strategy profile $\mathbf{s}$ of the input players as a bit vector $x \in \{0,1\}^n$ by setting $x_i = 0$ if $s_{X_i} = s_{X_i}^0$ and $x_i = 1$ otherwise. The gate players are said to *follow the NAND semantics* in a strategy profile, if for every gate g_k the following holds:

- if both players corresponding to the input bits of g_k play their one strategy, then the gate player G_k plays her zero strategy;
- if at least one of the players corresponding to the input bits of g_k plays her zero strategy, then the gate player G_k plays her one strategy.

We show that for the right choice of α, the set of α-PNE in $\mathcal{G}_\mu^d$ is the same as the set of all strategy profiles where the gate players follow the NAND semantics.

Define

$$\varepsilon(\mu) = \frac{3^{d+d/2}}{\mu - 1}. \tag{5}$$

From our choice of μ, we obtain $3^{d/2} - \varepsilon(\mu) \geq 3^{d/2} - \frac{1}{2} > 1$. For any valid circuit C in canonical form and a valid choice of μ the following lemma holds for $\mathcal{G}^d_\mu$.

▶ **Lemma 2.** *Let $\mathbf{s}_X$ be any strategy profile for the input players $X_1, \dots, X_n$ and let $x \in \{0,1\}^n$ be the bit vector represented by $\mathbf{s}_X$. For any $\mu \geq 1 + 2 \cdot 3^{d+d/2}$ and any $1 \leq \alpha < 3^{d/2} - \varepsilon(\mu)$, there is a unique α-approximate PNE*[3] *in $\mathcal{G}^d_\mu$ where the input players play according to $\mathbf{s}_X$. In particular, in this α-PNE the gate players follow the NAND semantics, and the output player G_1 plays according to $C(x)$.*

Proof sketch. We first fix the input players to the strategies given by $\mathbf{s}_X$ and show that then all gate players follow the NAND semantics (switching to the strategy corresponding to the NAND of their input bits is an α-improving move). Secondly, we argue that the input players have no incentive to change their strategy in any α-PNE where all gate players follow the NAND semantics. Hence, every strategy profile for the input players can be extended to an α-PNE in $\mathcal{G}^d_\mu$ that is uniquely defined by the NAND semantics. ◀

We are now ready to show our main result of this section; using the circuit game described above, we show NP-hardness of deciding whether approximate equilibria with additional properties exist.

▶ **Theorem 3.** *The following problems are NP-hard, even for* unweighted *polynomial congestion games of degree $d \geq 1$, for all $\alpha \in [1, 3^{d/2})$ and all $z > 0$:*

- *"Does there exist an α-approximate PNE in which a certain subset of players are playing a specific strategy profile?"*
- *"Does there exist an α-approximate PNE in which a certain resource is used by at least one player?"*
- *"Does there exist an α-approximate PNE in which a certain player has cost at most z?"*

Proof sketch. We use reductions from the NP-hard problem Circuit Satisfiability. For a circuit C we consider the game $\mathcal{G}^d_\mu$ as described above and focus on the output player G_1. Using Lemma 2 we get a one-to-one correspondence between satisfying assignments for C and α-PNE in $\mathcal{G}^d_\mu$ where G_1 plays her one strategy. ◀

5 Hardness of Existence

In this section we show that it is NP-hard to decide whether a polynomial congestion game has an α-PNE. For this we use a black-box reduction: our hard instance is obtained by combining any (weighted) polynomial congestion game $\mathcal{G}$ without α-PNE (i.e., the game from Section 3) with the circuit gadget of the previous section. To achieve this, it would be convenient to make some assumptions on the game $\mathcal{G}$, which however do not influence the existence or nonexistence of approximate equilibria.

[3] Which, as a matter of fact, is actually also an *exact* PNE.

Structural Properties of $\mathcal{G}$

Without loss of generality, we assume that a weighted polynomial congestion game of degree d has the following structural properties.

- *No player has an empty strategy.* If, for some player i, $\emptyset \in S_i$, then this strategy would be α-dominating for i. Removing i from the game description would not affect the (non)existence of (approximate) equilibria[4].
- *No player has zero weight.* If a player i had zero weight, her strategy would not influence the costs of the strategies of the other players. Again, removing i from the game description would not affect the (non)existence of equilibria.
- *Each resource e has a monomial cost function with a strictly positive coefficient*, i.e. $c_e(x) = a_e x^{k_e}$ where $a_e > 0$ and $k_e \in \{0, \dots, d\}$. If a resource had a more general cost function $c_e(x) = a_{e,0} + a_{e,1}x + \dots + a_{e,d}x^d$, we could split it into at most $d+1$ resources with (positive) monomial costs, $c_{e,0}(x) = a_{e,0}$, $c_{e,1}(x) = a_{e,1}x$, $\dots$, $c_{e,d}(x) = a_{e,d}x^d$. These monomial cost resources replace the original resource, appearing on every strategy that included e.
- *No resource e has a constant cost function.* If a resource e had a constant cost function $c_e(x) = a_{e,0}$, we could replace it by new resources having monomial cost. For each player i of weight w_i, replace resource e by a resource e_i with monomial cost $c_{e_i}(x) = \frac{a_{e,0}}{w_i}x$, that is used exclusively by player i on her strategies that originally had resource e. Note that $c_{e_i}(w_i) = a_{e,0}$, so that this modification does not change the player's costs, neither has an effect on the (non)existence of approximate equilibria. If a resource has cost function constantly equal to zero, we can simply remove it from the description of the game.

For a game having the above properties, we define the (strictly positive) quantities

$$a_{\min} = \min_{e\in E} a_e, \quad W = \sum_{i\in N} w_i, \quad c_{\max} = \sum_{e\in E} c_e(W). \tag{6}$$

Note that $c_{\max}$ is an upper bound on the cost of any player on any strategy profile.

Rescaling of $\mathcal{G}$

In our construction of the combined game we have to make sure that the weights of the players in $\mathcal{G}$ are smaller than the weights of the players in the circuit gadget. We introduce the following rescaling argument.

For any $\gamma \in (0, 1]$ define the game $\tilde{\mathcal{G}}_\gamma$, where we rescale the player weights and resource cost coefficients in $\mathcal{G}$ as

$$\tilde{a}_e = \gamma^{d+1-k_e} a_e, \quad \tilde{w}_i = \gamma w_i, \quad \tilde{c}_e(x) = \tilde{a}_e x^{k_e}. \tag{7}$$

This changes the quantities in (6) for $\tilde{\mathcal{G}}_\gamma$ to (recall that $k_e \geq 1$)

$$\begin{aligned}
\tilde{a}_{\min} &= \min_{e\in E} \tilde{a}_e = \min_{e\in E} \gamma^{d+1-k_e} a_e \geq \gamma^d \min_{e\in E} a_e = \gamma^d a_{\min},\\
\tilde{W} &= \sum_{i\in N} \tilde{w}_i = \sum_{i\in N} \gamma w_i = \gamma W,\\
\tilde{c}_{\max} &= \sum_{e\in E} \tilde{c}_e(\tilde{W}) = \sum_{e\in E} \tilde{a}_e(\gamma W)^{k_e} = \sum_{e\in E} \gamma^{d+1} a_e W^{k_e} = \gamma^{d+1} \sum_{e\in E} c_e(W) = \gamma^{d+1} c_{\max}.
\end{aligned}$$

In $\tilde{\mathcal{G}}_\gamma$ the player costs are all uniformly scaled as $\tilde{C}_i(\mathbf{s}) = \gamma^{d+1} C_i(\mathbf{s})$, so that the Nash dynamics and the (non)existence of equilibria are preserved.

[4] By this we mean, if $\mathcal{G}$ has (resp. does not have) α-PNE, then $\tilde{\mathcal{G}}$, obtained by removing player i from the game, still has (resp. still does not have) α-PNE.

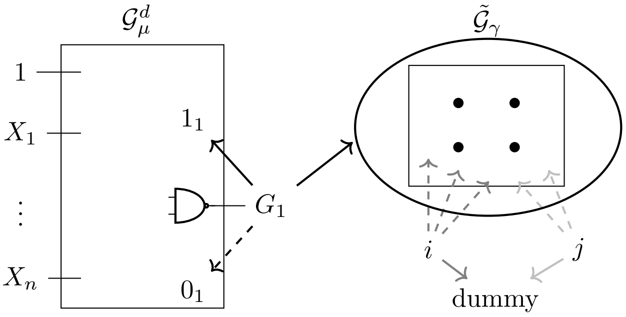


Figure 4 Combination of a circuit game (on the left) and a game without approximate equilibria (on the right). Changes to the subgames are indicated by solid arrows. The new one strategy of G_1 consists of 1_1 and all resources in $\tilde{\mathcal{G}}_\gamma$, while the zero strategy stays unchanged. The players of $\tilde{\mathcal{G}}_\gamma$ get a new strategy (the dummy resource), and keep their old strategies playing in $\tilde{\mathcal{G}}_\gamma$.

The next lemma formalizes the combination of both game gadgets and, furthermore, establishes the gap-introduction in the equilibrium factor. Using it, we will derive our key hardness tool of Theorem 5.

▶ **Lemma 4.** *Fix any integer $d \geq 2$ and rational $\alpha \geq 1$. Suppose there exists a weighted polynomial congestion game $\mathcal{G}$ of degree d that does* not *have an α-approximate PNE. Then, for any circuit C there exists a game $\tilde{\mathcal{G}}_C$ with the following property: the sets of α-approximate PNE and exact PNE of $\tilde{\mathcal{G}}_C$ coincide and are in one-to-one correspondence with the set of satisfying assignments of C. In particular, one of the following holds: either*

1. *C has a satisfying assignment, in which case $\tilde{\mathcal{G}}_C$ has an exact PNE (and thus, also an α-approximate PNE); or*
2. *C has no satisfying assignments, in which case $\tilde{\mathcal{G}}_C$ has no α-approximate PNE (and thus, also no exact PNE).*

Proof. Let $\mathcal{G}$ be a congestion game as in the statement of the theorem having the above mentioned structural properties. Recalling that weighted polynomial congestion games of degree d have d-PNE [3], this implies that $\alpha < d < 3^{d/2}$. Fix some $0 < \varepsilon < 3^{d/2} - \alpha$ and take $\mu \geq 1 + \frac{3^{d+d/2}}{\min\{\varepsilon,1\}}$; in this way $\alpha < 3^{d/2} - \varepsilon \leq 3^{d/2} - \varepsilon(\mu)$.

Given a circuit C we construct the game $\tilde{\mathcal{G}}_C$ as follows. We combine the game $\mathcal{G}_\mu^d$ whose Nash dynamics model the NAND semantics of C, as described in Section 4, with the game $\tilde{\mathcal{G}}_\gamma$ obtained from $\mathcal{G}$ via the aforementioned rescaling. We choose $\gamma \in (0, 1]$ sufficiently small such that the following three inequalities hold for the quantities in (6) for $\mathcal{G}$:

$$\gamma W < 1, \quad \gamma \sum_{e \in E} a_e < \frac{\mu}{\mu - 1}\left(\frac{3}{2}\right)^d, \quad \gamma\alpha^2 < \frac{a_{\min}}{c_{\max}}. \tag{8}$$

Thus, the set of players in $\tilde{\mathcal{G}}_C$ corresponds to the (disjoint) union of the static, input and gate players in $\mathcal{G}_\mu^d$ (which all have weights 1) and the players in $\tilde{\mathcal{G}}_\gamma$ (with weights $\tilde{w}_i$). We also consider a new dummy resource with constant cost $c_{\text{dummy}}(x) = \frac{\tilde{a}_{\min}}{\alpha}$. Thus, the set of resources corresponds to the (disjoint) union of the gate resources $0_k, 1_k$ in $\mathcal{G}_\mu^d$, the resources in $\tilde{\mathcal{G}}_\gamma$, and the dummy resource. We augment the strategy space of the players as follows:

- each input player or gate player of $\mathcal{G}_\mu^d$ that is *not* the output player G_1 has the same strategies as in $\mathcal{G}_\mu^d$ (i.e. either the zero or the one strategy);
- the zero strategy of the output player G_1 is the same as in $\mathcal{G}_\mu^d$, but her one strategy is augmented with *every* resource in $\tilde{\mathcal{G}}_\gamma$; that is, $s_{G_1}^1 = \{1_1\} \cup E(\tilde{\mathcal{G}}_\gamma)$;
- each player i in $\tilde{\mathcal{G}}_\gamma$ keeps her original strategies as in $\tilde{\mathcal{G}}_\gamma$, and gets a new dummy strategy $s_{i,\text{dummy}} = \{\text{dummy}\}$.

A graphical representation of the game $\tilde{\mathcal{G}}_C$ can be seen in Figure 4.

To finish the proof, we need to show that every α-PNE of $\tilde{\mathcal{G}}_C$ is an exact PNE and corresponds to a satisfying assignment of C; and, conversely, that every satisfying assignment of C gives rise to an exact PNE of $\tilde{\mathcal{G}}_C$ (and thus, an α-PNE as well).

Suppose that $\mathbf{s}$ is an α-PNE of $\tilde{\mathcal{G}}_C$, and let $\mathbf{s}_X$ denote the strategy profile restricted to the input players of $\mathcal{G}_\mu^d$. Then, as in the proof of Lemma 2, every gate player that is not the output player must respect the NAND semantics, and this is an α-dominating strategy. For the output player, either $\mathbf{s}_X$ is a non-satisfying assignment, in which case the zero strategy of G_1 was α-dominating, and this remains α-dominating in the game $\tilde{\mathcal{G}}_C$ (since only the cost of the one strategy increased for the output player); or $\mathbf{s}_X$ is a satisfying assignment. In the second case, we now argue that the one strategy of G_1 remains α-dominating. The cost of the output player on the zero strategy is at least $c_{0_1}(2) = \lambda\mu 2^d$, and the cost on the one strategy is at most

$$c_{1_1}(2)+\sum_{e\in E}\tilde{c}_e(1+\gamma W) = \mu 2^d+\sum_{e\in E}\gamma^{d+1-k_e}a_e(1+\gamma W)^{k_e} < \mu 2^d+\gamma\sum_{e\in E}a_e 2^d < \mu 2^d+\frac{\mu}{\mu-1}3^d,$$

where we used the first and second bounds from (8). Thus, the ratio between the costs is at least

$$\frac{\lambda\mu 2^d}{\mu 2^d+\frac{\mu}{\mu-1}3^d} = \lambda\left(\frac{1}{1+\frac{1}{\mu-1}\left(\frac{3}{2}\right)^d}\right) > 3^{d/2}\left(\frac{1}{1+\frac{1}{\mu-1}3^d}\right) > 3^{d/2}-\varepsilon(\mu) > \alpha.$$

Given that the gate players must follow the NAND semantics, the input players are also locked to their strategies (i.e. they have no incentive to change) due to the proof of Lemma 2. The only players left to consider are the players from $\tilde{\mathcal{G}}_\gamma$. First we show that, since $\mathbf{s}$ is an α-PNE, the output player must be playing her one strategy. If this was not the case, then each dummy strategy of a player in $\tilde{\mathcal{G}}_\gamma$ is α-dominated by any other strategy: the dummy strategy incurs a cost of $\frac{\tilde{a}_{\min}}{\alpha} \geq \gamma^d\frac{a_{\min}}{\alpha}$, whereas any other strategy would give a cost of at most $\tilde{c}_{\max} = \gamma^{d+1}c_{\max}$ (this is because the output player is not playing any of the resources in $\tilde{\mathcal{G}}_\gamma$). The ratio between the costs is thus at least

$$\frac{\gamma^d a_{\min}}{\gamma^{d+1}c_{\max}\alpha} = \frac{a_{\min}}{\gamma c_{\max}\alpha} > \alpha.$$

Since the dummy strategies are α-dominated, the players in $\tilde{\mathcal{G}}_\gamma$ must be playing on their original sets of strategies. The only way for $\mathbf{s}$ to be an α-PNE would be if $\mathcal{G}$ had an α-PNE to begin with, which yields a contradiction. Thus, the output player is playing the one strategy (and hence, is present in every resource in $\tilde{\mathcal{G}}_\gamma$). In such a case, we can conclude that each dummy strategy is now α-dominating. If a player i in $\tilde{\mathcal{G}}_\gamma$ is not playing a dummy strategy, she is playing at least one resource in $\tilde{\mathcal{G}}_\gamma$, say resource e. Her cost is at least $\tilde{c}_e(1+\tilde{w}_i) = \tilde{a}_e(1+\tilde{w}_i)^{k_e} > \tilde{a}_e \geq \tilde{a}_{\min}$ (the strict inequality holds since, by the structural properties of our game, all of $\tilde{a}_e$, $\tilde{w}_i$ and k_e are strictly positive quantities). On the other hand, the cost of playing the dummy strategy is $\frac{\tilde{a}_{\min}}{\alpha}$. Thus, the ratio between the costs is greater than α.

We have concluded that, if $\mathbf{s}$ is an α-PNE of $\tilde{\mathcal{G}}_C$, then $\mathbf{s}_X$ corresponds to a satisfying assignment of C, all the gate players are playing according to the NAND semantics, the output player is playing the one strategy, and all players of $\tilde{\mathcal{G}}_\gamma$ are playing the dummy strategies. In this case, we also have observed that each player's current strategy is α-dominating, so the strategy profile is an exact PNE. To finish the proof, we need to argue that every satisfying assignment gives rise to a unique α-PNE. Let $\mathbf{s}_X$ be the strategy profile corresponding to this assignment for the input players in $\mathcal{G}_\mu^d$. Then, as before, there is one and exactly one α-PNE $\mathbf{s}$ in $\tilde{\mathcal{G}}_C$ that agrees with $\mathbf{s}_X$; namely, each gate player follows the NAND semantics, the output player plays the one strategy, and the players in $\tilde{\mathcal{G}}_\gamma$ play the dummy strategies. ◀

By approximating all numbers occurring in the construction of Lemma 4 (weights, coefficients, approximation factor) by rationals, we obtain a polynomial-time reduction from Circuit Satisfiability, and thus the following theorem.

▶ **Theorem 5.** *For any integer $d \geq 2$ and rational $\alpha \geq 1$, suppose there exists a weighted polynomial congestion game which does not have an α-approximate PNE. Then it is NP-complete to decide whether (weighted) polynomial congestion games of degree d have an α-approximate PNE.*

Proof. Let $d \geq 2$ and $\alpha \geq 1$. Let $\mathcal{G}$ be a weighted polynomial congestion game of degree d that has no α-PNE; this means that for every strategy profile $\mathbf{s}$ there exists a player i and a strategy $s_i' \neq s_i$ such that $C_i(s_i, \mathbf{s}_{-i}) > \alpha \cdot C_i(s_i', \mathbf{s}_{-i})$. Note that the functions C_i are polynomials of degree d and hence they are continuous on the weights w_i and the coefficients a_e appearing on the cost functions. Hence, any arbitrarily small perturbation of the w_i, a_e does not change the sign of the above inequality. Thus, without loss of generality, we can assume that all w_i, a_e are rational numbers.

Next, we consider the game $\tilde{\mathcal{G}}_\gamma$ obtained from $\mathcal{G}$ by rescaling, as in the proof of Lemma 4. Notice that the rescaling is done via the choice of a sufficiently small γ, according to (8), and hence in particular we can take γ to be a sufficiently small rational. In this way, all the player weights and coefficients in the cost of resources are rational numbers scaled by a rational number and hence rationals.

Finally, we are able to provide the desired NP reduction from Circuit Satisfiability. Given a Boolean circuit C' built with 2-input NAND gates, transform it into a valid circuit C in canonical form. From C we can construct in polynomial time the game $\tilde{\mathcal{G}}_C$ as described in the proof of Lemma 4. The "circuit part", i.e. the game $\mathcal{G}_\mu^d$, is obtained in polynomial time from C, as in the proof of Theorem 3; the description of the game $\tilde{\mathcal{G}}_\gamma$ involves only rational numbers, and hence the game can be represented by a constant number of bits (i.e. independent of the circuit C). Similarly, the additional dummy strategy has a constant delay of ${}^{\tilde{a}_{\min}}/_{\alpha}$, and can be represented with a single rational number. Merging both $\mathcal{G}_\mu^d$ and $\tilde{\mathcal{G}}_\gamma$ into a single game $\tilde{\mathcal{G}}_C$ can be done in linear time. Since C has a satisfying assignment iff $\tilde{\mathcal{G}}_C$ has an α-PNE (or α-PNE), this concludes that the problem described is NP-hard.

The problem is clearly in NP: given a weighted polynomial congestion game of degree d and a strategy profile $\mathbf{s}$, one can check if $\mathbf{s}$ is an α-PNE by computing the ratios between the cost of each player in $\mathbf{s}$ and their cost for each possible deviation, and comparing these ratios with α. ◀

Combining the hardness result of Theorem 5 together with the nonexistence result of Theorem 1 we get the following corollary, which is the main result of this section.

▶ **Corollary 6.** *For any integer $d \geq 2$ and rational $\alpha \in [1, \alpha(d))$, it is NP-complete to decide whether (weighted) polynomial congestion games of degree d have an α-approximate PNE, where $\alpha(d) = \tilde{\Omega}(\sqrt{d})$ is the same as in Theorem 1.*

Notice that, in the proof of Lemma 4 and Theorem 5, we constructed a polynomial-time reduction from Circuit Satisfiability to the problem of determining whether a given congestion game has an α-PNE. Not only does this reduction map YES-instances of one problem to YES-instances of the other, but it also induces a bijection between the sets of satisfying assignments of a circuit C and α-PNE of the corresponding game $\tilde{\mathcal{G}}_C$. That is, this reduction is *parsimonious*. As a consequence, we can directly lift hardness of problems associated with counting satisfying assignments to Circuit Satisfiability into problems associated with counting equilibria in congestion games:

▶ **Corollary 7.** *Let $k \geq 1$ and $d \geq 2$ be integers and $\alpha \in [1, \alpha(d))$ where $\alpha(d) = \tilde{\Omega}(\sqrt{d})$ is the same as in Theorem 1. Then*

- *it is #P-hard to count the number of α-approximate PNE of (weighted) polynomial congestion games of degree d;*
- *it is NP-hard to decide whether a (weighted) polynomial congestion game of degree d has at least k distinct α-approximate PNE.*

Proof. The hardness of the first problem comes from the #P-hardness of the counting version of Circuit Satisfiability (see, e.g., [29, Ch. 18]). For the hardness of the second problem, it is immediate to see that the following problem is NP-complete, for any fixed integer $k \geq 1$: given a circuit C, decide whether there are at least k distinct satisfying assignments for C (simply add "dummy" variables to the description of the circuit). ◀

6 General Cost Functions

In this final section we leave the domain of polynomial latencies and study the existence of approximate equilibria in general congestion games having arbitrary (nondecreasing) cost functions. Our parameter of interest, with respect to which both our positive and negative results are going to be stated, is the number of players n. We start by showing that n-PNE always exist:

▶ **Theorem 8.** *Every weighted congestion game with n players and arbitrary (nondecreasing) cost functions has an n-approximate PNE.*

Proof. Fix a weighted congestion game with $n \geq 2$ players, some strategy profile $\mathbf{s}$, and a possible deviation s_i' of player i. First notice that we can write the change in the cost of any other player $j \neq i$ as

$$\begin{aligned} C_j(s_i', \mathbf{s}_{-i}) - C_j(\mathbf{s}) &= \sum_{e \in s_j} c_e(x_e(s_i', \mathbf{s}_{-i})) - \sum_{e \in s_j} c_e(x_e(\mathbf{s})) \\ &= \sum_{e \in s_j \cap (s_i' \setminus s_i)} [c_e(x_e(s_i', \mathbf{s}_{-i})) - c_e(x_e(\mathbf{s}))] \\ &+ \sum_{e \in s_j \cap (s_i \setminus s_i')} [c_e(x_e(s_i', \mathbf{s}_{-i})) - c_e(x_e(\mathbf{s}))] \qquad (9) \end{aligned}$$

Furthermore, we can upper bound this by

$$\begin{aligned} C_j(s'_i, \mathbf{s}_{-i}) - C_j(\mathbf{s}) &\leq \sum_{e \in s_j \cap (s'_i \setminus s_i)} [c_e(x_e(s'_i, \mathbf{s}_{-i})) - c_e(x_e(\mathbf{s}))] \\ &\leq \sum_{e \in s'_i} c_e(x_e(s'_i, \mathbf{s}_{-i})) \\ &= C_i(s'_i, \mathbf{s}_{-i}), \end{aligned} \tag{10}$$

the first inequality holding due to the fact that the second sum in (9) contains only nonpositive terms (since the latency functions are nondecreasing).

Next, define the social cost $C(\mathbf{s}) = \sum_{i \in N} C_i(\mathbf{s})$. Adding the above inequality over all players $j \neq i$ (of which there are $n-1$) and rearranging, we successively derive:

$$\begin{aligned} \sum_{j \neq i} C_j(s'_i, \mathbf{s}_{-i}) - \sum_{j \neq i} C_j(\mathbf{s}) &\leq (n-1) C_i(s'_i, \mathbf{s}_{-i}) \\ (C(s'_i, \mathbf{s}_{-i}) - C_i(s'_i, \mathbf{s}_{-i})) - (C(\mathbf{s}) - C_i(\mathbf{s})) &\leq (n-1) C_i(s'_i, \mathbf{s}_{-i}) \\ C(s'_i, \mathbf{s}_{-i}) - C(\mathbf{s}) &\leq n C_i(s'_i, \mathbf{s}_{-i}) - C_i(\mathbf{s}). \end{aligned} \tag{11}$$

We conclude that, if s'_i is an n-improving deviation for player i (i.e., $nC_i(s'_i, \mathbf{s}_{-i}) < C_i(\mathbf{s})$), then the social cost must strictly decrease after this move. Thus, any (global or local) minimizer of the social cost must be an n-PNE (the existence of such a minimizer is guaranteed by the fact that the strategy spaces are finite). ◀

The proof not only establishes the existence of n-approximate equilibria in general congestion games, but also highlights a few additional interesting features. First, due to the key inequality (11), n-PNE are reachable via sequences of n-improving moves, in addition to arising also as minimizers of the social cost function. These attributes give a nice "constructive" flavour to Theorem 8. Secondly, exactly because social cost optima are n-PNE, the *Price of Stability*[5] of n-PNE is optimal (i.e., equal to 1) as well. Another, more succinct way, to interpret these observations is within the context of *approximate potentials* (see, e.g., [6, 10, 9]); (11) establishes that the social cost itself is always an n-approximate potential of any congestion game.

Next, we design a family of games $\mathcal{G}_n$ that do not admit $\Theta\left(\frac{n}{\ln n}\right)$-PNE, thus nearly matching the upper bound Theorem 8. In the game $\mathcal{G}_n$ there are $n = m+1$ players $0, 1, \dots, m$, where player i has weight $w_i = 1/2^i$. In particular, this means that for any $i \in \{1, \dots, m\}$: $\sum_{k=i}^{m} w_k < w_{i-1} \leq w_0$. Furthermore, there are $2(m+1)$ resources $a_0, a_1, \dots, a_m, b_0, b_1, \dots, b_m$, where resources a_i and b_i have the same cost function c_i given by

$$c_{a_0}(x) = c_{b_0}(x) = c_0(x) = \begin{cases} 1, & \text{if } x \geq w_0, \\ 0, & \textit{otherwise}; \end{cases}$$

and for all $i \in \{1, \dots, m\}$,

$$c_{a_i}(x) = c_{b_i}(x) = c_i(x) = \begin{cases} \frac{1}{\xi}\left(1 + \frac{1}{\xi}\right)^{i-1}, & \text{if } x \geq w_0 + w_i, \\ 0, & \textit{otherwise}. \end{cases}$$

Where $\xi = \Phi_{n-1}$ is the positive solution of $(x+1)^{n-1} = x^n$.

[5] The Price of Stability (PoS) is a well-established and extensively studied notion in algorithmic game theory, originally studied in [2, 12]. It captures the minimum approximation ratio of the social cost between equilibria and the optimal solution (see, e.g., [7, 9]); in other words, it is the best-case analogue of the the Price of Anarchy (PoA) notion of Koutsoupias and Papadimitriou [25].

The strategy set of player 0 and of all players $i \in \{1, \dots, m\}$ are, respectively,

$$S_0 = \{\{a_0, \dots, a_m\}, \{b_0, \dots, b_m\}\}, \qquad \text{and} \qquad S_i = \{\{a_0, \dots, a_{i-1}, b_i\}, \{b_0, \dots, b_{i-1}, a_i\}\}.$$

Analysing the costs of strategy profiles in $\mathcal{G}_n$ (see [8]) we get the following theorem.

▶ **Theorem 9.** *For any integer $n \geq 2$, there exist weighted congestion games with n players and general cost functions that do not have α-approximate PNE for any $\alpha < \Phi_{n-1}$, where $\Phi_m \sim \frac{m}{\ln m}$ is the unique positive solution of $(x+1)^m = x^{m+1}$.*

Similar to the spirit of the rest of our paper so far, we'd like to show an NP-hardness result for deciding existence of α-PNE for general games as well. We do exactly that in the following theorem, where now α grows as $\tilde{\Theta}(n)$. Again, we use the circuit gadget and combine it with the game from the previous nonexistence Theorem 9. The main difference to the previous reductions is that now n is part of the input. On the other hand we are not restricted to polynomial latencies, so we use step functions having a single breakpoint.

▶ **Theorem 10.** *Let $\varepsilon > 0$, and let $\tilde{\alpha} : \mathbb{N}_{\geq 2} \longrightarrow \mathbb{Q}$ be any (polynomial-time computable) sequence such that $1 \leq \tilde{\alpha}(n) < \frac{\Phi_{n-1}}{1+\varepsilon} = \tilde{\Theta}(n)$, where $\Phi_m \sim \frac{m}{\ln m}$ is the unique positive solution of $(x+1)^m = x^{m+1}$. Then, it is NP-complete to decide whether a (weighted) congestion game with n players has an $\tilde{\alpha}(n)$-approximate PNE.*

7 Discussion and Future Directions

In this paper we showed that weighted congestion games with polynomial latencies of degree d do not have α-PNE for $\alpha < \alpha(d) = \Omega\left(\frac{\sqrt{d}}{\ln d}\right)$. For general cost functions, we proved that n-PNE always exist whereas α-PNE in general do not, where n is the number of players and $\alpha < \Phi_{n-1} = \Theta\left(\frac{n}{\ln n}\right)$. We also transformed the nonexistence results into complexity-theoretic results, establishing that deciding whether such α-PNE exist is itself an NP-hard problem.

We now identify two possible directions for follow-up work. A first obvious question would be to reduce the nonexistence gap between $\Omega\left(\frac{\sqrt{d}}{\ln d}\right)$ (derived in Theorem 1 of this paper) and d (shown in [3]) for polynomials of degree d; similarly for the gap between $\Theta\left(\frac{n}{\ln n}\right)$ (Theorem 9) and n (Theorem 8) for general cost functions and n players. Notice that all current methods for proving upper bounds (i.e., existence) are essentially based on potential function arguments; thus it might be necessary to come up with novel ideas and techniques to overcome the current gaps.

A second direction would be to study the complexity of *finding* α-PNE, when they are guaranteed to exist. For example, for polynomials of degree d, we know that d-improving dynamics eventually reach a d-PNE [3], and so finding such an approximate equilibrium lies in the complexity class PLS of local search problems (see, e.g., [24, 33]). However, from a complexity theory perspective the only known lower bound is the PLS-completeness of finding an *exact* equilibrium for *unweighted* congestion games [14] (and this is true even for $d = 1$, i.e., affine cost functions; see [1]). On the other hand, we know that $d^{O(d)}$-PNE can be computed in polynomial time (see, e.g., [5, 18, 15]). It would be then very interesting to establish a "gradation" in complexity (e.g., from NP-hardness to PLS-hardness to P) as the parameter α increases from 1 to $d^{O(d)}$.

References

1 Heiner Ackermann, Heiko Röglin, and Berthold Vöcking. On the impact of combinatorial structure on congestion games. *Journal of the ACM*, 55(6):1–22, December 2008. doi:10.1145/1455248.1455249.

2 Elliot Anshelevich, Anirban Dasgupta, Jon Kleinberg, Éva Tardos, Tom Wexler, and Tim Roughgarden. The price of stability for network design with fair cost allocation. *SIAM Journal on Computing*, 38(4):1602–1623, 2008. doi:10.1137/070680096.

3 Ioannis Caragiannis and Angelo Fanelli. On approximate pure Nash equilibria in weighted congestion games with polynomial latencies. In *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 133:1–133:12, 2019. doi:10.4230/LIPIcs.ICALP.2019.133.

4 Ioannis Caragiannis, Angelo Fanelli, Nick Gravin, and Alexander Skopalik. Efficient computation of approximate pure Nash equilibria in congestion games. In *Proceedings of the 52nd IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 532–541, 2011. doi:10.1109/focs.2011.50.

5 Ioannis Caragiannis, Angelo Fanelli, Nick Gravin, and Alexander Skopalik. Approximate pure nash equilibria in weighted congestion games: Existence, efficient computation, and structure. *ACM Trans. Econ. Comput.*, 3(1):2:1–2:32, March 2015. doi:10.1145/2614687.

6 Ho-Lin Chen and Tim Roughgarden. Network design with weighted players. *Theory of Computing Systems*, 45(2):302, 2008. doi:10.1007/s00224-008-9128-8.

7 George Christodoulou and Martin Gairing. Price of stability in polynomial congestion games. *ACM Transactions on Economics and Computation*, 4(2):1–17, 2015. doi:10.1145/2841229.

8 George Christodoulou, Martin Gairing, Yiannis Giannakopoulos, Diogo Poças, and Clara Waldmann. Existence and complexity of approximate equilibria in weighted congestion games. *CoRR*, abs/2002.07466, February 2020. arXiv:2002.07466.

9 George Christodoulou, Martin Gairing, Yiannis Giannakopoulos, and Paul G. Spirakis. The price of stability of weighted congestion games. *SIAM Journal on Computing*, 48(5):1544–1582, 2019. doi:10.1137/18M1207880.

10 George Christodoulou, Elias Koutsoupias, and Paul G. Spirakis. On the performance of approximate equilibria in congestion games. *Algorithmica*, 61(1):116–140, 2011. doi:10.1007/s00453-010-9449-2.

11 Vincent Conitzer and Tuomas Sandholm. New complexity results about Nash equilibria. *Games and Economic Behavior*, 63(2):621–641, 2008. doi:10.1016/j.geb.2008.02.015.

12 José R. Correa, Andreas S. Schulz, and Nicolás E. Stier-Moses. Selfish routing in capacitated networks. *Mathematics of Operations Research*, 29(4):961–976, 2004. doi:10.1287/moor.1040.0098.

13 Juliane Dunkel and Andreas S. Schulz. On the complexity of pure-strategy Nash equilibria in congestion and local-effect games. *Mathematics of Operations Research*, 33(4):851–868, 2008. doi:10.1287/moor.1080.0322.

14 Alex Fabrikant, Christos Papadimitriou, and Kunal Talwar. The complexity of pure nash equilibria. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC)*, pages 604–612, 2004. doi:10.1145/1007352.1007445.

15 Matthias Feldotto, Martin Gairing, Grammateia Kotsialou, and Alexander Skopalik. Computing approximate pure nash equilibria in shapley value weighted congestion games. In *Web and Internet Economics*, pages 191–204. Springer International Publishing, 2017. doi:10.1007/978-3-319-71924-5_14.

16 Dimitris Fotakis, Spyros Kontogiannis, Elias Koutsoupias, Marios Mavronicolas, and Paul Spirakis. The structure and complexity of Nash equilibria for a selfish routing game. *Theoretical Computer Science*, 410(36):3305–3326, 2009. doi:10.1016/j.tcs.2008.01.004.

17 Dimitris Fotakis, Spyros Kontogiannis, and Paul Spirakis. Selfish unsplittable flows. *Theoretical Computer Science*, 348(2):226–239, 2005. doi:10.1016/j.tcs.2005.09.024.

18 Yiannis Giannakopoulos, Georgy Noarov, and Andreas S. Schulz. An improved algorithm for computing approximate equilibria in weighted congestion games. *CoRR*, abs/1810.12806, October 2018. arXiv:1810.12806.

19 Itzhak Gilboa and Eitan Zemel. Nash and correlated equilibria: Some complexity considerations. *Games and Economic Behavior*, 1(1):80–93, March 1989. doi:10.1016/0899-8256(89)90006-7.

20 M. Goemans, Vahab Mirrokni, and A. Vetta. Sink equilibria and convergence. In *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 142–151, 2005. doi:10.1109/SFCS.2005.68.

21 Christoph Hansknecht, Max Klimm, and Alexander Skopalik. Approximate pure Nash equilibria in weighted congestion games. In *Proceedings of APPROX/RANDOM*, pages 242–257, 2014. doi:10.4230/LIPIcs.APPROX-RANDOM.2014.242.

22 Tobias Harks and Max Klimm. On the existence of pure Nash equilibria in weighted congestion games. *Mathematics of Operations Research*, 37(3):419–436, 2012. doi:10.1287/moor.1120.0543.

23 Tobias Harks, Max Klimm, and Rolf H Möhring. Strong equilibria in games with the lexicographical improvement property. *International Journal of Game Theory*, 42(2):461–482, 2012. doi:10.1007/s00182-012-0322-1.

24 David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. How easy is local search? *Journal of Computer and System Sciences*, 37(1):79–100, 1988. doi:10.1016/0022-0000(88)90046-3.

25 Elias Koutsoupias and Christos Papadimitriou. Worst-case equilibria. *Computer Science Review*, 3(2):65–69, 2009. doi:10.1016/j.cosrev.2009.04.003.

26 Lavy Libman and Ariel Orda. Atomic resource sharing in noncooperative networks. *Telecommunication Systems*, 17(4):385–409, August 2001. doi:10.1023/A:1016770831869.

27 Noam Nisan, Tim Roughgarden, Éva Tardos, and Vijay Vazirani, editors. *Algorithmic Game Theory*. Cambridge University Press, 2007.

28 Panagiota N. Panagopoulou and Paul G. Spirakis. Algorithms for pure Nash equilibria in weighted congestion games. *Journal of Experimental Algorithmics*, 11:27, February 2007. doi:10.1145/1187436.1216584.

29 Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

30 Robert W. Rosenthal. A class of games possessing pure-strategy Nash equilibria. *International Journal of Game Theory*, 2(1):65–67, 1973. doi:10.1007/BF01737559.

31 Robert W. Rosenthal. The network equilibrium problem in integers. *Networks*, 3(1):53–59, 1973. doi:10.1002/net.3230030104.

32 Tim Roughgarden. *Twenty Lectures on Algorithmic Game Theory*. Cambridge University Press, 2016.

33 Alejandro A. Schäffer and Mihalis Yannakakis. Simple local search problems that are hard to solve. *SIAM Journal on Computing*, 20(1):56–87, 1991. doi:10.1137/0220004.

34 Alexander Skopalik and Berthold Vöcking. Inapproximability of pure Nash equilibria. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC)*, pages 355–364, 2008. doi:10.1145/1374376.1374428.

On Packing Low-Diameter Spanning Trees

Julia Chuzhoy
Toyota Technological Institute at Chicago, IL, USA
https://ttic.uchicago.edu/~cjulia/
cjulia@ttic.edu

Merav Parter
The Weizmann Institute of Science, Rehovot, Israel
http://www.weizmann.ac.il/math/parter/home
merav.parter@weizmann.ac.il

Zihan Tan
Computer Science Department, University of Chicago, IL, USA
https://sites.google.com/view/zihantan
zihantan@uchicago.edu

Abstract

Edge connectivity of a graph is one of the most fundamental graph-theoretic concepts. The celebrated *tree packing* theorem of Tutte and Nash-Williams from 1961 states that every k-edge connected graph G contains a collection $\mathcal{T}$ of $\lfloor k/2 \rfloor$ edge-disjoint *spanning* trees, that we refer to as a *tree packing*; the *diameter* of the tree packing $\mathcal{T}$ is the largest diameter of any tree in $\mathcal{T}$. A desirable property of a tree packing for leveraging the high connectivity of a graph in distributed communication networks, is that its diameter is low. Yet, despite extensive research in this area, it is still unclear how to compute a tree packing of a low-diameter graph G, whose diameter is sublinear in $|V(G)|$, or, alternatively, how to show that such a packing does not exist.

In this paper, we provide first non-trivial upper and lower bounds on the diameter of tree packing. We start by showing that, for every k-edge connected n-vertex graph G of diameter D, there is a tree packing $\mathcal{T}$ containing $\Omega(k)$ trees, of diameter $O((101k \log n)^D)$, with edge-congestion at most 2.

Karger's edge sampling technique demonstrates that, if G is a k-edge connected graph, and $G[p]$ is a subgraph of G obtained by sampling each edge of G independently with probability $p = \Theta(\log n/k)$, then with high probability $G[p]$ is connected. We extend this result to show that the diameter of $G[p]$ is bounded by $O(k^{D(D+1)/2})$ with high probability. This immediately gives a tree packing of $\Omega(k/\log n)$ edge-disjoint trees of diameter at most $O(k^{D(D+1)/2})$. We also show that these two results are nearly tight for graphs with a small diameter: we show that there are k-edge connected graphs of diameter $2D$, such that any packing of k/α trees with edge-congestion η contains at least one tree of diameter $\Omega\left((k/(2\alpha\eta D))^D\right)$, for any k, α and η. Additionally, we show that if, for every pair u, v of vertices of a given graph G, there is a collection of k edge-disjoint paths connecting u to v, of length at most D each, then we can efficiently compute a tree packing of size k, diameter $O(D \log n)$, and edge-congestion $O(\log n)$. Finally, we provide several applications of low-diameter tree packing in the distributed settings of network optimization and secure computation.

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

Keywords and phrases Spanning tree, packing, edge-connectivity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.33

Category Track A: Algorithms, Complexity and Games

Related Version https://arxiv.org/abs/2006.07486

Funding *Julia Chuzhoy*: Part of the work was done while the author was a Weston visiting professor in the Department of Computer Science and Applied Mathematics, Weizmann Institute of Science. Supported in part by NSF grant CCF-1616584.
Merav Parter: Supported in part by an ISF grant no. 2084/18.
Zihan Tan: Part of the work was done while the author was visiting the Department of Computer Science and Applied Mathematics, Weizmann Institute of Science. Supported in part by NSF grant CCF-1616584.

1 Introduction

Edge connectivity of a graph is one of the most basic graph theoretic parameters, with various applications to network reliability and information dissemination. A key tool for leveraging high edge connectivity of a given graph is *tree packing*: a large collection of spanning trees that are (nearly) edge-disjoint. A celebrated result of Tutte [24] and Nash-Williams [19] shows that for every k-edge connected graph, there is a tree packing $\mathcal{T}$ containing $\lfloor k/2 \rfloor$ edge-disjoint trees. This beautiful theorem has numerous algorithmic applications, but unfortunately it provides no guarantee on the diameter of the individual trees in $\mathcal{T}$. In the worst case, trees in $\mathcal{T}$ may have diameter that is as large as $\Omega(|V(T)|)$, even if the diameter of the original graph is very small. Given a graph G and a collection $\mathcal{T}$ of trees in G, we say that the trees in $\mathcal{T}$ are *edge-disjoint* iff every edge of G lies in at most one tree of $\mathcal{T}$, and we say that they cause *edge-congestion* η iff every edge of G lies in at most η trees of $\mathcal{T}$. The *diameter* of a tree-packing $\mathcal{T}$ is the maximum diameter of any tree in $\mathcal{T}$.

The diameter of a graph is a central graph measure that determines the round complexity of distributed algorithms for various central graph problems, including minimum spanning tree, global minimum cut, shortest s-t path, and so on. All these problems admit a trivial lower bound of $\Omega(D)$ for the round complexity (where D is the diameter of the graph), and in fact a stronger lower bound of $\Omega(D + \sqrt{n})$, which is almost tight for general n-vertex graphs, that was shown by Das-Sarma et al. [23]. Despite attracting a significant amount of attention over the last decade (see e.g., [22, 10, 18, 2, 3, 16, 6, 1, 5, 4]), algorithms that exploit large edge connectivity of the input graph in the distributed setting are quite rare. The only examples that we are aware of are recent algorithms for minimum cut by Daga et al. [4] and by Ghaffari et al. [11].

Censor-Hillel et al. [2] presented several distributed algorithms, that, given a k-edge connected n-vertex graph of diameter D, computes a fractional tree packing of $\Omega(k/\log n)$ trees that are fractionally edge-disjoint[1] in $\widetilde{O}(D+\sqrt{n})$ rounds. These trees have been used to parallelize the flow of information, obtaining nearly optimal *throughput* for store-and-forward algorithms[2]. However, as these trees might have diameter as large as $\Omega(n)$ in the worst case, it is not clear how to use them in order to improve the *round complexity* of the problem at hand, as opposed to improving the throughput. In particular, in terms of optimizing the number of communication rounds, it may still be preferable to send the entire information over a single BFS tree rather than spreading it over *many* trees of potentially *large* diameter.

The problem of computing a low-diameter tree packing was studied later by Ghaffari [6] from the perspective of optimization. Specifically, he studied the multi-message broadcast problem, where a designated source vertex is required to send k messages to all other nodes in the network. Denoting by $\mathrm{OPT}(G)$ the minimum number of rounds required for the broadcast on an input graph G, he constructed a tree packing of size k, where both the diameter and the congestion are bounded by $\widetilde{O}(\mathrm{OPT}(G))$. While this approach provides a nearly optimal broadcast scheme, it does not provide absolute upper bounds on the diameter of the tree packing, and moreover, the congestion caused by the tree packing can be large.

[1] In the fractional setting, each tree T in the packing has a weight $w(T)$ and for each edge e, the sum of weights of all trees that contain e is at most 1.

[2] In this class of algorithms, the nodes can only forward the messages they receive (e.g., network coding is not allowed).

A recent work of Ghaffari and Kuhn [10] provides the following negative result for packing low-diameter trees into a graph: they show that for any large enough n and any $k \geq 1$, there is a k-edge-connected n-vertex graph of diameter $\Theta(\log n)$, such that, in any partitioning of the graph into spanning subgraphs, all but $O(\log n)$ of the subgraphs have diameter $\Omega(n/k)$. In light of this result, it is natural to consider the following key question:

(1) *Is it possible to compute a tree packing whose diameter is strongly* sublinear *in $|V(G)|$, provided that the diameter of the input graph G is* sublogarithmic *in $|V(G)|$?*

Our second key question aims at crystallizing the main challenge to computing low-diameter tree packing. So far, we have compared the diameter of the tree packing to the diameter of the original graph. However, as observed above, the results of [10] indicate that there may be a large gap between these two measures, even for graphs whose diameter is logarithmic in n. A more natural reference point is the following. We say that a graph G is (k, D)*-connected*, iff for every pair $u, v \in V(G)$ of distinct vertices, there are k edge-disjoint paths connecting u to v in G, such that the length of each path is bounded by D. Clearly, if there is a tree packing of edge-disjoint trees of diameter at most D into G, then G must be (k, D)-connected. The question is whether the reverse is also true, if we allow a small congestion and a small slack in the diameter of the trees. The celebrated result of Tutte and Nash-Williams shows that, if every pair of vertices in G has k edge-disjoint paths connecting them, then there are $\lfloor k/2 \rfloor$ edge-disjoint spanning trees in G. However, this result is not length-preserving, in the sense that the tree paths may be much longer than the original paths connecting pairs of vertices. Our goal is then to provide such a length-preserving transformation from collections of short edge-disjoint paths connecting pairs of nodes in G to a low-diameter tree packing.

(2) *Given a (k, D)-connected graph G, can one obtain a tree packing of $\widetilde{\Omega}(k)$ trees of diameter $\widetilde{O}(D)$ into G, with small edge-congestion?*

In this paper, we address both questions. For the first question, we show two efficient algorithms, that, given a k-edge connected n-vertex graph G of diameter at most D, construct a low-diameter tree packing. We complement this result by an almost matching lower bound. We address the second question by providing an efficient algorithm, that, given a (k, D)-connected graph G, computes a collection of k spanning trees of diameter at most $O(D \log n)$ each, that cause edge-congestion of $O(\log n)$.

Our Results

Our graph-theoretic results consider two main settings: in the first setting, the input graph is k-edge connected, and has diameter at most D; in the second setting, the input graph is (k, D)-connected. We only consider unweighted graphs, that is, all edge lengths are unit. Graphs are allowed to have parallel edges, unless we explicitly state that the graph is simple. Throughout the paper, we use the term *efficient algorithm* to refer to a sequential algorithm whose running time is polynomial in its input size.

Packing Trees into Low-Diameter Graphs. We prove the following two theorems that allow us to pack low-diameter trees into low-diameter graphs.

▶ **Theorem 1.** *There is an efficient randomized algorithm, that, given any positive integers D, n, k, and an n-vertex k-edge-connected graph G of diameter at most D, computes a collection $\mathcal{T}' = \{T'_1, \ldots, T'_{\lfloor k/2 \rfloor}\}$ of $\lfloor k/2 \rfloor$ spanning trees of G, such that each edge of G appears in at most two of the trees in $\mathcal{T}'$, and, with high probability, each tree $T'_i \in \mathcal{T}'$ has diameter $O((101k \ln n)^D)$.*

As we show later, the diameter bound of Theorem 1 is close to the best possible. Unfortunately, the trees in the packing provided by Theorem 1 may share edges. Next, we generalize the classical result of Karger [14] to obtain a packing of completely edge-disjoint trees of small diameter, in the following theorem.

▶ **Theorem 2.** *There is an efficient randomized algorithm that, given an n-vertex k-edge-connected graph G of diameter at most D, such that $k > 1000 \ln n$, computes a collection $\{T_1, \ldots, T_r\}$ of $r = \Omega(k/\ln n)$ edge-disjoint spanning trees of G, such that with probability $1 - 1/\mathsf{poly}(n)$, each resulting tree T_i has diameter $O(k^{D(D+1)/2})$.*

We note that while the diameter bound in Theorem 2 is slightly weaker than that obtained in Theorem 1, and the number of the spanning trees is somewhat lower, its advantage is that the resulting trees are guaranteed to be edge-disjoint. Moreover, the algorithm in Theorem 2 is very simple: we construct r graphs $G_1, \ldots, G_r$ with $V(G_i) = V(G)$ for all i, by sampling every edge of G into one of these graphs independently. We then compute a spanning tree T_i in each such graph G_i, and show that its diameter is suitably bounded. As such, this algorithm is easy to use in the distributed setting.

Lastly, we show that our upper bounds are close to the best possible if $k \gg D$, by proving the following lower bound.

▶ **Theorem 3.** *For all positive integers n, k, D, η, α such that $k/(4D\alpha\eta)$ is an integer and $n \geq 3k \cdot \left(\frac{k}{2D\alpha\eta}\right)^D$, there exists a k-edge connected simple graph G on n vertices of diameter at most $2D+2$, such that, for any collection $\mathcal{T} = \{T_1, \ldots, T_{k/\alpha}\}$ of k/α spanning trees of G that causes edge-congestion at most η, some tree $T_i \in \mathcal{T}$ has diameter at least $\frac{1}{4} \cdot \left(\frac{k}{2D\alpha\eta}\right)^D$.*

Note that, in particular, any collection $\mathcal{T}$ of $\Omega(k)$ trees that are either edge-disjoint, or cause a constant edge-congestion, must contain a tree of diameter $\Omega\left(\left(\frac{k}{cD}\right)^D\right)$ for some constant c. Even if we are willing to allow a polylogarithmic edge-congestion, and to settle for $\Theta(k/\mathsf{poly}\log n)$ trees, at least one of the trees must have diameter $\Omega\left(\left(\frac{k}{D\mathsf{poly}\log n}\right)^D\right)$. Moreover, we show that the lower bound from Theorem 3 continues to hold even for the weaker notion of *edge-independent* trees[3], introduced in [12].

Packing Trees into (k, D)-connected Graphs. We next consider (k, D)-connected graphs and show an algorithm that computes a tree packing, that is near-optimal in both the number of trees and in the diameter.

▶ **Theorem 4.** *There is an efficient randomized algorithm, that, given any positive integers D, k, n with $k \leq n$, and a (k, D)-connected n-vertex graph G, computes a collection $\mathcal{T} = \{T_1, \ldots, T_k\}$ of k spanning trees of G, such that, for each $1 \leq \ell \leq k$, tree T_ℓ has diameter at most $O(D \log n)$, and with probability at least $1 - 1/\mathsf{poly}(n)$, each edge of G appears in $O(\log n)$ trees of $\mathcal{T}$.*

Improved Distributed Algorithms for Highly Connected Graphs. We present several applications of low-diameter tree packing in the standard CONGEST model of distributed computation [21]. By the proof of Theorem 2 and the $O(\log n)$-approximation algorithm for edge connectivity by [10], we obtain the following result.

[3] A collection $\mathcal{T}$ of spanning trees is edge-independent, iff all trees in $\mathcal{T}$ are rooted at the same vertex v^*, and for every vertex $v \in V(G)$, if we denote by $\mathcal{P}(v)$ the collection of paths that contains, for each tree $T \in \mathcal{T}$, the unique path connecting v to v^* in T, then all paths in $\mathcal{P}(v)$ are edge-disjoint.

▶ **Theorem 5.** *There is a randomized distributed algorithm, that, given an n-vertex graph G of constant diameter $D = O(1)$ and an integer λ, with high probability solves the problem of $O(\log n)$-approximate verification of λ-edge connectivity in G in $\mathsf{poly}(\lambda \cdot \log n)$ rounds.*

This improves upon the state of the art bound of $O(\sqrt{n})$ for graphs with constant diameter $D \geq 3$, and $\lambda \leq n^c$ for some positive constant $c < 1/(2D^2)$. From now on, we restrict our attention to k-edge connected graphs with a constant diameter $D = O(1)$. We employ the modular approach for distributed optimization introduced by Ghaffari and Haeupler in [8] which is based on the notion of *low-congestion shortcuts*. Roughly speaking, these shortcuts augment vertex-disjoint connected subgraphs by adding nearly-edge disjoint subsets of "shortcut" edges (that is, edges that reduce the diameter of each subgraph). Using our tree packing construction, we provide improved shortcuts for highly connected graphs of small diameter. This immediately leads to $o(\sqrt{n})$-round algorithms for several classical graph problems. For example, we prove the following:

▶ **Theorem 6.** *There is a randomized distributed algorithm, that, given a k-edge connected weighted n-vertex graph G of diameter D, such that the nodes know an $O(\log n)$ approximation of k, computes an MST of G in $\widetilde{O}(\min\{\sqrt{n/k}+n^{D/(2D+1)}, n/k\})$ rounds with high probability.*

If the nodes do not know an $O(\log n)$-approximation of the value of k, then such an approximation can be computed in $\mathsf{poly}(k \log n)$ rounds for $D = O(1)$ using Theorem 5, w.h.p. For general graphs (of an arbitrary connectivity) with diameter $D = 3, 4$, Kitamura et al. [15] showed nearly optimal constructions of MST's (based on shortcuts) with round complexities of $\widetilde{O}(n^{1/4})$ and $\widetilde{O}(n^{1/3})$ respectively. Turning to lower bounds, we slightly modify the construction of Lotker et al. [17] to obtain a lower bound of $\Omega((n/k)^{1/3})$ rounds for computing an MST in k-edge connected graphs of diameter 4, assuming that $k = O(n^{1/4})$.

Finally, we consider the basic task of *information dissemination*, where a given source vertex s is required to send N bits of information to the designated target vertex t in a k-edge connected n-vertex graph. This problem was first addressed in [10], who showed a lower bound of $\Omega(\min\{N/\log^2 n, n/k\})$ rounds, provided that the diameter of the graph is $\Theta(\log n)$. Using our low-diameter tree packing we obtain the first improved upper bounds for sublogarithmic diameter. We also show a new lower bound for simple store-and-forward algorithms, for the regime where $D = o(\log n)$.

▶ **Theorem 7.** *There is a randomized distributed algorithm, that, given any k-edge connected n-vertex graph G of diameter D with a source vertex s and a destination vertex t, sends an input sequence of N bits from s to t. The number of rounds is bounded by $\widetilde{O}(N^{1-1/(D+1)}+N/k)$ with high probability.*

In addition, for all integers n, N, D and $k \leq n$, there exists a k-edge connected n-vertex graph $G = (V, E)$ of diameter $2D$, and a pair s, t of its vertices, such that sending N bits from s to t in a store-and-forward manner requires at least $\Omega(\min\{(N/(D \log n))^{1-1/(D+1)}, n/k\} + N/k + D)$ rounds.

Applications to Secure Distributed Computation. Recently, Parter and Yogev [20] presented a general simulation result that converts any non-secure distributed algorithm to an equivalent secure algorithm, while paying a small overhead in the number of rounds. This transformation is based on the combinatorial graph structure of low-congestion cycle cover, namely, a collection of nearly edge-disjoint short cycles that cover all edges in the graph. The security provided by [20] was limited to adversaries who can manipulate at most one edge of the graph in a given round; in fact if the graph is only 2-edge connected, no stronger security guarantees, in terms of the number of edges that an adversary is allowed to corrupt is possible. In this paper we provide technical tools for handling stronger adversaries, who collude with $f(k)$ edges in a k-edge connected graph in each given round. In order to do so, we define a stronger variant of cycle cover that is adapted to the highly connected setting. This generalization is formalized by the notion of k-connected cycle cover, in which each edge in the graph is covered by k almost-disjoint cycles. Our key contribution is an algorithm that transforms any tree packing with k trees of diameter D into a $(k-1)$-connected cycle cover with cycle length $O(D \log n)$ and congestion $\widetilde{O}(k \log n)$. This yields a simple secure simulation of distributed algorithms in the presence of an adversary who colludes with $O(k/\log n)$ edges of the graph in each round[4]. Finally, we also use low-diameter tree packing to provide a simple store-and-forward algorithm for the problem of secure broadcast.

Organization. We provide the proof of Theorem 1 in Section 2, the proof of Theorem 2 in Section 3, the proof of Theorem 3 in Section 4, and the proof of Theorem 4 in Section 5. We discuss applications of our graph theoretic results to distributed computation in Section 6. Lastly, we discuss open problems in Section 7. Due to lack of space, some of the proofs are only sketched; the full formal proofs are deferred to the full version of the paper.

2 Low-Diameter Tree Packing with Small Edge-Congestion: Proof of Theorem 1

We start by showing that, if we are given a graph G, and a collection $\{T_1, \ldots, T_k\}$ of edge-disjoint spanning trees of G, such that the diameter of the tree T_k is at most $2D$ (but other trees may have arbitrary diameters), then we can efficiently compute another collection $\{T'_1, \ldots, T'_{k-1}\}$ of edge-disjoint spanning trees of G, such that the diameter of each resulting tree T'_i is bounded by $O((101k \ln n)^D)$ with high probability.

▶ **Theorem 8.** *There is an efficient randomized algorithm, that, given any positive integers D, k, n, an n-vertex graph G, and a collection $\{T_1, \ldots, T_k\}$ of k spanning trees of G, such that the trees $T_1, \ldots, T_{k-1}$ are edge-disjoint, and the diameter of T_k is at most $2D$, computes a collection $\{T'_1, \ldots, T'_{k-1}\}$ of edge-disjoint spanning trees of G, such that, with probability at least $1 - 1/\mathsf{poly}(n)$, for each $1 \leq i \leq k-1$, the diameter of tree T'_i is bounded by $O((101k \ln n)^D)$.*

Theorem 1 easily follows by combining Theorem 8 with the results of Kaiser [13], who gave a short elementary proof of the tree-packing theorem of Tutte [24] and Nash-Williams [19]. His proof directly translates into an efficient algorithm, that, given a k-edge connected graph G, computes a collection of $\lfloor k/2 \rfloor$ edge-disjoint spanning trees of G. In order to complete the proof of Theorem 1, we use the algorithm of Kaiser [13] to compute an arbitrary collection $\mathcal{T} = \{T_1, \ldots, T_{\lfloor k/2 \rfloor}\}$ of edge-disjoint spanning trees of G, and compute another arbitrary BFS tree T^* of G. Since the diameter of G is at most D, the diameter of T^* is at most $2D$. We then apply Theorem 8 to the collection $\{T_1, \ldots, T_{\lfloor k/2 \rfloor}, T^*\}$ of spanning trees, to obtain another collection $\mathcal{T}' = \{T'_1, \ldots, T'_{\lfloor k/2 \rfloor}\}$ of spanning trees, such that each edge of G belongs to at most 2 trees of $\mathcal{T}'$, and with high probability, the diameter of each tree in $\mathcal{T}'$ is at most $O((101k \ln n)^D)$. We note that, since we allow parallel edges, the trees in the set $\{T_1, \ldots, T_{\lfloor k/2 \rfloor}, T^*\}$ are edge-disjoint in graph $G \cup E(T^*)$.

[4] We note that an adversary may choose a different set of $O(k/\log n)$ edges to listen to or to corrupt in each round.

The main technical tool that we use in order to prove of Theorem 8 is the following theorem, that allows one to "fix" a diameter of a connected graph using a low-diameter tree.

▶ **Theorem 9.** *Let H be a connected graph with $|V(H)| \leq n$, and let T be a rooted tree of depth D, such that $V(T) = V(H)$. For a real number $0 < p < 1$, let R be a random subset of the edges of T, where each edge $e \in E(T)$ is added to R independently with probability p. Then with probability at least $1 - \frac{D}{n^{48}}$, the diameter of the graph $H \cup R$ is at most $(\frac{101 \ln n}{p})^D$.*

Theorem 8 easily follows from Theorem 9: For each $1 \leq i < k$, we construct a graph G_i as follows. Start with $G_i = T_i$ for all $1 \leq i \leq k$. Compute a random partition $E_1, \ldots, E_{k-1}$ of the edges of $E(T_k)$, by adding each edge $e \in E(T_k)$ to a set E_i chosen uniformly at random from $\{E_1, \ldots, E_{k-1}\}$ independently from other edges. Using Theorem 9 with $p = 1/(k-1)$, it is immediate to see that with high probability, the diameter of each resulting graph G_i is bounded by $O((101k \ln n)^D)$. We then let T'_i be a BFS tree of graph G_i, rooted at an arbitrary vertex. In order to complete the proof of Theorem 1, it is now enough to prove Theorem 9.

Proof of Theorem 9. Recall that we are given a connected graph H with $|V(H)| \leq n$, and a rooted tree T of depth D, such that $V(T) = V(H)$, together with a parameter $0 < p < 1$. We let R be a random subset of $E(T)$, where each edge $e \in E(T)$ is added to R independently with probability p. Our goal is to show that the diameter of the graph $H \cup R$ is at most $\left(\frac{101 \ln n}{p}\right)^D$ with probability at least $1 - \frac{D}{n^{48}}$. Denote $V = V(H) = V(T)$. For each $0 \leq i \leq D$, let V_i be the set of nodes lying at level i of the tree T (that is, at distance i from the tree root), and denote $V_{\leq i} = \bigcup_{t=0}^{i} V_t$. Let $H' = H \cup R$.

We say that a node $x \in V$ is *good* if either (i) $x \in V_{\leq D-1}$; or (ii) $x \in V_D$, and there is an edge in R connecting x to a node in V_{D-1}. We assume that $V = \{v_1, \ldots, v_{n'}\}$, where the vertices are indexed in an arbitrary order. Given an ordered pair (x, x') of vertices in H, and a path P connecting x to x', let $\sigma(P)$ be a sequence of vertices that lists all the vertices appearing on P in their natural order, starting from vertex x (so in a sense, we think of P as a directed path). For an ordered pair $(x, x') \in V$ of vertices, let $P_{x,x'}$ be shortest path connecting x to x' in H, and among all such paths P, choose the one whose sequence $\sigma(P)$ is smallest lexicographically. Observe that $P_{x,x'}$ is unique, and, moreover, if some pair u, u' of vertices lie on $P_{x,x'}$, with u lying closer to x than u' on $P_{x,x'}$, then the sub-path of $P_{x,x'}$ from u to u' is precisely $P_{u,u'}$.

Let $M = \frac{50 \ln n}{p}$. For a pair x, x' of vertices of V, we let $B(x, x')$ be the bad event that length of $P_{x,x'}$ is greater than M and there is no good internal node on $P_{x,x'}$. Notice that event $B(x, x')$ may only happen if every inner vertex on $P_{x,x'}$ lies in V_D, and for each such vertex, the unique edge of T that is incident to it was not added to R. Therefore, the probability that event $B(x, x')$ happens for a fixed pair x, x' of vertices is at most $(1-p)^M = (1-p)^{(50 \ln n)/p} \leq n^{-50}$. Let B be the bad event that $B(x, x')$ happens for some pair $x, x' \in V$ of nodes. From the union bound over all pairs of nodes in V, the probability of B is bounded by n^{-48}.

Recall that H is a subgraph of H' and $\mathrm{dist}_H(\cdot, \cdot)$ is the shortest-path distance metric on H. We use the following immediate observation.

▶ **Observation 10.** *If the event B does not happen, then for every node $x \in V$, there is a good node $x' \in V$ such that* $\mathrm{dist}_H(x, x') \leq M$*.*

We prove Theorem 9 by induction on D. The base of the induction is when $D = 1$. In this case, T is a star graph. Let c denote the vertex that serves as the center of the star. For any pair $x_1, x_2 \in V$ of vertices, we denote by x'_1 the good node that is closest to x_1 in H, and we define x'_2 similarly for x_2. Notice that, from the definition of good vertices, either $x'_1 = c$, or it is connected to c by an edge of R, and the same holds for x'_2. Therefore, $\text{dist}_{H'}(x'_1, x'_2) \le 2$ must hold. If the event B does not happen, then, since H is a subgraph of H', $\text{dist}_{H'}(x_1, x_2) \le \text{dist}_{H'}(x_1, x'_1) + \text{dist}_{H'}(x'_1, x'_2) + \text{dist}_{H'}(x_2, x'_2) \le \text{dist}_{H}(x_1, x'_1) + \text{dist}_{H'}(x'_1, x'_2) + \text{dist}_{H}(x_2, x'_2) \le 2M + 2 \le \frac{101 \ln n}{p}$. Therefore, with probability at least $1 - n^{-48}$, $\text{dist}_{H'}(x_1, x_2) \le \frac{101 \ln n}{p}$.

Assume now that Theorem 9 holds for every connected graph H and every tree T of depth at most $D - 1$, with $V(T) = V(H)$. Consider now some connected graph H, and a rooted tree T of depth D, with $V(T) = V(H)$. We partition the edges of $E(T)$ into two subsets: set E_1 contains all edges incident to the vertices of V_D, and set E_2 contains all remaining edges. Let $E'_1 = E_1 \cap R$, and let $E'_2 = E_2 \cap R$. Notice that the definition of good vertices only depends on the edges of E'_1, and so the event B only depends on the random choices made in selecting the edges of E'_1, and is independent from the random choices made in selecting the edges of E'_2.

Let L be a subgraph of H', obtained by starting with $L = H$, and then adding all edges of E'_1 to the graph. Finally, we define a new graph $\hat{H}$, whose vertex set is $V_{\le D-1}$, and there is an edge between a pair of nodes w, w' in $\hat{H}$ iff the distance between w and w' in L is at most $M + 2$. We also let $\hat{T}$ be the tree obtained from T, by discarding from it all vertices of V_D and all edges incident to vertices of V_D. Observe that $V(\hat{H}) = V(\hat{T}) = V_{\le D-1}$. The idea is to use the induction hypothesis on the graph $\hat{H}$, together with the tree $\hat{T}$. In order to do so, we need to prove that $\hat{H}$ is a connected graph, which we do next.

▶ **Observation 11.** *If the event B does not happen, then graph $\hat{H}$ is connected.*

Proof. Assume that the event B does not happen, and assume for contradiction that graph $\hat{H}$ is not connected. Let $\mathcal{C} = \{C_1, \dots, C_r\}$ be the set of all connected components of graph $\hat{H}$. For every pair C_i, C_j of distinct components of $\mathcal{C}$, consider the set $\mathcal{P}_{i,j} = \{P_{x,x'} \mid x \in V(C_i), x' \in V(C_j)\}$ of paths (recall that $P_{x,x'}$ is the shortest path connecting x to x' in H with $\sigma(P_{x,x'})$ lexicographically smallest among all such paths). We let $P_{i,j}$ be a shortest path in $\mathcal{P}_{i,j}$. Choose two distinct components $C_i, C_j \in \mathcal{C}$, whose path $P_{i,j}$ has the shortest length, breaking ties arbitrarily. Assume that $P_{i,j}$ connects a vertex $v \in C_i$ to a vertex $u \in C_j$, so $P_{i,j} = P_{v,u}$. Recall that $H \subseteq L$, and so the path $P_{i,j}$ is contained in graph L. Since we did not add edge (u, v) to $\hat{H}$, the length of $P_{i,j}$ is greater than $M + 2$. Since we have assumed that event B does not happen, there is at least one good inner vertex on path $P_{i,j}$. Let X be the set of all good vertices that serve as inner vertices of $P_{i,j}$.

We first show that for each $x \in X$, $x \notin V(\hat{H})$ must hold. Indeed, assume for contradiction that $x \in V(\hat{H})$, so x belongs to some connected component of $V(\hat{H})$. Assume first that $x \in V(C_i)$. Recall that the sub-path of $P_{i,j}$ from x to u is precisely $P_{x,u}$, so this path lies in $\mathcal{P}_{i,j}$. But its length is less than the length of $P_{i,j}$, contradicting the choice of $P_{i,j}$. Otherwise, x belongs to some connected component C_ℓ of $\mathcal{C}$ with $\ell \ne i$. The sub-path of $P_{i,j}$ from v to x is precisely $P_{v,x}$, so this path must lie in $\mathcal{P}_{i,\ell}$. Since its length is less than the length of $P_{i,j}$, this contradicts the choice of the components C_i, C_j. We conclude that $x \notin V(\hat{H})$.

Since $V(\hat{H})$ contains all vertices of $V_{\le D-1}$, and every vertex in X is a good vertex, it must be the case that $X \subseteq V_D$. Consider again some vertex $x \in X$. Since x is a good vertex and $x \in V_D$, there must be an edge $e_x = (x, x') \in E'_1$, connecting x to some vertex $x' \in V_{\le D-1}$. In particular, x' must belong to some connected component of $\mathcal{C}$, and the edge e_x lies in graph L. Assume that $X = \{x_1, x_2, \ldots, x_q\}$, where the vertices are indexed in the order of their appearance on $P_{i,j}$, from v to u. Consider the sequence $\tilde{\sigma} = (v, x'_1, x'_2, \ldots, x'_q, u)$ of vertices. All these vertices belong to $V(\hat{H})$, and $v \in C_i$, while $u \in C_j$. For convenience, denote $v = x'_0 = x_0$ and $u = x'_{q+1} = x_{q+1}$. Then there must be an index $1 \leq a \leq q$, such that x'_a and x'_{a+1} belong to distinct connected components of $\mathcal{C}$. Note that the sub-path of $P_{i,j}$ between x_a and x_{a+1} is precisely $P_{x_a,x_{a+1}}$ – the shortest path connecting x_a to x_{a+1} in H. Since no good vertices lie between x_a and x_{a+1} on this path, and since we have assumed that event B does not happen, the length of this path is at most M. Therefore, there is a path in graph L, connecting x'_a to x'_{a+1}, whose length is at most $M + 2$. This path connects a pair of vertices that belong to different connected components of $\hat{H}$, contradicting the construction of $\hat{H}$. ◀

Consider now the tree $\hat{T}$ and the graph $\hat{H}$. Recall that $\hat{T}$ is a rooted tree of depth $D - 1$, $V(\hat{T}) = V(\hat{H})$, $|V(\hat{H})| \leq |V(H)| \leq n$, and, assuming the event B did not happen, $\hat{H}$ is a connected graph. Moreover, set E'_2 of edges is a subset of $E(\hat{T}) = E_2$, obtained by adding every edge of $E(\hat{T})$ to E'_2 with probability p, independently from other edges. Therefore, assuming that event B did not happen, we can use the induction hypothesis on the graph $\hat{H}$, the tree $\hat{T}$, and the set E'_2 of edges as R. Let B' be the bad event that the diameter of $\hat{H} \cup E'_2$ is greater than $(\frac{101 \ln n}{p})^{D-1}$. Note that the event B' only depends on the random choices made in selecting the edges of E'_2. From the induction hypothesis, the probability that B' happens is at most $\frac{D-1}{n^{48}}$.

Lastly, we show that, if neither of the events B, B' happens, then $\mathrm{diam}(H') \leq (\frac{101 \ln n}{p})^D$.

▶ **Observation 12.** *If neither of the events B, B' happens, then* $\mathrm{diam}(H') \leq (\frac{101 \ln n}{p})^D$.

Proof. Consider any pair $x_1, x_2 \in V$ of vertices. It is sufficient to show that, if events B, B' do not happen, then $\mathrm{dist}_{H'}(x_1, x_2) \leq (\frac{101 \ln n}{p})^D$.

Let x'_1 be a good node in $V(H)$ that is closest to x_1, and define x'_2 similarly for x_2. From Observation 10, $\mathrm{dist}_H(x_1, x'_1) \leq M$. If $x'_1 \in V_{\leq D-1}$, then we define $x''_1 = x'_1$, otherwise we let x''_1 be the node of V_{D-1} that is connected to x'_1 by an edge of E'_1, and we define x''_2 similarly for x_2. Therefore, $x''_1, x''_2 \in V_{\leq D-1} = V(\hat{H})$, and, assuming event B does not happen, $\mathrm{dist}_{H'}(x_1, x''_1) \leq M + 1$, and $\mathrm{dist}_{H'}(x_2, x''_2) \leq M + 1$. Since we have assumed that the bad event B' does not happen, $\mathrm{dist}_{\hat{H} \cup E'_2}(x''_1, x''_2) \leq (\frac{101 \ln n}{p})^{D-1}$. Recall that for every edge $e = (u, v) \in \hat{H} \cup E'_2$, if $e \in E'_2$ then $e \in E(H')$; otherwise, $e \in E(\hat{H})$, and there is a path in graph $H \cup E'_1$ of length at most $M + 2$ connecting u to v in H. Therefore, $\mathrm{dist}_{H'}(x''_1, x''_2) \leq (M + 2) \cdot \mathrm{dist}_{\hat{H}}(x''_1, x''_2) \leq (\frac{101 \ln n}{p})^{D-1} \cdot (M + 2)$.

Altogether, since $M = (50 \ln n)/p$,

$$\begin{aligned} \mathrm{dist}_{H'}(x_1, x_2) &\leq \mathrm{dist}_{H'}(x_1, x''_1) + \mathrm{dist}_{H'}(x''_1, x''_2) + \mathrm{dist}_{H'}(x_2, x''_2) \\ &\leq \left(\frac{101 \ln n}{p}\right)^{D-1} \cdot (M + 2) + (2M + 2) \\ &\leq \left(\frac{101 \ln n}{p}\right)^{D}. \end{aligned}$$ ◀

The probability that either B or B' happen is bounded by $\frac{D}{n^{48}}$. Therefore, with probability at least $1 - \frac{D}{n^{48}}$, neither of the events happens, and $\mathrm{diam}(H') \leq (\frac{101 \ln n}{p})^D$. ◀

3 Low-Diameter Packing of Edge-Disjoint Trees: Proof of Theorem 2

The main tool in the proof of Theorem 2 is the following theorem.

▶ **Theorem 13.** *Let k, D, n be any positive integers with $k > 1000 \ln n$, let $\frac{707 \ln n}{k} \leq p \leq 1$ be a real number, and let G be an n-vertex k-edge-connected graph of diameter D. Let G' be a sub-graph of G with $V(G') = V(G)$, where every edge $e \in E(G)$ is added to G' with probability p independently from other edges. Then, with probability at least $1 - 1/\mathsf{poly}(n)$, G' is a connected graph, and its diameter is bounded by $k^{D(D+1)/2}$.*

Karger [14] has shown that, if G is a k-connected graph, and G' is obtained by sub-sampling the edges of G with probability $\Omega(\log n/k)$, then G' is a connected graph with high probability. Theorem 13 further shows that the diameter of G' is with high probability bounded by $k^{D(D+1)/2}$, where D is the diameter of G.

Theorem 2 easily follows from Theorem 13: Let $r = \lfloor k/(707 \ln n) \rfloor$. We partition $E(G)$ into subsets $E_1, \ldots, E_r$ by choosing, for each edge $e \in E(G)$, an index i independently and uniformly at random from $\{1, 2, \ldots, r\}$ and then adding e to E_i. For each $1 \leq i \leq r$, we define a graph G_i by setting $V(G_i) = V(G)$ and $E(G_i) = E_i$. Finally, for each graph G_i, we compute an arbitrary BFS tree T_i, and return the resulting collection $\mathcal{T} = \{T_1, \ldots, T_r\}$ of trees. It is immediate to verify that the graphs $G_1, \ldots, G_r$ are edge-disjoint, and so are the trees of $\mathcal{T}$. Moreover, applying Theorem 13 to each graph G_i with $p = 1/r$, we get that with probability $1 - 1/\mathsf{poly}(n)$, $\operatorname{diam}(T_i) \leq 2 \operatorname{diam}(G_i) \leq O(k^{D(D+1)/2})$. Using the union bound over all $1 \leq i \leq r$ completes the proof of Theorem 2. It now remains to prove Theorem 13. We provide a proof sketch here; a formal proof appears in the full version of the paper.

Proof Sketch of Theorem 13: We use the well known result of Karger [14], that shows that the probability that the graph G' is not connected is at most $O(1/\mathsf{poly}(n))$. It remains to bound the diameter of G'. Throughout the proof, for a graph H, we denote by $\mathcal{D}(H, p)$ be the distribution of graphs, where the vertex set of the resulting graph is $V(H)$, and each edge of H is included in the graph with probability p independently from other edges.

Denote $G = (V, E)$, and let T be a BFS tree of G, rooted at an arbitrary node r of G. Since G has diameter at most D, the depth of T is at most D. Recall that $G' \sim \mathcal{D}(G, p)$. We define a different (but equivalent) sampling algorithm for generating a random graph G' from $\mathcal{D}(G, p)$ as follows. The algorithm consists of $D + 1$ phases. In the 0th phase, we sample all edges in $E \setminus E(T)$ independently with probability p each. For each $1 \leq i \leq D$, in the ith phase, we sample all edges that connect a vertex at distance $(D - i + 1)$ from r to a vertex at distance $(D - i)$ from r in T. Let E' be the set of all sampled edges at the end of this algorithm. We denote by $G' = (V, E')$ the final graph that we obtain. Clearly, G' is generated from the distribution $\mathcal{D}(G, p)$. We denote by T' the subgraph of T with $V(T') = V(T)$ and $E(T') = E(T) \cap E'$. Clearly, $T' \sim \mathcal{D}(T, p)$.

Consider a pair $u, u' \in V$ of distinct vertices. We say that they are *joined at phase i* for $0 \leq i \leq D$, if u and u' belong to the same connected component of the graph induced by all edges sampled in the first i phases, but they lie in different connected components of the graph induced by all edges sampled in the first $(i - 1)$ phases. Note that, if G' is connected, then every pair (u, u') of distinct vertices of V are joined at phase i for some $0 \leq i \leq D$. The following lemma allows us to bound the diameter of G'.

▶ **Lemma 14.** *For each $0 \leq i \leq D$, with probability $1 - O(1/\mathsf{poly}(n))$, for every pair x, y of vertices that are joined at phase i, x and y are at distance at most $7^i (101 \ln n/p)^{D+(D-1)+\cdots+(D-i)}$ in G'.*

Observe that, by applying the union bound over all $0 \le i \le D$, Lemma 14 implies Theorem 13, since $k \ge 707 \ln n/p$. We defer the proof of Lemma 14 to the full version of the paper, and only provide its proof sketch here. Assume for simplicity that the edges in $E \setminus E(T)$ only connect vertices that are at distance D from r in T (this also turns out to be the hardest case). The proof is by induction on i. In the base case where $i = 0$, let C be a connected component of the graph induced by all edges sampled in phase 0. Intuitively, we can view the algorithm as using a random subgraph of T to "fix" the diameter of C, like in Theorem 9. Therefore, with high probability, for every pair x, y of vertices of C, the distance from x to y in $C \cup T'$ is at most $(101 \ln n/p)^D$. Similarly, let C' be a connected component of the the subgraph of G induced by all edges sampled in phases $0, 1, \ldots, i$; we call C' a *phase-i cluster*. We view C' as consisting of a number of phase-$(i-1)$ clusters $C''_1, \ldots, C''_k$, connected to each other by edges that were sampled in the ith phase. Therefore, if $\hat{C}'$ is a graph obtained from C' by contracting each cluster $C''_1, \ldots, C''_k$ into a single vertex, then $\hat{C}'$ is a connected graph. Denote by T_i the subtree of T induced by all nodes that are at distance at most $(D - i)$ from r in T, and denote $T'_i = T' \cap T_i$. Clearly $T'_i \sim \mathcal{D}(T_i, p)$. We can again view our algorithm as using a random subgraph T'_i of T_i to "fix" the diameter of $\hat{C}'$, like in Theorem 9. Therefore, with high probability, for every pair x, y of vertices of $\hat{C}'$, the distance from x to y in $\hat{C}' \cup T'_i$ is at most $(101 \ln n/p)^{D-i}$. Note however that every vertex of $\hat{C}'$ is in fact a contracted level-$(i-1)$ cluster. Moreover, from the induction hypothesis, if C'' is a level-$(i-1)$ cluster, and x', y' is a pair of vertices in C'', then with high probability, the distance from x' to y' in $C'' \cup T'$ is at most $7^{i-1} \cdot (101 \ln n/p)^{D+(D-1)+\cdots+(D-i+1)}$. Therefore, with high probability, the distance between a pair $u, v \in V(C')$ of vertices in $C' \cup T'$ is at most $7^i \cdot (101 \ln n/p)^{D+(D-1)+\cdots+(D-i)}$.

4 Lower Bound: Proof of Theorem 3

In this section we provide the proof of Theorem 3. We start by proving the following slightly weaker theorem; we then extend it to obtain the proof of Theorem 3.

▶ **Theorem 15.** *For all positive integers k, D, η, α such that $k/(4D\alpha\eta)$ is an integer, there exists a k-edge connected graph G with $|V(G)| = O\left(\left(\frac{k}{2D\alpha\eta}\right)^D\right)$ and diameter at most $2D$, such that, for any collection $\mathcal{T} = \{T_1, \ldots, T_{k/\alpha}\}$ of k/α spanning trees of G that causes edge-congestion at most η, some tree $T_i \in \mathcal{T}$ has diameter at least $\frac{1}{4} \cdot \left(\frac{k}{2D\alpha\eta}\right)^D$.*

Notice that the main difference from Theorem 3 is that the graph G is no longer required to be simple; the number of vertices of $V(G)$ is no longer fixed to be a prescribed value; and the diameter of G is $2D$ instead of $2D + 2$.

Proof. For a pair of integers $w > 1, D \ge 1$, we let $T_{w,D}$ be a tree of depth D, such that every vertex lying at levels $0, \ldots, D-1$ of $T_{w,D}$ has exactly w children. In other words, $T_{w,D}$ is the full w-ary tree of depth D. We denote $N_{w,D} = |V(T_{w,D})| = 1 + w + w^2 + \cdots + w^D \le w^{D+1}/(w-1)$. We assume that for every inner vertex $v \in V(T_{w,D})$, we have fixed an arbitrary ordering of the children of v, denoted by $a_1(v), \ldots, a_w(v)$.

A *traversal* of a tree T is an ordering of the vertices of T. A *post-order traversal* on a tree T, $\pi(T)$, is defined as follows. If the tree consists of a single node v, then $\pi(T) = (v)$. Otherwise, let r be the root of the tree and consider the sequence $(a_1(r), \ldots, a_w(r))$ of its children. For each $1 \le i \le w$, let T_i be the sub-tree of T rooted at the vertex $a_i(r)$. We then let $\pi(T)$ be the concatenation of $\pi(T_1), \pi(T_2), \ldots, \pi(T_w)$, with the vertex r appearing at the end of the sequence; see Figure 1 for an illustration. For simplicity, we assume that $V(T_{w,D}) = \{v_1, v_2, \ldots, v_{N_{w,D}}\}$, where the vertices are indexed in the order of their appearance in $\pi(T_{w,D})$, so the traversal visits these vertices in this order.

Next, we define a graph $G_{w,D}$, as follows. The vertex set of $G_{w,D}$ is the same as the vertex set of $T_{w,D}$, namely $V(G_{w,D}) = V(T_{w,D})$. The edge set of $G_{w,D}$ consists of two subsets: $E_1 = E(T_{w,D})$, and another set E_2 of edges that contains, for each $1 \le i < N_{w,D}$, k parallel copies of the edge (v_i, v_{i+1}). We then set $E(G_{w,D}) = E_1 \cup E_2$. For convenience, we call the edges of E_1 *blue edges*, and the edges of E_2 *red edges*; see Figures 1 and 2.

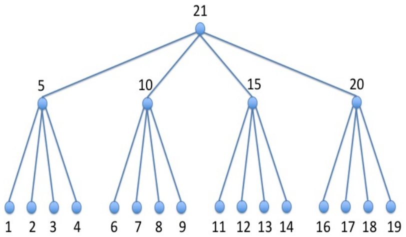


Figure 1 Tree $T_{4,2}$ with vertices indexed according to post-order traversal.

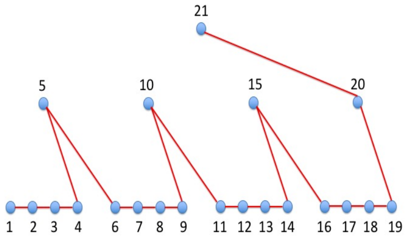


Figure 2 The edge set E_2 in $G_{4,2}$ (only a single copy of each edge is shown).

It is easy to verify that graph $G_{w,D}$ must be k-edge connected, since for any partition of $V(G_{w,D})$, there is some index $1 \le i < N_{w,D}$ with v_i, v_{i+1} separated by the partition, and so k parallel edges connecting v_i to v_{i+1} must cross the partition.

We now fix an integer $w = k/(2D\alpha\eta)$ (note that $w \ge 2$), and we let $T = T_{w,D}$ be the corresponding tree and $G = G_{w,D}$ the corresponding graph. For convenience, we denote $N_{w,D}$ by N. Recall that $N \le w^{D+1}/(w-1) = O\left(\left(\frac{k}{2D\alpha\eta}\right)^D\right)$. As observed before, G is k-edge connected. Since the depth of T is D, and $T \subseteq G$, it is easy to see that the diameter of G is at most $2D$.

We now consider any collection $\mathcal{T} = \{T_1, \ldots, T_{k/\alpha}\}$ of k/α spanning trees of G that causes edge-congestion at most η. Our goal is to show that some tree $T_i \in \mathcal{T}$ has diameter at least $\frac{1}{4} \cdot \left(\frac{k}{2D\alpha\eta}\right)^D$.

For convenience, we denote $V(G) = V(T) = V$. We say that a vertex $x \in V$ is an *ancestor* of a vertex $y \in V$ if x is an ancestor of y in the tree T, that is, $x \ne y$, and x lies on the unique path connecting y to the root of T.

Let $L \subseteq V$ be the set of vertices that serve as leaves of the tree T. We denote by $u = v_1$ a vertex of L that has the lowest index, and by u' the vertex of L with the largest index. It is easy to see that $u' = v_{N-D}$, as every vertex whose index is greater than that of u' is an ancestor of u'. For each $1 \le j \le k/\alpha$, we denote by P_j the unique path that connects u to u' in tree T_j. Let $\mathcal{P} = \{P_j \mid 1 \le j \le k/\alpha\}$. It is enough to show that at least one of the paths P_j has length at least $\frac{1}{4} \cdot \left(\frac{k}{2D\alpha\eta}\right)^D$. In order to do so, we show that $\sum_{j=1}^{k/\alpha} |E(P_j)|$ is sufficiently large. At a high level, we consider the red edges (v_i, v_{i+1}) (the edges of E_2), and show that many of the paths in $\mathcal{P}$ must contain copies of each such edge. This in turn will imply that $\sum_{P_j \in \mathcal{P}} |E(P_j)|$ is large, and that some path in $\mathcal{P}$ is long enough.

For each vertex $v_i \in L$ such that $v_i \ne u'$, we let $S_i = \{v_1, \ldots, v_i\}$, and we let $\overline{S}_i = \{v_{i+1}, \ldots, v_N\}$. Notice that, since $u \in S_i$ and $u' \in \overline{S}_i$, every path in $\mathcal{P}$ must contain an edge of $E_G(S_i, \overline{S}_i)$. Note that the only red edges in $E_G(S_i, \overline{S}_i)$ are the k parallel copies of the edge (v_i, v_{i+1}). In the next observation, we show that the number of blue edges in $E_G(S_i, \overline{S}_i)$ is bounded by Dw.

▶ **Observation 16.** *For each vertex $v_i \in L$ such that $v_i \neq u'$, for every blue edge $e \in E_G(S_i, \overline{S}_i)$, at least one endpoint of e must be an ancestor of v_i.*

Proof. We consider a natural layout of the tree T, where for every inner vertex x of the tree, its children $a_1(x), \ldots, a_w(x)$ are drawn in this left-to-right order (see Figure 3). Consider the path Q connecting the root of T to v_i, so every vertex on Q (except for v_i) is an ancestor of v_i. All vertices lying to the left of Q in the layout are visited before v_i by $\pi(T)$. All vertices lying to the right of Q, and on Q itself (excluding v_i) are visited after v_i. It is easy to see that the vertices of Q separate the two sets in T, and so the only blue edges connecting S_i to $\overline{S}_i$ are edges incident to the vertices of $V(Q) \setminus \{v_i\}$. ◀

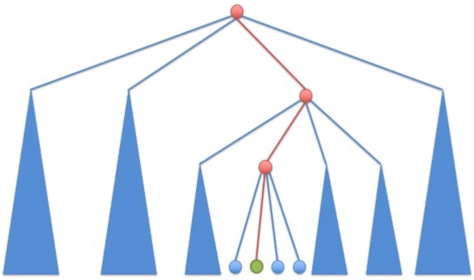


Figure 3 A layout of the tree T. Vertex v_i is shown in green and path Q in red. All vertices lying to the left of Q in this layout appear before v_i in $\pi(T)$, and all vertices lying to the right of Q or on Q (except for v_i) appear after v_i in $\pi(T)$.

Since every vertex of the tree T has at most w children, and since the depth of the tree is D, we obtain the following corollary of Observation 16.

▶ **Corollary 17.** *For each vertex $v_i \in L$ such that $v_i \neq u'$, at most Dw blue edges lie in $E_G(S_i, \overline{S}_i)$.*

Since the trees in $\mathcal{T}$ cause edge-congestion η, at most $Dw\eta$ trees of $\mathcal{T}$ may contain blue edges in $E_G(S_i, \overline{S}_i)$. Each of the remaining $\frac{k}{\alpha} - Dw\eta \geq \frac{k}{2\alpha}$ trees contains a copy of the red edge (e_i, e_{i+1}) (recall that $w = k/(2D\alpha\eta)$.) Therefore, $\sum_{P_j \in \mathcal{P}} |E(P_j)| \geq |L| \cdot \frac{k}{2\alpha} \geq \frac{Nk}{4\alpha}$, since $|L| \geq |N|/2$. We conclude that at least one path $P_j \in \mathcal{P}$ must have length at least $\frac{Nk}{4\alpha} / \frac{k}{\alpha} \geq \frac{N}{4}$, and so the diameter of T_j is at least $\frac{N}{4}$. Since $N \geq w^D \geq \left(\frac{k}{2D\alpha\eta}\right)^D$, the diameter of T_j is at least $\frac{1}{4} \cdot \left(\frac{k}{2D\alpha\eta}\right)^D$. ◀

We are now ready to complete the proof of Theorem 3. First, we show that we can turn the graph G into a simple graph, and ensure that $|V(G)| = n$, if $n \geq 3k \cdot \left(\frac{k}{2D\alpha\eta}\right)^D$. Let $G'_{w,D}$ be the graph obtained from $G_{w,D}$ as follows. For each $1 \leq i \leq N$, we replace the vertex v_i with a set $X_i = \{x_i^1, x_i^2 \ldots, x_i^k\}$ of k vertices that form a clique. For each $1 \leq i < N$, the k red edges connecting v_i to v_{i+1} are replaced by the perfect matching $\{(x_i^t, x_{i+1}^t)\}_{1 \leq t \leq k}$ between vertices of X_i and vertices of X_{i+1}. Each blue edge (v_i, v_j) is replaced by a new edge (x_i^1, x_j^1). Since $n \geq 3k \cdot \left(\frac{k}{2D\alpha\eta}\right)^D > k|V(G)| + k$, we add $n - k|V(G)| > k$ new vertices that form a clique, and for each newly-added vertex, we add an edge connecting it to x_N^1 (recall that the vertex v_N is the root of T). We denote $G' = G'_{w,D}$ for simplicity. It is not hard to see that G' has n vertices and it is k-edge connected. Moreover, G' has diameter at most $2D+2$, since its subgraph induced by vertices of $\{x^1_i\}_{1\le i\le N}$ has diameter $2D$, and every other vertex of G' is a neighbor of one of the vertices in $\{x^1_i\}_{1\le i\le N}$. The tree T' is defined exactly as before, except that every original vertex v_j is now replaced with its copy x^1_j. Let L denote the set of all leaf vertices in T'.

Assume that we are given a collection $\mathcal{T}=\{T_1,\dots,T_{k/\alpha}\}$ of k/α spanning trees of G' that causes edge-congestion at most η. For each $1\le i\le k/\alpha$, we denote by Q_i the unique path that connects x^1_1 to x^1_{N-D} in T_i and denote $\mathcal{Q}=\{Q_i\mid 1\le i\le k/\alpha\}$. For each every leaf vertex $x^1_j\in L$, we define a cut $(W_j,\overline{W_j})$ as follows: $W_j=\bigcup_{1\le s\le j}X_s$ and $\overline{W_j}=V(G')\setminus W_j$. Using reasoning similar to that in Corollary 17, it is easy to see that for every leaf vertex $x^1_j\in L$, the set $E_{G'}(W_j,\overline{W_j})$ of edges contains at most Dw blue edges – the edges of the tree T'. Since the trees in $\mathcal{T}$ cause edge-congestion at most η, at most $Dw\eta$ trees of $\mathcal{T}$ may contain blue edges in $E_{G'}(W_j,\overline{W_j})$. Therefore, for each of the remaining $\frac{k}{\alpha}-Dw\eta\ge\frac{k}{2\alpha}$ trees T_i, path Q_i must contain a red edge from $\{(x^t_j,x^t_{j+1})\}_{1\le t\le k}$. Therefore, the sum of lengths of all paths of $\mathcal{Q}$ is at least $\frac{Nk}{4\alpha}$, and so at least one path $Q_i\in\mathcal{Q}$ must have length at least $\frac{N}{4}$. We conclude that some tree $T_i\in\mathcal{T}$ has diameter at least $\frac{1}{4}\cdot\left(\frac{k}{2D\alpha\eta}\right)^D$.

Lastly, we extend our results to edge-independent trees. We use the same simple graph G' and the same tree T' as before, setting the congestion parameter $\eta=2$. Assume that we are given a collection $\mathcal{T}'=\{T'_1,\dots,T'_{k/\alpha}\}$ of k/α edge-independent spanning trees of G' and let $x\in V(G')$ be their common root vertex. For each $1\le i\le k/\alpha$, we denote by Q'_i the unique path that connects vertex x^1_1 to vertex x^1_{N-D} in tree T'_i, and we denote $\mathcal{Q}'=\{Q'_i\mid 1\le i\le k/\alpha\}$. Note that, for each $1\le i\le k/\alpha$, the path Q'_i is a sub-path of the path obtained by concatenating the path Q''_i, connecting x^1_1 to x in T'_i, with the path Q'''_i, connecting x^1_{N-D} to x in T'_i. Since the trees in $\mathcal{T}'$ are edge-independent, the paths in $\{Q''_i\}_{1\le i\le k/\alpha}$ are edge-disjoint and so are the paths in $\{Q'''_i\}_{1\le i\le k/\alpha}$. Therefore, the paths of $\mathcal{Q}'$ cause edge-congestion at most 2. The remainder of the proof is the same as before and is omitted here.

5 Tree Packing for (k,D)-Connected Graphs: Proof of Theorem 4

In this section we provide a proof sketch of Theorem 4. The full proof is deferred to the full version of the paper. The main tool that we use is the following theorem, whose proof appears in the full version of the paper.

▶ **Theorem 18.** *There is an efficient algorithm, that, given a (k,D)-connected graph G and a subset $S\subseteq V(G)$ of its vertices, computes a bi-partition (S',S'') of S, and a flow f from vertices of S'' to vertices of S', such that the following hold:*

1. *every vertex of S'' sends at least $k/2$ flow units;*
2. *every flow-path has length at most $2D$;*
3. *the total amount of flow through any edge is at most 3; and*
4. *$|S'|\le\frac{|S|}{2}+1$.*

Our algorithm consists of two phases. In the first phase, we define a partition of the vertices of G into layers $L_1,\dots,L_h$, where $h=O(\log n)$. Additionally, for each $1\le i\le h$, we define a flow f_i in graph G between vertices of L_i and vertices of $L_1\cup\cdots\cup L_{i-1}$. In the second phase, we use the layers and the flows in order to construct the desired set of spanning trees.

Phase 1: Partitioning into layers. We use a parameter $h=\Theta(\log n)$, whose exact value will be set later. We now define the layers $L_h,\dots,L_1$ in this order, and the corresponding flows $f_h,\dots,f_1$. In order to define the layer L_h, we let $S=V(G)$, and we apply Theorem

18 to the graph G and the set S of its vertices, to obtain a partition (S', S'') of S, with $|S'| \leq |S|/2 + 1$, and the flow f between the vertices of S'' and the vertices of S', where every vertex of S'' sends at least $k/2$ units of flow, each flow-path has length at most $2D$, and the edge-congestion caused by f is at most 3. We then set $L_h = S''$ and $f_h = f$, and continue to the next iteration.

Assume now that we have constructed layers $L_h, \ldots, L_i$. We now show how to construct layer L_{i-1}. Let $S = V(G) \setminus (L_h \cup \cdots \cup L_i)$. We apply Theorem 18 to the graph G and the set S of its vertices, to obtain a partition (S', S'') of S, with $|S'| \leq |S|/2 + 1$, and the corresponding flow f. We then set $L_{i-1} = S''$, $f_{i-1} = f$, and continue to the next iteration. If we reach an iteration where $|S| \leq 2$, we arbitrarily designate one of the two vertices as s and the other as s', and compute a flow of value at least k between the two vertices, such that the edge-congestion of the flow is at most 2, and every flow-path has length at most $2D$. We add vertex s' to the current layer, and we add vertex s to the final layer L_1. If we reach an iteration where $|S| = 1$, then we add the vertex of S to the final layer L_1 and terminate the algorithm. The number h of layers is chosen to be exactly the number of iterations in this algorithm. Notice that $h \leq 2 \log n$ must hold. Also observe that, for all $1 < i \leq h$, flow f_i originates at vertices of L_i, terminates at vertices of $L_1 \cup \cdots \cup L_{i-1}$, uses flow-paths of length at most $2D$, and causes edge-congestion at most 3.

Phase 2: Constructing the trees. In order to construct the spanning trees $T_1, \ldots, T_k$, we start by letting each tree contain all vertices of G and no edges. We then process every vertex $v \in V(G)$ one-by-one. Assume that $v \in L_i$, for some $1 \leq i \leq h$. Consider the following experiment. Let $\mathcal{Q}(v)$ be the set of all flow-paths that carry non-zero flow in f_i, and connect v to vertices of $L_1 \cup \cdots \cup L_{i-1}$. Let $F(v)$ be the total amount of flow that f_i sends on all paths $P \in \mathcal{Q}(v)$; recall that $F(v) \geq k/2$ must hold. We choose a path $P \in \mathcal{Q}(v)$ at random, where the probability to choose a path P is precisely $f_i(P)/F(v)$. We repeat this experiment k times, obtaining paths $P_1(v), \ldots, P_k(v)$. For each $1 \leq j \leq k$, we add all edges of $P_j(v)$ to T_j. Consider the graphs $T_1, \ldots, T_k$ at the end of this process. Notice that each such graph T_j may not be a tree. We fist show that the diameter of each such tree is $O(D \log n)$.

▷ Claim 19. For all $1 \leq j \leq k$, $\operatorname{diam}(T_j) \leq O(D \log n)$.

Proof. Fix an index $1 \leq j \leq k$. Let r be the unique vertex lying in L_1. We prove that for all $1 \leq i \leq h$, for every vertex $v \in L_i$, there is a path connecting v to r in T_j, of length at most $2D(i-1)$, by induction on i. The base of the induction is when $i = 1$ and the claim is trivially true. Assume now that the claim holds for layers $L_1, \ldots, L_{i-1}$. Let v be any vertex at layer L_i. Consider the path $P_j(v)$ that we have selected. Recall that this path has length at most $2D$, and it connect v to some vertex $u \in L_1 \cup \cdots \cup L_{i-1}$. By the induction hypothesis, there is a path P in T_j of length at most $2D(i-2)$, that connects u to r. Since all edges of $P_j(v)$ are added to T_j, the path $P_j(v)$ is contained in T_j. By concatenating path $P_j(v)$ with path P, we obtain a path connecting v to r, of length at most $2D(i-1)$. ◁

Lastly, using standard analysis of the Randomized Rounding technique, we show that, with probability at least $(1 - 1/\mathsf{poly}(n))$, every edge of G lies in at most $O(\log n)$ graphs $T_1, \ldots, T_k$. For each $1 \leq j \leq k$, we can now let T'_j be a BFS tree of the graph T_j, rooted at the vertex r. We conclude that each tree T'_j has diameter at most $O(D \log n)$, and the resulting set $\{T'_1, \ldots, T'_k\}$ of trees cause edge-congestion $O(\log n)$ with high probability.

6 Overview of the Applications to Distributed Computation

Our improved distributed algorithms in highly-connected graphs are based on the following basic tool, which follows by combining Karger's edge sampling and the diameter-fixing Theorem 9.

▷ Claim 20 (Basic Distributed Tool). There is a randomized algorithm that, given a k-edge connected n-vertex graph G and a congestion bound $\eta \in [1, k]$, computes, in $\widetilde{O}((101k \ln n/\eta)^D)$ rounds, a collection of k spanning trees that cause total edge-congestion at most $O(\eta \cdot \log n)$, and have diameter at most $O((101k \ln n/\eta)^D)$ each. Moreover, the algorithm can compute k spanning subgraphs with similar congestion and diameter bounds in $O(D + \eta \log n)$ rounds. The round complexity, the diameter, and the congestion bounds hold with high probability.

Approximation of Minimum-Cut. Ghaffari and Kuhn [10, 7] gave a very simple approach for finding an $O(\log n)$-approximation for the minimum cut problem that is based on Karger's edge sampling technique. The round complexity of their algorithm is $O(\sqrt{n})$ for constant diameter graphs. Combining Theorem 13 with Ghaffari and Kuhn's algorithm immediately leads to an $\widetilde{O}(\lambda)$ algorithm for graphs with constant diameter, where λ is the size of the minimum-cut.

To provide a more general approach for improved algorithms in highly-connected graphs, we next describe the notion of low-congestion shortcuts.

Low-Congestion Shortcuts. This notion, introduced by Ghaffari and Haeupler [9], provides a modular framework for solving global graph problems in the distributed setting.

▶ **Definition 21** (Low-Congestion Shortcuts, [9]). *Given a graph $G = (V, E)$, and a partition $S_1, \ldots, S_N$ of V into disjoint subsets, such that for all $1 \le i \le N$, graph $G[S_i]$ is connected, an (α, β)-shortcut is a collection $\{H_1, \ldots, H_N\}$ of subgraphs of G, that satisfy the following:*

(1) *for each edge $e \in E$, there are at most α subgraphs $G[S_i] \cup H_i$ containing e; and*

(2) *the diameter of each subgraph $G[S_i] \cup H_i$ is at most β.*

Ghaffari and Haeupler [9] showed that the quality of algorithms for several basic problems depend on the sum of α (i.e., congestion) and β (i.e., the dilation). The quantity of $\alpha + \beta$ is usually referred to as the *quality* of the shortcuts. As observed by [9] for every n-vertex graph G and any collection of vertex-disjoint subsets $S_1, \ldots, S_N$, there exist (α, β) shortcuts for with $\alpha + \beta = O(D + \sqrt{n})$. Our key result is in providing a nearly optimal construction for low-congestion shortcuts in highly connected graphs of constant diameter.

▶ **Theorem 22.** *[Improved Shortcuts in Highly Connected Graphs] There is a randomized algorithm that, for a sufficiently large n, given any k-connected n-vertex graph G of diameter $D = O(\log n / \log \log n)$, together with a partition $\{S_1, \ldots, S_N\}$ of $V(G)$, such that for all $1 \le i \le N$, $G[V_i]$ is a connected graph, w.h.p. computes (α, β) shortcuts, with*

$$\alpha + \beta = \widetilde{O}(\min\{\sqrt{n/k} + n^{D/(2D+1)}\}, n/k),$$

in $\widetilde{O}(\alpha + \beta)$ rounds.

The construction of the shortcuts from Theorem 22 serves the basis for the proof of Theorem 6. In the full version of the paper we describe further algorithmic applications of our results for additional graph problems. The proof of Theorem 7 is based on a careful implementation of Claim 20.

7 Open Problems

For brevity, let us say that a collection $\mathcal{T}$ of spanning trees of a (k, D)-connected graph G is an (α, D')-packing iff $|\mathcal{T}| \geq k/\alpha$ and the diameter of every tree in $\mathcal{T}$ is at most D'. A major remaining open question is: for which values of α and D' can we guarantee the existence of an (α, D')-packing $\mathcal{T}$ of edge-disjoint spanning tree in every (k, D)-connected graph. In particular, is the following statement true: every (k, D)-connected graph G contains a collection of $\Omega(k/\mathsf{poly}\log n)$ edge-disjoint trees of diameter $O(D \cdot \mathsf{poly}\log n)$ each. The only upper bounds that we have are the ones guaranteed by Theorem 2, and we do not have any lower bounds. We also do not have any upper bounds, except for those guaranteed by Theorem 1, if we allow a constant, or more generally any sub-logarithmic congestion. Additionally, obtaining an analogue of the algorithm from Theorem 4 in the distributed setting remains a very interesting open question.

Finally, most of our results are mainly meaningful for the setting where $k = \Omega(\log n)$. It will be very interesting to consider the case of small connectivity $k = O(1)$. One can show that any k-edge connected graph with $k = O(1)$ of diameter D is a $(k, \mathsf{poly}(D))$-connected graph. Is it possible to show that any k-edge-connected graph of diameter D, for some constant $k \geq 3$, has at least two edge-disjoint trees of depth at most $\mathsf{poly}(D)$?

References

1 Keren Censor-Hillel, Mohsen Ghaffari, George Giakkoupis, Bernhard Haeupler, and Fabian Kuhn. Tight bounds on vertex connectivity under vertex sampling. In *Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms*, pages 2006–2018. Society for Industrial and Applied Mathematics, 2015.

2 Keren Censor-Hillel, Mohsen Ghaffari, and Fabian Kuhn. Distributed connectivity decomposition. In *ACM Symposium on Principles of Distributed Computing, PODC '14, Paris, France, July 15-18, 2014*, pages 156–165, 2014.

3 Keren Censor-Hillel, Mohsen Ghaffari, and Fabian Kuhn. A new perspective on vertex connectivity. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 546–561. Society for Industrial and Applied Mathematics, 2014.

4 Mohit Daga, Monika Henzinger, Danupon Nanongkai, and Thatchaphol Saranurak. Distributed edge connectivity in sublinear time. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 343–354. ACM, 2019.

5 Michal Dory. Distributed approximation of minimum k-edge-connected spanning subgraphs. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*, pages 149–158. ACM, 2018.

6 Mohsen Ghaffari. Distributed broadcast revisited: Towards universal optimality. In *International Colloquium on Automata, Languages, and Programming*, pages 638–649. Springer, 2015.

7 Mohsen Ghaffari. *Improved Distributed Algorithms for Fundamental Graph Problems*. PhD thesis, MIT, USA, 2017. URL: https://groups.csail.mit.edu/tds/papers/Ghaffari/PhDThesis-Ghaffari.pdf.

8 Mohsen Ghaffari and Bernhard Haeupler. Distributed algorithms for planar networks ii: Low-congestion shortcuts, mst, and min-cut. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 202–219. SIAM, 2016.

9 Mohsen Ghaffari and Bernhard Haeupler. Distributed algorithms for planar networks II: low-congestion shortcuts, mst, and min-cut. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 202–219, 2016.

10 Mohsen Ghaffari and Fabian Kuhn. Distributed minimum cut approximation. In *International Symposium on Distributed Computing*, pages 1–15. Springer, 2013.

11 Mohsen Ghaffari and Krzysztof Nowicki. Faster algorithms for edge connectivity via random out contractions, 2019.

12 Alon Itai and Michael Rodeh. The multi-tree approach to reliability in distributed networks. *Information and Computation*, 79(1):43–59, 1988.

13 Tomáš Kaiser. A short proof of the tree-packing theorem. *Discrete Mathematics*, 312(10):1689–1691, 2012.

14 David R Karger. Random sampling in cut, flow, and network design problems. *Mathematics of Operations Research*, 24(2):383–413, 1999.

15 Naoki Kitamura, Hirotaka Kitagawa, Yota Otachi, and Taisuke Izumi. Low-congestion shortcut and graph parameters. In *33rd International Symposium on Distributed Computing (DISC 2019)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019.

16 Fabian Kuhn. A distributed perspective on graph connectivity and cuts. In *26th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '14, Prague, Czech Republic - June 23 - 25, 2014*, page 1, 2014.

17 Zvi Lotker, Boaz Patt-Shamir, and David Peleg. Distributed MST for constant diameter graphs. *Distributed Computing*, 18(6):453–460, 2006.

18 Danupon Nanongkai and Hsin-Hao Su. Almost-tight distributed minimum cut algorithms. In *International Symposium on Distributed Computing*, pages 439–453. Springer, 2014.

19 CSJA Nash-Williams. Edge-disjoint spanning trees of finite graphs. *Journal of the London Mathematical Society*, 1(1):445–450, 1961.

20 Merav Parter and Eylon Yogev. Low congestion cycle covers and their applications. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1673–1692. SIAM, 2019.

21 David Peleg. *Distributed Computing: A Locality-sensitive Approach.* SIAM, 2000.

22 David Pritchard and Ramakrishna Thurimella. Fast computation of small cuts via cycle space sampling. *ACM Transactions on Algorithms (TALG)*, 7(4):46, 2011.

23 Atish Das Sarma, Stephan Holzer, Liah Kor, Amos Korman, Danupon Nanongkai, Gopal Pandurangan, David Peleg, and Roger Wattenhofer. Distributed verification and hardness of distributed approximation. *SIAM Journal on Computing*, 41(5):1235–1265, 2012.

24 William Thomas Tutte. On the problem of decomposing a graph into n connected factors. *Journal of the London Mathematical Society*, 1(1):221–230, 1961.

Online Two-Dimensional Load Balancing

Ilan Cohen[1]
Jether Energy Ltd, Tel Aviv, Israel
ilanrcohen@gmail.com

Sungjin Im
Department of Computer Science and Engineering, University of California Merced, CA, USA
sim3@ucmerced.edu

Debmalya Panigrahi
Department of Computer Science, Duke University, Durham, NC, USA
debmalya@cs.duke.edu

Abstract

In this paper, we consider the problem of assigning 2-dimensional vector jobs to identical machines online so to minimize the maximum load on any dimension of any machine. For arbitrary number of dimensions d, this problem is known as vector scheduling, and recent research has established the optimal competitive ratio as $O\left(\frac{\log d}{\log \log d}\right)$ (Im et al. FOCS 2015, Azar et al. SODA 2018). But, these results do not shed light on the situation for small number of dimensions, particularly for $d = 2$ which is of practical interest. In this case, a trivial analysis shows that the classic list scheduling greedy algorithm has a competitive ratio of 3. We show the following improvements over this baseline in this paper:

- We give an improved, and tight, analysis of the list scheduling algorithm establishing a competitive ratio of 8/3 for two dimensions.
- If the value of OPT is known, we improve the competitive ratio to 9/4 using a variant of the classic best fit algorithm for two dimensions.
- For any fixed number of dimensions, we design an algorithm that is provably the best possible against a fractional optimum solution. This algorithm provides a proof of concept that we can simulate the optimal algorithm online up to the integrality gap of the natural LP relaxation of the problem.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases Online algorithms, scheduling, multi-dimensional

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.34

Category Track A: Algorithms, Complexity and Games

Funding *Sungjin Im*: Supported in part by NSF grants CCF-1409130, CCF-1617653, and CCF-1844939.
Debmalya Panigrahi: Supported in part by NSF grants CCF-1535972, CCF-1955703, an NSF CAREER Award CCF-1750140, and the Indo-US Virtual Networked Joint Center on Algorithms under Uncertainty.

1 Introduction

In the online load balancing problem, the goal is to allocate n jobs appearing online on a set of m identical machines so as to minimize the maximum load on any machine (called *makespan*). This problem was introduced in the 1960s by Graham [26, 27], who gave the list scheduling algorithm that assigns each arriving job to the machine with minimum

[1] Work done while at CWI Amsterdam, The Netherlands.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 34; pp. 34:1–34:21
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

load, and achieves a competitive ratio[2] of 2.[3] Since then, there has been a long line of work that aims to improve this constant below 2, both when the optimal value OPT is unknown [10, 37, 1, 19, 18, 11, 25, 28, 2], and when OPT is known [9, 40, 4, 38, 39, 21, 20, 12]. The current record is a competitive ratio of 1.916 due to Albers [2] for unknown OPT, and 1.5 due to Bohm et al. [12] for known OPT.

Recent research has further expanded the scope of this problem to vector jobs that have multiple dimensions, the resulting problem being called *vector scheduling* [15, 7, 43, 29, 8, 30]. As earlier, the goal is to minimize the makespan of the assignment, which now represents the maximum load across all machines *and all dimensions.* This problem arises in data centers where jobs with multiple resource requirements have to be allocated to machine clusters to make efficient use of limited resources such as CPU, memory, and network bandwidth [24, 44, 41, 17, 31, 32].

It is now known that the right dependence of the competitive ratio for vector scheduling on the number of dimensions d is $\Theta(\log d/\log\log d)$ [29]. While this gives a satisfactory answer when the number of dimensions is large, in the practical context, the number of dimensions is usually small since they represent distinct computational resources. In particular, the majority of the systems scheduling literature (e.g., see [24] and follow-up papers) considers only two resources, CPU and memory, since they often tend to be the most critical bottleneck resources. Unfortunately, the existing bounds for vector scheduling do not shed any light on this case since we are interested in optimizing the constant in the competitive ratio. In this paper, we initiate the study of the *online 2-dimensional scheduling* problem, or 2DSCHED in short.

1.1 Results and Techniques

Baseline. Graham's list scheduling algorithm can be naturally extended to $d > 1$ dimensions by assigning each job to the machine that minimizes the makespan after the assignment. This algorithm has a competitive ratio of 3 for $d = 2$ (and $d + 1$ for general d). To see this, assume wlog the optimum makespan OPT $= 1$ by scaling. Since the average load on each dimension is at most OPT, it follows that there is always some machine where the sum of loads on its two dimensions is at most 2. Consequently, this machine has a load of at most 2 on each of its dimensions. Now, note that the load of any single job cannot exceed OPT on any dimension; hence, the maximum load on a machine after the greedy assignment of a job cannot exceed 3.

Unfortunately, despite the aforementioned recent progress, no existing algorithms are known to have a competitive ratio better than 3. Thus, our goal is to break the 3-competitive ratio barrier for the problem. This requires a new approach since the existing analytical methods are based on potential functions, concentrations, or volume bounds, and they all seem to inevitably lose a considerable constant factor in the competitive ratio.

A Novel Analytical Technique: Characterizing Reachable States

Our new approach is to directly characterize the set of reachable states of the algorithm. To illustrate our approach, let us take a close look at the above analysis of list scheduling. For the analysis to be tight, a configuration (machine loads) must be created where half

[2] The *competitive ratio* of an online minimization problem is the maximum ratio between the objective of the algorithm and that of an (offline) optimal solution, see e.g., [13].

[3] The competitive ratio is actually $2 - 1/m$ for m machines, but we will ignore $o(1)$ terms throughout since we consider instances of arbitrary problem size in this paper.

the machines have a load of (roughly, ignoring lower order terms) $(2, 0)$ and the remaining half have a load of $(0, 2)$. If a job of load $(1, 1)$ now arrives, then the maximum load will increase to 3 no matter where the job is assigned. But, do we ever create such imbalance in the configurations of the machines?

To rule out such states/configurations, we need to define the set of reachable states of the algorithm. Our first contribution is to develop a general framework that allows us to characterize the set of reachable states of not only the greedy algorithm, but of a much larger class of algorithms that we call *priority-based algorithms*. Roughly speaking, these are algorithms where the newly arriving job is assigned to minimize a "disutility function" that maps the current load of the machines (i.e., the current state) and the load of the current job to a real number. For such algorithms, our main observation is that if m machines come to have load vectors $c = (c_1, c_2, ..., c_m)$ – meaning that this configuration is reachable – then any pair (c_i, c_j) is a reachable state for the same algorithm on just two machines. Furthermore, we identify a specific pair (c_i, c_j) that captures the essential characteristics of the algorithm under consideration.

Using this framework, we show that the greedy makespan minimization algorithm that we described above is 8/3-competitive – and our analysis is tight. We defer the lower bound to the full version of the paper.

▶ **Theorem 1** (Section 4). *There exists a priority-based algorithm that is* $8/3$*-competitive for the 2DSCHED problem with unknown* OPT*. Furthermore, this analysis is tight.*

If we know the value of OPT, then we obtain a better competitive ratio of 2.25 by using a different algorithm that explicitly minimizes the difference of the loads on the two dimensions without violating the preset threshold $\alpha \cdot \text{OPT}$, where $\alpha = 2.25$ is the desired competitive ratio. Again, this analysis is tight, the proof of which we defer to the full version of the paper. This "balance algorithm" can be thought of as a generalization of the popular *best fit* strategy used in bin packing (see [33, 35] for one-dimensional bin packing).

▶ **Theorem 2** (Section 5). *There exists a priority-based algorithm that is* 2.25*-competitive for the 2DSCHED problem with known* OPT*. Furthermore, this analysis is tight.*

Recall that the minimum makespan problem for $d = 1$ has been widely studied for both the known and unknown OPT scenarios, and our results obtain corresponding bounds for the 2DSCHED problem.

As further evidence of the generality of our analysis framework, we also analyze a natural extension of the popular *first fit* rule used for bin packing problems. (The reader is referred to [23, 6] for multi-dimensional first fit bin packing and [16] for a full survey on one-dimensional first fit bin packing.) In this algorithm, given a target competitive ratio, the algorithm assigns a new job to the first bin that does not violate the competitiveness guarantee. This can be implemented as stated if OPT is known, and has a tight competitive ratio of 2.5. (The proof of the next theorem is deferred to the full version of the paper.)

▶ **Theorem 3.** *The first fit algorithm is* 2.5*-competitive for the 2DSCHED problem with known* OPT*. Furthermore, this analysis is tight.*

We also show that if the first fit algorithm is suitably augmented with a framework for guessing the value of OPT and adjusting this guess over time, then it has a competitive ratio better than the naïve bound of 3 for unknown OPT. (The proof of the next theorem is also deferred to the full version of the paper.)

▶ **Theorem 4.** *The first fit algorithm is* 2.89*-competitive for the 2DSCHED problem with unknown* OPT*.*

While we only showcase the power of our framework by giving tight analyses of the algorithms described above, we believe that our framework has the potential to find more applications. This is because it reduces characterizing the reachable states for an arbitrary number of machines to those for only two machines, making the search space of the worst-case assignment much more tractable from an analytical perspective. In fact, our framework is easily extendable to arbitrary d, so that we only need to consider reachable states pretending that there are only d machines. While we currently know how to analytically characterize the reachable states only when $d = 2$, and therefore the results in this paper are only for this case, it is plausible (and an interesting direction of future work) to further extend such a characterization to higher dimensions analytically and/or numerically using the fact that the number of available machines is small. In that case, our framework would be useful in providing results for online vector scheduling in $d > 2$ dimensions, e.g., in cases of three or four dimensions that are also of practical interest.

A Near-optimal Algorithm

We now switch our attention to a different type of algorithmic result. Note that the competitive ratio of all the known algorithms for $d \geq 2$ are based on their comparison against the fractional optimum. That is, as long as the total load vector is $m \cdot \vec{1}$, and each job load vector is at most $\vec{1}$, an α-competitive algorithm produces a schedule where the load vector on each machine is at most $\alpha \cdot \vec{1}$. Note that when $d = 1$, the competitive ratio 2 is also obtained against the fractional optimum and even the best competitive ratio 1.916 against the actual optimum is not far from 2.

Our next result is to give an online algorithm whose competitive ratio nearly matches the best one can hope for against the fractional optimum for any fixed d. Here our high-level approach is as follows. We first use a variant of the algorithm in [29] to assign jobs to groups of machines, ensuring that every group receives at most $1 + \epsilon$ times its share of the load in an optimal fractional solution. Then, we assign jobs to machines within each group. We differentiate between "big" jobs and "small" jobs in this assignment. For the big jobs, we use discretization to bound the number of job types, and then use an optimal decision tree to make the actual assignments. Note that the optimal decision tree can be found *offline* for every possible job arrival pattern since the total number of big jobs in a group is small, and the one that matches the online sequence can be pressed into service in the online algorithm. To assign small jobs using the decision tree, we batch and encapsulate small jobs of similar load vectors into *bin* vectors. To enable this online, we pre-allocate some bin vectors. Thus, we can effectively reduce the problem of assigning small jobs to the scalar bin packing using pre-allocation and the decision tree.

▶ **Theorem 5** (Section 6). *For any $d \geq 1$ and $\epsilon > 0$, assuming that the value of the optimum makespan*[4] *is known a priori, there exists a deterministic online algorithm for the online vector load balancing problem whose competitive ratio is $(1 + \epsilon)c_d^*$, where c_d^* is the best competitive ratio one can hope for against the fractional optimum. Furthermore, the running time of the algorithm is polynomial in n for any fixed d, ϵ.*[5] *(For a more formal statement of this result, see Definition 23 and Theorem 24.)*

Before closing this section, we note the contrast between the first set of results based on the new analysis framework, and the last result that yields the nearly best competitive ratio against the fractional optimum. While the near optimality of the last solution is attractive,

[4] More accurately, the value of the fractional optimum makespan.

[5] More precisely, the running time is polynomial in n and $(d/\epsilon)^{d^{(d/\epsilon)^{O(d)}}}$.

the first set of algorithms are much more practical since the complexity of obtaining an optimal decision tree is likely to be prohibitive for the last algorithm. Furthermore, the last result does not give a numerical performance guarantee, unlike the first set of algorithms. Indeed, these two sets of results complement each other, and cumulatively provide the first insights into the 2DSched problem.

1.2 Related Work

The online load balancing problem for 1-dimensional jobs has had a long history. It was introduced by Graham in the 1960s [26, 27], who gave the list scheduling algorithm with a competitive ratio of 2. In the last three decades, there have been a series of results for improving the competitive ratio below 2 and obtaining lower bounds on the competitive ratio [10, 37, 1, 19, 18, 11, 25, 28, 2]. The best algorithm known is a 1.916-competitive algorithm due to Albers [2]. These results address the situation where the online algorithm does not know the value of OPT. Azar and Regev [9] introduced the problem of online load balancing when OPT is known, and called this problem *bin stretching*. For bin stretching, a series of results [40, 4, 38, 39, 21, 20, 12] have led to a 1.5-competitive algorithm due to Böhm et al. [12].

Recent research has expanded the scope of this problem to vector jobs, the resulting problem being called vector scheduling. Matching upper and lower bounds of $O(\log d/\log\log d)$ have been derived for d dimensions [15, 7, 43, 29]. Note that since the competitive ratio is super-constant, OPT can be assumed to be known by a standard guess and double trick. All these results are for an arbitrary number of machines and jobs. There is a large literature on variations, generalizations, and special cases of these problems, such as optimizing norms other than makespan, considering non-identical machines, focusing on a small constant number of machines, handling only jobs of small size, etc. that we omit here for brevity. The reader is referred to several excellent surveys on the topic, e.g., by Azar [5], Sgall [46, 47], Pruhs, Sgall, and Torng [45], Albers [3], etc.

The online load balancing problem is also related to the online bin packing problem, where the capacity of every machine is fixed and the goal is to minimize the number of machines used. For a single dimension, this problem has been studied since the work of Johnson in the 1970s; see, e.g., [34, 36] and surveys [22, 48]. For vector jobs, the problem was introduced by Garey et al. [23] and has been extensively studied in the last few years [7, 6, 8].

We note that some results of a flavor similar to Theorem 5 are known for other scheduling problems. Specifically, Lübbecke et al. [42] showed online algorithms of competitive ratios arbitrarily close to the optimum for the objective of minimizing total weighted completion time and its generalizations on unrelated machines. Various types of priority-based algorithms have been extensively studied for various scheduling problems. See [14] for the relevant pointers and follow-up works.

Roadmap

We present the general framework for analyzing priority based algorithms in Section 3 and use it to analyze the greedy algorithm in Section 4 for unknown OPT and the balance algorithm in Section 5 for known OPT. Finally, we present the near-optimal algorithm against the fractional optimum in Section 6.

2 Preliminaries

In this paper, we focus on the online 2-dimensional scheduling problem, or 2DSCHED in short. In this problem, a set of n jobs $\mathbf{V}$, indexed by $j \in [n]$, and represented by 2-dimensional non-negative vectors $v_j = (v_j(1), v_j(2))$ arrive in an online sequence. On arrival, a job must be assigned to one of a given set of m identical machines M, indexed by $i \in [m]$. The goal of the algorithm is to minimize the *makespan*, which is defined as the maximum load on any dimension of any machine. Formally, for any machine i, let V_i^j denote the set of vectors assigned to this machine after the arrival of the j'th vector. Then, we say that machine i's *configuration* is given by $c_i^j = \sum_{v_{j'} \in V_i^j} v_{j'}$, which is (we may omit the superscript j if it is clear from the context):

$$(c_i^j(1), c_i^j(2)) = \left(\sum_{v_{j'} \in V_i^j} v_{j'}(1), \sum_{v_{j'} \in V_i^j} v_{j'}(2) \right).$$

For any value k, we say $c_i \leq \vec{k}$ if $c_i(1) \leq k$ *and* $c_i(2) \leq k$; otherwise, we say $c_i \nleq \vec{k}$ if at least one of these inequalities are violated, i.e., if $c_i(1) > k$ *or* $c_i(2) > k$. Analogously, we define $c_i \geq \vec{k}$ if $c_i(1) \geq k$ *and* $c_i(2) \geq k$; otherwise, we say $c_i \ngeq \vec{k}$.

Let us denote the optimal offline configuration by OPT; overloading notation, let OPT also represent the makespan of this configuration. The online algorithm is said to be α-competitive if $c_i^n \leq \vec{a}$ for all $i \in [m]$, where $a = \alpha \cdot \text{OPT}$. We show two sets of results, the first when the value of OPT is known and the second when OPT is unknown to the algorithm. Note that if OPT is unknown, then its value is not used in the definition of the algorithm. Nevertheless, for the sake of the analysis, we normalize and set $\text{OPT} = 1$, which also implies that for all job vectors $v_j \in V$, we have $v_j(1), v_j(2) \in [0, 1]$, and $\sum_{j \in [n]} v_j/m \leq \vec{1}$. For convenience, we call the first coordinate the *left* coordinate, and the second coordinate the *right* coordinate. We call a job a *left vector* if its left coordinate is larger than or equal to its right coordinate, and a *right vector* otherwise.

3 Priority-based Algorithms

In this section, we give a framework to analyze a large class of algorithms that prioritize machines based on their current load. More specifically, such an algorithm computes a certain disutility for each machine only using its current load and the arriving job's load and assigns it to a machine with the least disutility. Thus, this class of algorithms are completely determined by the disutility function: $u : (c, g) \to [0, \infty]$ where $c \in [0, \infty)^2$ represents a machine's current load vector, and $g \in [0, \infty)^2$ a job's load vector. Formally, given a set $[m]$ of machines, the algorithm PRIORITY(u) assigns job j to machine $i^* := \arg\min_{i \in [m]} u(c_i^{j-1}, v_j)$ breaking ties arbitrarily but consistently. Here, as mentioned before, c_i^{j-1} denotes machine i's load just before assigning job j. After assigning job j, we update $c_{i^*}^j = c_{i^*}^{j-1} + v_j$ while keeping $c_i^j = c_i^{j-1}$ for all $i \neq i^*$. If $u(c_i^{j-1}, v_j) = \infty$ for all $i \in [m]$, then PRIORITY(u) declares failure.

3.1 Analysis Framework: Zooming in on Jobs Assigned to Two Machines

Now we present our general framework to analyze the above type of priority-based algorithms. The key to this framework is to define the set of reachable configurations.

▶ **Definition 6.** *We say that an ordered tuple $C = (c_1, c_2, \ldots, c_m)$, which we call a configuration, where $c_i \in [0, \infty)^2$, is reachable by* Priority(u) *if machines have load vectors $c_1, c_2, \ldots, c_m$ after* Priority(u) *assigns some sequence of jobs $[n]$ to machines $[m]$, such that $||v_j||_\infty \leq 1$ for all $j \in [n]$ and $\sum_{j \in [n]} v_j \leq m \cdot \vec{1}$. The set of reachable configurations is denoted by $\mathcal{R}_m(u)$.*

Unfortunately, it seems extremely challenging to characterize the reachable configurations for m machines in general. Our key observation is that the priority-based choices made by our algorithms allows us to focus on the loads of only two machines.

▶ **Observation 7.** *For any disutility function u and configuration $C = (c_1, c_2, \ldots, c_m) \in \mathcal{R}_m(u)$, and for any pair $i \neq j \in [m]$, we have $(c_i, c_j) \in \mathcal{R}_2(u)$.*

Proof. For notional convenience, say machines i and j have load vectors c_i and c_j, respectively. Consider the jobs that are assigned to machines i and j. If we assign those jobs to machines i and j pretending that no other machines exist, the two machines i and j each would get assigned exactly the same set of jobs. This is because Priority(u) prioritizes machines only based on their current respective load vector and the arriving job's load vector. Thus, we have shown $(c_i, c_j) \in \mathcal{R}_2(u)$. ◀

Now, the looming question is which pair of machines we should focus on. If α is the target competitive ratio we want to establish, we would like to focus on critical machines, i.e., where one of the coordinates has a load exceeding $\alpha - 1$ since they cannot accommodate a job of load vector $\vec{1}$. Also, we would like to focus on a pair where the "average" load between the two dimensions is not so high to draw a contradiction – more precisely, a convex combination of the load vectors of the two machines should be capped by $\vec{1}$. To denote such a pair, we will often use the notation $p = (p_f, p_s)$, where $p_f, p_s \in [0, \infty)^2$,

▶ **Definition 8.** *For an unordered pair of configuration $p = (p_f, p_s)$, we define*
- *$p \in \mathcal{L}(\alpha)$ iff $p_f + \vec{1} \nleq \vec{\alpha}$ and $p_s + \vec{1} \nleq \vec{\alpha}$, and we say p is* overloaded*; and*
- *$p \in \mathcal{F}$ iff for all $\lambda \in [0, 1]$, we have $\lambda \cdot p_f + (1 - \lambda) \cdot p_s \nleq \vec{1}$, and we say p is* overflown*.*

The next lemma shows that there will always be at least one pair of configurations that has not overflown.

▶ **Lemma 9.** *For any $C = (c_1, c_2, \ldots, c_m) \in \mathcal{R}_m(u)$ such that $c_i \neq \vec{0}$ for all $i \in [m]$, there exist $k \neq \ell \in [m]$ such that $(c_k, c_\ell) \notin \mathcal{F}$.*

Proof. Let $q = \frac{\sum_{i \in [m]} c_i}{m}$; note $q \leq \vec{1}$ since $C \in \mathcal{R}_m(u)$. Clearly, q is in the convex hull of the vectors, $c_1 \ldots, c_m$. However, the convex hull doesn't include $\vec{0}$. Since the convex hull is in two-dimensional space, this means there exists $\gamma \in (0, 1]$, such that $\gamma \cdot q$ is on the segment (c_k, c_ℓ) for some $k, \ell \in [m]$. Thus, there exists $\lambda \in [0, 1]$ such that

$$\lambda \cdot c_k + (1 - \lambda) \cdot c_\ell = \gamma \cdot q.$$

As $q \leq \vec{1}$, we have $(c_k, c_\ell) \notin \mathcal{F}$. ◀

The following observation is immediate from the definition of $\mathcal{L}(\alpha)$ and $\mathcal{F}$, it will be useful to first enlist the different cases in terms of these individual coordinate values.

▶ **Observation 10.** *For any $\alpha > 2$, if $(p_f, p_s) \in \mathcal{L}(\alpha) \setminus \mathcal{F}$, then we have:*
- *either $p_f(2), p_s(1) > 1$ and $p_f(1), p_s(2) < 1$;*
- *or, $p_f(1), p_s(2) > 1$ and $p_f(2), p_s(1) < 1$.*

If the algorithm Priority(u) reaches a state where no machine can accommodate another job of load $\vec{1}$, then using Lemma 9, we can find a pair of configurations in $\mathcal{L}(\alpha) \setminus \mathcal{F}$. Then, using the facts that the pair is overloaded yet not overflown, we can determine the sign of a certain function $V(p_f, p_s)$ defined on the configuration's load vectors; this function will be useful to draw a contradiction later. Formally, for a pair of configuration (p_f, p_s), define

$$V(p_f, p_s) := \frac{p_f(2) \cdot (p_s(1) - 1) + p_s(2) \cdot (1 - p_f(1))}{p_s(1) - p_f(1)} - 1 \tag{1}$$

▶ **Lemma 11.** *For $\alpha > 2$, if $(p_f, p_s) \in \mathcal{L}(\alpha) \setminus \mathcal{F}$, then we have $V(p_f, p_s) \leq 0$.*

Proof. Since $p_f + 1, p_s + 1 \nleq \vec{\alpha}$ and $\alpha > 2$, we can assume wlog that $p_f(2) > 1$; the other case $p_f(1) > 1$ can be handled similarly. Then, we must have

$$p_s(2) < 1 \text{ and } p_s(1) > 1 \text{ and } p_f(1) < 1,$$

since $(p_f, p_s) \in \mathcal{L}(\alpha) \setminus \mathcal{F}$; see Observation 10. Define

$$f(\lambda, k) := \lambda \cdot p_f(k) + (1 - \lambda) \cdot p_s(k).$$

Since $(p_f, p_s) \notin \mathcal{F}$, there must exist $\lambda^* \in [0, 1]$ such that $f(\lambda^*, 1), f(\lambda^*, 2) \leq 1$. Observe $f(\lambda, k)$ is monotonically decreasing in λ when $k = 1$, and monotonically increasing when $k = 2$. For

$$\tilde{\lambda} = \frac{p_s(1) - 1}{p_s(1) - p_f(1)} \in [0, 1], \text{ i.e., } 1 - \tilde{\lambda} = \frac{1 - p_f(1)}{p_s(1) - p_f(1)},$$

we have $f(\tilde{\lambda}, 1) = 1$. Since $f(\lambda, 1)$ is monotonically decreasing, $\lambda^* \geq \tilde{\lambda}$. Since $f(\lambda, 2)$ is monotonically increasing, we have

$$f(\tilde{\lambda}, 2) = \frac{p_f(2) \cdot (p_s(1) - 1) + p_s(2) \cdot (1 - p_f(1))}{p_s(1) - p_f(1)} \leq f(\lambda^*, 2) \leq 1,$$

as desired. ◀

Having established the sign of $V(p_f, p_s)$, we now observe that it is an increasing function in any of the coordinates of the two configurations p_f, p_s.

▶ **Observation 12.** *For $\alpha > 2$ and $p = (p_f, p_s) \in \mathcal{L}(\alpha) \setminus \mathcal{F}$ we have*

$$\left(\frac{\partial V(p_f, p_s)}{\partial p_f(1)}, \frac{\partial V(p_f, p_s)}{\partial p_f(2)}, \frac{\partial V(p_f, p_s)}{\partial p_s(1)}, \frac{\partial V(p_f, p_s)}{\partial p_s(2)} \right) > \vec{0}.$$

Proof. By taking partial derivatives on $p_f(1), p_f(2), p_s(1), p_s(2)$ respectively, we have

$$\begin{aligned} &\left(\frac{\partial V(p_f, p_s)}{\partial p_f(1)}, \frac{\partial V(p_f, p_s)}{\partial p_f(2)}, \frac{\partial V(p_f, p_s)}{\partial p_s(1)}, \frac{\partial V(p_f, p_s)}{\partial p_s(2)} \right) \\ &= \left(\frac{(p_f(2) - p_s(2)) \cdot (p_s(1) - 1)}{(p_f(1) - p_s(1))^2}, \frac{1 - p_s(1)}{p_f(1) - p_s(1)}, \frac{(p_f(2) - p_s(2)) \cdot (1 - p_f(1))}{(p_f(1) - p_s(1))^2}, \frac{p_f(1) - 1}{p_f(1) - p_s(1)} \right) \\ &> \vec{0}, \end{aligned}$$

where the last inequalities follow from Observation 10. ◀

▶ **Corollary 13.** *For any $\alpha > 2$, if $(p_f, p_s), (p'_f, p'_s) \in \mathcal{L}(\alpha) \setminus \mathcal{F}$, and $p_f \geq p'_f$ and $p_s \geq p'_s$, then we have $V(p_f, p_s) \geq V(p'_f, p'_s)$.*

We now have all the pieces for the refined analysis of $\textsc{Priority}(u)$. Suppose we want to show $\textsc{Priority}(u)$ is α-competitive. Towards this end, it is sufficient to show that if $C = (c_1, c_2, \ldots, c_m) \in \mathcal{R}_m(u)$, then we have $c_i + \vec{1} \leq \vec{\alpha}$ for some $i \in [m]$. For the sake of contradiction, suppose not. Then, by Lemmas 9 and 11, we have $(c_k, c_\ell) \in \mathcal{L}(\alpha) \setminus \mathcal{F}$ for some $k, \ell \in [m]$, and $V(c_k, c_\ell) \leq 0$. Further, by Observation 7, we know $(c_k, c_\ell) \in \mathcal{R}_2(u)$. This leads to the following lemma.

▶ **Lemma 14.** *If $\mathcal{R}_2(u) \cap \mathcal{L}(\alpha) \setminus \mathcal{F} = \emptyset$, then $\textsc{Priority}(u)$ is α-competitive.*

Note that this lemma allows us to analyze $\textsc{Priority}(u)$ pretending that there are only *two* machines. Now showing the condition $\mathcal{R}_2(u) \cap \mathcal{L}(\alpha) \setminus \mathcal{F} = \emptyset$ of the lemma depends on α and the disutility function u governing $\textsc{Priority}(u)$.

4 Greedy Algorithm: Unknown opt

In this section we consider the natural algorithm that assigns an arriving job to the machine yielding the minimum makespan. We recover this algorithm by setting the disutility function u to be the following:

$$\textsc{Max}(c, g) = ||c + g||_\infty \tag{2}$$

We call this algorithm $\textsc{Priority}(\textsc{Max})$. Our goal in this section is to prove the following theorem.

▶ **Theorem 15** (Upper Bound of Theorem 1). *$\textsc{Priority}(\textsc{Max})$ is $8/3$-competitive for the 2DSched problem.*

To show Theorem 15, we will set $\alpha = 8/3$ and use Lemma 14. We begin with an easy observation, which immediately follows from $||v_j||_\infty \leq 1$ for all jobs j. (The latter is a consequence of normalizing $\textsc{opt}$ in the analysis, and not an assumption on the input.)

▶ **Observation 16.** *For any $p = (p_s, p_f) \in \mathcal{R}_2(\textsc{Max})$, we have $|\, ||p_s||_\infty - ||p_f||_\infty| \leq 1$.*

It is straightforward to show $\textsc{Priority}(\textsc{Max})$ is 3-competitive using this observation. To obtain a tighter bound, we will show the following:

▶ **Lemma 17.** *For $\alpha = 8/3$, we have $(\mathcal{R}_2(\textsc{Max}) \cap \mathcal{L}(\alpha)) \setminus \mathcal{F} = \emptyset$.*

Note that Lemma 17 implies Theorem 15 by applying it to Lemma 14. So, the rest of this section is devoted to proving Lemma 17.

For a pair of configurations (p_f, p_s), define

$$H_1(p_f, p_s) := p_s(2) + p_f(1) - p_s(1) + 1$$
$$H_2(p_f, p_s) := p_s(2) + p_f(1) - p_f(2) + 1$$

▶ **Lemma 18.** *For a pair of configurations $p = (p_f, p_s)$, we have $p \notin \mathcal{R}_2(\textsc{Max})$ if*

$$\min\{H_1(p_f, p_s), H_2(p_f, p_s), H_1(p_s, p_f), H_2(p_s, p_f)\} < 0.$$

Proof. Assume for the sake of contraction that there exists $(p_f, p_s) \in \mathcal{R}_2(\textsc{Max})$ such that the minimum is non-negative. Further, assume that (p_f, p_s) is one among such configurations that is reachable by the minimum number of jobs assigned. Clearly, $(p_f, p_s) \neq ((0, 0), (0, 0))$. Since (p_f, p_s) is unordered, assume wlog that there exist c, g such that $p_f = c + g$ and $(c, p_s) \in \mathcal{R}_2(\textsc{Max})$ and $g \leq \vec{1}$ can be assigned to (a machine of load) c according to $\textsc{Priority}$ $(\textsc{Max})$, meaning $||c + g||_\infty \leq ||p_s + g||_\infty$. We consider two cases.

Case 1. $H_1(p_f, p_s) < 0$; this is symmetric to $H_2(p_s, p_f) < 0$. We have

$$H_1(c, p_s) = p_s(2) + c(1) - p_s(1) + 1 \leq p_s(2) + c(1) + g(1) - p_s(1) + 1 = H_1(p_f, p_s) < 0,$$

which is a contradiction to the minimality of (p_f, p_s).

Case 2. $H_2(p_f, p_s) < 0$; this is symmetric to $H_1(p_s, p_f) < 0$. In this case we have

$$0 > H_2(p_f, p_s) = p_s(2) + p_f(1) - p_f(2) + 1 \geq p_s(2) - p_f(2) + 1 \geq p_s(2) - p_f(2) + g(2);$$

hence we have $p_s(2) + g(2) < p_f(2)$. If $g(1) < p_f(2) - p_s(1)$, then

$$||c + g||_\infty = ||p_f||_\infty \geq p_f(2) > ||p_s + g||_\infty,$$

which is a contradiction to Priority(Max) assigning g to c. Therefore, we have $g(1) \geq p_f(2) - p_s(1)$. Then, we have

$$\begin{aligned} H_1(c, p_s) &= p_s(2) + c(1) - p_s(1) + 1 = p_s(2) + p_f(1) - g(1) - p_s(1) + 1 \\ &\leq p_s(2) + p_f(1) - p_f(2) + 1 = H_2(p_f, p_s) < 0, \end{aligned}$$

which is also a contradiction. ◀

We are now ready to prove Lemma 17.

Proof of Lemma 17. Since $(p_f, p_s) \in \mathcal{L}(\alpha) \setminus \mathcal{F}$ and $\alpha = 8/3$, we assume wlog that $p_f(2) > \alpha - 1 = 5/3$ and $p_s(1) > \alpha - 1 = 5/3$ (see Observation 10; the other case is symmetric). Then, we have

$$H_1(p_f, p_s) = p_s(2) + p_f(1) - p_s(1) + 1 \geq 0 \text{ by Lemma 18,}$$

which yields

$$p_s(2) + p_f(1) \geq 2/3.$$

Letting $p_f(1) = x$, we have

$$p_s(2) \geq 2/3 - x.$$

Note that

$$p_f \geq p'_f := (x, 5/3 + \epsilon); \quad \text{and} \quad p_s \geq p'_s := (5/3 + \epsilon, 2/3 - x)$$

for sufficiently small $\epsilon > 0$. Thus, $(p'_s, p'_f) \notin \mathcal{F}$. Also notice $(p'_s, p'_f) \in \mathcal{L}(\alpha)$. Therefore, by Corollary 13, we have

$$V(p_f, p_s) \geq V(p'_f, p'_s) \geq 0.$$

By taking the limit $\epsilon \to 0$, we have

$$V(p_f, p_s) > \lim_{\epsilon \to 0} V(p'_f, p'_s) = \frac{(3x-1)^2}{3 \cdot (5 - 3x)} \geq 0,$$

which is a contradiction to $(p_f, p_s) \in \mathcal{L}(\alpha) \setminus \mathcal{F}$ by Lemma 11. ◀

5 Balance Algorithm: Known opt

In this section, we consider another priority-based greedy algorithm, $\textsc{Priority}(\textsc{Bal})$. The rule $\textsc{Bal}$ is defined as follows:

$$\textsc{Bal}(c,g) = \begin{cases} d(c)\cdot d(g) & c+g \le \alpha\cdot\textsc{opt} \\ \infty & \text{otherwise,} \end{cases} \tag{Bal-α}$$

where $d(v) := v(2) - v(1)$ measures the signed difference between v's right load and left load.

In other words, $\textsc{Priority}(\textsc{Bal})$ tries to minimize the difference between the left and right loads over all machines, without violating a pre-defined threshold α. Note that this algorithm needs to know the value of $\textsc{opt}$, which is wlog assumed to be 1 by scaling. The $\textsc{Priority}(\textsc{Bal})$ algorithm keeps the machines in sorted order of the (signed) difference between the loads on the two coordinates (the left load minus the right load we say that the machines are maintained in left to right order where the rightmost (resp., leftmost) machine has the largest difference between the loads on the first and second coordinate (resp., second and first coordinate). Recall that a left (resp., right) vector is one whose first (resp., second) coordinate is larger. The $\textsc{Priority}(\textsc{Bal})$ algorithm assigns a left (resp., right) vector to the rightmost (resp., leftmost) machine that can accommodate it, i.e., whose load on any dimension does not exceed the desired competitive ratio α after the assignment. In order to achieve it, given a left (right) vector the algorithm would prefer to assign to the most unbalanced right (left) machine that can accommodate the vector.

▶ **Theorem 19** (Upper Bound of Theorem 2). *$\textsc{Priority}(\textsc{Bal})$, knowing $\textsc{opt}$ a priori, is 2.25-competitive for the 2DSched problem.*

To prove Theorem 19, thanks to Lemma 14, it suffices to show the following.

▶ **Lemma 20.** *For $\alpha = 2.25$, we have $(\mathcal{R}_2(\textsc{Bal}) \cap \mathcal{L}(\alpha)) \setminus \mathcal{F} = \emptyset$.*

The remainder of this section is devoted to proving Lemma 20. Instead of analysing directly $\mathcal{R}_2(\textsc{Bal})$, we introduce a slightly modified rule of $\textsc{Bal}$, which is not subject to α:

$$\textsc{Bal-No-Lim}(c,g) = d(c)\cdot d(g) \tag{3}$$

Note that $\mathcal{R}_2(\textsc{Bal-No-Lim}) \cap \{v \mid v \in [0,\alpha]^2\} \subseteq \mathcal{R}_2(\textsc{Bal})$ and the subtle difference between $\mathcal{R}_2(\textsc{Bal-No-Lim})$ and $\mathcal{R}_2(\textsc{Bal})$. The closure $\mathcal{R}_2(\textsc{Bal-No-Lim})$ attempts to assign a vector g to only mitigate the difference of the left and right loads of the two machines. If we can assign g to the machine i^* that achieves this, then this assignment would be exactly the same as $\textsc{Priority}(\textsc{Bal})$ would make. However, in the closure $\mathcal{R}_2(\textsc{Bal-No-Lim})$, if g would overflow machine i^*, it doesn't add it to the other machine even if it would be possible under $\textsc{Priority}(\textsc{Bal})$. This is summarized in the following observation.

▶ **Observation 21.** *For two pairs of configuration $p = (c_a, c_b), p' = (c_a + g, c_b)$, such that p' is reachable in $\mathcal{R}_2(\textsc{Bal})$ by assigning a job of load $g \in [0,1]^2$ into (a machine of load) c_a in the pair p, if $(c_a + g, c_b) \in \mathcal{R}_2(\textsc{Bal}) \setminus \mathcal{R}_2(\textsc{Bal-No-Lim})$ and $(c_a, c_b) \in \mathcal{R}_2(\textsc{Bal-No-Lim})$, then $c_b + g \not\le \vec{\alpha}$ and $(d(c_a) - d(c_b)) \cdot d(g) > 0$.*

For a pair of configuration (c_s, c_f), define

$$H_3(c_f, c_s) := d(c_s) - d(c_f) + 1 = c_s(2) - c_s(1) - c_f(2) + c_f(1) + 1$$

▶ **Lemma 22.** *For a pair of configuration $p = (p_s, p_f)$, if $H_3(p_f, p_s) < 0$ or $H_3(p_s, p_f) < 0$, then we have $p \notin \mathcal{R}_2(\textsc{Bal-No-Lim})$.*

Proof. Assume for the purpose of contradiction that $p \in \mathcal{R}_2(\textsc{Bal-No-Lim})$. Assume $H_3(p_f, p_s) < 0$, since the case $H_3(p_s, p_f) < 0$ is symmetric. There must exist a vector g such that $p_f = c + g$, and $(c, p_s) \in \mathcal{R}_2(\textsc{Bal-No-Lim})$, for which $\textsc{Priority}$ ($\textsc{Bal-No-Lim}$) assigned to c, hence we have

$$\textsc{Bal-No-Lim}(c) - \textsc{Bal-No-Lim}(p_s) = (d(c) - d(p_s)) \cdot d(g) \leq 0.$$

We consider two cases.

Case 1. $g(2) > g(1)$: Clearly, $g(2) - g(1) \leq 1$. However, $d(p_s) - d(p_f) + 1 = H_3(p_f, p_s) < 0$, hence

$$d(p_s) < d(p_f) - 1 = d(c + g) - 1 = c(2) + g(2) - c(1) - g(1) - 1 \leq c(2) - c(1) = d(c).$$

Therefore, $(d(c) - d(p_s)) \cdot (g(2) - g(1)) > 0$, which is a contradiction.

Case 2. $g(2) \leq g(1)$: We have

$$c(2) - c(1) - 1 \geq c(2) + g(2) - c(1) - g(1) - 1 = p_f(2) - p_f(1) - 1 > p_s(2) - p_s(1).$$

Therefore, we have $H_3(c, p_s) < 0$ hence $(c, p_s) \notin \mathcal{R}_2(\textsc{Bal-No-Lim})$, which is a contradiction. ◀

We now have all the pieces to prove Lemma 20.

Proof of Lemma 20. Assume for the purpose of contradiction that there exists a pair $p = (c_r, c_\ell) \in \mathcal{R}_2(\textsc{Bal}) \cap \mathcal{L}(\alpha)) \setminus \mathcal{F}$. Let $\langle p^\emptyset, p^1, \dots p^n = p\rangle$ a sequence of reachable pairs i.e. for all i, $p^i \in \mathcal{R}_2(\textsc{Bal})$ and p^{i+1} is reachable from p^i by a single vector assignment under $\textsc{Priority}(\textsc{Bal})$.

Case 1. For all $i \in [n]$, $p^i \in \mathcal{R}_2(\textsc{Bal-No-Lim})$. Since $p = (c_r, c_\ell) \in \mathcal{R}_2(\textsc{Bal-No-Lim}) \cap \mathcal{L}(\alpha)$, assume wlog that $c_r(2) > \alpha - 1$, $c_\ell(1) > \alpha - 1$. In addition, since $(c_r, c_\ell) \in \mathcal{R}_2(\textsc{Bal-No-Lim})$, by Lemma 22, we have $H_3(c_r, c_\ell) = c_\ell(2) - c_\ell(1) - c_r(2) + c_r - 1 \geq 0$. Therefore, we have

$$0 \leq H_3(c_r, c_\ell) \leq H_3((c_r(1), \alpha - 1), (\alpha - 1, c_\ell(1)) = 3 - 2 \cdot \alpha + c_r(1) + c_\ell(2).$$

By setting $c_r(1) = x$, we have $c_\ell(2) \geq 2 \cdot \alpha - 3 - x$. Note $x \in [0, 1]$ since $c_\ell(1) > 1$ and $(c_r, c_\ell) \notin \mathcal{F}$. For $\alpha = 2.25$, we have

$$V(c_r, c_\ell) > V((x, \alpha - 1), (\alpha - 1, 2 \cdot \alpha - 3 - x)) = \frac{(4x - 3)^2}{20 - 16x} \geq 0,$$

which is a contradiction to $p \in \mathcal{L}(\alpha) \setminus \mathcal{F}$ by Lemma 11.

Case 2. $p^i \notin \mathcal{R}_2(\textsc{Bal-No-Lim})$ for some $i \in [n]$. Let i be the first index such that $p^i \notin \mathcal{R}_2(\textsc{Bal-No-Lim})$, let $p^{i-1} = (c_a, c_b)$, $p^i = (c_a + g, c_b)$. Note that $c_r \geq c_a$ and $c_\ell \geq c_b$. By Observation 21 we have $c_b + g \nleq \vec{\alpha}$. Assuming wlog that $c_b(1) + g(1) > \alpha$, we have $c_b(1) > \alpha - g(1) > 1$ since $g \leq \vec{1}$ and $\alpha > 2$. For the same reason, we have $c_r(2) > \alpha - 1$, and $c_r(1), c_\ell(2) \leq 1$ (since $(c_r, c_\ell) \in \mathcal{L}(\alpha) \setminus \mathcal{F}$).

Since $V(c_r, c_\ell)$ is monotone increasing, we have $V(c_r, c_\ell) > V(\langle c_a(1) + g, \alpha - 1\rangle, c_b)$. Moreover, by our assumption $p^{i-1} = (c_a, c_b) \in \mathcal{R}_2(\textsc{Bal-No-Lim})$, by Lemma 22, we have

$$\begin{array}{rcccl} H_3(c_a, c_b) & = & c_b(2) - c_b(1) - c_a(2) + c_a(1) + 1 & \geq & 0, \\ c_a(1) & \geq & \max\{c_a(2) - c_b(2) + c_b(1) - 1, 0\} & \geq & \max\{c_b(1) - c_b(2) - 1, 0\}. \end{array}$$

Recall that $c_r(1) \leq 1$, and we have $c_a(1) \geq c_b(1) - c_b(2) - 1$, and $g(1) > \alpha - c_b(1)$. Therefore, $1 \geq c_r(1) \geq g(1) + c_a(1) \geq \alpha - c_b(1) + c_a(1) \geq \alpha - c_b(2) - 1$, which yields $c_b(2) \geq \alpha - 2$. Thus,

$$V(c_r, c_\ell) > V(\langle \alpha - c_b(1) + \max\{c_b(1) - c_b(2) - 1, 0\}, \alpha - 1\rangle, \langle c_b(1), c_b(2)\rangle).$$

We lower bound V by considering two cases:

Case A. If $c_b(1) - c_b(2) < 1$, then

$$V(c_r, c_\ell) > V(\langle \alpha - c_b(1), \alpha - 1\rangle, \langle c_b(1), c_b(2)\rangle) \geq V(\langle \alpha - c_b(1), \alpha - 1\rangle, \langle c_b(1), c_b(1) - 1\rangle).$$

Case B. If $c_b(1) - c_b(2) \geq 1$, then

$$V(c_r, c_\ell) > V(\langle \alpha - c_b(2) - 1, \alpha - 1\rangle, \langle c_b(1), c_b(2)\rangle) \geq V(\langle \alpha - c_b(2), \alpha - 1\rangle, \langle 1 + c_b(2), c_b(2)\rangle).$$

Setting $x = c_b(1) - 1 (\geq \alpha - 2)$ in the first case and $x = c_b(2)$ in the second case, we get that $x \geq \alpha - 2$ in both cases. So, we have

$$V(c_r, c_\ell) > V(\langle \alpha - 1 - x, \alpha - 1\rangle, \langle 1 + x, x\rangle) \geq 0.$$

By setting $\alpha = 2.25$, for $x \geq \alpha - 2 = 0.25$, we have

$$V(c_r, c_\ell) > V(\langle \alpha - 1 - x, \alpha - 1\rangle, \langle 1 + x, x\rangle) = \frac{(2x-1)^2}{8x-1} \geq 0,$$

which is a contradiction to $p \in \mathcal{L}(\alpha) \setminus \mathcal{F}$ by Lemma 11. ◀

6 A Nearly Optimal Algorithm Against the Fractional Optimal Solution

Recall that all algorithms we developed and analyzed were based on the two most obvious lower bounds for the optimal solution, the total load vector of all jobs and the maximum job size on any dimension. Therefore, the benchmark we used can do better than the offline optimum solution. For example, consider three job vectors $(1, 1, 0), (1, 0, 1), (0, 1, 1)$ to be scheduled on 2 machines. Since one of the two machines must receive at least two jobs, the optimum makespan cannot be smaller than 2. However, this instance still has an average load of 1 on all dimensions and no job has size greater than 1 on any dimensions. In other words, the benchmark can distribute all jobs equally across all machines. For this reason, we will call this benchmark the fractional optimum solution.

▶ **Definition 23.** *For any number of dimensions $d \geq 1$, the optimum competitive ratio c_d^* against the fractional optimum solution is defined as*

$$\inf_A \sup_J \frac{\max_{i \in [m], k \in [d]} \Lambda_i^{J,m}(k)}{\max\{\|\sum_{j \in J} v_j / m\|_\infty, \max_{j \in J, k \in [d]} v_j(k)\}},$$

where A denotes an arbitrary deterministic online algorithm, J an arbitrary sequence of jobs, m the number of machines, and $\Lambda_i^{J,m}$ the load vector of machine i under the assignment of jobs J to machines $[m]$ by the algorithm A.

Our goal is to develop and analyze an algorithm that performs nearly as well as the fractional optimum solution.

▶ **Theorem 24.** *For any $d \geq 1$ and $\epsilon > 0$, assuming that the value of the optimum makespan*[6] *is known a priori, there exists a deterministic online algorithm for the vector scheduling problem whose competitive ratio is $(1+\epsilon)c^*_d$. Further, the running time is polynomial in n for any fixed d, ϵ.*[7]

6.1 Assigning Jobs to Groups

The first stage of the algorithm is executed only when $m \geq (1 + \frac{1}{\epsilon})\alpha$ where $\alpha := \frac{250}{\epsilon^3} \log d$. We group machines so that every group has exactly α machines; to simplify the notation we assume that α is an integer to omit ceilings. The only one possible group that has less than α machines is discarded. We assign jobs to the (remaining) groups and obtain the following lemma using an algorithm and analysis very similar to [29]; hence, we defer the details to the full version of the paper.

▶ **Lemma 25.** *For a sufficiently small $\epsilon > 0$, there exists an online algorithm that assigns jobs to the groups, each consisting of $\alpha := \frac{250}{\epsilon^3} \log d$ machines, such that every group's total load is at most $(1+\epsilon)\alpha d\vec{1}$.*

Note that the average load vector a machine should handle increases by a factor of at most $\frac{m}{m-(\alpha-1)} \leq \frac{1+1/\epsilon}{1+1/\epsilon-1} = 1+\epsilon$. We also slightly modify each job's load vector: For each job j, we minimally increase v_j, so that we have $v_j(k) \geq (\epsilon/d)||v_j||_\infty$. This is wlog since increasing job load vectors can only increase the algorithm's makespan and we fixed the optimum to be 1.

▶ **Lemma 26.** *The preprocessing step increases the total load vector to at most $(1+\epsilon)m\vec{1}$.*

Proof. For the sake of contradiction, suppose the total load is more than $(1+\epsilon)m$ on some dimension. It means the load increased by more than ϵm on the dimension. We know that job j contributes to the increase by at most $(\epsilon/d)||v_j||_\infty$. Thus, the increase is at most $\sum_j (\epsilon/d)||v_j||_\infty$. However, we know $\sum_j ||v_j||_\infty \leq md$ since the total load of all jobs across all dimensions is md. Therefore, the increase is at most ϵm, which is a contradiction. ◀

Since we are only concerned with assigning jobs to groups of machines at this stage, to simplify notation we pretend each group is a machine. By a machine i, we mean the i-th group which consists of α machines.

We now restate the problem: We are given $m' = \lfloor \frac{m}{\alpha} \rfloor$ machines. Let $[n]$ denote the set of all jobs arriving, which satisfies the following properties.

- Property (i): The total job load vector, i.e., $\sum_{j\in[n]} v_j$ is at most $m'\alpha(1+\epsilon)^2\vec{1}$.
- Property (ii): For all $j \in [n]$, $||v_j||_\infty \leq 1$.
- Property (iii): For all $j \in [n]$, $\min_k v_j(k) \geq (\epsilon/d)||v_j||_\infty$.

Our goal is to assign jobs to m' machines so that each group receives jobs of total load at most $(1+7\epsilon)\alpha\vec{1}$, which would immediately imply Lemma 25 by scaling ϵ.

The algorithm has two procedures. The algorithm pretends there are two sets M_1 and M_2 of machines, where $|M_1| = |M_2| = m'$. The first procedure assigns all jobs to machines M_1 and identifies a set J^2 of jobs, which will be assigned to machines M_2 by the second procedure. However, this is a shadow process: What really happens is that only jobs in

[6] More accurately, the value of the fractional optimum makespan.

[7] More precisely, the running time is polynomial in n and $(d/\epsilon)^{d^{(d/\epsilon)^{O(d)}}}$.

$[n] \setminus J^2$ remain on machines M_1 and the other jobs J^2 are assigned to machines M_2. Further, the algorithm pairs machines between M_1 and M_2 arbitrarily and combine the load vectors of the paired machines. To prove Lemma 25, it suffices to show the following two lemmas (with scaling ϵ):

▶ **Lemma 27.** *The makespan of the assignment of $[n] \setminus J^2$ to M_1 is at most $((1+\epsilon)^5\alpha+1)\vec{1} \leq (1+6\epsilon)\alpha\vec{1}$.*

▶ **Lemma 28.** *The makespan of the assignment of J^2 to M_2 is at most $\epsilon\alpha\vec{1}$.*

We are now ready to describe the algorithm.

- **First procedure (assignment by a potential function):** Let $\beta := (1+\epsilon)^3$. Let $f(x) := \beta^x$. Each job j is assigned to a machine $i \in M_1$ such that $\Phi(j)$ is minimized. For every $i \in M_1$, let $\Lambda^1_{i,j}$ denote machine i's load vector right after assigning job j to some machine in M_1. If $\Lambda^1_{i,j}(k) \geq \beta\alpha + 1$, then j is added to queue J^2 so that it can be scheduled by the second procedure.

$$\Phi_{i,k}(j) := f\left(\Lambda^1_{i,j}(k) - \frac{\beta}{m'} \sum_{j' \in [j]} v_{j'}(k)\right) \qquad \forall i \in M_1, j \in [n], k \in [d]$$

$$\Phi(j) := \sum_{i \in M_1} \sum_{k=1}^{d} \Phi_{i,k}(j)$$

- **Second procedure (assignment by greedy):** This procedure is only concerned with the jobs J^2 that are passed from the first procedure. It allocates each job in J^2 (in the order that the jobs arrive in) to one of the machines in M_2 such that the resulting makespan, $\max_{i \in M_2, k \in [d]} \Lambda^2_{i,j}(k)$ is minimized; here $\Lambda^2_{i,j}$ is analogously defined as $\Lambda^1_{i,j}$ is defined in the first procedure.

Note that Lemma 27 immediately follows due to the way the first procedure is defined. The proof of Lemma 28 constitutes the heart of the analysis in this stage of the algorithm. Since this closely follows techniques in [29], we defer the details of this analysis to the full version of the paper.

6.2 Assigning Jobs to Machines Within Each Group

We need to define a fair amount of notation to formally describe our algorithm. We assume that the input consists of m machines and the average load of all jobs to be assigned is at most $(1+\epsilon)m$ on all dimensions for some $\epsilon > 0$.

For ease of reference, we list the following definitions.

- Let $\beta := \frac{\epsilon}{2md}$; $\Delta := \frac{\epsilon^2}{d(1+1/\beta)^d}$; and $\delta := \epsilon\beta\Delta/(2dm)$.
- A vector is said to be a type vector if it is in $\{0, \beta, 2\beta, \cdots, 1\}^d \setminus \{\vec{0}\}$ and has 1 on at least one dimension. Let $\mathcal{Q}$ denote the set of all type vectors. Note that $|\mathcal{Q}| \leq (1+1/\beta)^d \leq (2/\beta)^d$.
- We say a job j is small if $||v_j||_\infty < \Delta$; otherwise it is big.
- The volume of a job j is defined as its total size over all dimensions.

We discretize big jobs and small jobs in different manners:

- Big jobs: For a big job j, we round its load on every dimension down to the nearest integer multiple of δ. Let $\mathcal{B}$ denote the set of all possible load vectors of big jobs after discretization.

- Small jobs: For a small job j, let $p_j = ||v_j||_\infty$. Then, we discretize v_j/p_j by rounding each entry down to the nearest integer multiple of β. Let q_j denote the resulting discretized vector of v_j/p_j; note that $q_j \in \mathcal{Q}$. After rounding, we can express each small job j as $p_j q_j$.

We are now ready to describe our algorithm. Below, assume that jobs are already discretized. We will later show that the effect of discretization is negligible on the competitive ratio.

6.2.1 Building a decision tree

We build a decision tree T to assign big jobs. To simplify the analysis later, we assume wlog that the total load vector T receives is exactly $m(1+4\epsilon)\vec{1}$. Each node of the decision tree T corresponds to the current loads of all the m machines. Each node u has at most $m|\mathcal{B}|$ children. Each edge (u, w) is associated with a pair (i, j) where $j \in \mathcal{B}$ and $i \in [m]$, meaning that if a big job j is assigned to machine i, then the machine loads vectors change from u to w. Since the total volume of jobs is at most $(1 + 4\epsilon)md \leq 5md$ and every big job has volume at least Δ, the decision tree has depth at most $5md/\Delta$ and the number of nodes is at most $(m|\mathcal{B}|)^{5md/\Delta}$. We only need to keep nodes whose machine load vectors don't contradict the assumption that the total load of all jobs on each dimension is exactly $(1 + 4\epsilon)m$.

Given that every node of the decision tree T corresponds to a configuration (machine load vectors) that can be reached via a valid sequence of big jobs along with their assignment, our goal is to compute the minimum makespan we can achieve from each node u. Formally, if u is a leaf node, define $g(u)$ to be the makespan norm of the machine load vectors corresponding to u. Otherwise, let $u_{i,j}$ denote u's child such that the edge $(u, u_{i,j})$ is associated with (i, j). Then, define $g(u) := \min_i \max_j g(u_{i,j})$.

We can use the tree T to assign big jobs as follows. Let u be the node corresponding to the current machine load vectors. If a big job j arrives, then we assign j to machine i with the minimum $g(u_{i,j})$.

The following observation is immediate due to the optimal nature of the decision tree for big jobs. In other words, the observation says that the decision tree yields a nearly optimal algorithm against the fractional optimum.

▶ **Observation 29.** *Let r denote the root of T. Then, $g(r) \leq (1 + 4\epsilon)c^*_d$.*

Proof. Since we assumed that the total load vector is exactly $(1+4\epsilon)m\vec{1}$, the denominator in Definition 23 is exactly $(1 + 4\epsilon)$. Since we know the decision tree gives an optimal algorithm for big jobs, we have $g(r)/(1 + 4\epsilon) \leq c^*_d$, as desired. ◀

6.2.2 Batching smalls jobs of the same type

We will first describe how we batch small jobs and assign them using the above decision tree T assuming that we can wait until we collect enough volume of jobs for each type. For each type vector $q \in \mathcal{Q}$, we create a buffer $F(q)$. The buffer has capacity Δ/ϵ. When a small job j of vector $p_j q_j$ arrives, we add it to buffer $F(q_j)$; j uses p_j space of the buffer. There are two events that trigger emptying a buffer. When we empty a buffer $F(q)$, we encapsulate the load vectors of all jobs in $F(q)$ into a "bin" vector $(\Delta/\epsilon)q$, and assign it using the decision tree T. The buffer is emptied when either we cannot add a job j since it would exceed the capacity Δ/ϵ or after all jobs arrive.

This procedure is well defined due to the following lemma.

▶ **Lemma 30.** *Every bin vector is in $\mathcal{B}$.*

Proof. Consider any bin vector $(\Delta/\epsilon)q$. To show that this is in $\mathcal{B}$, we need to show the following three: (i) it has size at least Δ on some dimensions; (ii) each of its entries is a multiple of δ; and (iii) it has size no more than 1 on every dimension. First, (i) follows since $||q||_\infty = 1$ due to the way we defined small job types. To see (ii), consider q's entry on each dimension – we know that its value must be $\ell\beta$ for some integer ℓ. So, it suffices to show that $(\Delta/\epsilon)(\ell\beta)/\delta$ is an integer. Recall that $\delta := \epsilon\beta\Delta/(2dm)$. Thus, we have, $(\Delta/\epsilon)(\ell\beta)/\delta = \frac{\Delta}{\epsilon}(\ell\beta)\frac{2dm}{\epsilon\beta\Delta}$, which is an integer assuming that $1/\epsilon$ is an integer. To see (iii), note that the maximum size over all dimensions is at most $(\Delta/\epsilon) = \frac{\epsilon}{d(1+1/\beta)^d} < 1$. ◀

6.2.3 Batching small jobs online

In the online setting we cannot wait to aggregate small jobs of the same type. To handle this issue, we pre-allocate one "bin" vector of each type. That is, before any jobs arrive, we pretend that one job of each load vector q arrives and assign it using the decision tree T. Then, batching jobs of the same type vector q in $F(q)$ is actually done on the machine that received the bin vector. Therefore, we can assign small jobs upon their arrival without waiting.

We have fully described our online algorithm to assign jobs upon their arrival. We now shift our focus to the analysis. When a job j is encapsulated into a type vector v of type $q \in \mathcal{Q}$, we say v contains job j.

▶ **Observation 31.** *Every bin vector of each type $q \in \mathcal{Q}$, possibly except one, has total size of jobs at least $(1-\epsilon)(\Delta/\epsilon)$.*

Proof. Since small jobs are aggregated only when they are of the same type, for each type $q \in \mathcal{Q}$, we can focus on the scalar quantities, job sizes p_j and the buffer size Δ/ϵ. The observation follows from the fact that we empty buffer $B(q)$ only when the total size of jobs in the buffer $B(q)$ exceeds $(1-\epsilon)(\Delta/\epsilon)$, or at the end after all jobs arrive. The only exception is due to the pre-allocation, which corresponds to emptying the buffer at the end. ◀

▶ **Lemma 32.** *If the total job load vector is at most $(1+\epsilon)m\vec{1}$, then the decision tree, due to batching and preallocation, receives jobs of total load vector at most $(1+4\epsilon)m\vec{1}$.*

Proof. By Observation 31, the total load vector T receives is at most $(1+\epsilon)m\vec{1}/(1-\epsilon) \le (1+3\epsilon)m\vec{1}$, plus $\sum_{q\in\mathcal{Q}}(\Delta/\epsilon)q$. Further, we have $\sum_{q\in\mathcal{Q}}(\Delta/\epsilon)q = |\mathcal{Q}|(\Delta/\epsilon)\vec{1} \le (1+1/\beta)^d \cdot \frac{\epsilon^2}{d(1+1/\beta)^d}\frac{1}{\epsilon}\vec{1} \le \epsilon\vec{1}$. ◀

Therefore, the algorithm sends to the decision tree T big jobs (including bin vectors which are big) of total load vector at most $(1+4\epsilon)m\vec{1}$. Note that sending less loads only helps our algorithm. By Observation 29, our algorithm's makespan is at most $(1+4\epsilon)c^*_d$-competitive if all jobs are discretized. We now show that when replacing each discretized vector with the original vector, every machine's load increases by at most $2\epsilon\vec{1}$. Knowing that the optimum makespan is 1, this will mean that the competitive ratio of our algorithm is at most $(1+6\epsilon)c^*_d$-competitive. By scaling ϵ appropriately, we obtain Theorem 24.

We complete the analysis by proving the following lemma.

▶ **Lemma 33.** *Restoring the discretized jobs load vectors to their original vectors increases each machine's load vector by at most $2\epsilon\vec{1}$.*

Proof. For the sake of contradiction, suppose the total load increases by more than 2ϵ on some fixed dimension d on some fixed machine i. In the first case, suppose at least ϵ increase was due to big jobs. Then, since discretizing a big job reduces its load by less than δ on each dimension, this means that the total number of big jobs assigned to the machine i is at least ϵ/δ. Since a big job has a volume Δ or more, the total volume of jobs assigned to machine i is at least $(\epsilon/\delta) \cdot \Delta = \epsilon/(\epsilon\Delta/2dm) \cdot \Delta = 2dm$, which is a contradiction to the fact that each machine has makespan at most $(1+4\epsilon)$, thus volume at most $(1+4\epsilon)d$. Now suppose at least ϵ increase was due to small jobs. Let S be the set of all small jobs assigned to machine i. We know that discretizing a small job j reduces its load by at most $p_j\beta$ on the fixed dimension d. Thus, we have $\sum_{j \in S} p_j\beta > \epsilon$. As a result, we have $\sum_{j \in S} p_j > \epsilon/\beta = \epsilon/(\epsilon/2md) = 2md$. This implies that the total volume of the jobs in S is at least $\sum_{j \in S} p_j \geq 2md$, which is a contradiction as before. ◀

6.3 Putting the Pieces Together

In Section 6.1 we showed if we use the first phase of the algorithm, then we can assign jobs to groups of machines so that each group has $m' = O(\frac{1}{\epsilon^3}\log d)$ machines and receives load at most $(1+\epsilon)m'\vec{1}$. Otherwise, we can pretend all jobs are assigned to the single group of all machines. In either case, we can assign jobs to groups of machines so that each group has $m' = O(\frac{1}{\epsilon^4}\log d)$ machines and receives load at most $(1+\epsilon)m'\vec{1}$. Then, using the procedure in 6.2, we can assign jobs to machines within group, so that each machine's load vector is at most $(1+6\epsilon)c_d^*\vec{1}$. Thus, we have found an online algorithm whose competitive ratio is $(1+\epsilon)c_d^*$ by appropriately scaling ϵ.

It now remains to show the running time of our algorithm. Since the running time is mostly dominated by the second phase, we will focus on the second phase. It is an easy exercise to see the running time is polynomially bounded by the size of decision tree and n. As discussed, the number of nodes is $(m|\mathcal{B}|)^{5md/\Delta}$, where $|\mathcal{B}| \leq (2/\delta)^d$. By the above discussion, we have $m = O(\frac{1}{\epsilon^4}\log d)$. Recall that $\beta := \frac{\epsilon}{2md}$, $\Delta := \frac{\epsilon^2}{d(1+1/\beta)^d}$, $\delta := \epsilon\beta\Delta/(2dm)$. By calculation, one can show that the tree size is $(d/\epsilon)^{d^{(d/\epsilon)^{O(d)}}}$. Thus, we have shown the running time.

This completes the proof of Theorem 24.

7 Open Problems

This paper gives the first non-trivial results for the online vector scheduling problem with a small number of dimensions. The most interesting open question is to better understand the competitive ratio of "practical" algorithms when $d > 2$. For instance, what is the competitive ratio of Priority(Max) when $d > 2$? Or, can we extend Priority(Bal) for $d = 2$ to higher dimensions? Even for $d = 2$, in our analysis, we used as the lower bound the fractional optimum where job vectors can be fractionally assigned to machines. This is inherently limited by the integrality gap of the fractional assignment. Can we obtain better lower bounds for the true optimum, and thereby improve the competitive ratio for the problem? For instance, for $d = 2$, is there a 2-competitive algorithm?

References

1 Susanne Albers. Better bounds for online scheduling. *SIAM J. Comput.*, 29(2):459–473, 1999.

2 Susanne Albers. On randomized online scheduling. In *STOC*, pages 134–143, 2002.

3 Susanne Albers. Online algorithms: a survey. *Math. Program.*, 97(1-2):3–26, 2003. doi:10.1007/s10107-003-0436-0.

4 Susanne Albers and Matthias Hellwig. Semi-online scheduling revisited. *Theor. Comput. Sci.*, 443:1–9, 2012.

5 Yossi Azar. On-line load balancing. In *Online Algorithms, The State of the Art.*, pages 178–195, 1996.

6 Yossi Azar, Ilan Reuven Cohen, Amos Fiat, and Alan Roytman. Packing small vectors. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1511–1525, 2016.

7 Yossi Azar, Ilan Reuven Cohen, Seny Kamara, and F. Bruce Shepherd. Tight bounds for online vector bin packing. In *Symposium on Theory of Computing Conference, STOC 2013, Palo Alto, CA, USA, June 1-4, 2013*, pages 961–970, 2013.

8 Yossi Azar, Ilan Reuven Cohen, and Debmalya Panigrahi. Randomized algorithms for online vector load balancing. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 980–991, 2018.

9 Yossi Azar and Oded Regev. On-line bin-stretching. *Theor. Comput. Sci.*, 268(1):17–41, 2001.

10 Yair Bartal, Amos Fiat, Howard J. Karloff, and Rakesh Vohra. New algorithms for an ancient scheduling problem. *J. Comput. Syst. Sci.*, 51(3):359–366, 1995.

11 Yair Bartal, Howard J. Karloff, and Yuval Rabani. A better lower bound for on-line scheduling. *Inf. Process. Lett.*, 50(3):113–116, 1994.

12 Martin Böhm, Jiří Sgall, Rob van Stee, and Pavel Veselý. A two-phase algorithm for bin stretching with stretching factor 1.5. *J. Comb. Optim.*, 34(3):810–828, 2017.

13 Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis.* Cambridge University Press, USA, 1998.

14 Allan Borodin, Morten N Nielsen, and Charles Rackoff. (incremental) priority algorithms. *Algorithmica*, 37(4):295–326, 2003.

15 Chandra Chekuri and Sanjeev Khanna. On multidimensional packing problems. *SIAM J. Comput.*, 33(4):837–851, 2004.

16 Edward Grady Coffman Jr, Michael Randolph Garey, and David Stifler Johnson. Approximation algorithms for bin packing: A survey. *Approximation algorithms for NP-hard problems*, pages 46–93, 1996.

17 Richard Cole, Vasilis Gkatzelis, and Gagan Goel. Mechanism design for fair division: allocating divisible items without payments. In *ACM Conference on Electronic Commerce, EC '13, Philadelphia, PA, USA, June 16-20, 2013*, pages 251–268, 2013.

18 Ulrich Faigle, Walter Kern, and György Turán. On the performance of on-line algorithms for partition problems. *Acta Cybern.*, 9(2):107–119, 1989.

19 Rudolf Fleischer and Michaela Wahl. Online scheduling revisited. In *Algorithms - ESA 2000, 8th Annual European Symposium, Saarbrücken, Germany, September 5-8, 2000*, pages 202–210, 2000.

20 Michaël Gabay, Nadia Brauner, and Vladimir Kotov. Improved lower bounds for the online bin stretching problem. *4OR*, 15(2):183–199, 2017.

21 Michaël Gabay, Vladimir Kotov, and Nadia Brauner. Online bin stretching with bunch techniques. *Theor. Comput. Sci.*, 602:103–113, 2015.

22 Gábor Galambos and Gerhard J. Woeginger. An on-line scheduling heuristic with better worst case ratio than graham's list scheduling. *SIAM J. Comput.*, 22(2):349–355, 1993.

23 M. R. Garey, Ronald L. Graham, David S. Johnson, and Andrew C. Yao. Resource constrained scheduling as generalized bin packing. *J. Comb. Theory, Ser. A*, 21(3):257–298, 1976.

24 Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, and Ion Stoica. Dominant resource fairness: Fair allocation of multiple resource types. In *Proc. 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2011.

25 Todd Gormley, Nick Reingold, Eric Torng, and Jeffery Westbrook. Generating adversaries for request-answer games. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2000, January 9-11, 2000, San Francisco, CA, USA.*, pages 564–565, 2000.

26 R. L. Graham. Bounds for certain multiprocessing anomalies. *Siam Journal on Applied Mathematics*, 1966.

27 R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17:416–429, 1969.

28 J. F. Rudin III. Improved bound for the on-line scheduling problem. *PhD thesis, The University of Texas at Dallas*, 2001.

29 Sungjin Im, Nathaniel Kell, Janardhan Kulkarni, and Debmalya Panigrahi. Tight bounds for online vector scheduling. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 525–544, 2015.

30 Sungjin Im, Nathaniel Kell, Debmalya Panigrahi, and Maryam Shadloo. Online load balancing on related machines. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 30–43, 2018.

31 Sungjin Im, Janardhan Kulkarni, and Kamesh Munagala. Competitive algorithms from competitive equilibria: Non-clairvoyant scheduling under polyhedral constraints. In *Proc. 46th ACM Symposium. on Theory of Computing (STOC)*, pages 313–322, 2014.

32 Sungjin Im, Janardhan Kulkarni, and Kamesh Munagala. Competitive flow time algorithms for polyhedral scheduling. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 506–524, 2015.

33 David S Johnson. *Near-optimal bin packing algorithms.* PhD thesis, Massachusetts Institute of Technology, 1973.

34 David S. Johnson. Fast algorithms for bin packing. *J. Comput. Syst. Sci.*, 8(3):272–314, 1974.

35 David S. Johnson, Alan Demers, Jeffrey D. Ullman, Michael R Garey, and Ronald L. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM Journal on computing*, 3(4):299–325, 1974.

36 David S. Johnson, Alan J. Demers, Jeffrey D. Ullman, M. R. Garey, and Ronald L. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM J. Comput.*, 3(4):299–325, 1974.

37 David R. Karger, Steven J. Phillips, and Eric Torng. A better algorithm for an ancient scheduling problem. *J. Algorithms*, 20(2):400–430, 1996.

38 Hans Kellerer and Vladimir Kotov. An efficient algorithm for bin stretching. *Oper. Res. Lett.*, 41(4):343–346, 2013.

39 Hans Kellerer, Vladimir Kotov, and Michaël Gabay. An efficient algorithm for semi-online multiprocessor scheduling with given total processing time. *J. Scheduling*, 18(6):623–630, 2015.

40 Hans Kellerer, Vladimir Kotov, Maria Grazia Speranza, and Zsolt Tuza. Semi on-line algorithms for the partition problem. *Oper. Res. Lett.*, 21(5):235–242, 1997.

41 Gunho Lee, Byung-Gon Chun, and Randy H Katz. Heterogeneity-aware resource allocation and scheduling in the cloud. In *Proceedings of the 3rd USENIX Workshop on Hot Topics in Cloud Computing, HotCloud*, volume 11, 2011.

42 Elisabeth Lübbecke, Olaf Maurer, Nicole Megow, and Andreas Wiese. A new approach to online scheduling: Approximating the optimal competitive ratio. *ACM Trans. Algorithms*, 13(1):15:1–15:34, 2016. doi:10.1145/2996800.

43 Adam Meyerson, Alan Roytman, and Brian Tagiku. Online multidimensional load balancing. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 16th International Workshop, APPROX 2013, and 17th International Workshop, RANDOM 2013, Berkeley, CA, USA, August 21-23, 2013.*, pages 287–302, 2013.

44 Lucian Popa, Gautam Kumar, Mosharaf Chowdhury, Arvind Krishnamurthy, Sylvia Ratnasamy, and Ion Stoica. Faircloud: sharing the network in cloud computing. In *ACM SIGCOMM*, pages 187–198. ACM, 2012.

45 Kirk Pruhs, Jiří Sgall, and Eric Torng. Online scheduling. *Handbook of scheduling: algorithms, models, and performance analysis*, pages 15–1, 2004.

46 Jiří Sgall. On-line scheduling. In *Online Algorithms*, pages 196–231, 1996.

47 Jiří Sgall. Online scheduling. In *Algorithms for Optimization with Incomplete Information, 16.-21. January 2005*, 2005.

48 Jiří Sgall. Online bin packing: Old algorithms and new results. In *CiE*, volume 8493 of *Lecture Notes in Computer Science*, pages 362–372. Springer, 2014.

Conditionally Optimal Approximation Algorithms for the Girth of a Directed Graph

Mina Dalirrooyfard
MIT, Cambridge, MA, USA
minad@mit.edu

Virginia Vassilevska Williams
MIT, Cambridge, MA, USA
virgi@mit.edu

Abstract

The girth is one of the most basic graph parameters, and its computation has been studied for many decades. Under widely believed fine-grained assumptions, computing the girth exactly is known to require $mn^{1-o(1)}$ time, both in sparse and dense m-edge, n-node graphs, motivating the search for fast approximations. Fast good quality approximation algorithms for undirected graphs have been known for decades. For the girth in directed graphs, until recently the only constant factor approximation algorithms ran in $O(n^\omega)$ time, where $\omega < 2.373$ is the matrix multiplication exponent. These algorithms have two drawbacks: (1) they only offer an improvement over the mn running time for dense graphs, and (2) the current fast matrix multiplication methods are impractical. The first constant factor approximation algorithm that runs in $O(mn^{1-\varepsilon})$ time for $\varepsilon > 0$ and all sparsities m was only recently obtained by Chechik et al. [STOC 2020]; it is also combinatorial.

It is known that a better than 2-approximation algorithm for the girth in dense directed unweighted graphs needs $n^{3-o(1)}$ time unless one uses fast matrix multiplication. Meanwhile, the best known approximation factor for a combinatorial algorithm running in $O(mn^{1-\varepsilon})$ time (by Chechik et al.) is 3. Is the true answer 2 or 3?

The main result of this paper is a (conditionally) tight approximation algorithm for directed graphs. First, we show that under a popular hardness assumption, any algorithm, even one that exploits fast matrix multiplication, would need to take at least $mn^{1-o(1)}$ time for some sparsity m if it achieves a $(2-\varepsilon)$-approximation for any $\varepsilon > 0$. Second we give a 2-approximation algorithm for the girth of unweighted graphs running in $\tilde{O}(mn^{3/4})$ time, and a $(2+\varepsilon)$-approximation algorithm (for any $\varepsilon > 0$) that works in weighted graphs and runs in $\tilde{O}(m\sqrt{n})$ time. Our algorithms are combinatorial.

We also obtain a $(4+\varepsilon)$-approximation of the girth running in $\tilde{O}(mn^{\sqrt{2}-1})$ time, improving upon the previous best $\tilde{O}(m\sqrt{n})$ running time by Chechik et al. Finally, we consider the computation of roundtrip spanners. We obtain a $(5+\varepsilon)$-approximate roundtrip spanner on $\tilde{O}(n^{1.5}/\varepsilon^2)$ edges in $\tilde{O}(m\sqrt{n}/\varepsilon^2)$ time. This improves upon the previous approximation factor $(8+\varepsilon)$ of Chechik et al. for the same running time.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis; Theory of computation → Sparsification and spanners; Theory of computation → Graph algorithms analysis

Keywords and phrases Shortest cycle, Girth, Graph algorithms, Approximation algorithms, Fine-grained complexity, Roundtrip Spanner

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.35

Category Track A: Algorithms, Complexity and Games

Related Version `https://arxiv.org/abs/2004.11445`

Funding *Mina Dalirrooyfard*: Partially supported by an NSF CAREER award and NSF Grant CCF-1514339.

Virginia Vassilevska Williams: Supported by an NSF CAREER Award, NSF Grants CCF-1528078, CCF-1514339 and CCF-1909429, a BSF Grant BSF:2012338, a Google Research Fellowship and a Sloan Research Fellowship.

Acknowledgements The authors would like to thank the anonymous reviewers for their insightful comments.

1 Introduction

One of the most basic and well-studied graph parameters is the *girth*, i.e. the length of the shortest cycle in the graph. Computing the girth in an m-edge, n-node graph can be done by computing all pairwise distances, that is, solving the All-Pairs Shortest Paths (APSP) problem. This gives an $\tilde{O}(mn)$ time algorithm for the general version of the girth problem: directed or undirected integer weighted graphs and no negative weight cycles[1].

The $\tilde{O}(mn)$ running time for the exact computation of the girth is known to be tight, up to $n^{o(1)}$ factors, both for sparse and dense weighted graphs, under popular hardness hypotheses from fine-grained complexity [21, 14]. In unweighted graphs or graphs with integer weights of magnitude at most M, one can compute the girth in $\tilde{O}(Mn^{\omega})$ time [19, 12, 17, 8] where $\omega < 2.373$ is the exponent of $n \times n$ matrix multiplication [22, 13]. This improves upon mn only for somewhat dense graphs with small weights, and moreover is not considered very practical due to the large overhead of fast matrix multiplication techniques.

Due to the subcubic equivalences of [21], however, it is known that even in unweighted dense graphs, any algorithm that computes the girth in $O(n^{3-\varepsilon})$ time needs to use fast matrix multiplication techniques, unless one can obtain a subcubic time combinatorial Boolean Matrix Multiplication (BMM) algorithm. Thus, under popular fine-grained complexity assumptions, if one wants to have a fast combinatorial algorithm, or an algorithm that is faster than mn for sparser graphs, one needs to resort to *approximation*.

Fast approximation algorithms for the girth in undirected graphs have been known since the 1970s, starting with the work of Itai and Rodeh [12]. The current strongest result shows a 2-approximation in $\tilde{O}(n^{5/3})$ time [18]; note that if the graph is dense enough this algorithm is sublinear in the input. Such good approximation algorithms are possible for undirected graphs because of known strong structural properties. For instance, as shown by Bondy and Simonovits [4], for any integer $k \geq 2$, if a graph has at least $100kn^{1+1/k}$ edges, then it must contain a $2k$ cycle, and this gives an immediate upper bound on the girth. There are no such structural results for directed graphs, making the directed girth approximation problem quite challenging.

Zwick [24] showed that if the maximum weight of an edge is M, one can obtain in $\tilde{O}(n^{\omega}\log(M/\varepsilon)/\varepsilon)$ time a $(1+\varepsilon)$-approximation for APSP, and this implies the same for the girth of directed graphs. As before, however, this algorithm does not run fast in sparse graphs, and can be considered impractical.

The first nontrivial approximation algorithms (both for sparse graphs and combinatorial) for the girth of directed graphs were achieved by Pachocki et al. [15]. The current best result by Chechik et al. [6, 7] achieves for every integer $k \geq 1$, a randomized $O(k \log k)$-approximation algorithm running in time $\tilde{O}(m^{1+1/k})$. The best approximation factor that Chechik et al. obtain in $O(mn^{1-\varepsilon})$ time for $\varepsilon > 0$ is 3, in $\tilde{O}(m\sqrt{n})$ time.

[1] If the weights are nonnegative, running Dijkstra's algorithm suffices. If there are no negative weight cycles, one can use Johnson's trick to make the weights nonnegative at the cost of a single SSSP computation which can be achieved for instance in $\tilde{O}(m\sqrt{n}\log M)$ time if M is the largest edge weight magnitude via Goldberg's algorithm [11], so as long as the weights have at most $\tilde{O}(\sqrt{n})$ bits, the total time is $\tilde{O}(mn)$.

What should be the best approximation factor attainable in $O(mn^{1-\varepsilon})$ time for $\varepsilon > 0$? It is not hard to show (see e.g. [20], the construction in Thm 4.1.3) that graph triangle detection can be reduced to triangle detection in a directed graph whose cycle lengths are all divisible by 3. This, coupled with the combinatorial subcubic equivalence between triangle detection and BMM [21] implies that any $O(n^{3-\varepsilon})$ time algorithm for $\varepsilon > 0$ that achieves a $(2-\delta)$-approximation for the girth implies an $O(n^{3-\varepsilon/3})$ time algorithm for BMM, and hence fast matrix multiplication techniques are likely necessary for faster $(2-\varepsilon)$-approximation of the directed girth.

1.1 Our results

We first give a simple extension to the above hardness argument for $(2-\varepsilon)$-approximation, giving a conditional lower bound on the running time of $(2-\varepsilon)$-girth approximation algorithms under the so called k-Cycle hardness hypothesis [3, 16, 14].

The k-Cycle hypothesis states that for every $\varepsilon > 0$, there is a k such that k-cycle in m-edge directed unweighted graphs cannot be solved in $O(m^{2-\varepsilon})$ time (on a $O(\log n)$ bit word-RAM).

The hypothesis is consistent with all known algorithms for detecting k-cycles in directed graphs, as these run at best in time $m^{2-c/k}$ for various small constants c [23, 2, 14, 9], even using powerful tools such as matrix multiplication. Moreover, as shown by Lincoln et al. [14] any $O(mn^{1-\varepsilon})$ time algorithm (for $\varepsilon > 0$) that, for odd k, can detect k-cycles in n-node m-edge directed graphs with $m = \Theta(n^{1+2/(k-1)})$, would imply an $O(n^{k-\delta})$ time algorithm for k-clique detection for $\delta > 0$. If the cycle algorithm is "combinatorial", then the clique algorithm would be "combinatorial" as well, and since all known $O(n^{k-\delta})$ time k-clique algorithms use fast matrix multiplication, such a result for k-cycle would be substantial.

In Section 5, with a very simple reduction we show:

▶ **Theorem 1.** *Suppose that for some constants $\varepsilon > 0$ and $\delta > 0$, there is an $O(m^{2-\varepsilon})$ time algorithm that can compute a $(2-\delta)$-approximation of the girth in an m-edge directed graph. Then for every constant k, one can detect whether an m-edge directed graph contains a k-cycle, in $O(m^{2-\varepsilon})$ time, and hence the k-Cycle Hypothesis is false.*

Thus, barring breakthroughs in Cycle and Clique detection algorithms, we know that the best we can hope for using an $O(mn^{1-\varepsilon})$ time algorithm for the girth of directed graphs is a 2-approximation.

The main result of this paper is the first ever $O(mn^{1-\varepsilon})$ time for $\varepsilon > 0$ 2-approximation algorithm for the girth in directed graphs. This result is conditionally tight via the above discussion.

▶ **Theorem 2.** *There is an $\tilde{O}(mn^{3/4})$ time randomized algorithm that 2-approximates the girth in directed unweighted graphs whp. For every $\varepsilon > 0$, there is a $(2+\varepsilon)$-approximation algorithm for the girth in directed graphs with integer edge weights that runs in $\tilde{O}(m\sqrt{n}/\varepsilon)$ time. The algorithms are randomized and are correct whp.*

If one wanted to obtain a $(4+\varepsilon)$-approximation to the girth via Chechik et al.'s $O(k \log k)$ approximation algorithms, the best running time one would be able to achieve is $\tilde{O}(m\sqrt{n})$. Here we show how to get an improved running time for a $(4+\varepsilon)$ approximation.

▶ **Theorem 3.** *For every $\varepsilon > 0$, there is a $(4+\varepsilon)$-approximation algorithm for the girth in directed graphs with integer edge weights that runs in $\tilde{O}(mn^{\sqrt{2}-1}/\varepsilon)$ time. The algorithm is randomized and correct whp.*

In fact, we obtain a generalization of the above algorithms that improves upon the algorithms of Chechik et al. for all constants k.

▶ **Theorem 4.** *For every $\varepsilon > 0$ and integer $k \geq 1$, there is a $(2k+\varepsilon)$-approximation algorithm for the girth in directed graphs with integer edge weights that runs in $\tilde{O}(mn^{\alpha_k}/\varepsilon)$ time, where $\alpha_k > 0$ is the solution to $\alpha_k(1+\alpha_k)^{k-1} = 1 - \alpha_k$. The algorithms are randomized and correct whp.*

For example, let's consider α_1 in the above theorem. It is the solution to $\alpha_1 = 1 - \alpha_1$, giving $\alpha_1 = 1/2$ and recovering the result of Theorem 2 for weighted graphs. On the other hand, α_2 is the solution to $\alpha_2(1+\alpha_2) = 1 - \alpha_2$, which gives $\alpha_2 = \sqrt{2} - 1$ and recovering Theorem 3. Finally, say we wanted to get a $6 + \varepsilon$ approximation, then we need α_3, which is the solution to $\alpha_3(1+\alpha_3)^2 = 1 - \alpha_3$, giving $\alpha_3 \leq 0.354$, and thus there's an $\tilde{O}(mn^{0.354}/\varepsilon)$ time $(6+\varepsilon)$-approximation algorithm. Note that there is only one positive solution to the equation defining α_k in Theorem 4.

As k grows, α_k grows as $\Theta(\log k/k)$, and so the algorithm from Theorem 4 has similar asymptotic guarantees as the algorithm of Chechik et al. as it achieves an $O(\ell \log \ell)$ approximation in $\tilde{O}(mn^{1/\ell})$ time. The main improvements lie in the improved running time for small constant approximation factors.

Our approximation algorithms on weighted graphs can be found in section 4. If we are aiming for an algorithm running in $T(n,m)$ time, we first suppose that the maximum edge weight of the graph is M and we obtain an algorithm in $T(n,m)\log M$ time. We show how to remove the $\log M$ factor using standard techniques in the full version [10].

Roundtrip Spanners. Both papers that achieved nontrivial combinatorial approximation algorithms for the directed girth were also powerful enough to compute sparse approximate roundtrip spanners.

A c-approximate roundtrip spanner of a directed graph $G = (V,E)$ is a subgraph $H = (V,E')$ of G such that for every $u,v \in V$, $d_H(u,v) + d_H(v,u) \leq c \cdot (d_G(u,v) + d_G(v,u))$. Similar to what is known for spanners in undirected graphs, it is known [5] that for every integer $k \geq 2$ and every n, every n-node graph contains a $(2k-1+o(1))$-approximate roundtrip spanner on $O(kn^{1+1/k}\log n)$ edges; the $o(1)$ error can be removed if the edge weights are at most polynomial in n and the result then is optimal, up to log factors under the Erdös girth conjecture.

The best algorithms to date for computing sparse roundtrip spanners, similarly to the girth, achieve an $O(k\log k)$ approximation in $\tilde{O}(m^{1+1/k})$ time [7]. The best constant factor approximation achieved for roundtrip spanners in $O(mn^{1-\varepsilon})$ time for $\varepsilon > 0$ is again achieved by Chechik et al.: a $(8+\varepsilon)$ approximate $O(n^{1.5})$-edge (in expectation) roundtrip spanner can be computed in $\tilde{O}(m\sqrt{n})$ expected time. We improve this latter result:

▶ **Theorem 5.** *There is an $\tilde{O}(m\sqrt{n}\log^2(M)/\varepsilon^2)$ time randomized algorithm that computes a $(5+\varepsilon)$-approximate roundtrip spanner on $\tilde{O}(n^{1.5}\log^2(M)/\varepsilon^2)$ edges whp, for any n-node m-edge directed graph with edge weights in $\{1,\ldots,M\}$.*

2 Preliminary Lemmas

We begin with some preliminary lemmas. The first two will allow us to decrease all degrees to roughly m/n, while keeping the number of vertices and edges roughly the same. The last lemma, implicit in [6], is a crucial ingredient in our algorithms.

The following lemma was proven by Chechik et al. [6]:

▶ **Lemma 6.** *Given a directed graph $G = (V, E)$ with $|V| = n, |E| = m$, we can in $O(m+n)$ time construct a graph $G' = (V', E')$ with $V \subseteq V'$, so that $|V'| \leq O(n)$, $|E'| \leq O(m+n)$, for every $v \in V'$, $deg(v) \leq \lceil m/n \rceil$, and so that for every $u, v \in V$, $d_{G'}(u, v) = d_G(u, v)$, and so that any path p between some nodes $u \in V$ and $v \in V$ in G' (possibly $u = v$) is in one-to-one correspondence with a path in G of the same length.*

The proof of the above lemma introduces edges of weight 0, even if the graph was originally unweighted. In the lemma below which is proved in the the full version [10], we show how for an unweighted graph we can achieve essentially the same goal, but without adding weighted edges. This turns out to be useful for our unweighted girth approximation.

▶ **Lemma 7.** *Given a directed unweighted graph $G = (V, E)$ and $|V| = n, |E| = m$, we can in $\tilde{O}(m+n)$ time construct an unweighted graph $G' = (V', E')$ with $V \subseteq V'$, so that $|V'| \leq O(n \log n)$, $|E'| \leq O(m + n \log n)$, for every $v \in V'$ out-deg$(v) \leq \lceil m/n \rceil$, and so that there is an integer t such that for every $u, v \in V$, $d_{G'}(u, v) = t \cdot d_G(u, v)$, and so that any path p between some nodes $u \in V$ and $v \in V$ in G' (possibly $u = v$) is in one-to-one correspondence with a path in G of length $1/t$ of the length of p.*

In particular, the lemma will imply that the girth of G' is exactly t times the girth of G, and that given a c-roundtrip spanner of G', one can in $\tilde{O}(m+n)$ time obtain from it a c-roundtrip spanner of G. We note that it is easy to obtain the same result but where both the in- and out-degrees are $O(m/n)$ (see the proof in the full version [10]).

Now we can assume that the degree of each node is no more than $O(m/n)$. This will allow us for instance to run Dijkstra's algorithm or BFS from a vertex within a neighborhood of w nodes in $\tilde{O}(mw/n)$ time.

Another assumption we can make without loss of generality is that our given graph G is strongly connected. In linear time we can compute the strongly connected components and then run any algorithm on each component separately. We know that any two vertices in different components have infinite roundtrip distance.

A final lemma (implicit in [6]) will be very important for our algorithms:

▶ **Lemma 8.** *Let $G = (V, E)$ be a directed graph with $|V| = n$ and integer edge weights in $\{1, \ldots, M\}$. Let $S \subseteq V$ with $|S| > c \log n$ (for $c \geq 100/\log(10/9)$) and let d be a positive integer. Let R be a random sample of $c \log n$ nodes of S and define $S' := \{s \in S \mid d(s, r) \leq d, \forall r \in R\}$. Suppose that for every $s \in S$ there are at most $0.2|S|$ nodes $v \in V$ so that $d(s, v), d(v, s) \leq d$. Then $|S'| \leq 0.8|S|$.*

Proof. The proof will consist of two parts. First we will show that the number of ordered pairs $s, s' \in S$ for which $d(s, s'), d(s', s) \leq d$ is small. Then we will show that if $|S'| > 0.8|S|$, then with high probability, the number of ordered pairs $s, s' \in S$ for which $d(s, s'), d(s', s) \leq d$ is large, thus obtaining a contradiction.

(1) If for every $s \in S$ there are at most $0.2|S|$ nodes $v \in V$ so that $d(s, v), d(v, s) \leq d$, then the number of ordered pairs $s, s' \in S$ for which $d(s, s'), d(s', s) \leq d$ is clearly at most $0.2|S|^2$.

(2) Suppose now that $|S'| > 0.8|S|$. First, consider any $s \in S$ for which there are at least $0.1|S|$ nodes $s' \in S$ such that $d(s, s') > d$. The probability that $d(s, r) \leq d$ for all $r \in R$ is then at most $0.9^{c \log n} \leq 1/n^{100}$. Thus, via a union bound, with high probability at least $1 - 1/n^{99}$, for every $s \in S'$, there are at least $0.9|S|$ nodes $s' \in S$ such that $d(s, s') \leq d$.

Now, if $|S'| > 0.8|S|$, with high probability, there are at least $0.8|S| \times 0.9|S| = 0.72|S|^2$ ordered pairs (s, s') with $s, s' \in S$ and $d(s, s') \leq d$. There are at most $\binom{|S|}{2} \leq |S|^2/2$ ordered pairs (s, s') such that exactly one of $\{d(s, s') \leq d, d(s', s) \leq d\}$ holds. Hence, with high probability there are at least $0.22|S|^2 > 0.2|S|^2$ ordered pairs (s, s') with $s, s' \in S$ and both $d(s, s') \leq d$ and $d(s', s) \leq d$. Contradiction. ◀

3 2-Approximation for the Girth in Unweighted Graphs

Here we show how to obtain a genuine 2-approximation for the girth in unweighted graphs.

▶ **Theorem 9.** *Given a directed unweighted graph G on m edges and n nodes, one can in $\tilde{O}(mn^{3/4})$ time compute a 2-approximation to the girth.*

Note that this is the first part of Theorem 2. The pseudocode for the algorithm of Theorem 9 can be found in Algorithm 1, and we will refer to it at each stage of the proof.

We will consider two cases for the girth: when it is $\geq n^\delta$ and when it is $< n^\delta$, for some $\delta > 0$ we will eventually set to $1/4$. We will assume that all out-degrees in the graph are $O(m/n)$.

3.1 Large girth

Pick a random sample R of $100n^{1-\delta}\log n$ nodes, run BFS to and from each $s \in R$. Return

$$\min_{s\in R}\min_{v\neq s} d(s,v) + d(v,s).$$

If the girth is $\geq n^\delta$, with high probability, R will contain a node s on the shortest cycle C. Since any cycle must contain two distinct nodes, $\min_{s\in R}\min_{v\neq s} d(s,v)+d(v,s)$ is the weight of a shortest cycle that contains some node of R, and with high probability it must be the girth. Thus in $\tilde{O}(mn^{1-\delta})$ time we have computed the girth exactly. See Procedure HighGirth in Algorithm 1.

3.2 Small girth

Now let us assume that the girth is at most n^δ. For a vertex u and integer $j \in \{0,\dots,n^\delta\}$, define

$$B^j(u) := \{x \in V \mid d(u,x) = j\} \text{ and } \bar{B}^j(u) := \{x \in V \mid d(u,x) \leq j\}.$$

We will try all choices of integers i from 3 to n^δ to estimate the girth when it is $\leq i$.

Our algorithm first computes a random sample Q of size $O(n^{1-t}\log n)$ for a parameter t, does BFS from and to all nodes in Q, and computes for each $i \in \{1,\dots,n^\delta\}$, $V'_i = \{v \in V \mid \exists q \in Q: \ d(v,q) \leq i \text{ and } d(q,v) \leq i\}$. The running time needed to do this for all $i \leq n^\delta$ is $\tilde{O}(mn^{1-t+\delta})$ [2].

If $V'_i \neq \emptyset$, the girth of G must be $\leq 2i$.

Now, pick the smallest i for which $V'_{i+1} \neq \emptyset$. Then $V'_k = \emptyset$ for all $k \leq i$, and we have certified that the girth is $\leq 2i+2$. If the girth is $\geq i+1$, we already have a 2-approximation. Otherwise, the girth must be $\leq i$.

Consider any $u \in V$, and $j \leq i$. Suppose that for all $j \leq i$, $|B^j(u)| \leq 100n^t$. Then, for u and for all $v \in B^j(u)$ for $j \leq i$, we could compute the distances from u to v in G efficiently: We do this by running BFS from u but stopping when a vertex outside of $\cup_{j=0}^{i} B^j(u)$ is found. Note that the number of vertices in $\cup_{j=0}^{i} B^j(u)$ is $O(n^t \cdot i)$, and since we assumed that the degree of every vertex is $O(m/n)$, we get a total running time of $O(mn^{t-1} \cdot i)$. If this works for all vertices u, then we would be able to compute all distances up to i exactly in total time $O(m \cdot in^t) \leq O(mn^{t+\delta})$.

[2] The running time is actually less, $\tilde{O}(n^{2-t+\delta} + mn^{1-t})$ but this won't matter for our algorithm.

Unfortunately, however, some $B^j(u)$ balls can be larger than $100n^t$. In this case, for every $j \leq i$, we will compute a small set of nodes $B'^j(u)$ that will be just as good as $B^j(u)$ for computing short cycles.

▷ Claim 10. Fix i: $1 \leq i \leq n^\delta$. Suppose that for every $j \leq i$ we are given black box access to sets $B'^j(u) \subseteq \bar{B}_j(u)$ of nodes such that (1) In $t(n)$ time we can check whether a node is in $B'^j(u)$, (2) $|B'^j(u)| \leq 100n^t$ whp, and (3) for any cycle C of length $\leq i$ containing u, and every $j \leq i$, any node of C that is in $B^j(u)$ is also in $B'^j(u)$.

Then there is an $O(mn^{t-1+\delta}t(n))$ time algorithm that can find a shortest cycle through u, provided that cycle has length $\leq i$.

Proof. Let us assume that there is some cycle C of length $\leq i$ containing u. Also, assume that we are given the sets $B'^j(u)$ for all $j \leq i$ as in the statement of the lemma.

Then we can compute a modified BFS out of u. We will show by induction that when considering distance $j \leq i$, our modified BFS will have found a set $N_j(u)$ of nodes such that for every $x \in N_j(u)$, $d(u,x) \leq j$, and so that for any cycle C of length $\leq i$ containing u, any node of C that is in $B^j(u)$ is also in $N_j(u)$.

Initially, $N_0(u) = \{u\}$, so the base case is fine. Let's make the induction hypothesis for j that for every $x \in N_j(u)$, $d(u,x) \leq j$, and for a shortest cycle C of length $\leq i$ containing u, any node of C that is in $B^j(u)$ is also in $N_j(u)$.

Our modified BFS proceeds as follows: Given $N_j(u)$, we go through each $z \in N_j(u)$, and if $z \in B'^j(u)$, we go through all out-neighbors y of z, and if y has not been visited until now, we place y into $N_{j+1}(u)$. See Procedure ModBFS in Algorithm 1 (parameter t is set to 1/2 here).

Clearly, since $d(u,z) \leq j$ (by the induction hypothesis), we have that $d(u,y) \leq j+1$ for each out-neighbor y of z. Now consider a shortest cycle C containing u of length $\leq i$. To complete the induction we only care about $j < |C|$.

Assume that the induction hypothesis for j holds. Let x be the node on C at distance $j+1$ from u along C, and let x' be its predecessor on C, i.e. the node on C at distance j from u along C. Since C is a shortest cycle containing u and since $x' \neq x$, we must have that $d(u,x') = j$ so that $x' \in B^j(u)$. Also, either $u = x$, or $d(u,x) = j+1$ and so $x \in B^{j+1}(u)$.

We know by the induction hypothesis that $x' \in N_j(u)$ and also that $x' \in B'^j(u)$ by the definition of $B'^j(u)$. Thus, we would have gone through the edges out of x', and x would have been discovered. If $u = x$, then the cycle C will be found. Otherwise, $d(u,x) = j+1$, and x cannot have been visited until now, so our modified BFS will insert x into $N_{j+1}(u)$ thus completing the induction.

The running time of the modified BFS is determined by the fact that there are $i \leq n^\delta$ levels, each of $N_j(u) \cap B'^j(u)$ contains $\leq O(n^t)$ nodes, and we traverse the $O(m/n)$ edges out of every $x \in N_j(u) \cap B'^j(u)$. The running time is thus asymptotically $t(n) \times n^\delta \times n^t \times m/n$ which is $O(mn^{t+\delta-1}t(n))$. ◁

Now we want to explain how to compute the sets $B'^j(u)$. We use Lemma 8 from the preliminaries. Suppose that the girth is at most i and for every $k \leq i$, $V'_k = \emptyset$.

Let u be a node on a cycle C of length at most i. Let x be any node on C so that $x \in B^j(u)$ for some integer $j \leq i$. Then we must have that for every $y \in \bar{B}^j(u)$:

$$d(x,y) \leq d(x,u) + d(u,y) \leq |C| - d(u,x) + d(u,y) \leq i - j + j = i.$$

This inequality is crucial for our algorithm. See Figure 1 for a depiction of it.

In other words, we obtain that x is in $\{w \in B^j(u) \mid d(w,y) \leq i, \ \forall y \in \bar{B}^j(u)\}$.

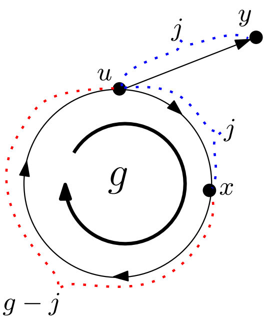


Figure 1 Here there is a cycle of length g containing u. A node x on the cycle is at distance j from u along the cycle and another node y is at distance $\leq j$ from u. Then the distance from x to y is at most g since one way to go from x to y is to go from x to u along the cycle at a cost of $g - j$, and then from u to y at a cost of $\leq j$. If the cycle is a shortest cycle containing u and if $x \neq u$, then the distance in the graph from u to x is j, as the path along the cycle needs to be a shortest path.

Suppose that we are able to pick a random sample $R^j(u)$ of $c \log n$ vertices from $\bar{B}^j(u)$ (we will show how later). Then we can define

$$\bar{B}'^j(u) := \{z \in \bar{B}^j(u) \mid d(z, y) \leq i, \ \forall y \in R^j(u)\}.$$

Using Lemma 8 we will show that if $|\bar{B}^j(u)| \geq 10n^t$, then $|\bar{B}'^j(u)| \leq 0.8\bar{B}^j(u)$ and if x is in $\{w \in B^j(u) \mid d(w, y) \leq i, \ \forall y \in \bar{B}^j(u)\}$, then whp $x \in \bar{B}'^j(u)$. We will then repeat the argument to obtain $B'^j(u)$ of size $O(n^t)$.

Consider any $s \in V$ with at least $0.2|\bar{B}^j(u)|$ nodes $v \in V$ so that $d(s, v), d(v, s) \leq i$. As $|\bar{B}^j(u)| \geq 10n^t$ (as otherwise we would be done), $0.2|\bar{B}^j(u)| \geq 2n^t$, and so with high probability, for s with the property above, our earlier random sample Q contains some q with $d(s, q), d(q, s) \leq i$, and so $V'_i \neq \emptyset$ which we assumed didn't happen. Thus with high probability, for every $s \in V$, there are at most $0.2|\bar{B}^j(u)|$ nodes $v \in V$ so that $d(s, v), d(v, s) \leq i$. Hence we also have that every $z \in \bar{B}^j(u)$ has at most $0.2|\bar{B}^j(u)|$ nodes $v \in V$ so that $d(s, v), d(v, s) \leq i$.

Thus we can apply Lemma 8 to $\bar{B}^j(u)$ and conclude that $|\bar{B}'^j(u)| \leq 0.8|\bar{B}^j(u)|$, while also any node $x \in B^j(u)$ on the cycle C (containing u) is also in $\bar{B}'^j(u)$.

We will iterate this process until we arrive at a subset of $\bar{B}^j(u)$ that is smaller than $10n^t$ and still contains all $x \in B^j(u)$ on an $\leq i$-length cycle C.

We do this as follows. Let $B^j_0(u) = \bar{B}^j(u)$. For each $k = 0, \ldots, 2\log n$, let $R^j_k(u)$ be a random sample of $O(\log n)$ vertices of $B^j_k(u)$. Define $B^j_{k+1}(u) = \{z \in B^j_k(u) \mid d(z, y) \leq i, \ \forall y \in \cup_{\ell=0}^k R^j_\ell(u)\}$. We get that for each k, $|B^j_k(u)| \leq 0.8^k|\bar{B}^j(u)|$ so that at the end of the last iteration, $|B^j_{2\log n}(u)| \leq 10n^t$ and we can set $B'^j(u)$ to $B^j_{2\log n}(u)$.

It is not immediately clear how to obtain the random sample $R^j_k(u)$ from $B^j_k(u)$ as $B^j_k(u)$ is unknown. We do it in the following way, adapting an argument from Chechik et al. [7]. For each $j \leq i$ and $k \leq 2\log n$ we independently obtain a random sample $S_{j,k}$ of V by sampling each vertex independently with probability $p = 100 \log n/n^t$. For each of the (in expectation) $O(n^{1-t+\delta} \log^2(n))$ vertices in the sets $S_{j,k}$ we run BFS to and from them, to obtain all their distances.

Now, for $j \leq i$ and k, to obtain the random sample $R^j_k(u)$ of the unknown $B^j_k(u)$, we assume that we already have $R^j_\ell(u)$ for $\ell < k$, and define

$$T^j_k(u) = \{s \in S_{j,k} \mid s \in \bar{B}^j(u) \text{ and } d(s, y) \leq i, \ \forall y \in \cup_{\ell<k} R^j_\ell(u)\}.$$

Forming the set $T_k^j(u)$ is easy since we have the distances $d(s,v)$ for all $s \in S_{j,k}$ and $v \in V$, so we can check whether $s \in \bar{B}^j(u)$ and $d(s,y) \leq i,\ \forall y \in \cup_{\ell<k} R_\ell^j(u)$ in polylogarithmic time for each $s \in S_{j,k}$. See Procedure RandomSamples in Algorithm 1.

Now since $S_{j,k}$ is independent from all our other random choices, $T_k^j(u)$ is a random sample of $B_k^j(u)$ essentially created by selecting each vertex with probability p. If $B_k^j(u) \geq 100n^t$, with high probability, $T_k^j(u)$ has at least $10 \log n$ vertices so we can pick $R_k^j(u)$ to be a random sample of $10 \log n$ vertices of $T_k^j(u)$, and they will also be a random sample of $10 \log n$ vertices of $B_k^j(u)$.

Once we have the sets $R_k^j(u)$ for each u and $j \leq i$, $k \leq 2 \log n$, we run our modified BFS from each u from Claim 10 where when we are going through the vertices $x \in N_j(u)$ we check whether $x \in B'^j(u)$ by checking whether $d(x,r) \leq i$ for every $r \in \cup_k R_k^j(u)$. This only gives a polylogarithmic overhead so we can run the modified BFS in time $\tilde{O}(mn^{t-1+\delta})$ time. We can run it through all $u \in V$ in total time $\tilde{O}(mn^{t+\delta})$ time, and in this time we will be able to compute the length of the shortest cycle if that cycle is of length $\leq i$.

Putting it all together. In $\tilde{O}(mn^{1-\delta})$ time we compute the girth exactly if it is $\geq n^\delta$. In $\tilde{O}(mn^{1-t+\delta})$ time, we obtain i so that we have a 2-approximation of the girth if the girth is $> i$. In additional $\tilde{O}(mn^{1-t+\delta} + mn^{t+\delta})$ time we compute the girth exactly if it is $\leq i$.

To optimize the running time we set $t = 1/2$, $1 - \delta = 0.5 + \delta$, obtaining $\delta = 1/4$, and a running time of $\tilde{O}(mn^{3/4})$. The final algorithm is in Algorithm 1.

4 Weighted Graphs: Girth and Roundtrip Spanner

One of the main differences between our weighted and unweighted algorithms is that for weighted graphs we do not go through each distance value i up to n^δ, but we instead process intervals of possible distance values $[(1+\varepsilon)^i, (1+\varepsilon)^{i+1})$ for small $\varepsilon > 0$. This will affect the approximation, so that we will get a $(2+O(\varepsilon))$-approximation. However, it will also enable us to have a smaller running time of $\tilde{O}(m\sqrt{n}\log(M)/\varepsilon)$, and to be able to output an $\tilde{O}(n^{1.5}\log(M)/\varepsilon)$-edge $(5+O(\varepsilon))$-approximate roundtrip spanner in $\tilde{O}(m\sqrt{n}\log(M)/\varepsilon^2)$ time, where M is the maximum edge weight.

Fix $\varepsilon > 0$. For a vertex u and integer j, define (differently from the previous section)

$$B^j(u) := \{x \in V \mid (1+\varepsilon)^j \leq d(u,x) < (1+\varepsilon)^{j+1}\} \text{ and } \bar{B}^j(u) := \{x \in V \mid d(u,x) < (1+\varepsilon)^{j+1}\}.$$

We include a boundary case $B^\emptyset(u) := \{x \in V \mid d(u,x) = 0\}$. Recall that we originally started with a graph with positive integer weights, but our transformation to vertices of degree $O(m/n)$ created some 0 weight edges. We note that any distance of 0 involves at least one of the auxiliary vertices and no roundtrip distance can be 0.

In our algorithms including our $(2+\epsilon)$-approximation algorithm, we do a restricted version of Dijkstra from every vertex where before running these Dijkstras, we need to efficiently sample a set of vertices $R^j(u)$ of size $O(\log n)$ from a subset of $B^j(u)$, without computing the set $B^j(u)$. The following lemma is given as input the target approximation factor 2β, a parameter i as an estimated size of cycles the algorithm is handling at a given stage and a parameter α as the target running time $\tilde{O}(mn^\alpha)$ of our algorithms. It outputs the sample sets in this running time. The proof of the lemma is similar to the sampling method of the previous section and is included in the full version [10].

Algorithm 1 2-Approximation algorithm for the girth in unweighted graphs.

```
Procedure HighGirth($G = (V, E)$)
    Let $R \subseteq V$ be a uniform random sample of $100n^{3/4} \log n$ nodes.
    foreach $s \in R$ do
        Do BFS from $s$ in $G$
    Let $g$ be the length of the shortest cycle found by the BFS searches.
    Return $g$.
Procedure RandomSamples($G = (v, E), i$)
    foreach $j \in \{1, \dots, i\}$ do
        foreach $k \in \{1, \dots, 2 \log n\}$ do
            Let $S_{j,k} \subseteq V$ be a uniform random sample of $100\sqrt{n} \log n$ vertices.
            foreach $s \in S_{j,k}$ do
                Do BFS to and from $s$ to compute for all $v$, $d(s, v)$ and $d(v, s)$.
    foreach $u \in V$ do
        foreach $j \in \{1, \dots, i\}$ do
            $R^j(u) \leftarrow \emptyset$.
            foreach $k \in \{1, \dots, 2 \log n\}$ do
                $T^j_k(u) \leftarrow \{s \in S_{j,k} \mid d(u, s) \leq j$ and for all $y \in R^j(u) : \ d(s, y) \leq i\}$.
                if $|T^j_k(u)| < 10 \log n$ then
                    $R^j(u) \leftarrow R^j(u) \cup T^j_k(u)$
                    Exit this loop (over $k$).
                else
                    Let $R^j_k(u)$ be a uniform random sample of $10 \log n$ nodes from
                      $T^j_k(u)$.
                    $R^j(u) \leftarrow R^j(u) \cup R^j_k(u)$.
    Return the sets $R^j(u)$ for all $j \leq i$, $u \in V$, and $d(s, v), d(v, s)$ for all $s \in \cup_{j,k} S_{j,k}$
      and $v \in V$.
Procedure ModBFS($G = (v, E), u, i, R^1(u), \dots, R^i(u)), d(\cdot)$
    // $d(\cdot)$ contains $d(s, v), d(v, s)$ for all $s \in \cup_{j,k} S_{j,k}$ and $v \in V$.
    $Visited \leftarrow$ empty hash table
    $N_0 \leftarrow \{u\}$
    $Visited.insert(u)$
    foreach $j$ from $0$ to $i - 1$ do
        $N_{j+1} \leftarrow$ empty linked list
        foreach $x \in N_j$ do
            if for every $s \in R^j(u)$, $d(x, s) \leq i$ then
                foreach $y$ s.t. $(x, y) \in E$ and $y \notin Visited$ do
                    if $y = u$ then
                        Stop and return $j + 1$
                    $N_{j+1}.insert(y)$
                    $Visited.insert(y)$
    Return $\infty$ // No $\leq i$ length cycle found through $u$
```

```
Procedure $\textit{GirthApprox}(G = (V,E))$
    $g_{high} \leftarrow \textsc{HighGirth}(G)$
    Let $Q \subseteq V$ be a uniform random sample of $100n^{1/2}\log n$ nodes.
    foreach $s \in Q$ do
        Do BFS from and to $s$ in $G$
    Let $i$ be the minimum integer s.t. $\exists s \in Q$ and $\exists v \in V$ with $d(s,v) \leq i+1$ and
      $d(v,s) \leq i+1$.
    $g_{med} \leftarrow 2(i+1)$
    Let $i$ be the min of $i$ and $n^{1/4}$
    Run $\textsc{RandomSamples}(G,i)$ to obtain sets $R^j(u)$ for all $j \leq i$, $u \in V$, and $d(\cdot)$
      containing $d(s,v), d(v,s)$ for all $s \in \cup_{j,k} S_{j,k}$ and $v \in V$
    foreach $u \in V$ do
        $g_u \leftarrow \textsc{ModBFS}(G, u, i, R^1(u), \ldots, R^i(u), d(\cdot))$
    $g \leftarrow \min\{g_{high}, g_{med}, \min_{u \in V} g_u\}$
    Return $g$
```

▶ **Lemma 11.** *Let M be the maximum edge weight of the graph and suppose that $i \in \{1,\ldots,\log_{1+\epsilon} Mn\}$, $\beta > 0$ and $0 < \alpha < 1$ are given. Suppose that Q is a given sampled set of size $\tilde{O}(n^\alpha)$ vertices. Let $d = \beta(1+\epsilon)^{i+1}$. Let $V_i' = \{v \in V \mid \exists q \in Q : \ d(v,q) \leq d$ and $d(q,v) \leq d\}$. In $\tilde{O}(mn^\alpha)$ time, for every $u \in V$ and every $j = \{1,\ldots,\log_{(1+\epsilon)}(Mn)\}$, one can output a sample set $R_i^j(u)$ of size $O(\log^2 n)$ from $\bar{Z}_i^j(u) = \bar{B}^j(u) \setminus V_i'$, where the number of vertices in $Z_i^j(u) = B^j(u) \setminus V_i'$ of distance at most d from all vertices in $R_i^j(u)$ is at most $O(n^{1-\alpha})$ whp.*

Now we focus on our $(2+O(\epsilon))$-approximation algorithm for the girth and $(5+O(\epsilon))$-approximate roundtrip spanner. We are going to prove the following Theorem, which consists of Theorem 5 and the second part of Theorem 2 with a $\log M$ factor added to their running times.

▶ **Theorem 12.** *Let G be an n-node, m-edge directed graph with edge weights in $\{1,\ldots,M\}$. Let $\varepsilon > 0$. One can compute a $(5+\varepsilon)$-roundtrip spanner on $\tilde{O}(n^{1.5}\log^2 M/\varepsilon^2)$ edges in $\tilde{O}(m\sqrt{n}\log^2(M)/\varepsilon^2)$ time, whp. In $\tilde{O}(m\sqrt{n}\log(M)/\varepsilon)$ time, whp, one can compute a $(2+\varepsilon)$-approximation to the girth.*

We will start with a sampling approach, similar to that in the unweighted girth approximation. The pseudocode of the girth algorithm can be found in the full version [10].

▶ **Lemma 13.** *Let $G = (V,E)$ be a directed graph with $|V| = n$ and integer edge weights in $\{1,\ldots,M\}$. Let d be a positive integer, $\varepsilon \geq 0$, and let $Q \subseteq V$ be a random sample of $100\sqrt{n}\log n$ vertices. In $\tilde{O}(m\sqrt{n})$ time we can compute shortest paths trees $T^{in}(q), T^{out}(q)$ into and out of each $q \in Q$. Let H be the subgraph of G consisting of the edges of these trees $T^{in}(q), T^{out}(q)$. Let $V' = \{v \in V \mid \exists q \in Q,\ d(v,q) \leq d$ and $d(q,v) \leq d\}$. Then:*

- ***Girth approximation:*** *If $V' \neq \emptyset$, then the girth of G is at most $2d$.*
- ***Additive distance approximation:*** *For any $u,v \in V$, if the shortest u to v path contains a node of V', then $d_H(u,v) \leq d(u,v) + 2d$.*
- ***Sparsity:*** *The number of edges in H is $\tilde{O}(n^{1.5})$.*

Proof. Given a directed $G = (V,E)$ with $|V| = n$, $|E| = m$ and edge weights in $\{1,\ldots,M\}$, let us first take a random sample $Q \subseteq V$ of $100\sqrt{n}\log n$ vertices. Run Dijkstra's algorithm from and to every $q \in Q$. Determine $V' \subseteq V$ defined as those $v \in V$ for which there is some

$q \in Q$ with $d(v,q), d(q,v) \leq d$. If $V' \neq \emptyset$, we get that the girth of G is at most $2d$. Suppose that we insert all edges of the in- and out- shortest paths trees rooted at all $q \in Q$ into a subgraph H. Then we have only inserted $\tilde{O}(n^{1.5})$ edges as each tree has $\leq n-1$ edges.

Consider some $u, v \in V$ such that there is some node $x \in V'$ on the shortest $u-v$ path. Let $q \in Q$ be such that $d(x,q), d(q,x) \leq d$. Then

$$d_H(u,v) \leq d(u,q) + d(q,v) \leq d(u,x) + d(x,q) + d(q,x) + d(x,v) \leq d(u,v) + 2d. \qquad \blacktriangleleft$$

Our approach below will handle the roundtrip spanner and the girth approximation at the same time.

We will try all choices of integers i from 0 to $\log_{1+\varepsilon}(Mn)$ to estimate roundtrip distances in the interval $[(1+\varepsilon)^i, (1+\varepsilon)^{i+1})$, and to estimate the girth if it is $< (1+\varepsilon)^{i+1}$.

Fix a choice for i for now.

Our algorithm first applies the approach of Lemma 13 by setting $d = (1+\varepsilon)^{i+2}$ (we will see later why). We compute a random sample Q of size $O(\sqrt{n}\log n)$, do Dijkstra's from and to all nodes in Q, and add the edges of the computed shortest paths trees to our roundtrip spanner H. We also compute

$$V_i' = \{v \in V \mid \exists q \in Q: \ d(v,q) \leq (1+\varepsilon)^{i+2} \text{ and } d(q,v) \leq (1+\varepsilon)^{i+2}\}.$$

By Lemma 13, if $V_i' \neq \emptyset$, the girth of G must be $\leq 2(1+\varepsilon)^{i+2}$. For the choice of i where $(1+\varepsilon)^i \leq g \leq (1+\varepsilon)^{i+1}$, we will get an approximation factor of $2(1+\varepsilon)^2 \leq 2(1+3\varepsilon)$. Just as with the algorithm for unweighted graphs, we can pick the minimum i so that $V_i' \neq \emptyset$, use $2(1+\varepsilon)^{i+2}$ as one of our girth estimates and then proceed from now on with a single value $i-1$ considering only the interval $[(1+\varepsilon)^{i-1}, (1+\varepsilon)^i)$.

By Lemma 13, we also get that for any $u, v \in V$ for which the u-v shortest path contains a node of V_i', H gives a good additive estimate of $d(u,v)$, i.e. $d(u,v) \leq d_H(u,v) \leq d(u,v) + 2(1+\varepsilon)^{i+2}$.

Suppose that also $(1+\varepsilon)^i \leq d(u \leftrightarrows v) \leq (1+\varepsilon)^{i+1}$, and that we somehow also get a good estimate for $d(v,u)$ (either because the v-u shortest path contains a node of V', or by adding more edges to H), so that also $d(v,u) \leq d_H(v,u) \leq d(v,u) + 2(1+\varepsilon)^{i+2}$. Then,

$$d(u \leftrightarrows v) \leq d_H(u \leftrightarrows v) \leq d(u \leftrightarrows v) + 4(1+\varepsilon)^{i+2} \leq d(u \leftrightarrows v)(1 + 4(1+3\varepsilon)) \leq d(u \leftrightarrows v)(5+12\varepsilon).$$

In other words, we would get a $5 + O(\varepsilon)$-roundtrip spanner, as long as by adding $\tilde{O}(n^{1.5})$ edges to H, we can get a good additive approximation to the weights of the u-v shortest paths that do not contain nodes of V_i', for all u, v with $(1+\varepsilon)^i \leq d(u \leftrightarrows v) \leq (1+\varepsilon)^{i+1}$. We will in fact compute these shortest paths exactly. For the girth g itself, we will show how to compute it exactly, if no node of V_i' hit the shortest cycle, where i is such that $(1+\varepsilon)^{i-1} \leq g \leq (1+\varepsilon)^i$.

Fix i. Let $Z_i = V \setminus V_i'$ and $d = (1+\varepsilon)^{i+1}$. We can focus on the subgraph induced by Z_i.

Consider any $u \in Z_i$, and $j \leq i$. Define $Z_i^j(u) = Z_i \cap B^j(u)$ and $\bar{Z}_i^j(u) = Z_i \cap \bar{B}^j(u)$. We also add the boundary case $Z_i^{\emptyset} = Z_i \cap B^{\emptyset}(u) = \{x \in Z_i \mid d(u,x) = 0\}$.

If for all $j \in \{\emptyset\} \cup \{1, \ldots, i\}$, $|Z_i^j(u)| \leq 100\sqrt{n}$, running Dijkstra's algorithm from u in the graph induced by Z_i, up to distance $(1+\varepsilon)^{i+1}$ would be cheap. Unfortunately, however, some $Z_i^j(u)$ balls can be larger than $100\sqrt{n}$. In this case, similarly to our approach for the unweighted case, we will replace $Z_i^j(u)$ with a set $Z_i'^j(u) \subseteq \bar{Z}_i^j(u)$ of size $O(\sqrt{n})$ with the guarantee that for any $v \in V$ with $(1+\varepsilon)^i \leq d(u \leftrightarrows v) < (1+\varepsilon)^{i+1}$ for which the shortest u-v path does not contain a node of V_i', every node of this u-v shortest path that is in $Z_i^j(u)$ is also in $Z_i'^j(u)$.

The following lemma shows how to use such replacement sets.

▶ **Lemma 14.** *Let u and i be fixed. Suppose that for every $j \in \{\emptyset\} \cup \{1, \dots, i\}$ we are given black box access to sets $Z'^j_i(u) \subseteq \bar{Z}^j_i(u)$ of nodes such that (1) Checking whether a node z is in $Z'^j_i(u)$ takes $t(n)$ time, (2) $|Z'^j_i(u)| \le 100\sqrt{n}$ whp, and (3) for any v such that $(1+\varepsilon)^i \le d(u \leftrightarrows v) \le (1+\varepsilon)^{i+1}$, and every $j \le i$, every node on the shortest path P from u to v that is in $Z^j_i(u)$ is also in $Z'^j_i(u)$.*

Then there is an $\tilde{O}(m \log(M) t(n)/(\varepsilon\sqrt{n}))$ time algorithm that finds a shortest path from u to any v with $(1+\varepsilon)^i \le d(u \leftrightarrows v) \le (1+\varepsilon)^{i+1}$ and s.t. the shortest u-v path does not contain a node of V'_i. The algorithm returns $\tilde{O}(n^{0.5}\log(M)/\varepsilon)$ edges whose union contains all these shortest paths.

Proof. Assume we have the sets $Z'^j_i(u)$ for $j \in \{\emptyset\} \cup \{1, \dots, i\}$ as in the statement of the lemma.

Then we will define a modified Dijkstra's algorithm out of u. The algorithm begins by placing u in the Fibonacci heap with $d[u] = 0$ and all other vertices with $d[\cdot] = \infty$. When a vertex x is extracted from the heap with estimate $d[x]$, we determine the j for which $(1+\varepsilon)^j \le d[x] < (1+\varepsilon)^{j+1}$; here j could be the boundary case that we called $\emptyset$ if $d[x] = 0$. Then we check whether x is in Z'^j_i. If it is not, we ignore it and extract a new vertex from the heap. Otherwise if $x \in Z'^j_i$, we go through all its out-edges (x, y), and if $d[y] > d[x] + w(x, y)$, we update $d[y] = d[x] + w(x, y)$. For any new cycle to u found, we update the best weight found, and in the end we return it.

Since we only go through the edges of at most $O(\sqrt{n}\log(Mn)/\varepsilon)$ vertices and the degrees are all $O(m/n)$, the runtime is $O(m\log(Mn)/(\varepsilon\sqrt{n}))$. For the same reason, the modified shortest paths tree whose edges we add to our roundtrip spanner has at most $O(\sqrt{n}\log(Mn)/\varepsilon)$ edges.

Let v be such that $(1+\varepsilon)^i \le d(u \leftrightarrows v) \le (1+\varepsilon)^{i+1}$ and for which the shortest u-v path does not contain a node of V'_i. We will show by induction that our modified Dijkstra's algorithm will compute the shortest path from u to v exactly.

The induction will be on the distance from u. Let's call the nodes on the shortest u to v path, $u = u_0, u_1, \dots, u_t = v$. The induction hypothesis for u_k is that u_k is extracted from the heap with $d[u_k] = d(u, u_k)$. Let us show that u_{k+1} will also be extracted from the heap with $d[u_{k+1}] = d(u, u_{k+1})$. The base case is clear since u is extracted first.

When u_k is extracted from the heap, by the induction hypothesis, $d[u_k] = d(u, u_k)$. Let j be such that $(1+\varepsilon)^j \le d[u_k] < (1+\varepsilon)^{j+1}$. As no node on the u-v shortest path is in V'_i, we get that $u_k \in Z^j_i$. By the assumptions in the lemma, we also have that $u_k \in Z'^j_i$. Thus, when u_k is extracted, we will go over its edges. In particular, (u_k, u_{k+1}) will be scanned, and $d[u_{k+1}]$ will be set to (or it already was) $d[u_k] + w(u_k, u_{k+1}) = d(u, u_{k+1})$. This completes the induction.

It is also not hard to see that the girth will be computed exactly if u is on a shortest cycle, the girth is in $[(1+\varepsilon)^i, (1+\varepsilon)^{i+1})$ and V'_i is empty. ◀

Now we compute the sets $Z'^j_i(u)$. First consider $u, v \in V$ with $(1+\varepsilon)^i \le d(u \leftrightarrows v) < (1+\varepsilon)^{i+1}$. Let x be any node on the u to v roundtrip path (cycle) so that $x \in Z^j_i(u)$ for some integer $j \le i$. Recall that this means $(1+\varepsilon)^j \le d(u, x) < (1+\varepsilon)^{j+1}$. Then for every y with $d(u, y) < (1+\varepsilon)^{j+1}$ and so for each $y \in \bar{Z}^j_i(u)$ we must have (see Figure 2) that

$$d(x, y) \le d(x, u) + d(u, y) \le d(u \leftrightarrows v) - d(u, x) + d(u, y) \le d(u \leftrightarrows v) - (1+\varepsilon)^j + (1+\varepsilon)^{j+1}$$

$$= d(u \leftrightarrows v) + \varepsilon(1+\varepsilon)^j \le d(u \leftrightarrows v) + \varepsilon(1+\varepsilon)^i \le d(u \leftrightarrows v)(1+\varepsilon) \le (1+\varepsilon)^{i+2}.$$

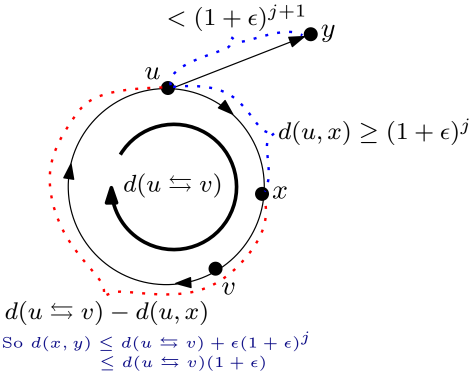

$< (1+\epsilon)^{j+1}$
y
u
$d(u,x) \geq (1+\epsilon)^j$
$d(u \leftrightarrows v)$
x
v
$d(u \leftrightarrows v) - d(u,x)$
So $d(x,y) \leq d(u \leftrightarrows v) + \epsilon(1+\epsilon)^j$
$\leq d(u \leftrightarrows v)(1+\epsilon)$

Figure 2 Here u and v have roundtrip distance more than $(1+\varepsilon)^j$. A node x on the shortest u-v path is at distance at least $(1+\varepsilon)^j$ from u, and another node y is at distance at most $(1+\varepsilon)^{j+1}$ from u. Then the distance from x to y is at most $d(u \leftrightarrows v)(1+\varepsilon)$ since one way to go from x to y is to go from x to u along the u-v roundtrip cycle at a cost of at most $d(u \leftrightarrows v) - (1+\varepsilon)^j$, and then from u to y at a cost of at most $(1+\varepsilon)^{j+1}$.

In other words, x must be in $\{w \in \bar{Z}_i^j(u) \mid d(w,y) \leq (1+\varepsilon)^{i+2},\ \forall y \in \bar{Z}_i^j(u)\}$.

We apply Lemma 11 for $\beta = (1+\epsilon)$ and $\alpha = 1/2$. It outputs sets $R_i^j(u)$ of size $O(\log^2 n)$ vertices, where the number of vertices in $\bar{Z}_i^j(u)$ that are at distance $(1+\epsilon)^{i+2}$ from all vertices in $R_i^j(u)$ is $O(\sqrt{n})$. So all vertices $x \in Z_i^j(u)$ that are in a roundtrip path $u - v$ with $(1+\varepsilon)^i \leq d(u \leftrightarrows v) < (1+\varepsilon)^{i+1}$ are in this set, so we let $Z_i'^j(u) = \{w \in \bar{Z}_i^j(u) | d(w,y) \leq (1+\epsilon)^{i+2},\ \forall y \in R_i^j(u)\}$.

Now that we have the random samples, we implement the modified Dijkstra's algorithm from Lemma 14 with only a polylogarithmic overhead as follows:

Fix some j. Let's look at the vertices x with $(1+\varepsilon)^j \leq d[x] < (1+\varepsilon)^{j+1}$ that the modified Dijkstra's algorithm extracts from the heap. Since $d[x]$ is always an overestimate, $d(u,x) \leq d[x] < (1+\varepsilon)^{j+1}$, and so $x \in \bar{B}^j(u)$. Now, since x is already in $\bar{B}^j(u)$, to check whether $x \in Z_i'^j(u)$, we only need to check whether $x \in Z_i$ (easy) and whether $d(x,y) \leq (1+\varepsilon)^{i+2}$ for all $y \in R_i^j(u)$ (this takes $O(\log^2 n)$ time since we have all the distances to the nodes in the random samples).

The final running time is $\tilde{O}(m\sqrt{n}\log^2(M)/\varepsilon^2)$ since we need to run the above procedure $O(\log(Mn)/\varepsilon)$ times, once for each i, and each procedure costs $\tilde{O}(m\log(M)\sqrt{n}/\varepsilon)$ time. As we mentioned before, to estimate the girth to within a $(2+\varepsilon)$-factor, we do not need to run the procedure for all i but (as with the algorithm for unweighted graphs), only for the minimum i for which $V'_{i+1} \neq \emptyset$. Thus the running time for the girth becomes $\tilde{O}(m\sqrt{n}\log(M)/\varepsilon)$.

4.1 $(4+\epsilon)$-Approximation Algorithm for the Girth in $\tilde{O}(mn^{\sqrt{2}-1})$ Time

In this section we are going to prove the modified version of Theorem 3, where a $\log M$ factor is added to the running time with M being the maximum edge weight.

▶ **Theorem 15.** *For every $\varepsilon > 0$, there is a $(4+\varepsilon)$-approximation algorithm for the girth in directed graphs with edge weights in $\{1,\ldots,M\}$ that runs in $\tilde{O}(mn^{\sqrt{2}-1}\log(M)/\varepsilon)$ time.*

Proof. Suppose that we want an $\tilde{O}(mn^{\alpha})$ time girth approximation algorithm. Let $\beta = 2(1+\epsilon)$. As a first step, we sample a set Q of $\tilde{O}(n^{\alpha})$ vertices and do in and out Dijkstra from them.

We let $V_i' = \{v \in V \mid \exists q \in Q : \ d(v,q) \leq \beta(1+\varepsilon)^{i+1} \text{ and } d(q,v) \leq \beta(1+\varepsilon)^{i+1}\}$. If $V_i' \neq \emptyset$ for some i, then we have that the girth g is at most $2\beta(1+\epsilon)^{i+1}$. If $(1+\epsilon)^i \leq g \leq (1+\epsilon)^{i+1}$, this is a $2\beta(1+\epsilon) \leq 4(1+3\epsilon) = 4 + O(\epsilon)$ approximation.

So take the minimum i where $V_{i+1}' \neq \emptyset$. Let $g' = (1+\epsilon)^{i+1}$ be our current upper bound for the girth g. We initially mark all vertices “on”, meaning that they are not processed yet. For each on vertex u, we either find the smallest cycle of length at most g' passing through u where all vertices of the cycle are on, or conclude that there is no cycle of length at most g' passing through u. When a vertex u is processed, we mark it as “off”. We proceed until all vertices are off.

We apply Lemma 11 for $\beta = 2(1+\epsilon)$. Note that since $V_i' = \emptyset$, $Z_i^j(u) = B^j(u)$ is all the vertices at distance $[(1+\epsilon)^j, (1+\epsilon)^{j+1})$ from u. The lemma outputs sets $R_i^j(u) \subseteq Z_i^j(u)$, where $|R_i^j(u)| = O(\log^2 n)$ and the number of vertices in $B^j(u)$ at distance $\beta g'$ from $R_i^j(u)$ is at most $O(n^{1-\alpha})$ whp. Fix some on vertex u. We do modified Dijkstra from u up to vertices with distance at most $g'/2$ from u as follows:

We begin by placing u in the Fibonacci heap with $d[u] = 0$ and all other on vertices with $d[\cdot] = \infty$. When a vertex x is extracted from the heap with estimate $d[x]$, we determine the j for which $(1+\varepsilon)^j \leq d[x] < (1+\varepsilon)^{j+1}$; here j could be the boundary case that we called $\emptyset$ if $d[x] = 0$. Then we check whether $d(x,r) \leq g' - (1+\epsilon)^j + (1+\epsilon)^{j'+1}$ for all $r \in R_i^{j'}(u)$ for all j'. If x does not satisfy this condition, we ignore it and extract a new vertex from the heap. Otherwise, we go through all its out-edges (x,y), and if $d[y] > d[x] + w(x,y)$, we update $d[y] = d[x] + w(x,y)$. We stop when the vertex u extracted from the heap has $d[u] > g'/2$.

Let $S_i(u)$ be the set of all the vertices visited in the modified out-Dijkstra. Simillarly, let $T_i(u)$ be all the vertices visited in the analogous modified in-Dijkstra (using an analogous version of Lemma 11).

Suppose that there is a vertex v with $d(u \leftrightarrows v) \leq g'$, where all vertices in the uv cycle C are on. Without loss of generality, suppose that $d_C(u,v) \leq g'/2$. So $d(u,v) \leq g'/2$. Moreover, suppose that $v \in Z_i^j(u)$, i.e. $(1+\epsilon)^j \leq d(u,v) \leq (1+\epsilon)^{j+1}$. So for any vertex $w \in Z_i^{j'}(u)$ for some j' we have that $d(v,w) \leq d(v,u) + d(u,w) \leq g' - (1+\epsilon)^j + (1+\epsilon)^{j'+1}$. Since all vertices on the uv path that is part of the cycle are on and the length of this path is at most $g'/2$, we visit v in the out-Dijkstra, i.e. $v \in S_i(u)$. Similarly, if $d(v,u) \leq g'/2$, we visit v in the in-Dijkstra and so $v \in T_i(u)$.

If both $S_i(u)$ and $T_i(u)$ have size at most n^α, we do Dijkstra from u in the induced subgraph on $S_i(u) \cup T_i(u)$, and see if there is a cycle of length at most g' passing through u (and find the smallest such cycle), which takes $O(\frac{m}{n} n^\alpha)$ time. We take the length of this cycle as one of our estimates. The modified in and out Dijkstras take $O(\log^2 n . \frac{\log nM}{\varepsilon} . n^\alpha . \frac{m}{n})$, as checking the conditions for each x extracted from the heap takes $O(\log^2 n . \frac{\log nM}{\varepsilon})$ time. So in $\tilde{O}(\frac{\log M}{\varepsilon} n^\alpha . \frac{m}{n})$ time we process u and mark it as ”off and proceed to another vertex.

Suppose $S_i(u)$ has size bigger than n^α (the case where $T_i(u)$ has size bigger than n^α is similar). Note that by Lemma 11 we have $|S_i(u)| \leq O(n^{1-\alpha})$ because for each $r \in R_i^j(u)$, we have that $d(x,r) \leq g' - (1+\varepsilon)^j + (1+\varepsilon)^{j+1} \leq g' + \varepsilon(1+\varepsilon)^j \leq g' + \varepsilon g'/2 \leq \beta g'$. So it is a subset of vertices that are at distance at most $\beta g'$ from all samples in R_i^j for all j. Our new goal is the following: We want to either find the smallest cycle of length at most g' passing through $S_i(u)$ that contains no off vertices, or say that there is no cycle of length $\leq g'$ passing through any of the vertices in $S_i(u)$ whp.

For this, we do another Modified Dijkstra from u as follows:

We begin by placing u in the Fibonacci heap with $d[u] = 0$ and all other on vertices with $d[\cdot] = \infty$. When a vertex x is extracted from the heap with estimate $d[x]$, we determine the j for which $(1+\varepsilon)^j \leq d[x] < (1+\varepsilon)^{j+1}$; here j could be the boundary case that we called

$\emptyset$ if $d[x] = 0$. Then we check whether $d(x, r) \leq \beta g' = 2(1+\varepsilon)g'$ for all $r \in R_i^j(u)$. If it is not, we ignore it and extract a new vertex from the heap. Otherwise, we go through all its out-edges (x, y), and if $d[y] > d[x] + w(x, y)$, we update $d[y] = d[x] + w(x, y)$. We stop when the vertex u extracted from the heap has $d[u] > 3g'/2$.

We show that if there is a cycle of length at most g' going through $v \in S_i(u)$ containing to off vertex, all vertices of the cycle are among the vertices we visit in the modified Dijkstra: Suppose that $d(w \leftrightarrows v) \leq g'$, and suppose that $v \in Z_i^j(u)$ and $w \in Z_i^{j'}(u)$. Then for every $r \in R_i^{j'}(u)$, we have that $d(w, r) \leq d(w, v) + d(v, r) \leq g' - d(v, w) + g' - (1+\varepsilon)^j + (1+\varepsilon)^{j'+1}$. Since $d(v, w) \geq (1+\varepsilon)^{j'} - (1+\varepsilon)^{j+1}$, we have $d(w, r) \leq 2g' + \varepsilon(1+\varepsilon)^{j'} + \varepsilon(1+\varepsilon)^j \leq 2g' + 3\varepsilon g'/2 + \varepsilon g'/2 = \beta g'$. Since the uw path that goes through v is a path of length at most $\beta g'$ that has no off vertices, we visit w in the modified Dijkstra.

By Lemma 11 the total number of vertices visited in the modified Dijkstra is at most $O(n^{1-\alpha})$. Let the subgraph on these vertices be G'. We recurse on G', and find a $4 + O(\varepsilon)$ approximation of the girth in G'. The girth in G' is a lower bound on the minimum cycle of length $\leq g'$ passing through any vertex in $S_i(u)$ that has no off vertex. We take this value as one of our estimates. So we have processed all vertices in $S_i(u)$ and we mark them off. This takes $O(\frac{m}{n}.\frac{\log M}{\varepsilon}.((n^{1-\alpha})^{1+\alpha}))$, and we have marked off at least n^α vertices. So we spend $O(\frac{m}{n}.\frac{\log M}{\varepsilon}.n^{1-\alpha^2-\alpha})$ for processing each vertex. Letting $1 - \alpha^2 - \alpha = \alpha$, we have that $\alpha = \sqrt{2} - 1$. So the total running time is $\tilde{O}(mn^{\sqrt{2}-1} \log(M)/\varepsilon)$. Our final estimate of the girth is the minimum of all the estimates we get through processing vertices. ◀

4.2 $(2k + \epsilon)$-Approximation Algorithm For the Girth

In this section we are going to prove a modified version of Theorem 4, where a $\log M$ factor is added to the running time with M being the maximum edge weight. The proof is a generalization of the proof of Theorem 15.

▶ **Theorem 16.** *For every $\varepsilon > 0$ and integer $k \geq 1$, there is a $(2k+\varepsilon)$-approximation algorithm for the girth in directed graphs with edge weights in $\{1, \ldots, M\}$ that runs in $\tilde{O}(mn^{\alpha_k} \log(M)/\varepsilon)$ time, where $\alpha_k > 0$ is the solution to $\alpha_k(1+\alpha_k)^{k-1} = 1 - \alpha_k$.*

Suppose that we are aiming for a $2k(1 + O(\epsilon))$ approximation algorithm for the girth, in $\tilde{O}(mn^\alpha \log M/\varepsilon)$ time, where we set α later. So basically we want to spend $\tilde{O}(\frac{m}{n}\frac{\log M}{\varepsilon}n^\alpha)$ per vertex. Let $\beta = k + k^2\epsilon + k\epsilon = k + O(\varepsilon)$. As before, first we sample a set Q of $\tilde{O}(n^\alpha)$ and do in and out Dijkstra from each vertex $q \in Q$. Let i_{min} be the minimum number i such that the set $V_i' = \{v \in V \mid \exists q \in Q : \ d(v, q) \leq \beta(1+\varepsilon)^{i+1}$ and $d(q, v) \leq \beta(1+\varepsilon)^{i+}\}$ is non-empty. So our initial estimate of the girth is $2\beta(1+\epsilon)^{i_{min}+1}$.

Let $i = i_{min} - 1$ and let $g' = (1+\varepsilon)^{i+1}$ be our estimate of the girth. Initially we mark all vertices as “on”, and as we process each vertex, we either find a smallest cycle of length at most g' with no “off” vertex, or we say that there is no cycle of length at most g' passing through it whp, and we mark the vertex as off.

We apply Lemma 11 for $\beta = k + k^2\epsilon + k\epsilon$ and the set Q as input. It gives us the sets $R_i^j(u)$ of size $O(\log^2 n)$ for all j, such that the number of vertices in $\bar{B}^j(u) = \{w \in V | d(u, w) \leq (1+\epsilon)^{j+1}\}$ that are at distance at most $\beta g'$ from all $r \in R_i^j(u)$ is at most $O(n^{1-\alpha})$ whp.

We take an on vertex u and do “modified” Dijkstra from (to) u, stopping at distance $g'/2$, such that the set of vertices we visit contains any cycle of length g' that passes through u that has no off vertex. We explain this modified Dijkstra later.

We call the set of vertices that we visit in the modified out-Dijkstra $S_i^1(u)$. If $S_i^1(u) \leq n^\alpha$, we do an analogous modified in-Dijkstra from u, and let $T_i^1(u)$ be the set of vertices visited in this in-Dijkstra. If $T_i^1(u) \leq n^\alpha$, then we do Dijkstra from u in the subgraph induced by

$S_i^1(u) \cup T_i^1(u)$, and hence find a smallest cycle of length $\le g'$ that passes through u with no off vertex. We take the length of this cycle as one of our estimates for the girth. If there is no such cycle, we don't have any estimate from u. Now we mark u as off and proceed the algorithm by taking another on vertex. Our modified Dijkstras takes $O(\log^2 n.\frac{\log Mn}{\varepsilon}.\frac{m}{n}|S|)$ time if S is the set of vertices visited by the Dijkstra. Hence for processing u we spend $O(\log^2 n.\frac{\log Mn}{\varepsilon}.\frac{m}{n}n^{\alpha})$ time.

So suppose that either $S_i^1(u)$ or $T_i^1(u)$ have size bigger than n^{α}. Without loss of generality assume that $|S_i^1(u)| \ge n^{\alpha}$ (the other case is analogous). For $1 \le l \le k$, define sets $S_i^l(u)$ as the set of on vertices $w \in V$ such that there is a path of length at most $(2l-1)g'/2$ from u to w that contains no off vertex, and if $w \in B^j(u)$, then for all $r \in R_i^{j'}(u)$ for all j', we have $d(w,r) \le (l + l^2\varepsilon)g' + (1+\epsilon)^{j'+1} - (1+\epsilon)^j$. Once we explain our modified Dijkstras, it will be clear that S_i^1 defined here is indeed the set of vertices visited in the first modified out-Dijkstra.

We set $S_i^0(u) = \{u\}$. We prove the following useful lemma in the full version [10].

▶ **Lemma 17.** *For all $l \in \{1, \dots, k\}$, we have that $S_i^{l-1}(u) \subseteq S_i^l(u)$. Moreover, if $w \in V$ is in a cycle of length at most g' with some vertex in $S_i^{l-1}(u)$ such that the cycle contains no off vertex, then we have $w \in S_i^l(u)$.*

Our algorithm will do at most k modified Dijkstras from u, where we prove that the set of vertices visited in the lth Dijkstra is $S_i^l(u)$. After performing each Dijkstra we decide if we continue to the next modified Dijkstra from u or proceed to another on vertex.

Suppose that at some point we know that the set $S_i^{l-1}(u)$ is the set of vertices visited in the $(l-1)$th modified Dijkstra, and we want to proceed to the lth Dijkstra. Our new goal is the following: We want to catch a minimum cycle of length $\le g'$ passing through S_i^l with no off vertex. For this, we do the lth modified Dijkstra form u as follows.

We begin by placing u in the Fibonacci heap with $d[u] = 0$ and all other on vertices with $d[\cdot] = \infty$. When a vertex x is extracted from the heap with estimate $d[x]$, we determine the j for which $(1+\varepsilon)^j \le d[x] < (1+\varepsilon)^{j+1}$; here j could be the boundary case that we called $\emptyset$ if $d[x] = 0$. Then we check whether $d(x,r) \le (l + l^2\varepsilon)g' - (1+\epsilon)^j + (1+\epsilon)^{j'+1}$ for all $r \in R_i^{j'}(u)$ for all j'. If x does not satisfy this condition, we ignore it and extract a new vertex from the heap. Otherwise, we go through all its out-edges (x,y), and if $d[y] > d[x] + w(x,y)$, we update $d[y] = d[x] + w(x,y)$. We stop when the vertex u extracted from the heap has $d[u] > (2l-1)g'/2$.

It is clear by definition that the set of vertices that this modified Dijkstra visits is $S_i^l(u)$. Now if $|S_i^l(u)| \le c(|S_i^{l-1}(u)|.n^{\alpha})^{\frac{1}{1+\alpha}}$ for some constant c, we recurse on the subgraph induced by $S_i^l(u)$, i.e. $G[S_i^l(u)]$, to get an $2k + O(\varepsilon)$ approximation of the girth on this subgraph. The girth in $G[S_i^l(u)]$ is a lower bound on the minimum cycle of length $\le g'$ passing through $S_i^{l-1}(u)$ with no off vertex. So we take this value as one of our estimates and we mark all vertices of $S_i^{l-1}(u)$ as off. The running time of this recursion is $\tilde{O}(\frac{m}{n}\frac{\log M}{\varepsilon}|S_i^l(u)|^{1+\alpha})$ as the average degree is $O(\frac{m}{n})$. Since we process $S_i^{l-1}(u)$ vertices in this running time, we spend $\tilde{O}(\frac{m}{n}\frac{\log M}{\varepsilon}.|S_i^l(u)|/|S_i^{l-1}(u)|) \le \tilde{O}(\frac{m}{n}.\frac{\log M}{\varepsilon}.n^{\alpha})$ for each vertex.

Note that $|S_i^k(u)| \le O(n^{1-\alpha})$. This is because for all $x \in S_i^k \cap B^j(u)$ and for all $r \in R_i^j(u)$, we have that $d(x,r) \le (k+k^2\varepsilon)g' + (1+\epsilon)^{j+1} - (1+\epsilon)^j \le (k+k^2\varepsilon)g' + \epsilon(1+\epsilon)^j \le (k+k^2\varepsilon)g' + \epsilon(2k-1)g'/2 \le (k+k^2\epsilon+k\varepsilon)g' = \beta g'$. So $S_i^k(u)$ is a subset of all vertices in $B^j(u)$ with distance at most $\beta g'$ from all $r \in R_i^j(u)$, and so by Lemma 11 it has size at most $O(n^{1-\alpha})$.

When all vertices are marked off, we take the minimum value of all the estimates as our estimate for g.

Since we have that $S_i^l(u) \leq O(n^{1-\alpha})$, if we set α appropriately, for some $l < k$ we have that $|S_i^{l+1}(u)| \leq (|S_i^l(u)|.n^{\alpha})^{\frac{1}{1+\alpha}}$. For $k = 1$, setting $\alpha = 1/2$ gives us the algorithm of Theorem 12. For $k > 1$, the following lemma determines α. The proof of the lemma can be found in the full version [10].

▶ **Lemma 18.** *For $k > 1$, let the sets S_i^l for $l = 1, \dots, k$ be such that $S_i^l \subseteq S_i^{l+1}$ for all $l < k$, $S_i^1 \geq n^{\alpha}$ and $S_i^k \leq O(n^{1-\alpha})$. Let $0 < \alpha < 1$ satisfy $\alpha(1+\alpha)^{k-1} = 1 - \alpha$. Then there is $l < k$ and a constant c such that $|S_i^{l+1}| \leq c(|S_i^l|.n^{\alpha})^{\frac{1}{1+\alpha}}$.*

Note that for $k = 2$, Lemma 18 sets $\alpha = \sqrt{2} - 1$ and thus gives us the algorithm of Theorem 15.

5 Hardness

Our hardness result is based on the following k-Cycle hypothesis (see [14, 3, 16]).

▶ **Hypothesis 1** (k-Cycle Hypothesis)**.** *In the word-RAM model with $O(\log m)$ bit words, for any constant $\varepsilon > 0$, there exists a constant integer k, so that there is no $O(m^{2-\varepsilon})$ time algorithm that can detect a k-cycle in an m-edge graph.*

All known algorithms for detecting k-cycles in directed graphs with m edges run at best in time $m^{2-c/k}$ for various small constants c [23, 2, 14, 9], even using powerful tools such as fast matrix multiplication. Refuting the k-Cycle Hypothesis above would resolve a big open problem in graph algorithms. Moreover, as shown by Lincoln et al. [14] any algorithm for directed k-cycle detection, for k-odd, with running time $O(mn^{1-\varepsilon})$ for $\varepsilon > 0$ whenever $m = \Theta(n^{1+2/(k-1)})$ would imply an $O(n^{k-\delta})$ time algorithm for k-clique detection for $\delta > 0$. If the cycle algorithm is "combinatorial", then the clique algorithm would be "combinatorial" as well, and since all known $O(n^{k-\delta})$ time k-clique algorithms use fast matrix multiplication, such a result for k-cycle would be substantial.

We will show that under Hypothesis 1, approximating the girth to a factor better than 2 would require $mn^{1-o(1)}$ time, and so up to this hypothesis, our approximation algorithm is optimal for the girth in unweighted graphs.

▶ **Theorem 19.** *Suppose that for some constants $\varepsilon > 0$ and $\delta > 0$, there is an $O(m^{2-\varepsilon})$ time algorithm that can compute a $(2-\delta)$-approximation of the girth in an m-edge directed graph. Then for every constant k, one can detect whether an m-edge directed graph contains a k-cycle, in $O(m^{2-\varepsilon})$ time, and hence the k-Cycle Hypothesis is false.*

Proof. The proof is relatively simple. Suppose that for some constants $\varepsilon > 0$ and $\delta > 0$, there is an $O(m^{2-\varepsilon})$ time algorithm that can compute a $(2-\delta)$-approximation of the girth in an m-edge directed graph.

Now let $k \geq 3$ be any constant integer and let G be an n-node, m-edge graph. First randomly color each vertex of G with one of k colors. Let C be any k-cycle in G. With probability $1/k^k$, for each $i = 0, \dots, k-1$, the ith vertex of C is colored i.

Now, for each $0 \leq i \leq k-1$, let V_i be the vertices colored i. For each vertex $u \in V_i$, and each directed edge (u, v) out of u, keep (u, v) if and only if $v \in V_{i+1}$ where the indices are taken mod k. This builds a graph G' which is a subgraph of G and contains a k-cycle if G does with probability $\geq 1/k^k$.

G' has two useful properties. (1) Any cycle of G' has length divisible by k, and (2) (which follows from (1)) the girth of G' is k if G' contains a k-cycle and it is $\geq 2k$ otherwise.

As G' has at most m edges (it is a subgraph of G), we can use our supposedly fast $2-\delta$ approximation algorithm to determine whether the girth is k or larger in $O(m^{2-\varepsilon})$ time. By iterating the construction $O(k^k \log n)$ times, we get that the k-cycle problem in G can be solved in $\tilde{O}(k^k m^{2-\varepsilon})$ time, and as k is a constant, we are done. The approach can be derandomized with standard techniques (e.g. [1]). ◀

References

1 N. Alon, R. Yuster, and U. Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995.

2 N. Alon, R. Yuster, and U. Zwick. Finding and counting given length cycles. *Algorithmica*, 17:209–223, 1997.

3 Bertie Ancona, Monika Henzinger, Liam Roditty, Virginia Vassilevska Williams, and Nicole Wein. Algorithms and hardness for diameter in dynamic graphs. In *Proceedings of ICALP*, page to appear, 2019.

4 A. Bondy and M. Simonovits. Cycles of even length in graphs. *Journal of Combinatorial Theory*, 16:97–105, 1974.

5 Ruoxu Cen and Ran Duan. Roundtrip spanners with $(2k-1)$ stretch. *CoRR*, abs/1911.12411, 2019. `arXiv:1911.12411`.

6 Shiri Chechik, Yang P. Liu, Omer Rotem, and Aaron Sidford. Improved girth approximation and roundtrip spanners. *CoRR*, abs/1907.10779, 2019. `arXiv:1907.10779`.

7 Shiri Chechik, Yang P. Liu, Omer Rotem, and Aaron Sidford. Improved girth approximation and roundtrip spanners. In *Proceedings of STOC*, page to appear, 2020.

8 Marek Cygan, Harold N. Gabow, and Piotr Sankowski. Algorithmic applications of baur-strassen's theorem: Shortest cycles, diameter, and matchings. *J. ACM*, 62(4):28:1–28:30, September 2015.

9 Mina Dalirrooyfard, Thuy Duong Vuong, and Virginia Vassilevska Williams. Graph pattern detection: Hardness for all induced patterns and faster non-induced cycles. In *Proceedings of STOC 2019*, page to appear, 2019.

10 Mina Dalirrooyfard and Virginia Vassilevska Williams. Conditionally optimal approximation algorithms for the girth of a directed graph. *arXiv preprint*, 2020. `arXiv:2004.11445`.

11 A.V. Goldberg. Scaling algorithms for the shortest paths problem. In *Proc. SODA*, pages 222–231, 1993.

12 A. Itai and M. Rodeh. Finding a minimum circuit in a graph. *SIAM J. Computing*, 7(4):413–423, 1978.

13 François Le Gall. Powers of tensors and fast matrix multiplication. In *International Symposium on Symbolic and Algebraic Computation, ISSAC '14, Kobe, Japan, July 23-25, 2014*, pages 296–303, 2014.

14 Andrea Lincoln, Virginia Vassilevska Williams, and R. Ryan Williams. Tight hardness for shortest cycles and paths in sparse graphs. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1236–1252, 2018.

15 Jakub Pachocki, Liam Roditty, Aaron Sidford, Roei Tov, and Virginia Vassilevska Williams. Approximating cycles in directed graphs: Fast algorithms for girth and roundtrip spanners. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1374–1392. SIAM, 2018.

16 Maximilian Probst, Virginia Vassilevska Williams, and Nicole Wein. New algorithms and hardness for incremental single-source shortest paths in directed graphs. In *Proceedings of STOC*, page to appear, 2020.

17 Liam Roditty and Virginia Vassilevska Williams. Minimum weight cycles and triangles: Equivalences and algorithms. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 180–189. IEEE Computer Society, 2011.

18 Liam Roditty and Virginia Vassilevska Williams. Subquadratic time approximation algorithms for the girth. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 833–845. SIAM, 2012.

19 R. Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. *JCSS*, 51:400–403, 1995.

20 V. Vassilevska. Efficient algorithms for path problems. *Ph.D. Thesis in Computer Science, Carnegie Mellon University*, 2008.

21 V. Vassilevska Williams and R. Williams. Subcubic equivalences between path, matrix and triangle problems. In *Proc. FOCS*, pages 645–654, 2010.

22 Virginia Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 887–898, 2012.

23 R. Yuster and U. Zwick. Detecting short directed cycles using rectangular matrix multiplication and dynamic programming. In *Proc. SODA*, pages 247–253, 2004.

24 U. Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *J. ACM*, 49(3):289–317, 2002.

Symmetric Arithmetic Circuits

Anuj Dawar

Department of Computer Science and Technology, University of Cambridge, UK
anuj.dawar@cl.cam.ac.uk

Gregory Wilsenach

Department of Computer Science and Technology, University of Cambridge, UK
gregory.wilsenach@cl.cam.ac.uk

Abstract

We introduce symmetric arithmetic circuits, i.e. arithmetic circuits with a natural symmetry restriction. In the context of circuits computing polynomials defined on a matrix of variables, such as the determinant or the permanent, the restriction amounts to requiring that the shape of the circuit is invariant under row and column permutations of the matrix. We establish unconditional, nearly exponential, lower bounds on the size of any symmetric circuit for computing the permanent over any field of characteristic other than 2. In contrast, we show that there are polynomial-size symmetric circuits for computing the determinant over fields of characteristic zero.

2012 ACM Subject Classification Theory of computation → Circuit complexity; Theory of computation → Algebraic complexity theory

Keywords and phrases arithmetic circuits, symmetric arithmetic circuits, Boolean circuits, symmetric circuits, permanent, determinant, counting width, Weisfeiler-Leman dimension, Cai-Fürer-Immerman constructions

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.36

Category Track A: Algorithms, Complexity and Games

Related Version A full version of the paper is available at `https://arxiv.org/abs/2002.06451`.

Funding Research funded in part by EPSRC grant EP/S03238X/1.

1 Introduction

Valiant's conjecture [23], that VP $\neq$ VNP, is often referred to as the algebraic counterpart to the conjecture that P $\neq$ NP. It has proved as elusive as the latter. The conjecture is equivalent to the statement that there is no polynomial-size family of arithmetic circuits for computing the permanent of a matrix, over any field of characteristic other than 2. Here, arithmetic circuits are circuits with input gates labelled by variables from some set X or constants from a fixed field $\mathbb{F}$, and internal gates labelled with the operations $+$ and $\times$. The output of such a circuit is some polynomial in $\mathbb{F}[X]$, and we think of the circuit as a compact representation of this polynomial. In particular, if the set of variables X form the entries of an $n \times n$ matrix, i.e. $X = \{x_{ij} \mid 1 \leq i, j \leq n\}$, then PERM_n denotes the polynomial $\sum_{\sigma \in \mathbf{Sym}_n} \prod x_{i\sigma(i)}$, which is the permanent of the matrix.

While a lower bound for the size of general arithmetic circuits computing the permanent remains out of reach, lower bounds have been established for some restricted classes of circuits. For example, it is known that there is no sub-exponential family of *monotone* circuits for the permanent [17, 18]. An exponential lower bound for the permanent is also known for *depth-3* arithmetic circuits [15] over finite fields. In both these cases, the exponential lower bound obtained for the permanent also applies to the determinant, i.e. the family of polynomials $\{\mathrm{DET}_n\}_{n \in \mathbb{N}}$, where DET_n is $\sum_{\sigma \in \mathbf{Sym}_n} \mathrm{sgn}(\sigma) \prod x_{i\sigma(i)}$. However, the determinant is in VP and so there do exist polynomial-size families of circuits for the determinant.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 36; pp. 36:1–36:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In this paper, we consider a new restriction on arithmetic circuits based on a natural notion of symmetry, and we show that it distingushes between the determinant and the permanent. That is to say, we are able to show nearly exponential lower bounds on the size of any family of symmetric arithmetic circuits for computing the permanent, while establishing the existence of polynimial-size symmetric circuits for computing the determinant. We prove the upper bound on the determinant for fields of characteristic zero, and conjecture that it holds for all fields. Our lower bound for the permanent is established for all fields of characteristic other than 2, which is the best that can be hoped for as the permanent and the determinant coincide for fields of characteristic 2.

We next define (informally) the notion of symmetry we use. A formal definition follows in Section 3. The permanent and the determinant are not symmetric polynomials in the usual meaning of the word, in that they are not invariant under arbitrary permutations of their variables. However, they do have natural symmetries, i.e. permutations of the variables induced by row and column permutations. Specifically, PERM_n is invariant under arbitrary permutations of the rows and columns of the matrix (x_{ij}), while DET_n is invariant under simultaneous permutations of the rows and columns. We say that an arithmetic circuit C (seen as a labelled directed acyclic graph) for computing DET_n is *symmetric* if the action of any permutation $\sigma \in \mathbf{Sym}(n)$ on its input variables (i.e. taking x_{ij} to $x_{\sigma(i)\sigma(j)}$) extends to an automorphism of C. Similarly, a circuit C for computing PERM_n is symmetric if the action of $(\sigma, \pi) \in \mathbf{Sym}(n) \times \mathbf{Sym}(n)$ on the inputs (taking x_{ij} to $x_{\sigma(i)\pi(j)}$) extends to an automorphism of C.

This notion of symmetry has been studied previously in the context of Boolean circuits for deciding graph properties, or properties of relational structures (see [13, 20, 2]). Specifically, such symmetric circuits arise naturally in the translation into circuit form of specifications of properties in a logic or similar high-level formalism. Similarly, we can think of a symmetric arithmetic circuit as a straight-line program which treats the rows and columns of a matrix as being indexed by unordered sets. Many natural algorithms have this property. For example, Ryser's formula for computing the permanent naturally yields a symmetric circuit.

Polynomial-size families of symmetric Boolean circuits with threshold gates form a particularly robust class, with links to fixed-point logics [2]. This allows us to deploy methods for proving inexpressiblity in such logics to prove lower bounds on the size of symmetric circuits. A close link has also been established between the power of such circuits and linear programming extended formulations with a geometric notion of symmetry [5]. Our lower bound for the permanent is established by first giving a symmetry-preserving translation of arithmetic circuits to Boolean circuits with threshold gates, and then establishing a lower bound there for computing the permanent of a 0-1-matrix.

The lower bounds for symmetric Boolean circuits are based on a measure we call the *counting width* of graph parameters (the term is introduced in [11]). This is also sometimes known as the Weisfeiler-Leman dimension. In short, we have, for each k an equivalence relation $\equiv^k$, known as the k-dimensional Weisfeiler-Leman equivalence, that is a coarse approximation of isomorphism, getting finer with increasing k. The counting width of a graph parameter μ is the smallest k, as a function of the graph size n, such that μ is constant on $\equiv^k$-classes of graphs of size n. From known results relating Boolean circuits and counting width [2, 5], we know that the existence of subexponential size symmetric circuits computing μ implies a sub-linear upper bound on its counting width. Hence, using the standard relationship between the permanent of a 0-1-matrix and the number of perfect matchings in a bipartite graph, we obtain our lower bound for the permanent in fields of characteristic zero by showing a linear lower bound on the counting width of $\mu(G)$ – the number of perfect matchings in G. Indeed, showing the same for $\mu(G) \pmod p$ for every prime $p > 2$ establishes the lower bound for the permanent in all odd positive characteristics.

The linear lower bound on the counting width of the number of perfect matchings is a result of interest in its own right, quite apart from the lower bounds it yields for circuits for the permanent. Indeed, there is an interest in determining the counting width of concrete graph parameters (see, for instance, [4]), and the result here is somewhat surprising. The decision problem of determining whether a graph has any perfect matching is known to have constant counting width. Indeed, the width is 2 for bipartite graphs [7]. For general graphs, it is known to be strictly greater than 2 but still bounded above by a constant [3].

Related Work. Landsberg and Ressayre [19] establish an exponential lower bound on the complexity of the permanent (specifically over the complex field $\mathbb{C}$) under an assumption of symmetry, and it is instructive to compare our results with theirs. Their lower bound is for the *equivariant determinantal complexity* of the permanent. The determinantal complexity (DC) of a polynomial $p \in \mathbb{F}[X]$ refers to the size of the smallest matrix M with entries that are affine linear forms in X such that $\det(M) = p$. Since every polynomial in VP has DC that is at most quasi-polynomial, an exponential lower bound on the DC of the permanent would show that circuits computing PERM_n must have size at least 2^{n^δ} for some positive δ, and hence separate VP from VNP. Landsberg and Ressayre establish exponential lower bounds on any *equivariant* determinantal representation of the permanent, that is one that preserves all the symmetries of the permanent function. This includes not just the permutations on entries that we consider, but the entire projective symmetry group. This does not, unfortunately, yield any lower bounds for symmetric circuits in the sense we consider. This is because the translation of circuits to determinantal representations that establishes that every polynomial in VP has DC at most quasi-polynomial does not preserve symmetries. A symmetric circuit does not, in general, yield a determinantal representation invariant under row-column permutations, let alone the much richer group of symmetries considered in [19]. The latter group also includes continuous group actions that have no counterpart in the realm of circuits and is specific to algebraically-closed fields of characteristic zero. It remains an interesting question to investigate whether there is a deeper connection between the lower bounds presented here and their results.

Outline. In Section 2 we introduce some preliminary definitions and notation. In Section 3, we introduce the key definitions and properties of symmetric circuits. Section 4 establishes the upper bound for symmetric circuit size for the determinant, by translating Le Verrier's method to symmetric circuits. Finally the lower bound for the permanent is established in Sections 5 and 6. The first of these gives the symmetry-preserving translation from arithmetic circuits to Boolean circuits with threshold gates, and the second gives the main construction proving the linear lower bound for the counting width of the number of perfect matchings in a bipartite graph.

2 Background

In this section we discuss relevant background and introduce notation.

We write $\mathbb{N}$ for the positive integers and $\mathbb{N}_0$ for the non-negative integers. For $m \in \mathbb{N}_0$, $[m]$ denotes the set $\{1, \dots, m\}$. For a set X we write $\mathcal{P}(X)$ to denote the powerset of X.

2.1 Counting Width

For any $k \in \mathbb{N}$, the k-dimensional Weisfeiler-Leman equivalence (see [8]), denoted $\equiv^k$ is an equivalence relation on graphs that provides an over-approximation of isomorphism in the sense that for isomorphic graphs G and H, we have $G \equiv^k H$ for all k. Increasing values of

k give finer relations, so $G \equiv^{k+1} H$ implies $G \equiv^{k} H$ for all k. The equivalence relation is decidable in time $n^{\mathcal{O}(k)}$, where n is the size of the graphs. If $k \geq n$, then $G \equiv^{k} H$ implies that G and H are isomorphic. The Weisfeiler-Leman equivalences have been widely studied and they have many equivalent characterizations in combinatorics, logic, algebra and linear optimization. One of their many uses has been to establish inexpressibility results in logic. These can be understood through the notion of *counting width*.

A *graph parameter* is a function from graphs to $\mathbb{N}$ which is isomorphism invariant. Examples are the chromatic number, the number of connected components or the number of perfect matchings. For a graph parameter μ and any fixed $n \in \mathbb{N}$, there is a smallest value of k such that μ is $\equiv^{k}$-invariant. This motivates the definition.

▶ **Definition 1.** *For any graph parameter μ, the* counting width *of μ is the function $\nu : \mathbb{N} \to \mathbb{N}$ such that $\nu(n)$ is the smallest k such that for all graphs G, H of size n, if $G \equiv^{k} H$, then $\mu(G) = \mu(H)$.*

The *counting width* of a class of graphs $\mathcal{C}$ is the counting width of its indicator function. This notion of counting width for classes of graphs was introduced in [11], which we here extend to graph parameters. Note that for any graph parameter $\nu(n) \leq n$.

Cai, Fürer and Immerman [8] first showed that there is no fixed k for which $\equiv^{k}$ coincides with isomorphism. Indeed, in our terminology, they construct a class of graphs with counting width $\Omega(n)$. Since then, many classes of graphs have been shown to have linear counting width, including the class of Hamiltonian graphs and the class of 3-colourable graphs (see [5]). In other cases, such as the class of graphs that contain a perfect matching, it has been proved that they have counting width bounded by a constant [3]. Our interest in counting width stems from the relation between this measure and lower bounds for symmetric circuits. Roughly, if a class of graphs is recognized by a family of polynomial-sized symmetric threshold circuits, it has bounded counting width (a more precise statement is given in Theorem 13).

Our lower bound construction in Section 6 is based on the graphs constructed by Cai et al. [8]. While we review some of the details of the construction in Section 6, a reader unfamiliar with the construction may wish to consult a more detailed introduction. The original construction can be found in [8] and a version closer to what we use is given in [10].

2.2 Circuits

We provide a general definition that incorporates both Boolean and arithmetic circuits.

▶ **Definition 2** (Circuit). *A* circuit *over the* basis $\mathbb{B}$ *with* variables X *and* constants K *is a directed acyclic graph with a labelling where each vertex of in-degree* 0 *is labelled by an element of $X \cup K$ and each vertex of in-degree greater than* 0 *is labelled by an element of* $\mathbb{B}$.

Let $C = (G, W)$, where $W \subset G \times G$, be a circuit with constants K. We call the elements of G *gates*, and the elements of W *wires*. We call the gates with in-degree 0 *input gates* and gates with out-degree 0 *output gates*. We call those input gates labelled by elements of K *constant gates*. We call those gates that are not input gates *internal gates*. For $g, h \in G$ we say that h is a *child* of g if $(h, g) \in W$. We write $\mathrm{child}(g)$ to denote the set of children of g. We write C_g to denote the sub-circuit of C rooted at g. Unless otherwise stated we always assume a circuit has exactly one output gate.

If K is a field $\mathbb{F}$, and $\mathbb{B}$ is the set $\{+, \times\}$, we have an *arithmetic circuit* over $\mathbb{F}$. If $K = \{0, 1\}$, and $\mathbb{B}$ is a collection of Boolean functions, we have a *Boolean circuit* over the basis $\mathbb{B}$. We define two Boolean bases here. The *standard basis* $\mathbb{B}_{\mathrm{std}}$ contains the functions

$\wedge$, $\vee$, and $\neg$. The *threshold basis* $\mathbb{B}_t$ is the union of $\mathbb{B}_{\text{std}}$ and $\{t_{\geq k} : k \in \mathbb{N}\}$, where for each $k \in \mathbb{N}$, $t_{\geq k}$ is defined for a string $\vec{x} \in \{0,1\}^*$ so that $t_{\geq k}(\vec{x}) = 1$ if, and only if, the number of 1s in $\vec{x}$ at least k. We call a circuit defined over this basis a *threshold circuit*. Another useful function is $t_{=k}$, which is defined by $t_{=k}(x) = t_{\geq k}(x) \wedge \neg t_{\geq k+1}(x)$. We do not explicitly include it in the basis as it is easily defined in $\mathbb{B}_t$.

In general, we require that a basis contain only functions that are invariant under all permutations of their inputs (we define this notion formally in Definition 4). This is the case for the arithmetic functions $+$ and $\times$ and for all of the Boolean functions in $\mathbb{B}_t$ and $\mathbb{B}_{\text{std}}$. Let C be a circuit defined over such a basis with variables X and constants K. We evaluate C for an assignment $M \in K^X$ by evaluating each gate labelled by some $x \in X$ to $M(x)$ and each gate labelled by some $k \in K$ to k, and then recursively evaluating each gate according to its corresponding basis element. We write $C[M](g)$ to denote the value of the gate g and $C[M]$ to denote the value of the output gate. We say that C computes the function $M \mapsto C[M]$.

It is conventional to consider an arithmetic circuit C over $\mathbb{F}$ with variables X to be computing a polynomial in $\mathbb{F}[X]$, rather than a function $\mathbb{F}^X \to \mathbb{F}$. This polynomial is defined via a similar recursive evaluation, except that now each gate labelled by a variable evaluates to the corresponding formal variable, and we treat addition and multiplication as ring operations in $\mathbb{F}[X]$. Each gate then evaluates to some polynomial in $\mathbb{F}[X]$. The polynomial computed by C is the value of the output gate.

For more details on arithmetic circuits see [22] and for Boolean circuits see [24].

3 Symmetric Circuits

In this section we discuss different symmetry conditions for functions and polynomials. We also introduce the notion of a symmetric circuit.

3.1 Symmetric Functions

There is a natural extension of a group action on a set X to functions on X and powers of X.

▶ **Definition 3.** *For any group G, we say that a function $F : K^X \to K$, along with an action of G on X is a G-*symmetric function*, if for every $\sigma \in G$, $\sigma F = F$.*

We are interested in some specific group actions, and we define these next.

▶ **Definition 4.**

- *If $G = \textbf{Sym}(X)$, we call a G-symmetric function $F : K^X \to K$* fully symmetric*.*
- *If $G = \textbf{Sym}(X) \times \textbf{Sym}(Y)$, we call a G-symmetric function $F : K^{X\times Y} \to K$* matrix symmetric*.*
- *If $G = \textbf{Sym}(X)$ we call a G-symmetric $F : K^{X\times X} \to K$, with the natural action of G on $X \times X$* square symmetric*.*

Examples of fully symmetric functions are those that appear as labels of gates in a circuit, including $+$, $\times$, $\wedge$, $\vee$ and $t_{\geq k}$. Matrix symmetric functions are those where the input is naturally seen as a matrix and the result in invariant under aribtrary row and column permutations. The canonical example for us of a matrix symmetric function is the permanent. The determinant is not matrix symmetric over fields of characteristic other than 2, but it is square symmetric. The determinant is also invariant under taking matrix transposes, and we also consider this variation.

▶ **Definition 5.** *Let G be the group generated by the diagonal of $\textbf{Sym}(X) \times \textbf{Sym}(X)$ and the permutation $\sigma_t \in \textbf{Sym}(X) \times \textbf{Sym}(X)$ defined such that $\sigma_t(x, y) = (y, x)$. A function $F : K^{X\times X} \to K$ that is G-symmetric with respect to the natural action of G on $X \times X$ is said to be* transpose symmetric*.*

Finally, another useful notion of symmetry in functions is where the inputs are naturally partitioned into sets.

▶ **Definition 6.** *If $X = \biguplus_{i \in I} X_i$, $G = \prod_{i \in I} \textbf{\textit{Sym}}(X_i)$, and $F : K^X \to K$ is G-symmetric with respect to the natural action of G on X, we say that it is* partition symmetric*.*

Unless otherwise stated we treat the permanent, perm : $\mathbb{F}^{X \times Y} \to \mathbb{F}$ as a matrix symmetric function, and the determinant det : $\mathbb{F}^{X \times X} \to \mathbb{F}$ as a transpose symmetric function.

3.2 Symmetric Circuits

Symmetric Boolean circuits have been considered in the literature, particularly in connection with definability in logic. In that context, we are considering circuits which take relational structures (such as graphs) as inputs and we require their computations to be invariant under re-orderings of the elements of the structure. Here, we generalize the notion to arbitrary symmetry groups, and also consider them in the context of arithmetic circuits. In order to define symmetric circuits, we first need to define the automorphisms of a circuit.

▶ **Definition 7** (Circuit Automorphism)**.** *Let $C = (G, W)$ be a circuit over the basis $\mathbb{B}$ with variables X and constants K. For $\sigma \in \textbf{\textit{Sym}}(X)$, we say that a bijection $\pi : G \to G$ is an* automorphism *extending σ if for every gate g in C we have that*

- *if g is a constant gate then $\pi(g) = g$,*
- *if g is a non-constant input gate then $\pi(g) = \sigma(g)$,*
- *if $(h, g) \in W$ is a wire, then so is $(\pi h, \pi g)$*
- *if g is labelled by $b \in \mathbb{B}$, then so is $\pi(g)$.*

We say that a circuit C with variables X is *rigid* if for every permutation $\sigma \in \textbf{Sym}(X)$ there is at most one automorphism of C extending σ.

We are now ready to define the key notion of a symmetric circuit.

▶ **Definition 8** (Symmetric Circuit)**.** *For a G-symmetric function $F : K^X \to K$, a circuit C computing F is said to be* symmetric *if for every $\sigma \in G$, the action of σ on X extends to an automorphism of C. We say C is* strictly symmetric *if it has no other automorphisms.*

For a gate g in a symmetric circuit C, the *orbit* of g, denoted by $\textbf{Orb}(g)$, is the the set of all $h \in C$ such that there exists an automorphism π of C with $\pi(g) = h$. We write $|\textbf{Orb}(C)|$ for the maximum size of an orbit in C, and call it the *orbit size* of C.

Though symmetric arithmetic circuits have not previously been studied, symmetric Boolean circuits have [13, 20, 2]. It is known that polynomial-size symmetric threshold circuits are more powerful than polynomial-size symmetric circuits over the standard basis [2]. In particular, the majority function is not computable by any family of polynomial-size symmetric circuits over the standard basis. On the other hand, it is also known [12] that adding any fully symmetric functions to the basis does not take us beyond the power of the threshold basis. Thus, $\mathbb{B}_t$ gives the robust notion, and that is what we use here. It is also this that has the tight connection with counting width mentioned above.

3.3 Polynomials

In the study of arithmetic complexity, we usually think of a circuit over a field $\mathbb{F}$ with variables in X as expressing a polynomial in $\mathbb{F}[X]$, rather than computing a function from $\mathbb{F}^X$ to $\mathbb{F}$. The distinction is significant when $\mathbb{F}$ is a finite field, as it is possible for distinct polynomials to represent the same function.

The definitions of symmetric functions given in Section 3.1 extend easily to polynomials. So, for a group G acting on X, a polynomial $p \in \mathbb{F}[X]$ is said to be G-symmetric if $\sigma p = p$ for all $\sigma \in G$. We define *fully symmetric*, *matrix symmetric*, *square symmetric* and *transpose symmetric* polynomials analogously. Every matrix symmetric polynomial is also square symmetric. Also, every transpose symmetric polynomial is square symmetric. The permanent PERM_n is both matrix symmetric and transpose symmetric, while the determinant DET_n is transpose symmetric, but not matrix symmetric. In this paper, we treat PERM_n as a matrix symmetric polynomial and DET_n as a transpose symmetric polynomial. It is clear that a G-symmetric polynomial determines a G-symmetric function.

An arithmetic circuit C expressing a G-symmetric polynomial is said to be *symmetric* if the action of each $\sigma \in G$ on the inputs of C extends to an automorphism of C.

Standard *symmetric polynomials* are, in our terminology, fully symmetric. In particular, the homogeneous polynomial $\sum_{i\in[n]} x_i^r$ is fully symmetric. There is a known lower bound of $\Omega(n \log r)$ on the size of any circuit expressing this polynomial [6]. Notably, the matching upper bound is achieved by a symmetric circuit. Similarly, we have tight upper and lower bounds for the elementary symmetric polynomials $\sum_{S\subseteq[n]:|S|=k} \prod_{i\in S} x_i$ over infinite fields [21]. Again, the upper bound is achieved by symmetric circuits.

The best known upper bound for general arithmetic circuits for expressing the permanent is given by Ryser's formula: $\mathrm{PERM}_n = (-1)^n \sum_{S\subseteq[n]} (-1)^{|S|} \prod_{i=1}^n \sum_{j\in S} x_{ij}$. It is easily seen that this expression is symmetric, and it yields a symmetric circuit of size $\mathcal{O}(2^n n^2)$. Our main result, Theorem 12, gives us a near matching lower bound on the size of symmetric circuits for expressing PERM_n.

A symmetric circuit C expressing a G-symmetric polynomial p is also a symmetric circuit computing the function determined by p. In establishing our upper bound for the determinant, we show the existence of small symmetric circuits for the polynomial, and hence also for the function. For the lower bound on the permanent, we show that there are no small symmetric circuits for computing the function, hence also none for the polynomial. For a discussion of functional lower bounds, as opposed to polynomial lower bounds, see [14].

4 An Upper-Bound for the Determinant

In this section we show that for any field $\mathbb{F}$ of characteristic 0 there is a polynomial-size family of symmetric arithmetic circuits over $\mathbb{F}$ computing $\{\mathrm{DET}_n\}$. We define this family using Le Verrier's method for calculating the characteristic polynomial of a matrix. We review this method briefly, and direct the reader to Section 3.4.1 in [16] for more detail.

The characteristic polynomial of an $n \times n$ matrix M is

$$\det(xI_n - M) = \prod_{i=1}^{n}(x - \lambda_i) = x^n - p_1 x^{n-1} + p_2 x^{n-2} - \ldots + (-1)^n p_n,$$

where $\lambda_1, \ldots, \lambda_n$ are the eigenvalues of M, counted with multiplicity. It is known that $p_n = \det(M)$ and $p_1 = \mathrm{Tr}(M)$. Le Verrier's method gives, for each $i \in [n]$, the linear recurrence given by $p_i = \frac{1}{i}[p_{i-1}s_1 - p_{i-2}s_2 + \ldots \pm s_i]$ where $p_0 = 1$ and for each $j \in [n]$, $s_j = \mathrm{Tr}(M^j)$.

The determinant can thus be computed as follows. First, for each $k \in [n]$ we compute entries in the matrix M^k. Then for each $k \in [n]$ we compute $s_k = \mathrm{Tr}(M^k)$. Finally, we recursively compute each p_i and output p_n. There is a natural arithmetic circuit Φ with variables $M = \{m_{ij} : i, j \in [n]\}$ implementing this algorithm.

To see that Φ is symmetric we pick some $\sigma \in \mathbf{Sym}(n) \times \mathbf{Sym}(n)$ such that σ is either in the diagonal or $\sigma(i,j) = (j,i)$ for all $i,j \in [n]$. We now retrace the description of the algorithm implemented by Φ and show how the action of σ on the input gates extends naturally to an automorphism π of Φ. For each input gate labelled by a variable v let $\pi(v) = \sigma(v)$, and let π fix each constant gate. For each gate v computing $(M^k)_{ij}$ let $\pi(v)$ be the gate computing $(M^k)_{\sigma(i,j)}$. For each $k \in [n]$ let v_k be the gate computing the trace of M^k. Then v_k is the sum of those gates computing the diagonal of M^k, which is fixed setwise by π, and so we let $\pi(v_k) = v_k$. We note that each p_i is defined in terms of $s_1, \dots, s_i$ and constants $-1, 1, \frac{1}{2}, \dots, \frac{1}{i}$. All of these gates are fixed by π and so we can take π to fix all remaining gates in the circuit.

It is possible to show that the construction of Φ for a given $n \in \mathbb{N}$ can be carried out in time $\mathcal{O}(n^3)$. In particular, we have the following.

▶ **Theorem 9.** *For $\mathbb{F}$ a field of characteristic 0, there exists a family of symmetric arithmetic circuits $(\Phi_n)_{n\in\mathbb{N}}$ over $\mathbb{F}$ computing $\{\mathrm{DET}_n\}$ as a family of transpose symmetric polynomials and for which the function $n \mapsto \Phi_n$ is computable in time $\mathcal{O}(n^3)$.*

Le Verrier's method explicitly involves multiplications by $\frac{1}{k}$ for $k \in [n]$, and so cannot be directly applied to fields of positive characteristic. There are many known algorithms for computing the determinant over fields of positive characteristic, and it seems reasonable to conjecture that some could be implemented symmetrically.

5 From Arithmetic To Boolean Circuits

In this section we establish the following symmetry-preserving translation from arithmetic circuits to threshold circuits.

▶ **Theorem 10.** *Let G be a group acting on a set of variables X. Let Φ be a symmetric arithmetic circuit over a field $\mathbb{F}$ with variables X and computing a G-symmetric function. Let $B \subseteq \mathbb{F}$ be finite. Then there is a symmetric threshold circuit C with variables X, such that for all $M \in \{0,1\}^X$ we have $C[M] = 1$ if, and only if, $\Phi[M] \in B$ and $|\boldsymbol{Orb}(C)| = |\boldsymbol{Orb}(\Phi)|$.*

We use Theorem 10 in Section 6 to transfer a lower bound on threshold circuits to arithmetic circuits, a crucial step in establishing our lower bound for the permanent.

We prove Theorem 10 by first establishing a similar translation from arithmetic circuits over a field $\mathbb{F}$ to Boolean circuits over a basis $\mathbb{B}^{\mathbb{F}}_{\mathrm{arth}}$ of partition symmetric functions. We then complete the proof by replacing each gate labelled by a partition symmetric function with an appropriate symmetric threshold circuit.

To enable this second step, we show that each partition symmetric function can be computed by a rigid strictly symmetric threshold circuit. The proof of this result follows from the fact that if a function $F : \{0,1\}^A \to \{0,1\}$ is partition symmetric, then its output for $h \in \{0,1\}^A$ depends only on the *number* of elements in each part of A that h maps to 1. We can thus evaluate F by counting the number of 1s in each part, a procedure which we now show can be implemented via a symmetric threshold circuit.

▶ **Lemma 11.** *Let F be a partition symmetric function. There exists a rigid strictly symmetric threshold circuit $C(F)$ computing F.*

Proof. Let $A := \biguplus_{q\in Q} A_q$ be a disjoint union of finite sets A_q indexed by Q, and $F : \{0,1\}^A \to \{0,1\}$ be a partition symmetric function. The fact that F is partition symmetric means that whether $F(h) = 1$ for some $h \in \{0,1\}^A$ is determined by the number of $a \in A_q$

(for each q) for which $h(a) = 1$. Write h_q for this number. Then, there is a set $c_F \subseteq \mathbb{N}_0^Q$ such that $F(h) = 1$ if, and only if, $(h_q)_{q\in Q} \in c_F$. Since each A_q is finite, so is c_F. Then $F(h) = 1$ if, and only if, the following Boolean expression is true: $\bigvee_{c\in c_F} \bigwedge_{q\in Q}(h_q = c(q))$. We can turn this expression into a circuit C with an OR gate at the output, whose children are AND gates, one for each $c \in c_F$, let us call it $\wedge_c$. The children of $\wedge_c$ are a set of gates, one for each $q \in Q$, let us call it $T_{c,q}$, which is labelled by $t_{=c(q)}$ and has as children all the inputs $a \in A_q$.

This circuit C is symmetric and rigid, but not necessarily strictly symmetric, as it may admit automorphisms that do not respect the partition of the inputs A as $\biguplus_{q\in Q} A_q$. To remedy this, we create pairwise non-isomorphic gadgets G_q, one for each $q \in Q$. Each G_q is a one-input, one-output circuit computing the identity function. For example, G_q could be a tower of single-input AND gates, and we choose a different height for each q. We now modify C to obtain $C(F)$ by inserting between each input $a \in A_q$ and each gate $T_{c,q}$ a copy G_q^a of the gadget G_q.

Clearly $C(F)$ computes F. We now argue $C(F)$ is rigid and strictly symmetric. To see that it is symmetric, consider any $\sigma \in \prod_{q\in Q} \mathbf{Sym}(A_q)$ in its natural action on A. This extends to an automorphism of $C(F)$ that takes the gadget G_q^a to $G_q^{\sigma a}$ while fixing all gates $T_{c,q}$ and $\wedge_c$. To see that there are no other automorphisms, suppose π is an automorphism of $C(F)$. It must fix the output OR gate. Also π cannot map a gate $T_{c,q}$ to $T_{c',q'}$ for $q' \neq q$ because the gadgets G_q and $G_{q'}$ are non-isomorphic. Suppose that π maps $\wedge_c$ to $\wedge_{c'}$. Then, it must map $T_{c,q}$ to $T_{c',q}$. Since the labels of these gates are $t_{=c(q)}$ and $t_{=c'(q)}$ respectively, we conclude that $c(q) = c'(q)$ for all q and therefore $c = c'$. ◀

We now define for each field $\mathbb{F}$ the basis $\mathbb{B}^{\mathbb{F}}_{\text{arth}}$. The functions in this basis are intended to be Boolean analogues of addition and multiplication. Let $Q \subseteq \mathbb{F}$ be finite, $A = \biguplus_{q\in Q} A_q$ be a disjoint union of non-empty finite sets, and $c \in \mathbb{F}$. We define a function $+^A_{Q,c} : \{0,1\}^A \to \{0,1\}$ that given $h \in \{0,1\}^A$ computes the sum over all q of the number of elements of A_q that h maps to 1, weighted by q, and returns 1 if this sum equals c. We also define an analogous function for multiplication $\times^A_{Q,c} : \{0,1\}^A \to \{0,1\}$. Formally, these functions are defined for $h \in \{0,1\}^A$ as follows: $+^A_{Q,c}(h) = 1$ if, and only if, $\sum_{q\in Q} |\{a \in A_q : h(a) = 1\}| \cdot q = c$ and $\times^A_{Q,c}(h) = 1$ if, and only if, $\prod_{q\in Q} q^{|\{a\in A_q : h(a)=1\}|} = c$. Both $+^A_{Q,c}$ and $\times^A_{Q,c}$ are partition symmetric. Let $\mathbb{B}^{\mathbb{F}}_{\text{arth}}$ be the set of all functions $+^A_{Q,c}$ and $\times^A_{Q,c}$.

We aim to prove Theorem 10 by first defining for a given symmetric arithmetic circuit a corresponding symmetric circuit over a partition symmetric basis. To ensure unambiguous evaluation, the circuit must include for each gate labelled by a partition symmetric function a corresponding partition on its children. Let C be a circuit with variables X and let g be a gate in C labelled by a partition symmetric function $F : \{0,1\}^A \to \{0,1\}$, where $A = \biguplus_{q\in Q} A_q$ is a disjoint union of finite non-empty sets. We associate with g a bijection $L_g : A \to \text{child}(g)$. We evaluate g for an input as follows. For $M \in \{0,1\}^X$ we let $L_g^M : A \to \{0,1\}$ be defined such that $L_g^M(a) = C[M](L_g(a))$ for all $a \in A$. Let $C[M](g) = F(L_g^M)$.

Proof of Theorem 10. For $v \in \Phi$ let Q_v be the set of possible evaluations of v if the input gates are assigned to 0 or 1, i.e. $Q_v = \{\Phi[M](v) : M \in \{0,1\}^X\}$. The restriction to 0-1-matrices ensures that Q_v is finite. Let z be the output gate of Φ. If $Q_z \subseteq B$ let C be the circuit consisting of a single gate labelled by 1 and if $Q_z \cap B = \emptyset$ let C consist of a single gate labelled by 0. Suppose that neither of these two cases hold.

We now construct a $\mathbb{B}^{\mathbb{F}}_{\text{arth}} \cup \mathbb{B}_{\text{std}}$-circuit D from Φ by replacing each internal gate v in Φ with a family of gates (v, q) for $q \in Q_v$ such that $D[M](v, q) = 1$ if, and only if, $\Phi[M](v) = q$. Each (v, q) is labelled by a function of the form $+^A_{Q,q}$ or $\times^A_{Q,q}$, depending on if v is an addition or multiplication gate. We also add a single output gate in D that has as children exactly those gates (z, q) where $q \in Q_z \cap B$. We define D from Φ recursively as follows. Let $v \in \Phi$.

- If v is an non-constant input gate in Φ let $(v, 1)$ be an input gate in D labelled by the same variable as v and let $(v, 0)$ be a NOT-gate with child $(v, 1)$.
- If v is a constant gate in Φ labelled by some field element q let (v, q) be a constant gate in D labelled by 1.
- Suppose v is an internal gate. Let $Q = \bigcup_{u \in \text{child}(v)} Q_v$. For $q \in Q$ let $A_q = \{u \in \text{child}(v) : q \in Q_u\}$. Let $A = \biguplus_{q \in Q} A_q$. For each $c \in Q_v$ let (v, c) be a gate in D such that if v is an addition gate or multiplication gate then (v, c) is labelled by $+^{A}_{Q,c}$ or $\times^{A}_{Q,c}$, respectively. The labelling function $L_{(v,c)} : A \to \text{child}(v, c)$ is defined for $u \in A$ such that if $u \in A_q$ then $L_{(v,c)}(u) = (u, q)$.

We add one final OR-gate w to form D with $\text{child}(w) = \{(z, q) : q \in B \cap Q_z\}$.

We now show that D is a symmetric circuit. Let $\sigma \in G$ and π be an automorphism of Φ extending σ. Let $\pi' : D \to D$ be defined such that for each gate $(v, c) \in D$, $\pi'(v, c) = (\pi(v), c)$ and for the output gate w, $\pi'(w) = w$. It can be verified by induction that π' is an automorphism of C extending σ.

We now show that $|\mathbf{Orb}(D)| = |\mathbf{Orb}(\Phi)|$. It suffices to prove that for $v, u \in \Phi$ and $c \in Q_v$ that $u \in \mathbf{Orb}(v)$ if, and only if, $(u, c) \in \mathbf{Orb}(v, c)$. The forward direction follows from the above argument establishing that D is symmetric. Let $v, u \in \Phi$ and $c \in Q_v$ and suppose $(u, c) \in \mathbf{Orb}(v, c)$. For each gate $t \in \Phi$ pick some $c_t \in Q_t$ such that if $t = u$ or $t = v$ then $c_t = c$ and for all $t_1, t_2 \in \Phi$, if $Q_{t_1} = Q_{t_2}$ then $c_{t_1} = c_{t_2}$. Let π' be an automorphism of D such that $\pi'(v, c) = (u, c)$. Let $\pi : \Phi \to \Phi$ be defined for $t \in \Phi$ such that $\pi'(t, c_t) = (\pi(t), c_t)$. We now show that π is an automorphism of Φ, and so $u \in \mathbf{Orb}(v)$. Note that, since π' preserves the labelling on the gates in D, it follows that for all $t \in \Phi$, $Q_t = Q_{\pi(t)}$ and so $c_{\pi(t)} = c_t$. Let $t, t' \in \Phi$ and suppose $\pi(t) = \pi(t')$. Then $\pi'(t, c_t) = (\pi(t), c_t) = (\pi(t), c_{\pi(t)}) = (\pi(t'), c_{\pi(t')}) = (\pi(t'), c_{t'}) = \pi'(t', c_{t'})$, and so $(t, c_t) = (t', c_{t'})$ and $t = t'$. It follows that π is injective, and so bijective. Let $t, s \in \Phi$. Then $t \in \text{child}(s) \iff (t, c_t) \in \text{child}(s, c_s) \iff \pi'(t, c_t) \in \text{child}(\pi'(s, c_s)) \iff (\pi(t), c_t) \in \text{child}(\pi(s), c_s) \iff \pi(t) \in \text{child}(\pi(s))$. The first and last equivalences follow from the construction of the circuit. The remaining conditions for π to be an automorphism can be easily verified.

Let $M \in \{0, 1\}^X$. We now show by induction that for all $v \in \Phi$ and $c \in Q_v$, $\Phi[M](v) = c$ if, and only if, $D[M](v, c) = 1$. Let $v \in \Phi$. If v is an input gate then the claim holds trivially. Suppose v is an internal gate and let $c \in Q_v$. Suppose v is an addition gate. Then (v, c) is labelled by the function $+^{A}_{Q,c}$ where $Q = \bigcup_{u \in \text{child}(v)} Q_u$, for $q \in Q$, $A_q = \{u \in \text{child}(v) : q \in Q_u\}$, and $A = \biguplus_{q \in Q} A_q$. Then

$$\begin{aligned}
\Phi[M](v) = c &\iff \sum_{u \in \text{child}(v)} \Phi[M](u) = c \iff \sum_{q \in Q} |\{u \in \text{child}(v) : \Phi[M](u) = q\}| \cdot q = c \\
&\iff \sum_{q \in Q} |\{u \in A_q : D[M](u, q) = 1\}| \cdot q = c \\
&\iff \sum_{q \in Q} |\{u \in A_q : L^{M}_{(v,c)}(u) = 1\}| \cdot q = c \\
&\iff D[M](v, c) = 1
\end{aligned}$$

A similar argument suffices if v is a multiplication gate. It follows that $D[M](w) = 1$ if, and only if, there exists $c \in B$ such that $D[M](z, c) = 1$ if, and only if, $\Phi[M] \in B$.

We define C from D by replacing each internal gate $(v, c) \in D$ labelled by some $F \in \mathbb{B}^{\mathbb{F}}_{\text{arth}}$ with the rigid strictly symmetric threshold circuit $C(F)$ computing F defined in Lemma 11. C computes the same function as D. Since $C(F)$ is symmetric, C is symmetric. Since $C(F)$ is rigid and *strictly* symmetric, $|\mathbf{Orb}(C)| = |\mathbf{Orb}(D)| = |\mathbf{Orb}(\Phi)|$. ◀

6 A Lower-Bound for the Permanent

We now establish the lower bound on symmetric arithmetic circuits for the permanent.

▶ **Theorem 12.** *If $\mathbb{F}$ is a field with char$(\mathbb{F}) \neq 2$, then for any $\epsilon > 0$ there is no family of symmetric arithmetic circuits over $\mathbb{F}$ of orbit size $\mathcal{O}(2^{n^{1-\epsilon}})$ computing $\{\mathrm{PERM}_n\}$.*

Our proof establishes something stronger. We actually show that there are no symmetric arithmetic circuits of orbit size $\mathcal{O}(2^{n^{1-\epsilon}})$ that compute the function perm(M) for matrices $M \in \mathbb{F}^{n\times n}$. Indeed, the lower bound holds even when restricting the input to matrices $M \in \{0,1\}^{n\times n}$. Theorem 12 is proved by showing lower bounds on the counting widths of functions which determine the number of perfect matchings in a bipartite graph. The connection of orbit size to counting width comes through the following theorem (see [2, 5]).

▶ **Theorem 13.** *Let $(C_n)_{n\in\mathbb{N}}$ be a family of symmetric threshold circuits of orbit size $s = \mathcal{O}(2^{n^{1-\epsilon}})$ for some $\epsilon > 0$ deciding a class of graphs $\mathcal{C}$. Then, the counting width of $\mathcal{C}$ is $\mathcal{O}(\frac{\log s}{\log n})$.*

If G is a bipartite graph, let $\mu(G)$ denote the number of perfect matchings in G and, for a prime number p, we write $\mu^p(G)$ for the congruence class of $\mu(G) \pmod p$. It is well known if G is a balanced bipartite graph with vertex bipartition $V(G) = A \cup B$, and $M_G \in \{0,1\}^{A\times B}$ is the biadjacency matrix of G, then the permanent of M_G (say, over the rational field $\mathbb{Q}$) is the number of distinct perfect matchings of G. Moreover, since M_G is a 0-1-matrix, $\mathrm{perm}_{\mathbb{F}}(M_G) = \mathrm{perm}_{\mathbb{F}'}(M_G)$ whenever $\mathbb{F}'$ is a subfield of $\mathbb{F}$. In particular, for any field $\mathbb{F}$ of characteristic zero, $\mathrm{perm}_{\mathbb{F}}(M_G) = \mathrm{perm}_{\mathbb{Q}}(M_G) = \mu(G)$ and for any field $\mathbb{F}$ of characteristic p, $\mathrm{perm}_{\mathbb{F}}(M_G) = \mathrm{perm}_{\mathbb{F}_p}(M_G) = \mu^p(G)$. To avoid unnecessary case distinctions, we write $\mu^c(G)$ where c is either 0 or a prime p, with the understanding that $\mu^0(G) = \mu(G)$. Then, we can say that for any field $\mathbb{F}$ with char$(\mathbb{F}) = c$, $\mathrm{perm}_{\mathbb{F}}(M_G) = \mu^c(G)$.

Combining Theorem 10 with Theorem 13 gives us the following consequences.

▶ **Corollary 14.** *If there exists a family of symmetric circuits of orbit size $s = \mathcal{O}(2^{n^{1-\epsilon}})$ over a field $\mathbb{F}$ of characteristic c computing $\{\mathrm{PERM}_n\}$, then the counting width of μ^c is $\mathcal{O}(\frac{\log s}{\log n})$.*

Proof. Let k be the counting width of μ^c. Then, by definition, we can find for each $n \in \mathbb{N}$, a pair of balanced bipartite graphs G_n and H_n on at most $2n$ vertices such that $G_n \equiv^{k(n)-1} H_n$ but $\mu(G_n) \neq \mu(H_n)$. Let $B_n = \{\mu(G_n)\}$. Then, by Theorem 10 and the assumption that there is a family of symmetric circuits over $\mathbb{F}$ computing $\{\mathrm{PERM}_n\}$ of orbit size $s = \mathcal{O}(2^{n^{1-\epsilon}})$, there is a family of symmetric Boolean threshold circuits of orbit size $s = \mathcal{O}(2^{n^{1-\epsilon}})$ which decides for a matrix $M \in \{0,1\}^{n\times n}$ whether perm$(M) \in B_n$. In other words, when $c = 0$, this family of circuits then decides whether a balanced bipartite graph G on $2n$ vertices has exactly $\mu(G_n)$ perfect matchings, and when $c = p$ for some prime p, it decides whether G has $\mu^p(G_n)$ perfect matchings, modulo p. It follows by Theorem 13 that the counting width of this decision problem is $\mathcal{O}(\frac{\log s}{\log n})$. Since the counting width of this decision problem is, by choice of G_n, k, it follows that $k = \mathcal{O}(\frac{\log s}{\log n})$. ◀

Thus, to establish Theorem 12, we aim to prove the following.

▶ **Theorem 15.** *There are, for each $k \in \mathbb{N}$, a pair of balanced bipartite graphs X and Y with $\mathcal{O}(k)$ vertices, such that $X \equiv^k Y$, and $\mu(X) - \mu(Y) = 2^l$ for some $l > 0$.*

Before giving the proof of Theorem 15 we show how Theorem 12 now follows.

Proof of Theorem 12. By Theorem 15, we have for each k, a pair of graphs X and Y with $\mathcal{O}(k)$ vertices such that $\mu(X) \neq \mu(Y)$ and $X \equiv^k Y$ thus, the counting width of μ is $\Omega(n)$. Moreover, since $\mu(X) - \mu(Y)$ is a power of 2, it follows that for any prime $p \neq 2$, $\mu(X) \not\equiv \mu(Y) \pmod{p}$. Hence, the counting width of μ^p is also $\Omega(n)$.

Suppose then that $\mathbb{F}$ is a field of characteristic $c \neq 2$ and that there is a family of symmetric arithmetic circuits over $\mathbb{F}$ of orbit size $s = \mathcal{O}(2^{n^{1-\epsilon}})$ computing $\{\mathrm{PERM}_n\}$. Then, it follows from Corollary 14 that the counting width of μ^c is at most $k = \mathcal{O}(\frac{\log s}{\log n}) = \mathcal{O}(n^{1-\epsilon})$, giving a contradiction. ◀

The construction used to prove Theorem 15 is an adaptation of a standard construction by Cai, Fürer and Immerman [8] which gives non-isomorphic graphs X and Y with $X \equiv^k Y$ for arbitrary k (see also [10]). We tweak it somewhat to ensure that both graphs have perfect matchings (indeed, they are both balanced bipartite graphs). The main innovation is in the analysis of the number of perfect matchings the graphs contain.

Gadgets. In what follows, $G = (V, E)$ is always a 3-regular 2-connected graph. From this, we first define a graph $X(G)$. The vertex set of $X(G)$ contains, for each edge $e \in E$, two vertices that we denote e_0 and e_1. For each vertex $v \in V$ with incident edges f, g and h, $X(G)$ contains five vertices. One of these we call the *balance vertex* and denote v_b. The other four are called *inner vertices* and there is one v_S, for each subset $S \subseteq \{f, g, h\}$ of even size. For each $v \in V$, the neighbours of v_b are exactly the four vertices of the form v_S. Moreover, for each $e \in \{f, g, h\}$, $X(G)$ contains the edge $\{e_1, v_S\}$ if $e \in S$ and the edge $\{e_0, v_S\}$ otherwise. There are no other edges in $X(G)$.

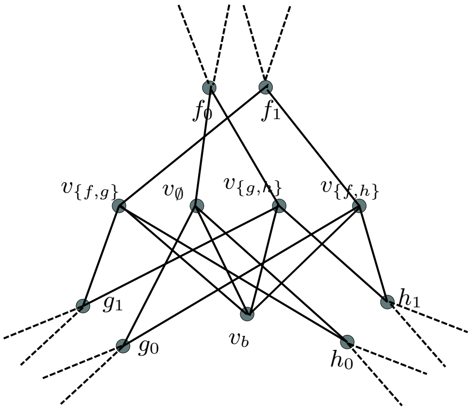


Figure 1 A gadget in $X(G)$ corresponding to vertex v with incident edges f, g, h.

The construction of $X(G)$ from G essentially replaces each vertex v with incident edges f, g and h with the gadget depicted in Figure 1, where the dashed lines indicate edges whose endpoints are in other gadgets. The vertices e_0, e_1 for each $e \in \{f, g, h\}$ are shared with neighbouring gadgets.

For any fixed vertex $x \in V$ with incident edges f, g, h, the graph $\tilde{X}_x(G)$ is obtained by modifying the construction of $X(G)$ so that, for the one vertex x, the gadget contains inner vertices x_S for subsets $S \subseteq \{f, g, h\}$ of odd size. Again, for each $e \in \{f, g, h\}$, $X(G)$ contains the edge $\{e_1, v_S\}$ if $e \in S$ and the edge $\{e_0, v_S\}$ otherwise.

If we remove the balance vertices v_b, the graphs $X(G)$ and $\tilde{X}_x(G)$ are essentially the Cai-Fürer-Immerman (CFI) graphs associated with G. The balance vertex v_b is adjacent to all the inner vertices associated with v and so does not alter the automorphism structure of $X(G)$ (or $\tilde{X}_x(G)$) at all. Nor do these vertices alter any other essential properties of the CFI construction. In particular, since G is connected, we have the following lemma.

▶ **Lemma 16.** *For any $x, y \in V$, $\tilde{X}_x(G)$ and $\tilde{X}_y(G)$ are isomorphic.*

With this in mind, we refer simply to the graph $\tilde{X}(G)$ to mean a graph $\tilde{X}_x(G)$ for some fixed x, and we refer to x as the special vertex of G.

By known properties of the CFI construction, we also have the following (see [10, Theorem 3]).

▶ **Lemma 17.** *If the treewidth of G is greater than k, then $X(G) \equiv^k \tilde{X}(G)$.*

The purpose of the balance vertices is to change the structure of the perfect matchings. Indeed, if we let CFI(G) denote the subgraph of $X(G)$ that excludes the balance vertices, it is easily seen that this contains no perfect matchings. It is a bipartite graph where one part contains the $4|V|$ inner vertices and the other part contains the $2|E| = 3|V|$ edge vertices and so no perfect matching is possible. But, $X(G)$ is a bipartite graph where in one part we have the $4|V|$ inner vertices and in the other the $3|V|$ edge vertices along with the $|V|$ balance vertices. In short, this is a 4-regular bipartite graph and so contains perfect matchings. We next analyse the structure of the set of such perfect matchings. In particular, we show that $X(G)$ and $\tilde{X}(G)$ contain different numbers of perfect matchings.

In the sequel, we write X to denote either one of the graphs $X(G)$ or $\tilde{X}(G)$, $V(X)$ to denote its vertices and $E(X)$ to denote its edges. We continue to use V and E for the vertices and edges of G. Also, for each $v \in V$, we write I_v to denote the set of four inner vertices in X associated with v.

Non-Uniform Matchings. Let $M \subseteq E(X)$ be a perfect matching in X. For each $v \in V$ and $e \in E$ incident on v, we define the projection $p^M(v, e)$ of M on (v, e) to be the value in $\{0, 1, 2\}$ which is the number of edges between $\{e_0, e_1\}$ and I_v that are included in M. These satisfy the following equations:

$p(u, e) + p(v, e) = 2$ for each edge $e = \{u, v\} \in E$; and

$p(v, f) + p(v, g) + p(v, h) = 3$ for each vertex $v \in V$ with incident edges f, g, h.

The first of these holds because M must include exactly one edge incident on each of e_0 and e_1. The second holds because M must include an edge between v_b and one vertex of I_v. Thus, the three remaining vertices in I_v must be matched with vertices among $f_0, f_1, g_0, g_1, h_0, h_1$.

One solution to the set of equations is obtained by taking the constant projection $p^M(v, e) = 1$ for all such pairs (v, e). Say that a matching M is uniform if $p^M(v, e) = 1$ everywhere and non-uniform otherwise.

▶ **Lemma 18.** *The number of non-uniform matchings in $X(G)$ is the same as in $\tilde{X}(G)$.*

Proof. It suffices to prove that for any non-constant projection p, the number of matchings M with $p^M = p$ is the same for both $X(G)$ and $\tilde{X}(G)$. For then, taking the sum over all possible projections gives the result. So, let p be a non-constant projection. Then, for some edge $e = \{u, v\} \in E$, we have $p(u, e) = 2$ and $p(v, e) = 0$. Then, let $X(G)^-$ and $\tilde{X}(G)^-$ be the subgraphs of $X(G)$ and $\tilde{X}(G)$ respectively obtained by removing the edges between $\{e_0, e_1\}$ and I_v. It is clear that any matching M in $X(G)$ with $p^M = p$ is also a perfect matching in $X(G)^-$, and similarly for $\tilde{X}(G)$. However, $X(G)^-$ and $\tilde{X}(G)^-$ are isomorphic. This follows by an argument analogous to the proof of Lemma 16. Since G is 2-connected, there is a path p from u to the special vertex x that does not involve the edge e. We can then define an isomorphism from $X(G)$ to $\tilde{X}(G)$ by mapping e_0 to e_1, for each edge f on the path p, mapping f_0 to f_1 and extending this using the induced automorphisms of the gadgets corresponding to $v_1, \ldots, v_{t-1}$. We conclude that the numbers of such matchings are the same for both. ◀

Now, we aim to show that the number of uniform matchings of $X(G)$ is different to that of $\tilde{X}(G)$. For this, it is useful to first analyse the orientations of the underlying graph G.

Orientations. An *orientation* of G is a directed graph obtained from G by assigning to each edge $\{u, v\} \in E$ a direction. There are exactly $2^{|E|}$ distinct orientations of G. We say that a vertex $v \in V$ is *odd* with respect to an orientation $\overrightarrow{G}$ of G if it has an odd number of incoming directed edges and *even* otherwise. For an orientation $\overrightarrow{G}$ of G, we write $\mathrm{odd}(\overrightarrow{G})$ for the set of its odd vertices. We say that the orientiation $\overrightarrow{G}$ is *odd* if $|\mathrm{odd}(\overrightarrow{G})|$ is odd, and we say it is *even* otherwise. Since G is 3-regular, it is not hard to see that $|V|$ is always even and $|E|$ is even if, and only if, $|V|/2$ is. Moreover, in all orientations of G, $|\mathrm{odd}(\overrightarrow{G})| = |E| \pmod 2$.

▶ **Lemma 19.** *If $|V|/2$ is even, then all orientations of G are even. If $|V|/2$ is odd, then all orientations of G are odd.*

Proof. Note that since G is 3-regular, $3|V| = 2|E|$, so $|V|$ is always even. Moreover, $|V|/2$ is even if, and only if, $|E|$ is. For an orientation $\overrightarrow{G}$, let $\mathrm{in}(v)$ denote the number of edges incoming to the vertex v. Then, $|E| = \sum_v \mathrm{in}(v)$. But, $\sum_v \mathrm{in}(v) \equiv |\mathrm{odd}(\overrightarrow{G})| \pmod 2$. ◀

Thus, we say that a graph G is *odd* if $|E|$ is odd, and hence all orientations of G are odd, and G is *even* if $|E|$ is even and hence all orientations of G are even.

We can now quantify exactly, for any set $S \subseteq V$ the number of distinct orientations $\overrightarrow{G}$ with $\mathrm{odd}(\overrightarrow{G}) = S$. To do this, we first establish an auxiliary lemma.

▶ **Lemma 20.** *If $G = (V, E)$ is even, then for every set $S \subseteq V$ with $|S|$ even, there is an orientation $\overrightarrow{G}$ of G with odd$(\overrightarrow{G}) = S$. Similarly if $G = (V, E)$ is odd, then for every set $S \subseteq V$ with $|S|$ odd, there is an orientation $\overrightarrow{G}$ of G with odd$(\overrightarrow{G}) = S$.*

Proof. It suffices to show, for any set $S \subseteq V$ and any pair of vertices $u, v \in V$, if there is an orientation $\overrightarrow{G}$ of G with $\mathrm{odd}(\overrightarrow{G}) = S$, then there is also an orientation $\overrightarrow{G}'$ with $\mathrm{odd}(\overrightarrow{G}') = S \triangle \{u, v\}$. Now, consider any simple path from u to v in G and let $\overrightarrow{G}'$ be the orientation obtained from $\overrightarrow{G}$ by reversing the direction of every edge on this path. ◀

▶ **Lemma 21.** *For every set $S \subseteq V$ with $|S| = |E| \pmod 2$, there are exactly $2^{|V|/2+1}$ distinct orientations $\overrightarrow{G}$ with odd$(\overrightarrow{G}) = S$.*

Proof. Let A be the $V \times E$ incidence matrix of the graph G. This defines a linear transformation from the vector space $\mathbb{F}_2^E$ to $\mathbb{F}_2^V$. The additive group of $\mathbb{F}_2^E$ has a natural action on the orientations of G: for a vector $\pi \in \mathbb{F}_2^E$, and an orientation $\overrightarrow{G}$, define $\pi\overrightarrow{G}$ to be the orientation obtained from $\overrightarrow{G}$ by changing the orientation of each edge e with $\pi(e) = 1$. Indeed, fixing one particular orientation $\overrightarrow{G}$, the action generates all orientations and gives a bijective correspondence between the vectors in $\mathbb{F}_2^E$ and the orientations of G. Similarly, the additive group of $\mathbb{F}_2^V$ has a natural action on the powerset of V: for a vector $\sigma \in \mathbb{F}_2^V$ and a set $S \subseteq V$, let σS be the set $S \triangle \{v \mid \sigma(v) = 1\}$. Again, for any fixed set S, this action generates all subsetes of V and gives a bijection between $\mathbb{F}_2^V$ and the powerset of V.

Then, it can be seen that $\mathrm{odd}(\pi\overrightarrow{G}) = (A\pi)\mathrm{odd}(\overrightarrow{G})$. Indeed, if $v \in V$ is a vertex with incident edges f, g, h, then $(A\pi)(v) = \pi(f)+\pi(g)+\pi(h) \pmod 2$. In other words $(A\pi)(v) = 1$ just in case the direction of an odd number of edges incident on v is flipped by π. Thus, the set of vertices $\{v \mid (A\pi)(v) = 1\}$ are exactly the ones that change from being odd to even or vice versa under the action of π, i.e. $\{v \mid (A\pi)(v) = 1\} = \mathrm{odd}(\overrightarrow{G}) \triangle \mathrm{odd}(\pi\overrightarrow{G})$ for any orientation $\overrightarrow{G}$.

Fixing a particular orientation $\vec{G}$, the action of $\mathbb{F}_2^E$ generates all orientation $\pi\vec{G}$, and A maps this to the collection of all sets $\text{odd}(\vec{G})\triangle\text{odd}(\pi\vec{G})$. Then, by Lemmas 19 and 20 the image of A consists of exactly the set of vectors with an even number of 1s. Hence, the image of A has dimension $|V|-1$ and so its kernel has size $2^{|E|}/2^{|V|-1}$. Since $|E| = 3|V|/2$, this is $2^{|V|/2+1}$. By linearity, the pre-image of any vector v in the image of A has exactly this size. Thus, for each even size set $T \subseteq V$, there are exactly $2^{|V|/2+1}$ vectors $\pi \in \mathbb{F}_2^E$ with $\text{odd}(\pi\vec{G}) = T\triangle\text{odd}(\vec{G})$. ◀

Matchings in Gadgets. Any uniform perfect matching M of X induces an orientation of G, which we denote $\vec{G}^M$: any edge $e = \{u, v\} \in E$ is oriented from u to v in $\vec{G}^M$ if M contains an edge between e_0 and a vertex in I_u and an edge between e_1 and a vertex in I_v.

Furthermore, every orientation arises from some perfect matching. To see this, consider again the gadget in Figure 1. This has eight subgraphs induced by taking the vertices $\{v_b\}\cup I_v$, together with exactly one vertex from each of the sets $\{f_0, f_1\}$, $\{g_0, g_1\}$ and $\{h_0, h_1\}$. We claim that each of these eight subgraphs contains a perfect matching. Indeed, it suffices to verify this for the two cases $S = I_v \cup \{v_b\} \cup \{f_0, g_0, h_0\}$ and $T = I_v \cup \{v_b\} \cup \{f_0, g_0, h_1\}$ as the other six are obtained from these by automorphisms of the gadget. In what follows, we also write S and T for the subgraphs of the gadget in Figure 1 induced by these sets. It is easily seen by inspection that S has exactly four perfect matchings and T has exactly two perfect matchings.

Hence, for any orientation $\vec{G}$, we get a matching $M \subseteq X$ with $\vec{G}^M = \vec{G}$ by choosing one matching from each gadget. To be precise, for each vertex $v \in V$, define the *relevant subgraph* of X at v to be the subgraph induced by $I_v \cup \{v_b\}$ along with the vertices e_1 for each edge e incoming at v in $\vec{G}$ and e_0 for each edge e outgoing at v in $\vec{G}$. In $X(G)$, the relevant subgraph at v is isomorphic to S if v is even in $\vec{G}$ and it is isomorphic to T if v is odd in $\vec{G}$. The same is true for all vertices in $\tilde{X}(G)$, apart from the special vertex x. For this one, the relevant subgraph is isomorphic to S if x is odd and to T if x is even. In either case, we get a perfect matching M with $\vec{G}^M = \vec{G}$ by independently choosing exactly one matching in each relevant subgraph. There are 4 such choices when the relevant subgraph is like S and 2 choices when it is like T.

Uniform Matchings. It follows that for any orientation $\vec{G}$ of G, the number of uniform perfect matchings M of $X(G)$ with $\vec{G}^M = \vec{G}$ is $2^{|\text{odd}(\vec{G})|}4^{|V|-|\text{odd}(\vec{G})|}$. The number of uniform perfect matchings in $\tilde{X}(G)$ depends on whether the special vertex x is odd in $\vec{G}$ or not. If it is, the number is $2^{|\text{odd}(\vec{G})|-1}4^{|V|-|\text{odd}(\vec{G})|+1}$ otherwise it is $2^{|\text{odd}(\vec{G})|+1}4^{|V|-|\text{odd}(\vec{G})|-1}$. Thus, if we denote the number of uniform perfect matchings in X by $\#MX$, then we have $\#MX(G) = \sum_{\vec{G}} 2^{|\text{odd}(\vec{G})|}4^{|V|-|\text{odd}(\vec{G})|}$ where the sum is over all orientations of G. Then, by Lemma 21, $\#MX(G) = 2^{|V|/2+1}\sum_{S\subseteq V\,:\,|S|\equiv|E| \pmod 2} 2^{|S|}4^{|V|-|S|}$. By the same token, $\#M\tilde{X}(G) = 2^{|V|/2+1}\sum_{S\subseteq V\,:\,|S|\not\equiv|E| \pmod 2} 2^{|S|}4^{|V|-|S|}$.

Finally, to show that $\#MX(G)$ and $\#M\tilde{X}(G)$ are different, let P_m denote the number $\sum_{S\subseteq[2m]:|S|\text{even}} 2^{|S|}4^{2m-|S|}$ and Q_m denote the number $\sum_{S\subseteq[2m]:|S|\text{odd}} 2^{|S|}4^{2m-|S|}$.

▶ **Lemma 22.** *For all $m \geq 1$, $P_m - Q_m = 4^m$.*

Proof. We prove this by induction on m. For $m = 1$, there are exactly two odd sized subsets and two even sized subsets of $[2m]$. So $P_m = 20$ and $Q_m = 16$. For larger values of m, we have the following identity, where S ranges over subsets of $[2m-2]$

$$\begin{aligned}P_m &= \sum_{|S|\text{even}} 2^{|S|}4^{2m-|S|} + 2\sum_{|S|\text{odd}} 2^{|S|+1}4^{2m-|S|-1} + \sum_{|S|\text{even}} 2^{|S|+2}4^{2m-|S|}\\ &= 16P_{m-1} + 16Q_{m-1} + 4P_{m-1}\\ &= 20P_{m-1} + 16Q_{m-1}.\end{aligned}$$

Here, in the first line, the first sum accounts for all even size subsets of [2m] that exclude the last two elements, the second one for those that include exactly one of the last two elements and the third sum for all that include the last two elements.

Similarly, we have

$$\begin{aligned}Q_m &= \sum_{|S|\text{odd}} 2^{|S|}4^{2m-|S|} + 2\sum_{|S|\text{even}} 2^{|S|+1}4^{2m-|S|-1} + \sum_{|S|\text{odd}} 2^{|S|+2}4^{2m-|S|}\\ &= 16Q_{m-1} + 16P_{m-1} + 4Q_{m-1}\\ &= 20Q_{m-1} + 16P_{m-1}.\end{aligned}$$

Thus, $P_m - Q_m = 4P_{m-1} - 4Q_{m-1}$. By, induction hypothesis, the right hand side is $4 \cdot 4^{m-1}$ and we're done. ◀

Proof of Theorem 15. By a standard expander graph construction (e.g. [1]), for any k, we can find a 3-regular graph G with treewidth at least k and $2n = \mathcal{O}(k)$ vertices. Then $X(G)$ and $\tilde{X}(G)$ both have $\mathcal{O}(k)$ vertices and by Lemma 17 we have $X(G) \equiv^k \tilde{X}(G)$. Moreover, $X(G)$ and $\tilde{X}(G)$ have the same number of non-uniform perfect matchings by Lemma 18. The number of uniform matchings is $2^{n+1}P_n$ in one case and $2^{n+1}Q_n$ in the other (which is which depends on whether n is even or odd). Either way, $|\mu(X(G)) - \mu(\tilde{X}(G))| = 2^{3n+1}$, which is a power of 2 as required. ◀

7 Concluding Discussion

We have introduced a novel restriction of arithmetic circuits, which is based on a natural notion of symmetry. On this basis, we have shown a fundamental difference between circuits for the determinant and the permanent. The former admits a description through polynomial-size symmetric circuits and the latter does not.

There are several ways in which our results could be tightened. The first would be to show the existence of polynomial-size circuits for computing the determinant over arbitrary fields. Our construction for fields of characteristic zero is based on Le Verrier's method, which does not easily transfer to other fields as it relies on division by arbitrarily large integers. There are general methods for simulating such division on small fields, but it is not immediately clear if they can be carried out symmetrically. However, there are many efficient ways of computing a determinant and it seems likely that some method that works on fields of positive characteristic could be implemented symmetrically. It should be noted, however, that Gaussian elimination is not such a method. Known results about the expressive power of fixed-point logic with counting (see, e.g. [9]) tell us that there is no polynomial-size family of symmetric circuits that can carry out Gaussian elimination. On the other hand, we do know that the determinant, even over finite fields, can be computed by exactly such a family of Boolean circuits, as shown by Holm [16]. It is when we restrict to *arithmetic* circuits, and also require symmetry, that the question is open.

The notions of symmetry used in our upper bound for the determinant and the lower bound for the permanent are slightly different. Essentially, we consider symmetric circuits for the determinant where we require that each *simultaneous* row and column permutation extend to an automorphism of the circuit, while for the permanent we require that each permutation generated by separate row and column permutations extend to an automorphism of the circuit. We could improve the result by showing that the lower bound for the permanent still holds even if we only require the circuits be symmetric with respect to simultaneous row and column permutations. We think this could be established by adapting our construction to analyse the counting width of the number of cycle covers of general graphs.

We could consider more general symmetries. For example, the determinant has other symmetries besides simultaneous row and column permutations. The construction we use already yields a circuit which is symmetric not only with respect to these but also transposition of rows and columns. We could consider a richer group that allowed for even permutations of the rows and columns. Could our upper bound be improved by constructing circuits for the determinant that are symmetric with respect to larger groups of permutations?

Finally, it is reasonable to think that there are polynomials in VP which do not admit polynomial-size symmetric arithmetic circuits, by analogy with the case of Boolean circuits. Can we give an explicit example of such a polynomial?

References

1 M. Ajtai. Recursive construction for 3-regular expanders. *Combinatorica*, 14:379–416, 1994.

2 M. Anderson and A. Dawar. On symmetric circuits and fixed-point logics. *Theory Comput. Syst.*, 60(3):521–551, 2017.

3 M. Anderson, A. Dawar, and B. Holm. Solving linear programs without breaking abstractions. *J. ACM*, 62, 2015.

4 V. Arvind, F. Fuhlbrück, J. Köbler, and O. Verbitsky. On Weisfeiler-Leman invariance: Subgraph counts and related graph properties. In *Fundamentals of Computation Theory – 22nd International Symposium, FCT 2019*, pages 111–125, 2019. doi:10.1007/978-3-030-25027-0_8.

5 A. Atserias, A. Dawar, and J. Ochremiak. On the power of symmetric linear programs. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS*, pages 1–13, 2019. doi:10.1109/LICS.2019.8785792.

6 W. Baur and V. Strassen. The complexity of partial derivatives. *Theor. Comput. Sci.*, 22, 1983. doi:10.1016/0304-3975(83)90110-X.

7 A. Blass, Y. Gurevich, and S. Shelah. On polynomial time computation over unordered structures. *Journal of Symbolic Logic*, 67(3):1093–1125, 2002.

8 J-Y. Cai, M. Fürer, and N. Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12(4):389–410, 1992.

9 A. Dawar. The nature and power of fixed-point logic with counting. *ACM SIGLOG News*, 2(1):8–21, 2015.

10 A. Dawar and D. Richerby. The power of counting logics on restricted classes of finite structures. In *CSL 2007:Computer Science Logic*, volume 4646 of *LNCS*, pages 84–98. Springer, 2007.

11 A. Dawar and P. Wang. Definability of semidefinite programming and Lasserre lower bounds for CSPs. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS*, 2017. doi:10.1109/LICS.2017.8005108.

12 A. Dawar and G. Wilsenach. Symmetric circuits for rank logic. In *27th EACSL Annual Conference on Computer Science Logic, CSL 2018*, pages 20:1–20:16, 2018.

13 L. Denenberg, Y. Gurevich, and S. Shelah. Definability by constant-depth polynomial-size circuits. *Information and Control*, 70:216–240, 1986.

14 M. A. Forbes, M. Kumar, and R. Saptharishi. Functional lower bounds for arithmetic circuits and connections to boolean circuit complexity. In *31st Conference on Computational Complexity, CCC 2016*, pages 33:1–33:19, 2016. doi:10.4230/LIPIcs.CCC.2016.33.

15 D. Grigoriev and M. Karpinski. An exponential lower bound for depth 3 arithmetic circuits. In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing*, pages 577–582, 1998. doi:10.1145/276698.276872.

16 B. Holm. *Descriptive Complexity of Linear Algebra.* PhD thesis, University of Cambridge, 2010.

17 M. Jerrum and M. Snir. Some exact complexity results for straight-line computations over semirings. *J. ACM*, 29:874–897, 1982. doi:10.1145/322326.322341.

18 N. Kayal and R. Saptharishi. A selection of lower bounds for arithmetic circuits. In M. Agrawal and V. Arvind, editors, *Perspectives in Computational Complexity.* Birkhäuser Basel, 2014.

19 J.M. Landsberg and N. Ressayre. Permanent v. determinant: An exponential lower bound assuming symmetry. In *Proc. ACM Conference on Innovations in Theoretical Computer Science*, pages 29–35. ACM, 2016. doi:10.1145/2840728.2840735.

20 M. Otto. The logic of explicitly presentation-invariant circuits. In *Computer Science Logic, 10th International Workshop, CSL '96, Annual Conference of the EACSL*, pages 369–384, 1996.

21 A. Shpilka and A. Wigderson. Depth-3 arithmetic circuits over fields of characteristic zero. *Computational Complexity*, 10:1–27, 2001. doi:10.1007/PL00001609.

22 A. Shpilka and A. Yehudayoff. Arithmetic circuits: A survey of recent results and open questions. *Foundations and Trends in Theoretical Computer Science*, 5(3-4):207–388, 2010. doi:10.1561/0400000039.

23 L. G. Valiant. Completeness classes in algebra. In *Proceedings of the 11h Annual ACM Symposium on Theory of Computing STOC*, pages 249–261, 1979. doi:10.1145/800135.804419.

24 H. Vollmer. *Introduction to Circuit Complexity – A Uniform Approach.* Texts in Theoretical Computer Science. An EATCS Series. Springer, 1999. doi:10.1007/978-3-662-03927-4.

An Efficient PTAS for Stochastic Load Balancing with Poisson Jobs

Anindya De
Department of Computer and Information Science,
University of Pennsylvania, Philadelphia, PA, USA
anindyad@cis.upenn.edu

Sanjeev Khanna
Department of Computer and Information Science,
University of Pennsylvania, Philadelphia, PA, USA
sanjeev@cis.upenn.edu

Huan Li
Department of Computer and Information Science,
University of Pennsylvania, Philadelphia, PA, USA
huanli@seas.upenn.edu

Hesam Nikpey
Department of Computer and Information Science,
University of Pennsylvania, Philadelphia, PA, USA
hesam@seas.upenn.edu

Abstract

We give the first polynomial-time approximation scheme (PTAS) for the stochastic load balancing problem when the job sizes follow Poisson distributions. This improves upon the 2-approximation algorithm due to Goel and Indyk (FOCS'99). Moreover, our approximation scheme is an efficient PTAS that has a running time double exponential in $1/\epsilon$ but nearly-linear in n, where n is the number of jobs and ϵ is the target error. Previously, a PTAS (not efficient) was only known for jobs that obey exponential distributions (Goel and Indyk, FOCS'99).

Our algorithm relies on several probabilistic ingredients including some (seemingly) new results on scaling and the so-called "focusing effect" of maximum of Poisson random variables which might be of independent interest.

2012 ACM Subject Classification Theory of computation → Discrete optimization; Theory of computation → Approximation algorithms analysis; Theory of computation → Scheduling algorithms

Keywords and phrases Efficient PTAS, Makespan Minimization, Scheduling, Stochastic Load Balancing, Poisson Distribution

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.37

Category Track A: Algorithms, Complexity and Games

Related Version `https://arxiv.org/abs/2006.12670`

Acknowledgements We thank the anonymous reviewers for their valuable feedback. This work was supported in part by NSF awards CCF-1617851, CCF-1763514, CCF-1910534, CCF-1926872, and CCF-1934876.

1 Introduction

We consider the following fundamental problem in scheduling theory: given n jobs with job sizes $w_1, \ldots, w_n \geq 0$, assign jobs to m machines such that the maximum load of any machine (i.e., the total size of jobs assigned to the machine) is minimized. In other words, we want to partition $[n]$ into sets $S_1, \ldots, S_m$ so as to minimize $\max_{i \in [m]} \sum_{j \in S_i} w_j$. Often referred to as *load balancing* or *makespan minimization*, this is one of the classical NP-complete problems and along with its many variants, has been extensively studied both in theoretical computer

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 37; pp. 37:1–37:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

science as well as operations research. While the exact problem is hard, this problem admits a polynomial time approximation scheme (PTAS) [12] and later work improved this to an efficient PTAS [16, 15, 14, 1]. Several other variants of this problem have also been studied – this includes (i) the *related machines case* where the machines can have different speeds [13]; (ii) the *unrelated machines case* where the job size itself depends on the machine on which it is scheduled [20]; and (iii) the *precedence constrained case* where there are precedence constraints on schedule of jobs [7, 5].

All the aforementioned variants of this problem have the common feature that the job sizes are known in advance to the algorithm designer. However, in many situations, there might be *uncertainty* in the job size. An obvious way to model this uncertainty is via the framework of stochastic optimization as follows – we have m machines and n jobs where the size of the i^{th} job is given by the random variable $\mathbf{W}_i$. If we now assign the jobs to m machines (given by $S_1, \ldots, S_m$), then the load of the j^{th} machine is given by the random variable $\sum_{i \in S_j} \mathbf{W}_i$. Similar to the case when the job sizes are *deterministic*, in *stochastic load balancing*, one would like to minimize the maximum load. However, since the maximum load (across machines) is itself a random variable – arguably, the most natural objective is to then minimize the expected maximum load. In other words, we seek to find a partition of $[n]$ into sets $S_1, \ldots, S_m$ so as to minimize $\mathbf{E}\left[\max_{j \in [m]} \sum_{i \in S_j} \mathbf{W}_i\right]$.

Throughout this paper, we assume that the random variables $\{\mathbf{W}_i\}_{i=1}^n$ are independent – that is the job sizes are independent of each other. Further, note that the algorithm designer is assumed to know the distribution of the random variables $\{\mathbf{W}_i\}_{i=1}^n$ (though, of course, not the actual realizations of the loads).

To our knowledge, Kleinberg, Rabani and Tardos [18] were the first to consider this problem in the algorithms community. They gave a $O(1)$-factor approximation algorithm for this problem. Soon thereafter, Goel and Indyk [9] considered the problem of obtaining better approximation for special classes of random variables – in particular, (i) if each $\{\mathbf{W}_i\}$ is an exponential random variable, they obtain a PTAS (though not an efficient one); (ii) if each $\{\mathbf{W}_i\}$ is a Poisson random variable, then they obtain a 2-approximation algorithm. In fact, this 2-approximation is obtained by considering the (deterministic) instance with loads $\{w_1, \ldots, w_n\}$ where $w_i = \mathbf{E}[\mathbf{W}_i]$ and then applying Graham's heuristic [10] on this instance.

Somewhat more complicated variants of this problem have also been considered – as an example, Gupta *et al.* [11] considered the problem of stochastic load balancing on unrelated machines. Here, the load of job i on machine j is given by a random variable $\mathbf{W}_{i,j}$. For this variant, [11] gave a $O(1)$ approximation algorithm (thus extending the guarantee of [18] to the case of unrelated machines). Similarly, Molinaro [22] considered the problem of minimizing the expected ℓ_p norm of the loads (the version we have can be seen as minimizing the expected ℓ_∞ norm of the loads). Despite all this impressive progress, the only case where we have a PTAS for stochastic load balancing is when all the loads $\{\mathbf{W}_i\}$ are exponential random variables. As the main result of this paper, we obtain an efficient PTAS for stochastic load balancing when all the loads are Poisson random variables.

▶ **Theorem 1.** *There is an algorithm* $\textsc{PoiScheduling}(n, m, \{\lambda_i\}_{i=1}^n, \epsilon)$ *that given an instance of the load balancing problem with n jobs and m machines where the size of the i^{th} job is* $\mathbf{W}_i = \mathsf{Poi}(\lambda_i)$ *(i.e. a Poisson random variable with mean λ_i), and a parameter $0 < \epsilon < 1$, outputs a job assignment whose expected maximum load satisfies $L \leq (1+\epsilon)L^*$, where L^* is the expected maximum load of an optimal assignment. The algorithm runs in time* $2^{2^{O(1/\epsilon^2)}} + O(n \log^2 n \log\log^2 n)$.

Theorem 1 is the first PTAS for stochastic load balancing with Poisson jobs. Prior to this result, the best known approximation algorithm for this setting was due to Goel and Indyk [9] (mentioned earlier) and had an approximation factor of 2. In fact, our PTAS is also an *efficient PTAS* – i.e., the running time remains polynomial in n even for some $\epsilon = o(1)$. In contrast, the PTAS from [9] for exponential random variables was *not* an efficient PTAS. Finally, we point out that our running time is doubly exponential in the error parameter ϵ. While this can be potentially improved to a singly exponential dependence in ϵ, it is unlikely to be improved further – in particular, [6] showed that under the ETH, any PTAS for even the deterministic load balancing problem must have a singly exponential dependence on ϵ[1].

1.1 Our techniques

At a high level, to design an algorithm for stochastic load balancing, we must come up with an *algorithmically tractable proxy* for the objective function $\mathbf{E}[\max_{j\in[m]} \sum_{i\in S_j} \mathbf{W}_i]$. However, the expected maxima of random variables (and more generally stochastic processes) can be notoriously difficult to reason about. Indeed, we point out that in the last fifty years, significant effort in probability theory has been devoted towards understanding the maximum of even simple families of random variables such as Gaussians [8, 24]. Despite this challenge, the hope is that by exploiting structural properties of Poisson random variables along with appropriate algorithmic primitives, we will be able to design an efficient PTAS for stochastic load balancing for Poisson jobs.

The starting points of our algorithm are two natural heuristics which have previously been analyzed in the context of stochastic load balancing.

1. The first heuristic is to construct an instance of (deterministic) load balancing where the size of the i^{th} job is $w_i = \mathbf{E}[\mathbf{W}_i]$. One can then apply the PTAS (say from [1]) to get an allocation of the n jobs into m machines. The obvious pitfall here is that the actual job size is a Poisson random variable which may typically be very far from its mean. In other words, this heuristic has a good guarantee provided

$$\mathbf{E}[\max_{j=1}^{m} \mathsf{Poi}(\mu_j)] \approx \max_{j=1}^{m}[\mathbf{E}\ [\mathsf{Poi}(\mu_j)]],$$

   where μ_j is the expected load size of the j^{th} machine[2]. Of course, the above relation may be far from true and indeed, we want to point out that while the left hand side $\mathbf{E}[\max_{j=1}^{m} \mathsf{Poi}(\mu_j)]$ is just a function of $\mu_1, \ldots, \mu_j$, it is far from being a linear function of $\mu_1, \ldots, \mu_j$. It is easy to create an instance where the optimum obtained by replacing each Poisson load by its expectation is a constant factor away from the true optimum.
   Despite this limitation, this heuristic is in fact of both theoretical and practical value. In particular, from a theoretical aspect, recall that $\mathsf{Poi}(\lambda)$ concentrates around λ (with standard deviation $\sqrt{\lambda}$). This can be leveraged to show that if the optimum allocation must necessarily have at least one machine with a (sufficiently) large load, then the allocation for the deterministic load balancing problem provides a near optimal allocation for the stochastic version as well.
2. The second heuristic is a *greedy algorithm* – namely, we first assign an arbitrary order to the jobs and *iteratively assign each job to the machine with the least current expected load.* This is the same as the Graham's rule [10], and is precisely how the authors of [9] obtained

[1] One can reduce an instance of deterministic load balancing to one of Poisson load balancing by scaling up all job sizes such that they all become at least $\omega(\epsilon^{-2}\log m)$. By Chernoff bounds and union bound this reduction preserves $(1+O(\epsilon))$-approximation.

[2] Note that $\mathsf{Poi}\,(\lambda_1) + \mathsf{Poi}\,(\lambda_2) = \mathsf{Poi}\,(\lambda_1+\lambda_2)$ if $\mathsf{Poi}\,(\lambda_1)$ and $\mathsf{Poi}\,(\lambda_2)$ are independent, so every machine's load is still a Poisson random variable.

a 2-approximation for load balancing Poisson jobs. The underlying rationale for this rule is the following cruical fact about Poisson random variables. Suppose $\mu_1 \geq \mu_2 \geq \mu_3 \geq \mu_4$ such that $\mu_1+\mu_4 = \mu_2+\mu_3$. Then, $\mathbf{E}[\max\{\mathsf{Poi}(\mu_1), \mathsf{Poi}\ (\mu_4)\}] \geq \mathbf{E}[\max\{\mathsf{Poi}(\mu_2), \mathsf{Poi}(\mu_3)\}]$. This fact can in fact be extended to prove that if there is an allocation such that the expected load is the same across all machines, then that is an optimum allocation. Of course, such an allocation might not exist – however, heuristically we might hope that if all the job sizes are small, then we can approximately equalize the expected load on the machines and that such an allocation might have a near-optimal expected maximum load.

It turns out that these heuristics (even when rigorously analyzed) are not sufficient to provide a PTAS for stochastic load balancing in all regimes of job sizes and (number of) machines. Therefore we need some other observations. The first crucial observation is that if there is a job size λ which is more than the average load (i.e. the total expected job size divided by m), then in the optimal allocation, such a job is assigned its own separate machine (Observation 17). This observation can be iteratively applied so that we are now left with job sizes $\{\lambda_i\}_{i=1}^{n'}$ and m' machines such that

$$\max_{i=1}^{n'}\{\lambda_i\} \leq \frac{\sum_{i=1}^{n'} \lambda_i}{m'} := \mu.$$

In other words, no job is larger than the average expected load across the m' machines, i.e., μ. With this simplification, we discuss another *familiar trick* in the context of allocation problems – namely we create a *rounded instance* such that each job size (now call it $\{\lambda'_i\}_{i=1}^{n'}$) is now in the interval $[\epsilon\mu, 2\mu]$. The rounding procedure we apply is identical to the rounding procedure used by [1] in the context of deterministic load balancing. A key property is that the number of different (expected) job sizes in this modified instance is a constant – i.e., only dependent on the target error parameter ϵ.

This rounding step highlights a key technical challenge our algorithm faces – namely, it is possible that by "multiplicatively dilating" the job sizes, the expected maximum load of the machines can change significantly. In other words, suppose $\mu_1, \ldots, \mu_m \geq 0$, then is it the case that for any $0 < \delta < 1$,

$$\mathbf{E}[\max_{j=1}^{m} \mathsf{Poi}((1+\delta)\mu_j)] \approx (1+O(\delta))\mathbf{E}[\max_{j=1}^{m} \mathsf{Poi}(\mu_j)]\ ? \qquad (1)$$

While intuitively this looks reasonable, it is not clear if this is true in full generality. Fortunately for us, we obtain the following dichotomy:

1. When μ is very large (this corresponds to the Case 1 in the analysis), we are able to show that the first heuristic above provides a PTAS – in other words, just substituting each stochastic job $\mathbf{W}_i$ with a deterministic job w_i such that $w_i = \mathbf{E}[\mathbf{W}_i]$ and then applying the PTAS for the deterministic case [1, 14, 15] gives a PTAS for the stochastic case. The underlying reason is that in this case, the expected maximum is essentially the same as the expected heaviest load across the m machines.

   The same algorithm also works if μ is in a "certain intermediate range" and m is sufficiently large (this corresponds to Case 3 in the analysis). In fact, in this case, even the greedy heuristic described earlier provides a PTAS. The underlying reason why the deterministic PTAS works is the following: in this regime, the expected maximum remains essentially the same even if all the loads were to go up by a factor of 2.
2. Outside of the above two cases, our heuristics (greedy or deterministic scheduling) fail to provably work. However, in these cases, we are able to prove (1). In other words, we are able to show that dilating or contracting each job size by a factor of $(1+\delta)$ affects

the expected maximum by only a factor of $1 \pm O(\delta)$. Thus, we can apply the rounding procedure from [1] to reduce to the case where the number of different job sizes is a constant. In fact, this is enough to obtain a PTAS for the stochastic load balancing problem though not an efficient PTAS.

Finally, to get an efficient PTAS, we leverage a third property of the "maximum of Poisson random variables" – namely, the so-called "focusing effect" [3, 2, 4]. Roughly speaking, it says that suppose we have m independent Poisson random variables (call them $\mathbf{X}_1, \ldots, \mathbf{X}_m$), each with mean μ, then there is an integer I such that $(\max_{i=1}^m \mathbf{X}_i) \in [I, I+1]$ with probability $1 - o(1)$ as $m \to \infty$. We extend this to (certain instances of) independent but not identically distributed Poisson random variables. Essentially such a "focusing effect", whenever it holds, allows us to express the expected maximum of the loads of m machines as a linear function of the allocation and then employ an integer linear program (ILP) to find the optimal allocation.

To explain how an ILP comes into the picture, first of all, we can assume that m (i.e., the number of machines) is sufficiently large in terms of the target error parameter δ. If this is not the case, then we can simply employ dynamic programming to find a good allocation (it is now an efficient PTAS because m is a constant). Once m is large, we show the following:

(a) Either there is a transition point $t = t(\mu_1, \ldots, \mu_m)$ such that $\max\{\mathsf{Poi}(\mu_1), \ldots, \mathsf{Poi}(\mu_m)\}$ sharply concentrates within $1 \pm O(\delta)$ of the transition point. In this case, we want to find the smallest t^*, for which there is an allocation with loads $\mu_1, \ldots, \mu_m$ such that $t^* = t(\mu_1, \ldots, \mu_m)$. We use ILP and binary search to find such t^*. Observe that in this case, the smallest such t^* will minimize the expected maximum load (up to $1 \pm O(\delta)$).

(b) Otherwise, there is a transition point $t = t(\mu, m)$ such that $\max\{\mathsf{Poi}(\mu_1), \ldots, \mathsf{Poi}(\mu_m)\}$ sharply concentrates in the set $[t-1, t]$. Observe that t only depends on μ and m, and hence can be easily computed. With the knowledge of t, we now use an ILP to find an assignment $\mu_1, \ldots, \mu_m$ which maximizes the probability that $\max\{\mathsf{Poi}(\mu_1), \ldots, \mathsf{Poi}(\mu_m)\} = t - 1$ and thus minimizes the expected maximum load.

1.2 Organization

In Section 2, we formally state the problem, establish some notations, and describe some properties of Poisson random variables that we will utilize. In Section 3 we give an overview of our concentration and scaling results for the maximum of Poisson random variables. In Section 4 we present our efficient polynomial-time approximation scheme and its analysis.

2 Preliminaries

We use $\mathsf{Poi}(\lambda)$ to denote a Poisson random variable with mean λ. Recall that $\mathbf{Pr}[\mathsf{Poi}(\lambda) = k] = e^{-\lambda} \cdot \frac{\lambda^k}{k!}$ for $k \in \mathbb{N}$. In the stochastic load balancing problem considered in this paper, we are given n jobs and m machines where the job sizes are independent Poisson random variables $\mathsf{Poi}(\lambda_1), \mathsf{Poi}(\lambda_2), \ldots, \mathsf{Poi}(\lambda_n)$. We will call λ_i the size of job i. Our goal is to assign the jobs to the machines so that the expected maximum load

$$L \stackrel{\text{def}}{=} \mathbf{E}\left[\max_{j=1}^{m} \sum_{i \in S_j} \mathsf{Poi}(\lambda_i)\right] \tag{2}$$

is minimized, where S_j is the set of jobs assigned to machine j.

It is well known that the sum of two independent Poisson random variables also follows a Poisson distribution, i.e. $\mathsf{Poi}\left(\lambda_1\right)+\mathsf{Poi}\left(\lambda_2\right) = \mathsf{Poi}\left(\lambda_1+\lambda_2\right)$. Therefore if we let $\mu_j = \sum_{i \in S_j} \lambda_i$, we can write (2) as $L = \mathbf{E}\left[\max_{j=1}^{m} \mathsf{Poi}\left(\mu_j\right)\right]$. We will call μ_j the load of machine j.

Henceforth, our analysis of Poisson random variables will mainly serve the purpose of characterizing the expected maximum load, and therefore we will use μ and $\mu_1, \mu_2, \ldots, \mu_m$ to denote the means when stating useful claims about Poisson distributions.

▶ **Definition 2.** *We write $\mathsf{M}(m, \mu)$ to denote the random variable whose value is the maximum of m i.i.d.* $\mathsf{Poi}\left(\mu\right)$*.*

In [9] the authors proved that Poisson distributions are log-concave:

▶ **Proposition 3** ([9])**.** *For any $t \geq 0$, the function*

$$f_t(\mu) = \log \mathbf{Pr}\left[\mathsf{Poi}\left(\mu\right) \leq t\right] \tag{3}$$

is decreasing and concave with respect to μ.

For any random variables $\mathbf{X}$ and $\mathbf{Y}$ taking values on $\mathbb{N}$, we say $\mathbf{X}$ *stochastically dominates* $\mathbf{Y}$, denoted by $\mathbf{X} \geq_{\mathrm{sd}} \mathbf{Y}$, if $\mathbf{Pr}\left[\mathbf{X} \geq k\right] \geq \mathbf{Pr}\left[\mathbf{Y} \geq k\right]$ holds for every $k \in \mathbb{N}$. Note that for independent $\mathbf{X}, \mathbf{Y}$ we have $\mathbf{Pr}\left[\max\left\{\mathbf{X}, \mathbf{Y}\right\} \geq k+1\right] = 1 - \mathbf{Pr}\left[\mathbf{X} \leq k\right]\mathbf{Pr}\left[\mathbf{Y} \leq k\right]$. Now by Proposition 3 we have the following:

▶ **Proposition 4** (Lemma 2.1 of [9])**.** *Given $0 \leq \mu_1 \leq \mu_1' \leq \mu_2' \leq \mu_2$ such that $\mu_1 + \mu_2 = \mu_1' + \mu_2'$, it holds that* $\max\left\{\mathsf{Poi}\left(\mu_1\right), \mathsf{Poi}\left(\mu_2\right)\right\} \geq_{\mathrm{sd}} \max\left\{\mathsf{Poi}\left(\mu_1'\right), \mathsf{Poi}\left(\mu_2'\right)\right\}$*.*

Poisson random variables satisfy exponential tail bounds:

▶ **Proposition 5** (Theorem 4.4, Theorem 4.5 of [21])**.** *Let $\mathbf{X}$ be a Poisson random variable with mean μ. For $0 < \delta < 1$ we have*

$$\mathbf{Pr}\left[\mathbf{X} \geq (1+\delta)\mu\right] \leq e^{-\mu\delta^2/3}, \tag{4}$$

$$\mathbf{Pr}\left[\mathbf{X} \leq (1-\delta)\mu\right] \leq e^{-\mu\delta^2/2}. \tag{5}$$

In our analysis we will need to use Stirling's approximation to deal with factorials:

▶ **Proposition 6** (Stirling's approximation [23])**.** *For any integer $n > 0$,*

$$e\left(\frac{n}{e}\right)^n \leq \sqrt{2\pi n}\left(\frac{n}{e}\right)^n e^{1/(12n+1)} \leq n! \leq \sqrt{2\pi n}\left(\frac{n}{e}\right)^n e^{1/12n} \leq en\left(\frac{n}{e}\right)^n. \tag{6}$$

3 Concentration and Scaling Results for Maximum of Poissons

In this section we present our concentration and scaling results for the maximum of independent Poisson random variables $\mathsf{Poi}\left(\mu_1\right), \mathsf{Poi}\left(\mu_2\right), \ldots, \mathsf{Poi}\left(\mu_m\right)$, which will be used to prove the correctness of our algorithm. Full proofs of these results are deferred to the full version of the paper.

Throughout we assume $\mu_1 \geq \mu_2 \geq \ldots \geq \mu_m \geq 0$, and define $\mu = (\sum_{j=1}^{m} \mu_j)/m$. We use δ as an error parameter, which measures how well the maximum of Poissons is concentrated. We consider five different cases based on the relationship between μ, m, and μ_j's, and state our results for each of them. Note that while the ranges of μ in these cases are disjoint, we prove our lemmas below for slightly overlapping ranges of μ for ease of analyzing our algorithm in Section 4.

Fix $\delta \in (0, 1/10]$. We prove our results for the following cases respectively:

Case 1: $\frac{6}{\delta^2} \log m < \mu$.

Case 2: $\frac{1}{2^{1/\delta+1}} \log m < \mu \le \frac{6}{\delta^2} \log m$ and $m \ge 2^{2^{\frac{2}{\delta}}}$.

Case 3: $\frac{1}{m^\delta} < \mu \le \frac{1}{2^{1/\delta+1}} \log m$, $m \ge 2^{\frac{2}{\delta} \log \frac{2}{\delta}}$, and $\forall j, \mu_j \in [\mu/4, 4\mu]$.

Case 4: $\frac{4 \log m}{m} < \mu \le \frac{1}{m^\delta}$, $m \ge 2^{100/\delta^2}$, and $\forall j, \mu_j \in [\mu/4, 4\mu]$.

Case 5: $\mu \le \frac{4 \log m}{m}$, $m \ge 1000(1/\delta) \log^2(1/\delta)$, and $\forall j, \mu_j \in [\mu/4, 4\mu]$.

For Case 1 we show that $\max_{j=1}^m \mathsf{Poi}(\mu_j)$ is concentrated within $(1 \pm O(\delta))\mu_1$. For each of Case 2, Case 3, and Case 4 we define a certain transition point and show that $\max_{j=1}^m \mathsf{Poi}(\mu_j)$ is concentrated around this point. For Case 5 we show that $\max_{j=1}^m \mathsf{Poi}(\mu_j)$ takes value 0 or 1 with high probability. For all cases we show that the maximum value is robust to contraction or dilation of μ_j's. In particular $\max_{j=1}^m \mathsf{Poi}(\mu_j)$ does not blow up by more than $1 + O(\delta)$ when we scale all μ_j's by $1 + \delta$. We call these *scaling results*.

▶ **Lemma 7** (Case 1). *Suppose $\delta \in (0, 1/10]$ and $\mu > \frac{6}{\delta^2} \log m$. Then*

$$\mu_1 \le \mathbf{E}\left[\max_{j=1}^m \mathsf{Poi}(\mu_j)\right] \le (1 + 5\delta)\mu_1. \tag{7}$$

▶ **Lemma 8** (Case 2). *Suppose $\delta \in (0, 1/10]$, $\frac{1}{2^{1/\delta+1}} \log m < \mu \le \frac{12}{\delta^2} \log m$, and $m \ge 2^{2^{\frac{2}{\delta}}}$. Define transition point $t_2 = t_2(\mu_1, \mu_1, \dots, \mu_m)$ as the largest integer satisfying*[3]

$$\sum_{j=1}^m \mathbf{Pr}\left[\mathsf{Poi}(\mu_j) \ge t_2\right] \ge \frac{1}{3}. \tag{8}$$

Then for any random variable $\mathbf{X}$ taking values on $\mathbb{N}$, we have

$$(1 - 6\delta)\mathbf{E}\left[\max\{t_2, \mathbf{X}\}\right] \le \mathbf{E}\left[\max\left\{\max_{j=1}^m \mathsf{Poi}(\mu_j), \mathbf{X}\right\}\right] \le (1 + 10\delta)\mathbf{E}\left[\max\{t_2, \mathbf{X}\}\right], \tag{9}$$

and

$$\mathbf{E}\left[\max\left\{\max_{j=1}^m \mathsf{Poi}((1+\delta)\mu_j), \mathbf{X}\right\}\right] \le (1 + 16\delta)\mathbf{E}\left[\max\left\{\max_{j=1}^m \mathsf{Poi}(\mu_j), \mathbf{X}\right\}\right]. \tag{10}$$

▶ **Lemma 9** (Case 3). *Suppose $\delta \in (0, 1/10]$, $\frac{1}{m^\delta} < \mu \le \frac{1}{2^{1/\delta+1}} \log m$, and $m \ge 2^{\frac{2}{\delta} \log \frac{2}{\delta}}$. Define transition point $t_3 = t_3(m, \mu) = \frac{\log m}{\log \frac{1}{\mu} + \log \log m}$. Then for any random variable $\mathbf{X}$ on $\mathbb{N}$ we have*

$$(1 - 4\delta)\max\{t_3, \mathbf{X}\} \le \max\{\mathsf{M}(m, \mu), \mathbf{X}\} \le (1 + 14\delta)\max\{t_3, \mathbf{X}\}, \tag{11}$$

and

$$\mathbf{E}\left[\max\{\mathsf{M}(m, 4\mu), \mathbf{X}\}\right] \le (1 + 20\delta)\,\mathbf{E}\left[\max\{\mathsf{M}(m, \mu), \mathbf{X}\}\right]. \tag{12}$$

[3] The choice of $\frac{1}{3}$ in the definition of t_2 is arbitrary. In principle any constant bounded away from both 1 and 0 suffices.

▶ **Lemma 10** (Case 4). *Suppose $\delta \in (0, 1/10]$, $\frac{4\log m}{m} < \mu \le \frac{2}{m^\delta}$, $m \ge 2^{100/\delta^2}$, and $\mu_1, \dots, \mu_m \in [\mu/4, 4\mu]$. Define transition point $t_4 = t_4(m, \mu) = \lceil \gamma_4(m, \mu) \rceil$ where $\gamma_4(m, \mu) = \frac{\log m}{\log \frac{4}{\mu} + \log\log m}$. Let $\mathbf{W}$ be a Bernoulli random variable taking values on $t_4 - 1$ and t_4 where $\mathbf{Pr}\left[\mathbf{W} = t_4 - 1\right] = \prod_{j=1}^m \mathbf{Pr}\left[\mathsf{Poi}\left(\mu_j\right) \le t_4 - 1\right]$. Then for any random variable $\mathbf{X}$ on $\mathbb{N}$ we have*

$$(1 - 5\delta)\mathbf{E}\left[\max\left\{\mathbf{W}, \mathbf{X}\right\}\right] \le \mathbf{E}\left[\max\left\{\max_{j=1}^m \mathsf{Poi}\left(\mu_j\right), \mathbf{X}\right\}\right] \le (1 + 16\delta)\mathbf{E}\left[\max\left\{\mathbf{W}, \mathbf{X}\right\}\right], \quad (13)$$

and

$$\mathbf{E}\left[\max\left\{\max_{j=1}^m \mathsf{Poi}\left((1+\delta)\mu_j\right), \mathbf{X}\right\}\right] \le (1 + 16\delta)\mathbf{E}\left[\max\left\{\max_{j=1}^m \mathsf{Poi}\left(\mu_j\right), \mathbf{X}\right\}\right]. \quad (14)$$

▶ **Lemma 11** (Case 5). *Suppose $\delta \in (0, 1/10]$, $\mu \le \frac{8\log m}{m}$, $m \ge 1000(1/\delta)\log^2(1/\delta)$, and $\forall j, \mu_j \in [\mu/4, 4\mu]$. Let $\mathbf{W}$ be a 0/1 Bernoulli random variable with $\mathbf{E}\left[\mathbf{W}\right] = 1 - \prod_{j=1}^m \mathbf{Pr}\left[\mathsf{Poi}\left(\mu_j\right) = 0\right] = 1 - e^{-m\mu}$. Then for any random variable $\mathbf{X}$ on $\mathbb{N}$ we have*

$$\max\left\{\mathbf{W}, \mathbf{X}\right\} \le \mathbf{E}\left[\max_{j=1}^m \mathsf{Poi}\left(\mu_j\right)\right] \le (1 + 10\delta)\max\left\{\mathbf{W}, \mathbf{X}\right\}, \quad (15)$$

and

$$\mathbf{E}\left[\max\left\{\max_{j=1}^m \mathsf{Poi}\left((1+\delta)\mu_j\right), \mathbf{X}\right\}\right] \le (1 + 10\delta)\,\mathbf{E}\left[\max\left\{\max_{j=1}^m \mathsf{Poi}\left(\mu_j\right), \mathbf{X}\right\}\right]. \quad (16)$$

4 An Efficient Polynomial-time Approximation Scheme

Our PTAS for stochastic load balancing is heavily inspired by the approach of [1, 15] for the deterministic load balancing problem. Thus, we first begin with a recap of their approach.

4.1 Recap of the PTAS for deterministic load balancing

Consider any instance of *deterministic* load balancing where the job sizes are $\{\lambda_i\}_{i=1}^n$ and we have m machines – the goal is to find an assignment with smallest maximum load. The algorithms in [1, 15] proceed in two phases: In phase I, we assign "big" jobs to separate machines. Here big jobs are the maximal set of jobs whose size is greater than the remaining average load. In other words, it is the maximal set $B \subseteq [n]$ satisfying that $\forall i \in B$, $\lambda_i > (\sum_{i \notin B} \lambda_i)/(m - |B|) := \mu$. Exploiting the convexity of the objective function (i.e. the function $\max\{x_1, x_2, \dots, x_m\}$), [1, 15] show that an optimum assignment (i) assigns the jobs in B to their own separate machines; (ii) assigns the remaining jobs to the remaining machines in a way such that each of these machines have a load between $\mu/2$ and 2μ.

With this, we are only left with the problem of assigning the small jobs, i.e., the jobs not in B. A second key step here is to round the sizes of the remaining jobs, such that (i) the number of different job sizes is now[4] $\tilde{O}(1/\epsilon)$ and (ii) the potential number of different assignments to any single machine is $2^{\tilde{O}(1/\epsilon)}$. Crucially, both these numbers are just dependent on the target error parameter ϵ. With this rounding, [1, 14] formulate the problem of finding an optimal assignment on the remaining (rounded) jobs as an integer linear program with

[4] Recall that $\tilde{O}(f)$ denotes $O(f \log^c f)$ for some constant c.

$2^{\tilde{O}(1/\epsilon)}$ variables – referred to as a *configuration-IP*. The *configuration-IP* can be solved in time exponential in the number of variables and linear in the input length using algorithms in [19, 17].

▶ **Theorem 12** ([17]). *There is an algorithm ILP that solves an integer linear program with p variables in time $p^{O(p)}O(n)$ where n is the length of input.*

This leads to an overall running time of $2^{2^{\tilde{O}(1/\epsilon)}} + O(n \log n)$ [1], where the running time $O(n \log n)$ comes from the pre-processing step of sorting the job sizes (to find the big jobs).

▶ **Theorem 13** ([1]). *There is an algorithm DetScheduling that given an instance of the load balancing problem with n jobs and m machines where the jobs have deterministic sizes $\lambda_1, \lambda_2, \ldots, \lambda_n$, and a parameter $0 < \epsilon < 1$, outputs a job assignment whose maximum load is at most $(1+\epsilon)$ of the maximum load of an optimum assignment. The algorithm runs in time $2^{2^{\tilde{O}(1/\epsilon)}} + O(n \log n)$.*

The authors in [15] then use a *sparsification technique* to show that the *configuration-IP* has an optimum solution with a small support size, which leads to an improved running time of $2^{O(1/\epsilon)\log^4(1/\epsilon)} + O(n \log n)$. Later by improving the runtime of solving the ILP, a running time of $2^{O(1/\epsilon)\log^2(1/\epsilon)} + O(n \log n)$ was achieved [16].

4.2 Overview of our approach

We now give an overview of our approach for the stochastic load balancing problem. Recall that we have n jobs and m machines where the i^{th} job has size $\mathsf{Poi}(\lambda_i)$. Similar to the deterministic case [1, 15], we define "big" jobs as the maximal set of jobs whose (expected) size is greater than the remaining (expected) average load, i.e. the maximal set $B \subseteq [n]$ satisfying that $\forall i \in B$

$$\lambda_i > \frac{\sum_{i \notin B} \lambda_i}{m - |B|}. \tag{17}$$

By Proposition 3, the objective function is convex with respect to the machine loads (similar to [1, 15]). Thus, we assign the jobs in B to separate machines (see Lines 2–5 of Algorithm 1).

Assigning the small jobs (i.e., the jobs outside B) is however somewhat more complicated. As stated in Section 1.1, there are two principal difficulties in applying the approaches from [1, 15] to handling the remaining jobs. One is to discretize the job sizes, and the other is to formulate the problem of minimizing the expected maximum load as an integer linear program. The key to circumventing both these difficulties lies in the (technical) results on concentration and scaling of maximum of Poisson random variables proven in Section 3. To understand their role, let us begin with some notation. Let $m^{(1)} = m - |B|$ denote the number of the remaining machines. Consider an assignment of the remaining jobs and let $\mathsf{Poi}(\mu_1), \mathsf{Poi}(\mu_2), \ldots, \mathsf{Poi}(\mu_{m^{(1)}})$ be the corresponding distributions of the machine loads. Suppose $\mu_1 \geq \mu_2 \geq \ldots \geq \mu_{m^{(1)}}$ and let $\mu = (\sum_{j=1}^{m^{(1)}} \mu_j)/m^{(1)}$. By an argument similar to the deterministic case [1, 15] (see Observation 18), we can restrict ourselves to the case when $\mu_j \in [\mu/2, 2\mu]$ for all $j \in [m^{(1)}]$. Let $\delta \in (0, 1)$ be a target error parameter (roughly speaking, we set $\delta \approx \Theta(\epsilon)$).

Our results in Section 3 first imply that when μ is sufficiently large in terms of $1/\delta$ and $m^{(1)}$ (the condition of Lemma 7), or μ is in a certain intermediate range but $m^{(1)}$ is large enough in terms of $1/\delta$ (the condition of Lemma 9), it suffices to find an assignment by running the deterministic load balancing algorithm in Theorem 13. When μ does not satisfy the above conditions but $m^{(1)}$ is sufficiently large in terms of $1/\delta$, we have the following dichotomy:

1. $\max_{j=1}^{m^{(1)}} \mathsf{Poi}\left(\mu_j\right)$ is concentrated within $(1 \pm O(\delta))$ of a certain transition point $t = t(\mu_1, \ldots, \mu_{m^{(1)}})$ (Lemma 8), or
2. takes value $\lceil t \rceil - 1$ or $\lceil t \rceil$ with high probability, where the transition point $t = t(m^{(1)}, \mu)$ only depends on $m^{(1)}$ and μ (Lemmas 10 and 11).

Further, in all of the cases,

$$\mathbf{E}\left[\max_{j=1}^{m} \mathsf{Poi}\left((1+\delta)\mu_j\right)\right] \leq (1+O(\delta))\mathbf{E}\left[\max_{j=1}^{m} \mathsf{Poi}\left(\mu_j\right)\right] \tag{18}$$

(Lemmas 8, 10, and 11). Finally, when m is sufficiently small in terms of δ, we can use dynamic programming to get an efficient PTAS.

Let us now see how the above structural results are useful in algorithm design − first of all, (18) immediately allows us to discretize the job sizes by rounding up their means to the nearest integer power of $(1 + \delta)$, with proper handling of jobs with size below a certain threshold[5]. Once the job sizes are rounded, the existence of the transition points (rather, the precise definitions of these transition points in Section 3) can be used to construct integer linear programs which can find a near optimal solution. As an example, if we are in Case 4 as defined above, by Lemma 10 minimizing the expected maximum is the same as finding $\mu_1, \mu_2, \ldots, \mu_{m^{(1)}}$ such that the probability that the maximum load is $\lceil t \rceil$ is minimized. This finishes our overview of the PTAS.

4.3 Our PTAS

We give a full description of our efficient PTAS in Algorithm 1 as PoiScheduling, which calls Rounded (Algorithms 2), ILPScheduling (Algorithm 3), and DPScheduling as subroutines. The detailed description of DPScheduling is deferred to the full version as it uses mostly standard ideas.

PoiScheduling first handles the "big" jobs in the same way as deterministic case (Lines 2–5 of Algorithm 1). As before μ denotes the remaining average machine load (i.e. RHS of (17)) and $m^{(1)}$ denote the number of the remaining machines. PoiScheduling then does one of the following for the remaining jobs:

1. When μ is large enough to meet the condition of Case 1, or μ and $m^{(1)}$ meet the condition of Case 3, PoiScheduling directly uses the algorithm for deterministic case as a blackbox (Lines 7–9 of Algorithm 1).
2. When μ and $m^{(1)}$ meet the condition of Case 2, Case 4, or Case 5, PoiScheduling first rounds the sizes of the remaining jobs in the same way as [15] (Line 6 of Algorithm 1). Then for Case 2 it uses integer linear programming in conjunction with a binary search to find the smallest transition point t_2 achievable by an assignment of the remaining jobs, where the ILPs have linear objective functions and *configuration-IP* from [1, 15] as feasibility constraints (Lines 10–12 of Algorithm 1). Case 4 and Case 5 are also handled by integer linear programs (Lines 13-15 and Lines 16-17 of Algorithm 1 respectively).
3. When none of the conditions of Case 1 - Case 5 is met, namely $m^{(1)} \leq 2^{2^{O(1/\epsilon)}}$ and $\mu \leq O(\epsilon^{-2} \log m)$, PoiScheduling finds an assignment by dynamic programming (Line 19 of Algorithm 1).

PoiScheduling uses Rounded, ILPScheduling and DPScheduling as subroutines. Roughly speaking, Rounded takes a multi-set of job sizes and a parameter δ as input and outputs a multi-set of job sizes such that the number of different job sizes only depends

[5] The specific rounding scheme we use is identical to the one used in [15].

on δ. ILPScheduling takes the jobs and the number of machines m, and a function f as input. The goal of ILPScheduling is to find an assignment with loads $\mu_1, \dots, \mu_m$ such that $\sum_{j=1}^{m} f(\mu_j)$ is minimized. DPScheduling takes the jobs and number of machine, and an error parameter ϵ as input and returns an assignment of jobs with at most $(1+\epsilon)$ error with respect to an optimum assignment by a dynamic programming.

By using the algorithm in [17] (Theorem 12) to solve the integer linear programs, PoiScheduling achieves a running time double exponential in $1/\epsilon$ and nearly-linear in n. Note that although the algorithm in Theorem 12 needs integral coefficients, we can compute the coefficients with a high precision (inverse polynomial precision) which is sufficient to get our desired approximation and does not affect our running time, so we don't get into the details. We also note that while it is possible to use the sparsification technique in [15] to improve our running time of solving integer linear programs to single exponential in $1/\epsilon$, our dynamic program for the case when $1/\epsilon \geq \Omega(\log\log m)$ still takes time double exponential in $1/\epsilon$.

The performance of PoiScheduling is characterized in Theorem 1. The performances of Rounded and DPScheduling are characterized in Lemmas 14 and 15 respectively. We do not give a separate lemma for ILPScheduling but analyze it in our proofs directly.

Note that Lemma 14 below only gives guarantees for how the job sizes and individual machine loads change after rounding; the lemma itself does not make assertions about the expected maximum load. Instead, guarantees for the latter will follow from our scaling results in Section 3.

▶ **Lemma 14** ([15]). *The algorithm $\{\lambda'_i\}_{i=1}^{n'} = \textsc{Rounded}(\{\lambda_i\}_{i=1}^{n}, \mu, \delta)$ runs in time $O(n)$. Suppose all $\lambda_i \leq \mu$, $\delta \in (0,1)$, and $\mu = (\sum_{i=1}^{n} \lambda_i)/m$ for an integer m. The number of different sizes in $\{\lambda'_i\}_{i=1}^{n'}$ is bounded by $O(\frac{1}{\delta}\log\frac{1}{\delta})$, and each $\lambda'_i = \delta\mu + k\delta^2\mu$ for some $k \in \mathbb{Z}_{\geq 0}, k \leq \frac{2}{\delta^2}$. n' is bounded by $O(m/\delta)$. For any assignment of $\{\lambda_i\}_{i=1}^{n}$ to m machines with loads $\mu_1, \mu_2, \dots, \mu_m$, there is an assignment of $\{\lambda'_i\}_{i=1}^{n'}$ to m machines with loads $\mu'_1, \mu'_2, \dots, \mu'_m$ such that all $\mu'_j \leq (1+5\delta)\mu_j$. Conversely, for any assignment $\{\lambda'_i\}_{i=1}^{n'}$ to m machines with loads $\mu'_1, \mu'_2, \dots, \mu'_m$, there is an assignment of $\{\lambda_i\}_{i=1}^{n}$ to m machines with loads $\mu_1, \mu_2, \dots, \mu_m$ such that all $\mu_j \leq (1+5\delta)\mu'_j$, and the latter assignment can be found in $O(n)$ time given the former assignment if both $\{\lambda_i\}_{i=1}^{n}$ and $\{\lambda'_i\}_{i=1}^{n'}$ are sorted.*

▶ **Lemma 15.** *Given jobs $\{\nu_i\}_{i=1}^{n^{(0)}}$, $\{\lambda_i\}_{i=1}^{n^{(1)}}$, number of machines m, and $\epsilon \in (0, \frac{1}{10}]$. Let $m^{(1)} = m - n^{(0)}$, $\mu = (\sum_{i=1}^{n^{(1)}} \lambda_i)/m^{(1)}$, and $\delta = \max\left\{\frac{\epsilon}{1000m^{(1)}}, 2^{-10^9/\epsilon^2}\right\}$. Suppose all $\lambda_i \leq \mu$, all $\nu_i > \mu$, and $\mu \leq 6000000\epsilon^{-2}\log m^{(1)}$. Then $\textsc{DPScheduling}(\{\nu_i\}_{i=1}^{n^{(0)}}, \{\lambda_i\}_{i=1}^{n^{(1)}}, m, \epsilon)$ finds in $O(n^{(0)}\epsilon^{-4}\log^2 m^{(1)}) + (m^{(1)}/\delta)^{O(\frac{1}{\delta}\log\frac{1}{\delta})}$ time an assignment with expected maximum load $L \leq (1+\epsilon)L^*$, where L^* is the expected maximum load of an optimum assignment.*

A detailed proof of Lemma 15 is deferred to the full version of the paper. First we show how to prove Theorem 1 using the lemma.

4.4 Proof of Theorem 1

The following lemma shows that if Algorithm 1 does not use dynamic programming to find an assignment, then we have a good approximation.

▶ **Lemma 16.** *If Algorithm 1 returns an assignment without going into Line 19, then $L \leq (1+\epsilon)L^*$, where L, L^* is the expected maximum load of the assignment returned by Algorithm 1 and the optimum assignment respectively.*

Algorithm 1 $\textsc{PoiScheduling}(n, m, \{\lambda_i\}_{i=1}^{n}, \epsilon)$.

```
Input  : Number of jobs $n$, number of machines $m$, job sizes $\{\lambda_i\}_{i=1}^{n}$, and $\epsilon \in (0,1)$
Output : A job assignment $\phi : [n] \to [m]$
$\mu \leftarrow (\sum_{i=1}^{n} \lambda_i)/m$, and sort $\{\lambda_i\}_{i=1}^{n}$ such that $\lambda_1 \le \lambda_2 \le \ldots \le \lambda_n$.
$n^{(1)} \leftarrow n$ and $m^{(1)} \leftarrow m$.
while $\lambda_{n^{(1)}} > \mu$ do
    Assign the job with size $\lambda_{n^{(1)}}$ to the empty machine $m^{(1)}$: $\phi(n^{(1)}) \leftarrow m^{(1)}$.
    $\mu \leftarrow \frac{m^{(1)}\mu - \lambda_{n^{(1)}}}{m^{(1)}-1}$, $n^{(1)} \leftarrow n^{(1)} - 1$, and $m^{(1)} \leftarrow m^{(1)} - 1$.
$\delta \leftarrow \frac{\epsilon}{1000}$ and $\{\lambda'_i\}_{i=1}^{n'} \leftarrow \textsc{Rounded}(\{\lambda_i\}_{i=1}^{n^{(1)}}, \mu, \delta)$.
if $\mu > \frac{6}{\delta^2} \log m^{(1)}$ or $\left(\frac{1}{(m^{(1)})^{\delta}} < \mu \le \frac{1}{2^{1/\delta+1}} \log m^{(1)}$ and $m^{(1)} \ge 2^{\frac{2}{\delta}\log\frac{2}{\delta}}\right)$ then
    // Case 1, Case 3
    Create a new instance with $m^{(1)}$ machines and deterministic job sizes $\{\lambda_i\}_{i=1}^{n^{(1)}}$.
    Run $\textsc{DetScheduling}$ in Theorem 13 to find a $(1 + \frac{\epsilon}{5})$-optimum assignment
      $\phi' : [n^{(1)}] \to [m^{(1)}]$.
else if $\frac{1}{2^{\delta+1}} \log m^{(1)} < \mu \le \frac{6}{\delta^2} \log m^{(1)}$ and $m^{(1)} \ge 2^{2^{2/\delta}}$ then
    // Case 2
    Use binary search to find the smallest $t_2 \in [\mu, 100\mu \log m^{(1)}]$ s.t. there is an
      assignment of $\{\lambda'_i\}_{i=1}^{n'}$ with loads $\{\mu_j\}_{j=1}^{m^{(1)}}$ s.t. $\sum_{j=1}^{m^{(1)}} \mathbf{Pr}[\mathsf{Poi}(\mu_j) > t_2] < \frac{1}{3}$; this
      is by $(\mathtt{opt}, \phi') \leftarrow \textsc{ILPScheduling}(\{\lambda'_i\}_{i=1}^{n'}, m^{(1)}, x \to \mathbf{Pr}[\mathsf{Poi}(x) > t_2])$ for
      each guess of $t_2$ and checking if $\mathtt{opt} < \frac{1}{3}$.
    $(\mathtt{opt}, \phi') \leftarrow \textsc{ILPScheduling}(\{\lambda'_i\}_{i=1}^{n'}, m^{(1)}, x \to \mathbf{Pr}[\mathsf{Poi}(x) > t_2])$.
else if $\frac{4 \log m^{(1)}}{m^{(1)}} < \mu \le \frac{1}{(m^{(1)})^{\delta}}$ and $m^{(1)} \ge 2^{100/\delta^2}$ then
    // Case 4
    $t_4 \leftarrow \left\lceil \frac{\log m^{(1)}}{\log\frac{4}{\mu} + \log\log m^{(1)}} \right\rceil$.
    Find an assignment of $\{\lambda'_i\}_{i=1}^{n'}$ with loads $\{\mu_j\}_{j=1}^{m^{(1)}}$ s.t. $\prod_{j=1}^{m^{(1)}} \mathbf{Pr}[\mathsf{Poi}(\mu_j) < t_4]$ is
      maximized, by
      $(\mathtt{opt}, \phi') \leftarrow \textsc{ILPScheduling}(\{\lambda'_i\}_{i=1}^{n'}, m^{(1)}, x \to -\ln(\mathbf{Pr}[\mathsf{Poi}(x) < t_4]))$.
else if $\mu \le \frac{4 \log m^{(1)}}{m^{(1)}}$ and $m^{(1)} \ge 1000(1/\delta)\log^2(1/\delta)$ then
    // Case 5
    Find an assignment of $\{\lambda'_i\}_{i=1}^{n'}$ with loads $\{\mu_j\}_{j=1}^{m^{(1)}}$ s.t. $\prod_{j=1}^{m^{(1)}} \mathbf{Pr}[\mathsf{Poi}(\mu_j) = 0]$ is
      maximized, by
      $(\mathtt{opt}, \phi') \leftarrow \textsc{ILPScheduling}(\{\lambda'_i\}_{i=1}^{n'}, m^{(1)}, x \to -\ln(\mathbf{Pr}[\mathsf{Poi}(x) = 0]))$.
else
    $\phi' \leftarrow \textsc{DPScheduling}(\{\lambda_i\}_{i=n^{(1)}+1}^{n}, \{\lambda_i\}_{i=1}^{n^{(1)}}, m, \epsilon)$.
If $\phi'$ is an assignment of the rounded jobs $\{\lambda'_i\}_{i=1}^{n'}$, use Lemma 14 to covert it to an
  assignment of $\{\lambda_i\}_{i=1}^{n^{(1)}}$ such every machine's load overflows by no more than $(1 + 5\delta)$.
Return the assignment $\phi \cup \phi'$.
```

Algorithm 2 Rounded($\{\lambda_i\}_{i=1}^n$, μ, δ).

```
Input  : Job sizes $\{\lambda_i\}_{i=1}^n$, $\mu$ s.t. all $\lambda_i \le \mu$, and $\delta \in (0,1)$
Output : Rounded job sizes $\{\lambda'_i\}_{i=1}^{n'}$
$n' \leftarrow 0$ and $S \leftarrow 0$.
for $i \leftarrow 1$ to $n$ do
    if $\lambda_i \ge \delta\mu$ then
        $n' \leftarrow n' + 1$.
        $\nu \leftarrow (1+\delta)^k \delta\mu$ where $k$ is the unique integer s.t.
          $(1+\delta)^{k-1}\delta\mu < \lambda_i \le (1+\delta)^k \delta\mu$.
        $\lambda_{n'} \leftarrow l\delta^2\mu$ where $l$ is the unique integer s.t. $(l-1)\delta^2\mu < \nu \le l\delta^2\mu$.
    else
        $S \leftarrow S + \lambda_i$.
$S^{\#} \leftarrow k\delta\mu$ where $k$ is the unique integer s.t. $(k-1)\delta\mu < S \le k\delta\mu$.
$\lambda'_i \leftarrow \delta\mu$ for each $i = n'+1, \dots, n' + S^{\#}/(\delta\mu)$ and $n' \leftarrow n' + S^{\#}/(\delta\mu)$.
return $\{\lambda'_i\}_{i=1}^{n'}$
```

Algorithm 3 ILPScheduling($\{\lambda_i\}_{i=1}^n$, m, f).

```
Input  : Job sizes $\{\lambda_i\}_{i=1}^n$, number of machines $m$, and a function $f : \mathbb{R} \to \mathbb{R}$
Output : An optimum value opt and a job assignment $\phi : [n] \to [m]$
$\mu \leftarrow (\sum_{i=1}^n \lambda_i)/m$.
Let $\pi_1 < \pi_2 < \dots < \pi_d$ be all different sizes in $\{\lambda_i\}_{i=1}^n$ and $\vec{\pi} \leftarrow (\pi_1, \pi_2, \dots, \pi_d)^T$.
Let $n_k$ be the number of jobs with size $\pi_k$ and $\vec{n} \leftarrow (n_1, n_2, \dots, n_d)^T$ (so
  $\sum_{k=1}^d n_k = n$).
$Q \leftarrow \left\{\vec{c} \in \mathbb{Z}_{\ge 0}^d : \vec{c}^T\vec{\pi} \le 4\mu\right\}$ (the set of possible assignments to a single machine).
Solve the following integer linear programming using Theorem 12:

    $\min \sum_{\vec{c} \in Q} x_{\vec{c}}\, f(\vec{c}^T\vec{\pi})$
    $s.t. \sum_{\vec{c} \in Q} x_{\vec{c}} = m$
          $\sum_{\vec{c} \in Q} x_{\vec{c}}\vec{c} = \vec{n}$
          $x_{\vec{c}} \in \mathbb{Z}_{\ge 0}, \ \forall \vec{c} \in Q.$                    (19)

Return the optimum value of the ILP and an assignment $\phi$ extracted from its
  solution.
```

We will provide a detailed proof of Lemma 16 in the next section. Roughly speaking, for Case 1 and Case 3, where we do not use the rounded jobs, using the guarantee of the deterministic load balancing algorithm we show that the maximum expected load of any machine is bounded. Then by using the concentration results from Section 3, we can bound the expected maximum load.

For three other cases, where the assignment obtained is for the rounded jobs, the proof consists of two main steps. In the first step, we compare the individual machine loads of the output assignment (of unrounded jobs) and the assignment of the rounded jobs given by ILPScheduling by Lemma 14. Then by the concentration and scaling results from Section 3, we bound the expected maximum load of the output assignment using the transition point of rounded jobs. In the second step, we bound the expected maximum of an optimum assignment of rounded jobs by that of an optimum assignment of unrounded jobs by scaling results from Section 3. Thus, we bound the expected maximum load of an output assignment by that of an optimum assignment.

Now we prove Theorem 1.

Proof of Theorem 1. First we analyze the running time when PoiScheduling does not use dynamic programming. If the algorithm invokes the efficient PTAS for deterministic case in Theorem 13, the running time is bounded by $2^{2^{\tilde{O}(1/\epsilon)}} + O(n \log n)$. Otherwise the algorithm will first do the rounding. By Lemma 14 after the rounding the number of different job sizes in $\{\lambda'_i\}_{i=1}^{n'}$ becomes $O(\frac{1}{\delta} \log \frac{1}{\delta}) = O(\frac{1}{\epsilon} \log \frac{1}{\epsilon})$ and all job sizes are between $\delta\mu$ and 2μ. Now for each machine of $1, 2, \ldots, m^{(1)}$, we only need to consider the assignments to it with load at most 4μ. Therefore each machine can have at most $O(1/\delta) = O(1/\epsilon)$ jobs, and the number of different job profiles that can be assigned to one machine is at most $(1/\epsilon)^{O(1/\epsilon)} = 2^{O(\frac{1}{\epsilon} \log \frac{1}{\epsilon})}$, which is the number of variables in the ILP. By Theorem 12 the ILP can be solved in $2^{2^{O(\frac{1}{\epsilon} \log \frac{1}{\epsilon})}} O(\log n)$ time. Since for Case 2 we need to do a binary search on interval $[\mu, 100\mu \log m^{(1)}]$, the total running time is bounded by $2^{2^{O(\frac{1}{\epsilon} \log \frac{1}{\epsilon})}} O(\log n \log \log n)$.

If PoiScheduling uses dynamic programming, we have $\delta \geq 2^{-10^9/\epsilon^2}$ and $m^{(1)} \leq 2^{2^{O(1/\epsilon)}}$. Therefore the total running time is bounded by $2^{2^{O(1/\epsilon^2)}} + O(n\epsilon^{-4} \log^2 n)$ by Lemma 15. Combining these two cases gives us the desired running time.

The approximation guarantee directly follows from Lemmas 16 and 15. ◀

5 Proof of Correctness using Concentration Results

In this section, we will prove the correctness of Algorithm 1 using the concentration results in Section 3. We refer to the case in Lemma 16 as the case with "large m" and the other case as the one with "small m". Here we only give the proof for large m but defer the proof for small m to the full version of the paper.

5.1 Analysis for large m

In this subsection, we prove that if Algorithm 1 does not call DPScheduling (Line 19), then $L < (1 + \epsilon)L^*$ where L^* is the expected maximum load of an optimum assignment and L is the expected maximum load of the returned assignment by Algorithm 1. In our analysis, by large m we mean that the condition on Line 7, 10, 13, or 16 is satisfied. By small m we mean that none of those conditions is satisfied, i.e. the algorithm calls DPScheduling. In this subsection, we will also consider the case when the algorithm finishes assigning all jobs before Line 6.

▶ **Lemma 16.** *If Algorithm 1 returns an assignment without going into Line 19, then $L \leq (1 + \epsilon)L^*$, where L, L^* is the expected maximum load of the assignment returned by Algorithm 1 and the optimum assignment respectively.*

Proof of Lemma 16. Let χ be the set of jobs each of which is assigned to a single machine in the first loop of Algorithm 1. Let Λ be the (multi-)set of the sizes of the jobs in χ. Let $\boldsymbol{\chi} = \{\mathsf{Poi}\,(\lambda)\,|\lambda \in \Lambda\}$ and $\mathbf{X} = \max_{\lambda\in\Lambda} \mathsf{Poi}\,(\lambda)$. Consider μ and $m^{(1)}$ as same as Algorithm 1 after Line 6. Suppose the algorithm returns an assignment where machine j has load μ_j. Let $\left\{\mu_j^{(1)}\right\}_{j=1}^{m^{(1)}}$ be the machine loads corresponding to the assignment ϕ' obtained *before* Line 20.

That is, if ϕ' is an assignment of the rounded jobs, $\mu_j^{(1)}$'s are the corresponding rounded loads. Without loss of generality, we assume $\mu_1^{(1)} \geq \mu_2^{(1)} \geq \ldots \geq \mu_{m^{(1)}}^{(1)}$. Let $\mu^{(1)} = \frac{\sum_{j=1}^{m^{(1)}} \mu_j^{(1)}}{m(1)}$. Note that if Algorithm 1 does not return an assignment using rounded jobs (Line 9), then $\mu = \mu^{(1)}$. There are six possible cases:

Case 0: The algorithm finishes assigning all jobs before Line 6. Then it must be the case that $n \leq m$. Therefore assigning every job to a single machine is the optimum solution.

Case 1: $\mu > \frac{6}{\delta^2} \log m^{(1)}$ and the algorithm goes to Lines 7–9. First we state an observation. Note that this is the same observation as **Observation 2.1** in [1] for deterministic case. Both our observation and the one in [1] follow from the convexity of the objective function with respect to the machine loads.

▶ **Observation 17.** *There is an optimum assignment such that each job in χ is solely assigned to a separate machine.*

Proof. Suppose in an optimum assignment there is a job in χ with size λ_i which is assigned to a machine with another job of size λ_j and suppose i is the biggest index such that a job with size λ_i shares its machine with another job. Let μ' be the value of μ in Algorithm 1 just before assigning λ_i to a machine, so we know $\lambda_i > \mu'$. By removing λ_j from its machine, all the other machines should have load at least μ', otherwise we can assign λ_j to a machine with load less than μ' and by Proposition 4, the expected maximum decreases which contradicts the optimality. But μ' is the average load of remaining machines (machines that do not contain any jobs from $\{\lambda_k | k > i\}$) and now each one has load more than μ', a contradiction. ◀

Let L^*_{DET} be the optimum answer of deterministic case for job sizes of $\{\lambda_i\}_{i=1}^{n^{(1)}}$ and $m^{(1)}$ machines, where $n^{(1)} = n - |\chi|$. Note that $L^* \geq L^*_{DET}$ and $L^* \geq \max_{\lambda \in \Lambda} \lambda$, by Observation 17. Since $\delta = \frac{\epsilon}{1000}$, by Lemma 7 we get:

$$\begin{aligned} L &= \mathbf{E}\left[\max\left\{\max_{j=1}^{m^{(1)}} \mathsf{Poi}\left(\mu_j\right), \mathbf{X}\right\}\right] \\ &\leq \mathbf{E}\left[\max\left\{\mathsf{Poi}\left((1+\epsilon/5)L^*_{DET}\right), \max_{j=2}^{m^{(1)}} \mathsf{Poi}\left(\mu_j\right), \mathbf{X}\right\}\right] \qquad \text{(by Theorem 13)} \\ &\leq (1+5\delta)\max\left\{(1+\epsilon/5)L^*_{DET}, \max_{\lambda\in\Lambda}\lambda\right\} \qquad \text{(by (7) in Lemma 7)} \\ &\leq (1+\epsilon/200)(1+\epsilon/5)L^* < (1+\epsilon)L^*. \qquad (20) \end{aligned}$$

Case 2: The algorithm goes to Lines 10–12. By calling ILPScheduling multiple times, Algorithm 1 finds the smallest transition point t_2 for rounded jobs on $m^{(1)}$ machines as defined in Lemma 8. Note that if $\hat{L}$ is the optimum expected maximum for rounded version, then $\hat{L} \leq (1+80\delta)L^*$ (by (10) in Lemma 8 and Lemma 14). Since $\delta = \frac{\epsilon}{1000}$, $m^{(1)} > 2^{2^{100/\epsilon}}$ and $\frac{1}{2^{1/\delta+1}} \log m^{(1)} < \mu^{(1)} \log m^{(1)} \leq (1+5\delta)\frac{6}{\delta^2} \log m^{(1)} < \frac{12}{\delta^2} \log m^{(1)}$ (Lemma 14), then by Lemma 8 we get:

$$\begin{aligned} L &= \mathbf{E}\left[\max\left\{\max_{j=1}^{m^{(1)}} \mathsf{Poi}\left(\mu_j\right), \mathbf{X}\right\}\right] \\ &\leq \mathbf{E}\left[\max\left\{\max_{j=1}^{m^{(1)}} \mathsf{Poi}\left((1+5\delta)\mu_j^{(1)}\right), \mathbf{X}\right\}\right] \qquad \text{(Lemma 14)} \end{aligned}$$

$$\begin{aligned}
\dots &\le (1+80\delta)\mathbf{E}\left[\max\left\{\max_{j=1}^{m^{(1)}}\mathsf{Poi}\left(\mu_j^{(1)}\right),\mathbf{X}\right\}\right] \quad \text{(by (10) in Lemma 8)}\\
&\le (1+10\delta)(1+80\delta)\mathbf{E}\left[\max\{t_2,\mathbf{X}\}\right] \quad \text{(by (9) in Lemma 8)}\\
&\le \frac{(1+10\delta)(1+80\delta)}{1-6\delta}\hat{L} \quad \text{(by (9) in Lemma 8)}\\
&\le \frac{(1+10\delta)(1+80\delta)^2}{1-6\delta}L^*\\
&< (1+\epsilon)L^*
\end{aligned}$$

Case 3: $\frac{1}{(m^{(1)})^\delta} < \mu \le \frac{1}{2^{1/\delta+1}}\log m^{(1)}$ and $m^{(1)} \ge 2^{\frac{2}{\delta}\log\frac{2}{\delta}}$, and the algorithm goes to Lines 7–9. First we state an observation which is the same as **Observation 2.2** in [1] for deterministic case and again follows from the convexity of the objective function with respect to the machine loads

▶ **Observation 18.** *If for each i, $\lambda_i \le \mu$, then for each load μ_j in an optimum assignment we have $\mu/2 \le \mu_j \le 2\mu$.*

Proof. Suppose a machine has load $\mu_j > 2\mu$ and a job assigned to the machine is λ_i. $\lambda_i \le \mu$, so $\mu_j - \lambda_i > \mu$, so any other machines has load at least μ, a contradiction. So $\mu_j \le 2\mu$.

Suppose a machine has load $\mu_j < \mu/2$. So there is a machine with load $\mu_k > \mu$, so it has at least two jobs assigned. So there is a job λ_i in μ_k such that $\lambda_i \le \mu_k/2$, by taking it from μ_k its load would be $\mu_k - \lambda_i \ge \mu_k/2 > \mu/2 > \mu_j$, a contradiction as by reassigning λ_i to μ_j by Proposition 4 the expected maximum decreases. So $\mu/2 \le \mu_j$. ◀

Since $\delta = \frac{\epsilon}{1000}$ and $m^{(1)} > 2^{\frac{2}{\delta}\log\frac{2}{\delta}}$, the conditions of Lemma 9 holds and we get:

$$\begin{aligned}
L &= \mathbf{E}\left[\max\left\{\max_{j=1}^{m^{(1)}}\mathsf{Poi}(\mu_j),\mathbf{X}\right\}\right]\\
&\le \mathbf{E}\left[\max\left\{\mathsf{M}(m^{(1)},(1+\epsilon/5)2\mu),\mathbf{X}\right\}\right] \quad \text{(Observation 18 and Theorem 13)}\\
&\le \mathbf{E}\left[\max\left\{\mathsf{M}(m^{(1)},4\mu),\mathbf{X}\right\}\right] \quad \text{(Proposition 3)}\\
&\le (1+20\delta)\,\mathbf{E}\left[\max\left\{\mathsf{M}(m^{(1)},\mu),\mathbf{X}\right\}\right] \quad \text{(by (12) in Lemma 9)}\\
&< (1+\epsilon)L^*.
\end{aligned}$$

Case 4: The algorithm goes to Lines 13–15. By putting $\delta = \frac{\epsilon}{1000}$ and Observation 18, an optimum assignment of rounded jobs satisfies the condition of Lemma 10. Let $\mathbf{W}$ be the Bernoulli random variable as described in Lemma 10 but with respect to the machine loads of an optimum assignment of the rounded jobs, then $(1-5\delta)\mathbf{E}[\max\{\mathbf{W},\boldsymbol{X}\}] \le \hat{L}$ where $\hat{L}$ is the optimum expected maximum of rounded version. By finding an assignment maximimizing $\prod_{j=1}^{m^{(1)}}\mathbf{Pr}[\mathsf{Poi}(\mu_j) \le t_4 - 1]$, the value of $\mathbf{E}[\max\{\mathbf{W},\boldsymbol{X}\}]$ would be minimized. Let the corresponding optimum $\mathbf{W}$ be $\mathbf{W}^*$. So we have $(1-5\delta)\mathbf{E}[\max\{\mathbf{W}^*,\mathbf{X}\}] \le (1-5\delta)\mathbf{E}[\max\{\mathbf{W},\mathbf{X}\}] \le \hat{L}$. Note that $\hat{L} \le (1+80\delta)L^*$ by (14) in Lemma 10 and Lemma 14.

As $\frac{4\log m}{m} < \mu^{(1)} \le (1+5\delta)\frac{1}{m^{\delta}} \le \frac{2}{m^{\delta}}$, by Lemma 10:

$$\begin{aligned}
L &= \mathbf{E}\left[\max\left\{\max_{j=1}^{m^{(1)}} \mathsf{Poi}\left(\mu_j\right), \mathbf{X}\right\}\right] \\
&\le \mathbf{E}\left[\max\left\{\max_{j=1}^{m^{(1)}} \mathsf{Poi}\left((1+5\delta)\mu_j^{(1)}\right), \mathbf{X}\right\}\right] \quad \text{(Lemma 14)} \\
&\le (1+80\delta)\mathbf{E}\left[\max\left\{\max_{j=1}^{m^{(1)}} \mathsf{Poi}\left(\mu_j^{(1)}\right), \mathbf{X}\right\}\right] \quad \text{(by (14) in Lemma 10)} \\
&\le (1+16\delta)(1+80\delta)\mathbf{E}\left[\max\left\{\mathbf{W}^*, \mathbf{X}\right\}\right] \quad \text{(by (13) in Lemma 10)} \\
&\le \frac{(1+16\delta)(1+80\delta)}{1-5\delta}\hat{L} \\
&\le \frac{(1+16\delta)(1+80\delta)^2}{1-20\delta}L^* \\
&< (1+\epsilon)L^*. \qquad (21)
\end{aligned}$$

Case 5: The algorithm returns an assignment on Line 17. With the same argument as the previous case, by maximizing $\prod_{j=1}^{m^{(1)}} \mathbf{Pr}\left[\mathsf{Poi}\left(\mu_j\right)=0\right]$, the value $\mathbf{E}\left[\max\left\{\mathbf{W}, \boldsymbol{X}\right\}\right]$ as defined in Lemma 11 would be minimized. Let the corresponding optimum $\mathbf{W}$ be $\mathbf{W}^*$. So we have $\mathbf{E}\left[\max\left\{\mathbf{W}^*, \mathbf{X}\right\}\right] \le \hat{L}$ where $\hat{L}$ is the optimum expected maximum of rounded jobs with $m^{(1)}$ machines and $\hat{L} \le (1+50\delta)L^*$ by (16) in Lemma 11 and Lemma 14. So we get:

$$\begin{aligned}
L &\le \mathbf{E}\left[\max\left\{\max_{j=1}^{m^{(1)}} \mathsf{Poi}\left((1+5\delta)\mu_j^{(1)}\right), \mathbf{X}\right\}\right] \quad \text{(Lemma 14)} \\
&\le (1+50\delta)\mathbf{E}\left[\max\left\{\max_{j=1}^{m^{(1)}} \mathsf{Poi}\left(\mu_j^{(1)}\right), \mathbf{X}\right\}\right] \quad \text{(by (16) in Lemma 11)} \\
&\le (1+10\delta)(1+50\delta)\mathbf{E}\left[\max\left\{\mathbf{W}^*, \mathbf{X}\right\}\right] \quad \text{(by (15)) in Lemma 11)} \\
&\le (1+10\delta)(1+50\delta)\hat{L} \\
&\le (1+10\delta)(1+50\delta)^2 L^* \\
&< (1+\epsilon)L^*.
\end{aligned}$$

So for all cases we have $L < (1+\epsilon)L^*$ and we are done. ◀

References

1 Noga Alon, Yossi Azar, Gerhard J Woeginger, and Tal Yadid. Approximation schemes for scheduling on parallel machines. *Journal of Scheduling*, 1(1):55–66, 1998.

2 Clive W Anderson, Stuart G Coles, and Jürg Hüsler. Maxima of poisson-like variables and related triangular arrays. *The Annals of Applied Probability*, pages 953–971, 1997.

3 CW Anderson. Extreme value theory for a class of discrete distributions with applications to some stochastic processes. *Journal of Applied Probability*, 7(1):99–113, 1970.

4 Keith M Briggs, Linlin Song, and Thomas Prellberg. A note on the distribution of the maximum of a set of poisson random variables. *arXiv preprint*, 2009. `arXiv:0903.4373`.

5 Chandra Chekuri and Michael Bender. An efficient approximation algorithm for minimizing makespan on uniformly related machines. *Journal of Algorithms*, 41(2):212–224, 2001.

6 Lin Chen, Klaus Jansen, and Guochuan Zhang. On the optimality of approximation schemes for the classical scheduling problem. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 657–668. SIAM, 2014.

7 Fabián A Chudak and David B Shmoys. Approximation algorithms for precedence-constrained scheduling problems on parallel machines that run at different speeds. *Journal of Algorithms*, 30(2):323–343, 1999.

8 D Darling. On the supremum of a certain gaussian process. *The Annals of Probability*, pages 803–806, 1983.

9 Ashish Goel and Piotr Indyk. Stochastic load balancing and related problems. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*, pages 579–586, 1999. doi:10.1109/SFFCS.1999.814632.

10 R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Tech. J.*, pages 1563–1581, 1966.

11 Anupam Gupta, Amit Kumar, Viswanath Nagarajan, and Xiangkun Shen. Stochastic load balancing on unrelated machines. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1274–1285. SIAM, 2018.

12 Dorit S Hochbaum and David B Shmoys. Using dual approximation algorithms for scheduling problems theoretical and practical results. *Journal of the ACM (JACM)*, 34(1):144–162, 1987.

13 Hochbaum, Dorit S and Shmoys, David B. A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach. *SIAM journal on computing*, 17(3):539–551, 1988.

14 Klaus Jansen. An EPTAS for scheduling jobs on uniform processors: Using an MILP relaxation with a constant number of integral variables. In *Automata, Languages and Programming, 36th International Colloquium, ICALP 2009, Rhodes, Greece, July 5-12, 2009, Proceedings, Part I*, pages 562–573, 2009. doi:10.1007/978-3-642-02927-1_47.

15 Klaus Jansen, Kim-Manuel Klein, and José Verschae. Closing the gap for makespan scheduling via sparsification techniques. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, pages 72:1–72:13, 2016. doi:10.4230/LIPIcs.ICALP.2016.72.

16 Klaus Jansen and Lars Rohwedder. On integer programming and convolution. In *10th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10-12, 2019, San Diego, California, USA*, pages 43:1–43:17, 2019. doi:10.4230/LIPIcs.ITCS.2019.43.

17 Ravi Kannan. Minkowski's convex body theorem and integer programming. *Math. Oper. Res.*, 12(3):415–440, 1987. doi:10.1287/moor.12.3.415.

18 Jon Kleinberg, Yuval Rabani, and Éva Tardos. Allocating bandwidth for bursty connections. *SIAM Journal on Computing*, 30(1):191–217, 2000.

19 Hendrik W. Lenstra. Integer programming with a fixed number of variables. *Math. Oper. Res.*, 8(4):538–548, 1983. doi:10.1287/moor.8.4.538.

20 Jan Karel Lenstra, David B Shmoys, and Eva Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical programming*, 46(1-3):259–271, 1990.

21 Michael Mitzenmacher and Eli Upfal. *Probability and computing: randomization and probabilistic techniques in algorithms and data analysis*. Cambridge university press, 2017.

22 Marco Molinaro. Stochastic ℓ_p load balancing and moment problems via the l-function method. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 343–354. SIAM, 2019.

23 Herbert Robbins. A remark on stirling's formula. *The American mathematical monthly*, 62(1):26–29, 1955.

24 Michel Talagrand. Majorizing measures: the generic chaining. *The Annals of Probability*, 24(3):1049–1103, 1996.

Tree Polymatrix Games Are PPAD-Hard

Argyrios Deligkas
Royal Holloway University of London, UK
Argyrios.Deligkas@rhul.ac.uk

John Fearnley
University of Liverpool, UK
John.Fearnley@liverpool.ac.uk

Rahul Savani
University of Liverpool, UK
Rahul.Savani@liverpool.ac.uk

Abstract

We prove that it is PPAD-hard to compute a Nash equilibrium in a tree polymatrix game with twenty actions per player. This is the first PPAD hardness result for a game with a constant number of actions per player where the interaction graph is acyclic. Along the way we show PPAD-hardness for finding an ϵ-fixed point of a 2D-LinearFIXP instance, when ϵ is any constant less than $(\sqrt{2}-1)/2 \approx 0.2071$. This lifts the hardness regime from polynomially small approximations in k-dimensions to constant approximations in two-dimensions, and our constant is substantial when compared to the trivial upper bound of 0.5.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness; Theory of computation → Exact and approximate computation of equilibria

Keywords and phrases Nash Equilibria, Polymatrix Games, PPAD, Brouwer Fixed Points

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.38

Category Track A: Algorithms, Complexity and Games

Related Version A full version of this paper, with all proofs included, is available [8], https://arxiv.org/abs/2002.12119.

1 Introduction

A *polymatrix game* is a succinctly represented many-player game. The players are represented by vertices in an *interaction graph*, where each edge of the graph specifies a two-player game that is to be played by the adjacent vertices. Each player picks a *pure strategy*, or *action*, and then plays that action in all of the edge-games that they are involved with. They then receive the *sum* of the payoffs from each of those games. A *Nash equilibrium* prescribes a mixed strategy to each player, with the property that no player has an incentive to unilaterally deviate from their assigned strategy.

Constant-action polymatrix games have played a central role in the study of equilibrium computation. The classical PPAD-hardness result for finding Nash equilibria in bimatrix games [4] uses constant-action polymatrix games as an intermediate step in the reduction [4,5]. Rubinstein later showed that there exists a constant $\epsilon > 0$ such that computing an ϵ-approximate Nash equilibrium in two-action bipartite polymatrix games is PPAD-hard [16], which was the first result of its kind to give hardness for constant ϵ.

These hardness results create polymatrix games whose interaction graphs contain cycles. This has lead researchers to study *acyclic* polymatrix games, with the hope of finding tractable cases. Kearns, Littman, and Singh claimed to produce a polynomial-time algorithm for finding a Nash equilibrium in a two-action tree *graphical game* [12], where graphical

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 38; pp. 38:1–38:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

games are a slight generalization of polymatrix games. However, their algorithm does not work, which was pointed out by Elkind, Goldberg, and Goldberg [10], who also showed that the natural fix gives an exponential-time algorithm.

Elkind, Goldberg, and Goldberg also show that a Nash equilibrium can be found in polynomial time for two-action graphical games whose interaction graphs contain only paths and cycles. They also show that finding a Nash equilibrium is PPAD-hard when the interaction graph has pathwidth at most four, but there appear to be some issues with their approach. Later work of Barman, Ligett, and Piliouras [1] provided a QPTAS for constant-action tree polymatrix games, and then Ortiz and Irfan [14] gave an FPTAS for this case. All three papers, [1, 10, 14], leave as a main open problem the question of whether it is possible to find a Nash equilibrium in a tree polymatrix game in polynomial time.

Our contribution. In this work we show that finding a Nash equilibrium in twenty-action tree polymatrix games is PPAD-hard. Combined with the known PPAD containment of polymatrix games [5], this implies that the problem is PPAD-complete. This is the first hardness result for polymatrix (or graphical) games in which the interaction graph is acyclic, and decisively closes the open question raised by prior work: tree polymatrix games cannot be solved in polynomial time unless PPAD is equal to P.

Our reduction produces a particularly simple class of interaction graphs: all of our games are played on *caterpillar* graphs (see Figure 3) which consist of a single path with small one-vertex branches affixed to every node. These graphs have pathwidth 1, so we obtain a stark contrast with prior work: two-action path polymatrix games can be solved in polynomial time [10], but twenty-action pathwidth-1-caterpillar polymatrix games are PPAD-hard.

Our approach is founded upon Mehta's proof that 2D-LinearFIXP is PPAD-hard [13]. We show that her reduction can be implemented by a synchronous arithmetic circuit with *constant width.* We then embed the constant-width circuit into a caterpillar polymatrix game, where each player in the game is responsible for simulating all gates at a particular level of the circuit. This differs from previous hardness results [5, 16], where each player is responsible for simulating exactly one gate from the circuit.

Along the way, we also substantially strengthen Mehta's hardness result for LinearFIXP. She showed PPAD-hardness for finding an exact fixed point of a 2D-LinearFIXP instance, and an ϵ-fixed point of a kD-LinearFIXP instance, where ϵ is polynomially small. We show PPAD-hardness for finding an ϵ-fixed point of a 2D-LinearFIXP instance when ϵ is any constant less than $(\sqrt{2}-1)/2 \approx 0.2071$. So we have lifted the hardness regime from polynomially small approximations in k-dimensions to constant approximations in two-dimensions, and our constant is substantial when compared to the trivial upper bound of 0.5.

Related work. The class PPAD was defined by Papadimitriou [15]. Years later, Daskalakis, Goldberg, and Papadimitriou (DGP) [5] proved PPAD-hardness for graphical games and 3-player normal form games. Chen, Deng, and Teng (CDT) [4] extended this result to 2-player games and proved that there is no FPTAS for the problem unless PPAD = P. The observations made by CDT imply that DGP's result also holds for polymatrix games with constantly-many actions (but with cycles in the interaction graph) for an exponentially small ϵ. More recently, Rubinstein [17] showed that there exists a *constant* $\epsilon > 0$ such that computing an $\epsilon - NE$ in binary-action bipartite polymatrix games is PPAD-hard (again with cycles in the interaction graph).

Etessami and Yiannakakis [11] defined the classes FIXP and LinearFIXP and they proved that LinearFIXP = PPAD. Mehta [13] strengthened these results by proving that two-dimensional LinearFIXP equals PPAD, building on the result of Chen and Deng who proved that 2D-discrete Brouwer is PPAD-hard [3].

On the positive side, Cai and Daskalakis [2], proved that NE can be efficiently found in polymatrix games where every 2-player game is zero-sum. Ortiz and Irfan [14] and Deligkas, Fearnley, and Savani [7] produced QPTASs for polymatrix games of bounded treewidth (in addition to the FPTAS of [14] for tree polymatrix games mentioned above). For general polymatrix games, the only positive result to date is a polynomial-time algorithm to compute a $(\frac{1}{2} + \delta)$-NE [9]. Finally, an empirical study on algorithms for exact and approximate NE in polymatrix games can be found in [6].

2 Preliminaries

Polymatrix games. An n-player *polymatrix game* is defined by an undirected *interaction graph* $G = (V, E)$ with n vertices, where each vertex represents a player, and the edges of the graph specify which players interact with each other. Each player in the game has m actions, and each edge $(v, u) \in E$ of the graph is associated with two $m \times m$ matrices $A^{v,u}$ and $A^{u,v}$ which specify a bimatrix game that is to be played between the two players, where $A^{v,u}$ specifies the payoffs to player v from their interaction with player u.

Each player in the game selects a single action, and then plays that action in *all* of the bimatrix games with their neighbours in the graph. Their payoff is the *sum* of the payoffs that they obtain from each of the individual bimatrix games.

A *mixed strategy* for player i is a probability distribution over the m actions of that player, a *strategy profile* is a vector $\mathbf{s} = (s_1, s_2, \dots, s_n)$ where s_i is a mixed strategy for player i. The *vector of expected payoffs* for player i under strategy profile $\mathbf{s}$ is $\mathbf{p}_i(\mathbf{s}) := \sum_{(i,j)\in E} A^{i,j} s_j$. The *expected payoff* to player i under $\mathbf{s}$ is $s_i \cdot \mathbf{p}_i(\mathbf{s})$. A strategy profile is a *mixed Nash equilibrium* if $s_i \cdot \mathbf{p}_i(\mathbf{s}) = \max \mathbf{p}_i(\mathbf{s})$ for all i, which means that no player can unilaterally change their strategy in order to obtain a higher expected payoff. In this paper we are interested in the problem of computing a Nash equilibrium of a *tree* polymatrix game, which is a polymatrix game in which the interaction graph is a tree.

Arithmetic circuits. For the purposes of this paper, each gate in an arithmetic circuit will operate only on values that lie in the range $[0, 1]$. In our construction, we will use four specific gates, called *constant introduction* denoted by c, *bounded addition* denoted by $+^b$, *bounded subtraction* denoted by $-^b$, and *bounded multiplication by a constant* denoted by $*^b c$. These gates are formally defined as follows.

- c is a gate with no inputs that outputs some fixed constant $c \in [0, 1]$.
- Given inputs $x, y \in [0, 1]$ the gate $x +^b y := \min(x + y, 1)$.
- Given inputs $x, y \in [0, 1]$ the gate $x -^b y := \max(x - y, 0)$.
- Given an input $x \in [0, 1]$, and a constant $c \geq 0$, the gate $x *^b c := \min(x * c, 1)$.

These gates perform their operation, but also clip the output value so that it lies in the range $[0, 1]$. Note that the constant c in the $*^b c$ gate is specified as part of the gate. Multiplication of two inputs is not allowed.

We will build arithmetic circuits that compute functions of the form $[0, 1]^d \to [0, 1]^d$. A circuit $C = (I, G)$ consists of a set $I = \{\mathtt{in}_1, \mathtt{in}_2, \dots, \mathtt{in}_d\}$ containing d input nodes, and a set $G = \{g_1, g_2, \dots, g_k\}$ containing k gates. Each gate g_i has a type from the set $\{c, +^b, -^b, *^b c\}$, and if the gate has one or more inputs, these are taken from the set $I \cup G$. The connectivity structure of the gates is required to be a directed acyclic graph.

The *depth* of a gate, denoted by $d(g)$ is the length of the longest path from that gate to an input. We will build *synchronous* circuits, meaning that all gates of the form $g_x = g_y +^b g_z$ satisfy $d(g_x) = 1 + d(g_y) = 1 + d(g_z)$, and likewise for gates of the form $g_x = g_y -^b g_z$. There are no restrictions on c-gates, or $*^b c$-gates.

The *width* of a particular level i of the circuit is defined to be $w(i) = |\{g_j : d(g_j) = i\}|$, which is the number of gates at that level. The *width* of a circuit is defined to be $w(C) = \max_i w(i)$, which is the maximum width taken over all the levels of the circuit.

Straight line programs. A convenient way of specifying an arithmetic circuit is to write down a straight line program (SLP) [11].

SLP 1 Example.

```
x ← 0.5
z ← x $+^b$ in1
x ← x $*^b$ 0.5
out1 ← z $+^b$ x
```

SLP 2 `if` and `for` example.

```
x ← in1 $*^b$ 1
for $i$ in $\{1, 2, \dots, 10\}$ do
    if $i$ is even then
        x ← x $+^b$ 0.1
    end
end
out1 ← x $*^b$ 1
```

Each line of an SLP consists of a statement of the form `v` ← `op`, where `v` is a *variable*, and `op` consists of exactly one arithmetic operation from the set set $\{c, +^b, -^b, *^b c\}$. The inputs to the gate can be any variable that is defined before the line, or one of the inputs to the circuit. We permit variables to be used on the left hand side in more than one line, which effectively means that we allow variables to be overwritten.

It is easy to turn an SLP into a circuit. Each line is turned into a gate, and if variable `v` is used as the input to gate g, then we set the corresponding input of g to be the gate g' that corresponds to the line that most recently assigned a value to `v`. SLP 1 above specifies a circuit with four gates, and the output of the circuit will be 0.75 $+^b$ $\texttt{in}_1$.

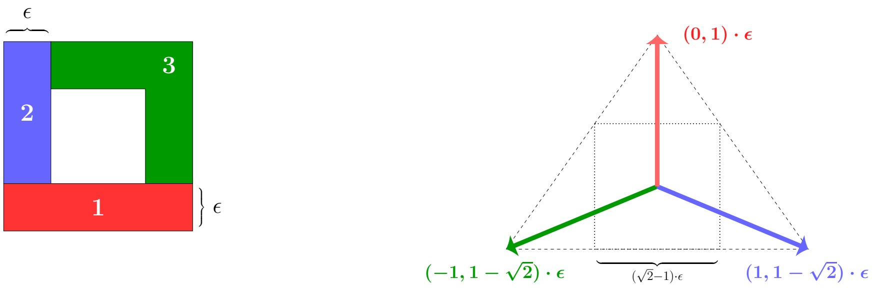
For the sake of brevity, we also allow `if` statements and `for` loops in our SLPs. These two pieces of syntax can be thought of as macros that help us specify a straight line program concisely. The arguments to an `if` statement or a `for` loop must be constants that do not depend on the value of any gate in the circuit. When we turn an SLP into a circuit, we unroll every `for` loop the specified number of times, and we resolve every `if` statement by deleting the block if the condition does not hold. So the example in SLP 2 produces a circuit with seven gates: two gates correspond to the lines `x` ← `in1` $*^b$ `1` and `out1` ← `x` $*^b$ `1`, while there are five gates corresponding to the line `x` ← `x` $+^b$ `0.1`, since there are five copies of the line remaining after we unroll the loop and resolve the `if` statements. The output of the resulting circuit will be 0.5 $+^b$ $\texttt{in}_1$.

Liveness of variables and circuit width. Our ultimate goal will be to build circuits that have small width. To do this, we can keep track of the number of variables that are *live* at any one time in our SLPs. A variable `v` is live at line i of an SLP if both of the following conditions are met.

- There exists a line with index $j \leq i$ that assigns a value to `v`.
- There exists a line with index $k \geq i$ that uses the value assigned to `v` as an argument.

The number of variables that are live at line i is denoted by $\text{live}(i)$, and the number of variables *used by* an SLP is defined to be $\max_i \text{live}(i)$, which is the maximum number of variables that are live at any point in the SLP.

▶ **Lemma 1.** *An SLP that uses w variables can be transformed into a polynomial-size synchronous circuit of width w.*


ϵ
3
2
1
ϵ
$(0,1)\cdot\epsilon$
$(-1,1-\sqrt{2})\cdot\epsilon$
$(\sqrt{2}-1)\cdot\epsilon$
$(1,1-\sqrt{2})\cdot\epsilon$

(a) Our stronger boundary conditions.

(b) The mapping from colors to vectors.

Figure 1 Reducing ϵ-`ThickDisBrouwer` to `2D-Brouwer`.

3 Hardness of 2D-Brouwer

In this section, we consider the following problem. It is a variant of two-dimensional Brouwer that uses only our restricted set of bounded gates.

▶ **Definition 2** (`2D-Brouwer`). *Given an arithmetic circuit $F : [0,1]^2 \to [0,1]^2$ using gates from the set $\{c, +^b, -^b, *^b\ c\}$, find $x \in [0,1]^2$ such that $F(x) = x$.*

As a starting point for our reduction, we will show that this problem is `PPAD`-hard. Our proof will follow the work of Mehta [13], who showed that the closely related `2D-LinearFIXP` problem is `PPAD`-hard. There are two differences between `2D-Brouwer` and `2D-LinearFIXP`.

- In `2D-LinearFIXP`, all internal gates of the circuit take and return values from $\mathbb{R}$ rather than $[0,1]$.
- `2D-LinearFIXP` takes a circuit that uses gates from the set $\{c, +, -, *c, \max, \min\}$, where none of these gates bound their outputs to be in $[0,1]$.

In this section, we present an altered version of Mehta's reduction, which will show that finding an ϵ-solution to `2D-Brouwer` is `PPAD`-hard for a constant ϵ.

Discrete Brouwer. The starting point for Mehta's reduction is the two-dimensional discrete Brouwer problem, which is known to be `PPAD`-hard [3]. This problem is defined over a discretization of the unit square $[0,1]^2$ into a grid of points $G = \{0, 1/2^n, 2/2^n, \dots, (2^n - 1)/2^n\}^2$. The input to the problem is a Boolean circuit $C : G \to \{1,2,3\}$ the assigns one of three colors to each point. The coloring will respect the following boundary conditions.

- We have $C(0,i) = 1$ for all $i \geq 0$.
- We have $C(i,0) = 2$ for all $i > 0$.
- We have $C(\frac{2^n-1}{2^n}, i) = C(i, \frac{2^n-1}{2^n}) = 3$ for all $i > 0$.

These conditions can be enforced syntactically by modifying the circuit. The problem is to find a grid square that is *trichromatic*, meaning that all three colors appear on one of the four points that define the square.

▶ **Definition 3** (`DiscreteBrouwer`). *Given a Boolean circuit $C : \{0,1\}^n \times \{0,1\}^n \to \{1,2,3\}$ that satisfies the boundary conditions, find a point $x, y \in \{0,1\}^n$ such that, for each color $i \in \{1,2,3\}$, there exists a point (x', y') with $C(x', y') = i$ where $x' \in \{x, x+1\}$ and $y' \in \{y, y+1\}$.*

Our first deviation from Mehta's reduction is to insist on the following stronger boundary condition, which is shown in Figure 1a.

- We have $C(i, j) = 1$ for all i, and for all $j \leq \epsilon$.
- We have $C(i, j) = 2$ for all $j > \epsilon$, and for all $i \leq \epsilon$.
- We have $C(i, j) = C(j, i) = 3$ for all $i > \epsilon$, and all $j \geq 1 - \epsilon$.

The original boundary conditions placed constraints only on the outermost grid points, while these conditions place constraints on a border of width ϵ. We call this modified problem ϵ-ThickDisBrouwer, which is the same as DiscreteBrouwer, except that the function is syntactically required to satisfy the new boundary conditions.

It is not difficult to produce a polynomial time reduction from DiscreteBrouwer to ϵ-ThickDisBrouwer. It suffices to increase the number of points in the grid, and then to embed the original DiscreteBrouwer instance into the $[\epsilon, 1 - \epsilon]^2$ square in the middle of the instance.

▶ **Lemma 4.** *DiscreteBrouwer can be reduced in polynomial time to ϵ-ThickDisBrouwer.*

Embedding the grid in $[0, 1]^2$. We now reduce ϵ-ThickDisBrouwer to 2D-Brouwer. One of the keys steps of the reduction is to map points from the continuous space $[0, 1]^2$ to the discrete grid G. Specifically, given a point $x \in [0, 1]$, we would like to determine the n bits that define the integer $\lfloor x \cdot 2^n \rfloor$.

Mehta showed that this mapping from continuous points to discrete points can be done by a linear arithmetic circuit. Here we give a slightly different formulation that uses only gates from the set $\{c, +^b, -^b, *^b c\}$. Let L be a fixed constant that will be defined later.

SLP 3 ExtractBit(x, b).

```
b ← 0.5
b ← x $-^b$ b
b ← b $*^b$ L
```

SLP 4 ExtractBits(x, b$_1$, b$_2$, . . . , b$_n$).

```
for $i$ in $\{1, 2, \ldots, n\}$ do
    ExtractBit(x, b$_i$)
    y ← b$_i$ $*^b$ 0.5
    x ← x $-^b$ y
    x ← x $*^b$ 2
end
```

SLP 3 extracts the first bit of the number $x \in [0, 1]$. The first three lines of the program compute the value $b = (x -^b 0.5) *^b L$. There are three possibilities.

- If $x \leq 0.5$, then $b = 0$.
- If $x \geq 0.5 + 1/L$, then $b = 1$.
- If $0.5 < x < 0.5 + 1/L$, then b will be some number strictly between 0 and 1.

The first two cases correctly decode the first bit of x, and we call these cases *good decodes*. We will call the third case a *bad decode*, since the bit has not been decoded correctly.

SLP 4 extracts the first n bits of x, by extracting each bit in turn, starting with the first bit. The three lines after each extraction erase the current first bit of x, and then multiply x by two, which means that the next extraction will give us the next bit of x. If any of the bit decodes are bad, then this procedure will break, meaning that we only extract the first n bits of x in the case where all decodes are good. We say that x is *well-positioned* if the procedure succeeds, and *poorly-positioned* otherwise.

Multiple samples. The problem of poorly-positioned points is common in PPAD-hardness reductions. Indeed, observe that we cannot define an SLP that always correctly extracts the first n bits of x, since this would be a discontinuous function, and all gates in our arithmetic circuits compute continuous functions. As in previous works, this is resolved by taking multiple samples around a given point. Specifically, for the point $p \in [0,1]^2$, we sample k points $p_1, p_2, \dots, p_k$ where $p_i = p + \left(\frac{i-1}{(k+1)\cdot 2^{n+1}}, \frac{i-1}{(k+1)\cdot 2^{n+1}}\right)$. Mehta proved that there exists a setting for L that ensures that there are at most two points that have poorly positioned coordinates. We have changed several details, and so we provide our own statement here.

▶ **Lemma 5.** *If $L = (k+2)\cdot 2^{n+1}$, then at most two of the points p_1 through p_k have poorly-positioned coordinates.*

Evaluating a Boolean circuit. Once we have decoded the bits for a well-positioned point, we have a sequence of 0/1 variables. It is easy to simulate a Boolean circuit on these values.

- The operator $\neg\, x$ can be simulated by $1 -^b x$.
- The operator $x \vee y$ can be simulated by $x +^b y$.
- The operator $x \wedge y$ can be simulated by applying De Morgan's laws and using $\vee$ and $\neg$.

Recall that C outputs one of three possible colors. We also assume, without loss of generality, that C gives its output as a *one-hot vector*. This means that there are three Boolean outputs $x_1, x_2, x_3 \in \{0,1\}^3$ of the circuit. The color 1 is represented by the vector $(1,0,0)$, the color 2 is represented as $(0,1,0)$, and color 3 is represented as $(0,0,1)$. If the simulation is applied to a point with well-positioned coordinates, then the circuit will output one of these three vectors, while if it is applied to a point with poorly positioned coordinates, then the circuit will output some value $x \in [0,1]^3$ that has no particular meaning.

The output. The key idea behind the reduction is that each color will be mapped to a displacement vector, as shown in Figure 1b. Here we again deviate from Mehta's reduction, by giving different vectors that will allow us to prove our approximation lower bound.

- Color 1 will be mapped to the vector $(0,1)\cdot\epsilon$.
- Color 2 will be mapped to the vector $(1, 1-\sqrt{2})\cdot\epsilon$.
- Color 3 will be mapped to the vector $(-1, 1-\sqrt{2})\cdot\epsilon$.

These are irrational coordinates, but in our proofs we argue that a suitably good rational approximation of these vectors will suffice. We average the displacements over the k different sampled points to get the final output of the circuit. Suppose that x_{ij} denotes output i from sampled point j. Our circuit will compute

$$\mathtt{disp}_x = \sum_{j=1}^{k} \frac{(x_{2j} - x_{3j})\cdot\epsilon}{k}, \qquad \mathtt{disp}_y = \sum_{j=1}^{k} \frac{(x_{1j} + (1-\sqrt{2})(x_{2j}+x_{3j}))\cdot\epsilon}{k}.$$

Finally, we specify $F : [0,1]^2 \to [0,1]^2$ to compute $F(x,y) = (x + \mathtt{disp}_x \cdot \epsilon, y + \mathtt{disp}_y \cdot \epsilon)$.

Completing the proof. To find an approximate fixed point of F, we must find a point where both $\mathtt{disp}_x$ and $\mathtt{disp}_y$ are close to zero. The dotted square in Figure 1b shows the set of displacements that satisfy $\|x - (0,0)\|_\infty \le (\sqrt{2}-1)\cdot\epsilon$, which correspond to the displacements that would be $(\sqrt{2}-1)\cdot\epsilon$-fixed points.

The idea is that, if we do not sample points of all three colors, then we cannot produce a displacement that is strictly better than an $(\sqrt{2}-1)\cdot\epsilon$-fixed point. For example, if we only have points of colors 1 and 2, then the displacement will be some point on the dashed line between the red and blue vectors in Figure 1b. This line touches the box of $(\sqrt{2}-1)\cdot\epsilon$-fixed points, but does not enter it. It can be seen that the same property holds for the other pairs of colors: we specifically chose the displacement vectors in order to maximize the size of the inscribed square shown in Figure 1b.

The argument is complicated by the fact that two of our sampled points may have poorly positioned coordinates, which may drag the displacement towards $(0,0)$. However, this effect can be minimized by taking a large number of samples. We show show the following lemma.

▶ **Lemma 6.** *Let $\epsilon' < (\sqrt{2}-1)\cdot\epsilon$ be a constant. There is a sufficiently large constant k such that, if $\|x - F(x)\|_\infty < \epsilon'$, then x is contained in a trichromatic square.*

Since ϵ can be fixed to be any constant strictly less than 0.5, we obtain the following.

▶ **Theorem 7.** *Given a `2D-Brouwer` instance, it is `PPAD`-hard to find a point $x \in [0,1]^2$ s.t. $\|x - F(x)\|_\infty < (\sqrt{2}-1)/2 \approx 0.2071$.*

Reducing `2D-Brouwer` to `2D-LinearFIXP` is easy, since the gates $\{c, +^b, -^b, *^b c\}$ can be simulated by the gates $\{c, +, -, *c, \max, \min\}$. This implies that it is `PPAD`-hard to find an ϵ-fixed point of a `2D-LinearFIXP` instance with $\epsilon < (\sqrt{2}-1)/2$.

It should be noted that an ϵ-approximate fixed point can be found in polynomial time if the function has a suitably small Lipschitz constant, by trying all points in a grid of width ϵ. We are able to obtain a lower bound for constant ϵ because our functions have exponentially large Lipschitz constants.

4 Hardness of 2D-Brouwer with a constant width circuit

In our reduction from `2D-Brouwer` to tree polymatrix games, the number of actions in the game will be determined by the width of the circuit. This means that the hardness proof from the previous section is not a sufficient starting point, because it produces `2D-Brouwer` instances that have circuits with high width. In particular, the circuits will extract $2n$ bits from the two inputs, which means that the circuits will have width at least $2n$.

Since we desire a constant number of actions in our tree polymatrix game, we need to build a hardness proof for `2D-Brouwer` that produces a circuit with constant width. In this section we do exactly that, by reimplementing the reduction from the previous section using gadgets that keep the width small.

Bit packing. We adopt an idea of Elkind, Goldberg, and Goldberg [10], to store many bits in a single arithmetic value using a *packed* representation. Given bits $b_1, b_2, \ldots, b_k \in \{0,1\}$, the packed representation of these bits is the value $\text{packed}(b_1, b_2, \ldots, b_k) := \sum_{i=1}^{k} b_i/2^i$. We will show that the reduction from the previous section can be performed while keeping all Boolean values in a single variable that uses packed representation.

Working with packed variables. We build SLPs that work with this packed representation, two of which are shown below.

SLP 5 FirstBit(x, b) +0 variables.

```
// Extract the first bit of x
   into b
b ← 0.5
b ← x $-^b$ b
b ← b $*^b$ L

// Remove the first bit of x
b ← b $*^b$ 0.5
x ← x $-^b$ b
x ← x $*^b$ 2
b ← b $*^b$ 2
```

SLP 6 Clear(I, x) +2 variables.

```
x' ← x $*^b$ 1
for $i$ in $\{1, 2, \dots, k\}$ do
    b ← 0
    FirstBit(x', b)
    if $i \in I$ then
        b ← b $*^b$ $\frac{1}{2^i}$
        x ← x $-^b$ b
    end
end
```

The FirstBit SLP combines the ideas from SLPs 3 and 4 to extract the first bit from a value $x \in [0, 1]$. Repeatedly applying this SLP allows us to read out each bit of a value in sequence. The Clear SLP uses this to set some bits of a packed variable to zero. It takes as input a set of indices I, and a packed variable $x = \text{packed}(b_1, b_2, \dots, b_k)$. At the end of the SLP we have $x = \text{packed}(b'_1, b'_2, \dots, b'_k)$ where $b'_i = 0$ whenever $i \in I$, and $b'_i = b_i$ otherwise.

It first copies x to a fresh variable x'. The bits of x' are then read-out using FirstBit. Whenever a bit b_i with $i \in I$ is decoded from x', we subtract $b_i/2^i$ from x. If $b_i = 1$, then this sets the corresponding bit of x to zero, and if $b_i = 0$, then this leaves x unchanged.

We want to minimize the the width of the circuit that we produce, so we keep track of the number of *extra* variables used by our SLPs. For FirstBit, this is zero, while for Clear this is two, since that SLP uses the fresh variables x' and b.

Packing and unpacking bits. We implement two SLPs that manipulated packed variables. The Pack(x, y, S) operation allows us to extract bits from $y \in [0, 1]$, and store them in x, while the Unpack(x, y, S) operation allows us to extract bits from x to create a value $y \in [0, 1]$. This is formally specified in the following lemma.

▶ **Lemma 8.** *Suppose that we are given* $\mathtt{x} = \text{packed}(b_1, b_2, \dots, b_k)$*, a variable* $\mathtt{y} \in [0, 1]$*, and a sequence of indices* $S = \langle s_1, s_2, \dots, s_j \rangle$*. Let* y_j *denote the* j*th bit of* y*. The following SLPs can be implemented using at most two extra variables.*

- *Pack(x, y, S) modifies* x *so that* $\mathtt{x} = \text{packed}(b'_1, b'_2, \dots, b'_k)$ *where* $b'_i = y_j$ *whenever there exists an index* $s_j \in S$ *with* $s_j = i$*, and* $b'_i = b_i$ *otherwise.*
- *Unpack(x, y, S) modifies* y *so that* $\mathtt{y} = \mathtt{y} +^b \sum_{i=1}^{j} b_{s_i}/2^i$

Simulating a Boolean operations. As described in the previous section, the reduction only needs to simulate or- and not-gates. Given $\mathtt{x} = \text{packed}(b_1, b_2, \dots, b_k)$, and three indices i_1, i_2, i_3, we implement two SLPs, which both modify x so that $\mathtt{x} = \text{packed}(b'_1, b'_2, \dots, b'_k)$. SLP 7 implements $\mathtt{Or}(\mathtt{x}, \mathtt{i}_1, \mathtt{i}_2, \mathtt{i}_3)$, which ensures that $b'_{i_3} = b_{i_1} \vee b_{i_2}$, and $b'_i = b_i$ for $i \neq i_3$. SLP 8 implements $\mathtt{Not}(\mathtt{x}, \mathtt{i}_1, \mathtt{i}_2)$, which ensures that $b'_{i_2} = \neg b_{i_1}$, and $b'_i = b_i$ for $i \neq i_2$.

These two SLPs simply unpack the input bits, perform the operation, and then pack the result into the output bit. The Or SLP uses the Unpack operation to set $\mathtt{a} = b_{i_1} +^b b_{i_2}$. Both SLPs use three extra variables: the fresh variable a is live throughout, and the pack and unpack operations use two extra variables. The variable b in the Not SLP is not live concurrently with a pack or unpack, and so does not increase the number of live variables. These two SLPs can be used to simulate a Boolean circuit using at most three extra variables.

SLP 7 Or($\mathtt{x}, \mathtt{i}_1, \mathtt{i}_2, \mathtt{i}_3$) +3 variables.

```
a ← 0
Unpack(x, a, $\langle \mathtt{i}_1 \rangle$)
Unpack(x, a, $\langle \mathtt{i}_2 \rangle$)
Pack(x, a, $\langle \mathtt{i}_3 \rangle$)
```

SLP 8 Not($\mathtt{x}, \mathtt{i}_1, \mathtt{i}_2$) +3 variables.

```
a ← 0
Unpack(x, a, $\langle \mathtt{i}_1 \rangle$)
b ← 1
a ← b $-^b$ a
Pack(x, a, $\langle \mathtt{i}_2 \rangle$)
```

▶ **Lemma 9.** *Let C be a Boolean circuit with n inputs and k gates. Suppose that $x = \text{packed}(b_1, \dots, b_n)$, gives values for the inputs of the circuit. There is an SLP* Simulate(C, x) *that uses three extra variables, and modifies x so that $x = \text{packed}(b_1, \dots, b_n, b_{n+1}, \dots, b_{n+k})$, where b_{n+i} is the output of gate i of the circuit.*

Implementing the reduction. Finally, we can show that the circuit built in Theorem 7 can be implemented by an SLP that uses at most 8 variables. This SLP cycles through each sampled point in turn, computes the x and y displacements by simulating the Boolean circuit, and then adds the result to the output.

▶ **Theorem 10.** *Given a* 2D-Brouwer *instance, it is* PPAD*-hard to find a point $x \in [0,1]^2$ with $\|x - F(x)\|_\infty < \frac{\sqrt{2}-1}{2}$ even for a synchronous circuit of width eight.*

5 Hardness for tree polymatrix games

Now we show that finding a Nash equilibrium of a tree polymatrix game is PPAD-hard. We reduce from the low-width 2D-Brouwer problem, whose hardness was shown in Theorem 10. Throughout this section, we suppose that we have a 2D-Brouwer instance defined by a synchronous arithmetic circuit F of width eight and depth n. The gates of this circuit will be indexed as $g_{i,j}$ where $1 \leq i \leq 8$ and $1 \leq j \leq n$, meaning that $g_{i,j}$ is the ith gate on level j.

Modifying the circuit. The first step of the reduction is to modify the circuit. First, we modify the circuit so that all gates operate on values in $[0, 0.1]$, rather than $[0, 1]$. We introduce the operators $+^b_{0.1}$, $-^b_{0.1}$, and $*^b_{0.1}$, which bound their outputs to be in $[0, 0.1]$. The following lemma states that we can rewrite our circuit using these new gates. The transformation simply divides all c-gates in the circuit by ten.

▶ **Lemma 11.** *Given an arithmetic circuit $F : [0,1]^2 \to [0,1]^2$ that uses gates from $\{c, +^b, -^b, *^b\}$, we can construct a circuit $F' : [0, 0.1]^2 \to [0, 0.1]^2$ that uses the gates from $\{c, +^b_{0.1}, -^b_{0.1}, *^b_{0.1}\}$, so that $F(x, y) = (x, y)$ if and only if $F'(x/10, y/10) = (x/10, y/10)$.*

Next we modify the structure of the circuit by connecting the two outputs of the circuit to its two inputs. Suppose, without loss of generality, that $g_{7,1}$ and $g_{8,1}$ are the inputs and that $g_{7,n}$ and $g_{8,n}$ are outputs. Note that the equality $x = y$ can be implemented using the gate $x = y *^b_{0.1} 1$. We add the following extra equalities, which are shown in Figure 2.

- We add gates $g_{9,n-1} = g_{7,n}$ and $g_{10,n-1} = g_{8,n}$.
- For each j in the range $2 \leq j < n - 1$, we add $g_{9,j} = g_{9,j+1}$ and $g_{10,j} = g_{10,j+1}$.
- We modify $g_{7,1}$ so that $g_{7,1} = g_{9,2}$, and we modify $g_{8,1}$ so that $g_{8,1} = g_{10,2}$.

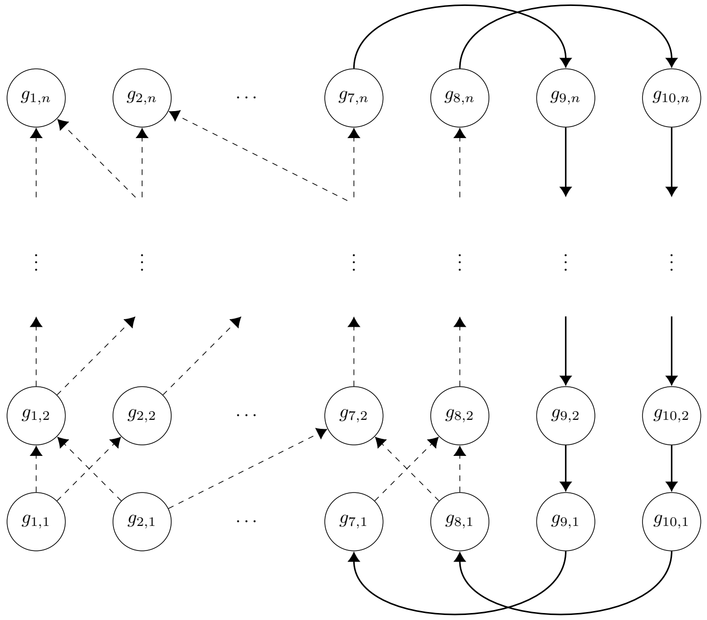


Figure 2 Extra equalities to introduce feedback of $g_{7,n}$ and $g_{8,n}$ to $g_{7,1}$ and $g_{8,1}$ respectively.

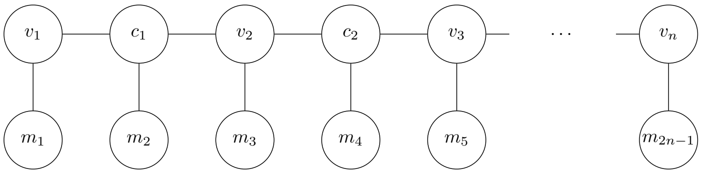


Figure 3 The structure of the polymatrix game.

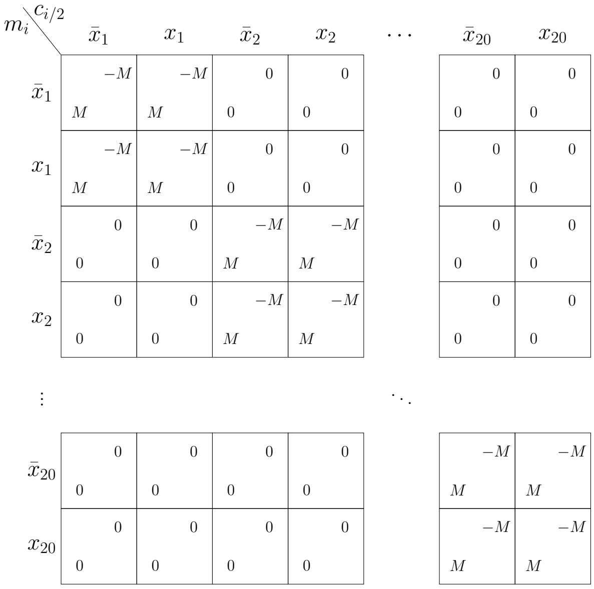
Note that these gates are backwards: they copy values from higher levels in the circuit to lower levels, and so the result is not a circuit, but a system of constraints defined by gates, with some structural properties. Firstly, each gate $g_{i,j}$ is only involved in constraints with gates of the form $g_{i',j+1}$ and $g_{i',j-1}$. Secondly, finding values for the gates that satisfy all of the constraints is PPAD-hard, since by construction such values would yield a fixed point of F.

The polymatrix game. The polymatrix game will contain three types of players.

- For each $i = 1, \dots, n$, we have a *variable* player v_i.
- For each $i = 1, \dots, n-1$, we have a *constraint* player c_i, who is connected to v_i and v_{i+1}.
- For each $i = 1, \dots, 2n-1$, we have a *mix* player m_i. If i is even, then m_i is connected to $c_{i/2}$. If i is odd, then m_i is connected to $v_{(i+1)/2}$.

The structure of this game is shown in Figure 3. Each player has twenty actions, which are divided into ten pairs, x_i and $\bar{x}_i$ for $i = 1, \dots, 10$.

Forcing mixing. The role of the mix players is to force the variable and constraint players to play specific mixed strategies: for every variable or constraint player j, we want $s_j(x_i) + s_j(\bar{x}_i) = 0.1$ for all i, which means that the same amount of probability is assigned to each pair of actions. To force this, each mix player plays a high-stakes hide-and-seek against their



$m_i \backslash c_{i/2}$	$\bar{x}_1$	x_1	$\bar{x}_2$	x_2	$\cdots$	$\bar{x}_{20}$	x_{20}
$\bar{x}_1$	$-M$ M	$-M$ M	0 0	0 0		0 0	0 0
x_1	$-M$ M	$-M$ M	0 0	0 0		0 0	0 0
$\bar{x}_2$	0 0	0 0	$-M$ M	$-M$ M		0 0	0 0
x_2	0 0	0 0	$-M$ M	$-M$ M		0 0	0 0
$\vdots$					$\ddots$		
$\bar{x}_{20}$	0 0	0 0	0 0	0 0		$-M$ M	$-M$ M
x_{20}	0 0	0 0	0 0	0 0		$-M$ M	$-M$ M

Figure 4 The hide and seek game that forces $c_{j/2}$ to play an appropriate mixed strategy. The same game is used to force $v_{(j-1)/2}$ mixes appropriately.

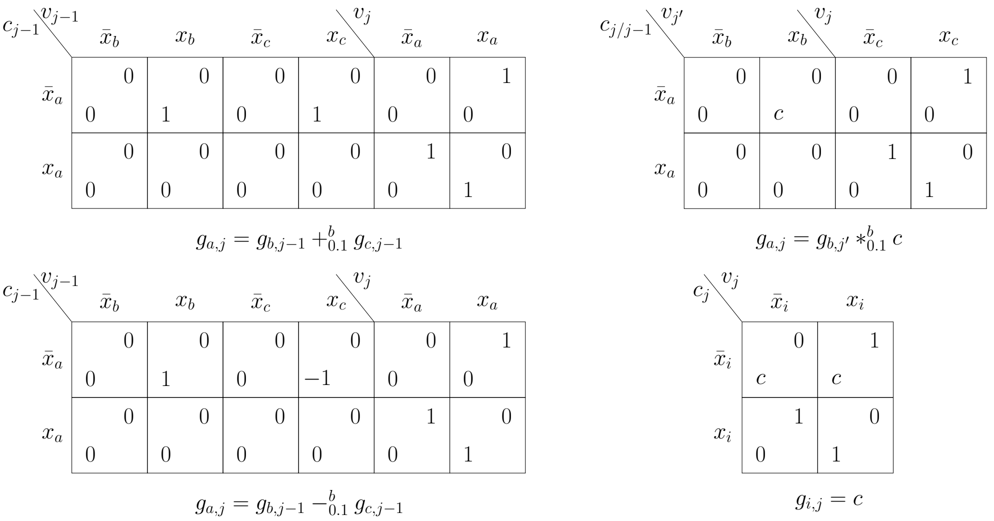
opponent, which is shown in Figure 4. This zero-sum game is defined by a 20×20 matrix Z and a constant M. The payoff Z_{ij} is defined as follows. If $i \in \{x_a, \bar{x}_a\}$ and $j \in \{x_a, \bar{x}_a\}$ for some a, then $Z_{ij} = M$. Otherwise, $Z_{ij} = 0$. For each i the player m_i plays against player j, which is either a constraint player $c_{i'}$ or a variable player $v_{i'}$. We define the payoff matrix $A^{m_i,j} = Z$ and $G^{j,m_i} = -Z$. The following lemma shows that if M is suitably large, then the variable and constraint players must allocate probability 0.1 to each of the ten action pairs.

▶ **Lemma 12.** *Suppose that all payoffs in the games between variable and constraint players use payoffs in the range $[-P, P]$. If $M > 40 \cdot P$ then in every mixed Nash equilibrium $\mathbf{s}$, the action s_j of every variable and constraint player j satisfies $s_j(x_i) + s_j(\bar{x}_i) = 0.1$ for all i.*

Gate gadgets. We now define the payoffs for variable and constraint players. Actions x_i and $\bar{x}_i$ of variable player v_j will represent the output of gate $g_{i,j}$. Specifically, the probability that player v_j assigns to action x_i will be equal to the output of $g_{i,j}$. In this way, the strategy of variable player v_j will represent the output of every gate at level j of the circuit. The constraint player c_j enforces all constraints between the gates at level j and the gates at level $j + 1$. To simulate each gate, we will embed one of the gate gadgets from Figure 5, which originated from the reduction of DGP [5], into the bimatrix games that involve c_j.

The idea is that, for the constraint player to be in equilibrium, the variable players must play x_i with probabilities that exactly simulate the original gate. Lemma 12 allows us to treat each gate independently: each pair of actions x_i and $\mathbf{s}_i$ must receive probability 0.1 in total, but the split of probability between x_i and $\mathbf{s}_i$ is determined by the gate gadgets.

Formally, we construct the payoff matrices A^{v_i,c_i} and $A^{c_i,v_{i+1}}$ for all $i < n$ by first setting each payoff to 0. Then, for each gate, we embed the corresponding gate gadget from Figure 5 into the matrices. For each gate $g_{a,j}$, we take the corresponding game from Figure 5, and embed it into the rows x_a and $\bar{x}_a$ of a constraint player's matrix. The diagrams specify specific actions of the constraint and variable players that should be modified.



c_{j-1} \ v_{j-1}					v_j	
	$\bar{x}_b$	x_b	$\bar{x}_c$	x_c	$\bar{x}_a$	x_a
$\bar{x}_a$	0 0	0 1	0 0	0 1	0 0	1 0
x_a	0 0	0 0	0 0	0 0	1 0	0 1

$g_{a,j} = g_{b,j-1} +^b_{0.1} g_{c,j-1}$

$c_{j/j-1}$ \ $v_{j'}$			v_j	
	$\bar{x}_b$	x_b	$\bar{x}_c$	x_c
$\bar{x}_a$	0 0	0 c	0 0	1 0
x_a	0 0	0 0	1 0	0 1

$g_{a,j} = g_{b,j'} *^b_{0.1} c$

c_{j-1} \ v_{j-1}					v_j	
	$\bar{x}_b$	x_b	$\bar{x}_c$	x_c	$\bar{x}_a$	x_a
$\bar{x}_a$	0 0	0 1	0 0	0 -1	0 0	1 0
x_a	0 0	0 0	0 0	0 0	1 0	0 1

$g_{a,j} = g_{b,j-1} -^b_{0.1} g_{c,j-1}$

c_j \ v_j	$\bar{x}_i$	x_i
$\bar{x}_i$	0 c	1 c
x_i	1 0	0 1

$g_{i,j} = c$

Figure 5 DGP polymatrix game gadgets.

For gates that originated in the circuit, the gadget is always embedded into the matrices $A^{v_{j-1},c_{j-1}}$ and A^{c_{j-1},v_j}, the synchronicity of the circuit ensures that the inputs for level j gates come from level $j-1$ gates. We have also added extra multiplication gates that copy values from the output of the circuit back to the input. These gates are of the form $g_{i,j} = g_{i',j+1}$, and are embedded into the matrices A^{v_j,c_j} and $A^{c_j,v_{j+1}}$.

The following lemma states that, in every Nash equilibrium, the strategies of the variable players exactly simulate the gates that have been embedded.

▶ **Lemma 13.** *In every mixed Nash equilibrium* **s** *of the game, the following are satisfied for each gate $g_{i,j}$.*

- *If $g_{i,j} = c$, then $s_{v_j}(x_i) = c$.*
- *If $g_{i,j} = g_{i_1,j-1} +^b_{0.1} g_{i_2,j-1}$, then $s_{v_j}(x_i) = s_{v_{j-1}}(x_{i_1}) +^b_{0.1} s_{v_{j-1}}(x_{i_2})$.*
- *If $g_{i,j} = g_{i_1,j-1} -^b_{0.1} g_{i_2,j-1}$, then $s_{v_j}(x_i) = s_{v_{j-1}}(x_{i_1}) -^b_{0.1} s_{v_{j-1}}(x_{i_2})$.*
- *If $g_{i,j} = g_{i_1,j'} *^b_{0.1} c$, then $s_{v_j}(x_i) = s_{v_{j'}}(x_{i_1}) *^b_{0.1} c$.*

Lemma 13 says that, in every Nash equilibrium of the game, the strategies of the variable players exactly simulate the gates, which by construction means that they give us a fixed point of the circuit F. Also note that it is straightforward to give a path decomposition for our interaction graph, where each node in the decomposition contains exactly two vertices from the game, meaning that the graph has pathwidth 1. So we have proved the following.

▶ **Theorem 14.** *It is* `PPAD`*-hard to find a Nash equilibrium of a tree polymatrix game, even when all players have at most twenty actions and the interaction graph has pathwidth 1.*

6 Open questions

For polymatrix games, the main open question is to find the exact boundary between tractability and hardness. Twenty-action pathwidth-1 tree polymatrix games are hard, but two-action path polymatrix games can be solved in polynomial time [10]. What about two-action tree polymatrix games, or path-polymatrix games with more than two actions?

For `2D-Brouwer` and `2D-LinearFIXP`, the natural question is: for which ϵ is it hard to find an ϵ-fixed point? We have shown that it is hard for $\epsilon = 0.2071$, while the case for $\epsilon = 0.5$ is trivial, since the point $(0.5, 0.5)$ must always be a 0.5-fixed point. Closing the gap between these two numbers would be desirable.

References

1 Siddharth Barman, Katrina Ligett, and Georgios Piliouras. Approximating Nash equilibria in tree polymatrix games. In *Proc. of SAGT*, pages 285–296, 2015.

2 Yang Cai and Constantinos Daskalakis. On minmax theorems for multiplayer games. In *Proc. of SODA*, pages 217–234, 2011. doi:10.1137/1.9781611973082.20.

3 Xi Chen and Xiaotie Deng. On the complexity of 2D discrete fixed point problem. *Theoretical Computer Science*, 410(44):4448–4456, 2009.

4 Xi Chen, Xiaotie Deng, and Shang-Hua Teng. Settling the complexity of computing two-player Nash equilibria. *Journal of the ACM*, 56(3):14:1–14:57, 2009.

5 Constantinos Daskalakis, Paul W. Goldberg, and Christos H. Papadimitriou. The complexity of computing a Nash equilibrium. *SIAM Journal on Computing*, 39(1):195–259, 2009.

6 Argyrios Deligkas, John Fearnley, Tobenna Peter Igwe, and Rahul Savani. An empirical study on computing equilibria in polymatrix games. In *Proc. of AAMAS*, pages 186–195, 2016.

7 Argyrios Deligkas, John Fearnley, and Rahul Savani. Computing constrained approximate equilibria in polymatrix games. In *Proc. of SAGT*, pages 93–105, 2017.

8 Argyrios Deligkas, John Fearnley, and Rahul Savani. Tree polymatrix games are PPAD-hard, 2020. arXiv:2002.12119.

9 Argyrios Deligkas, John Fearnley, Rahul Savani, and Paul G. Spirakis. Computing approximate Nash equilibria in polymatrix games. *Algorithmica*, 77(2):487–514, 2017.

10 Edith Elkind, Leslie Ann Goldberg, and Paul W. Goldberg. Nash equilibria in graphical games on trees revisited. In *Proc. of EC*, pages 100–109, 2006.

11 Kousha Etessami and Mihalis Yannakakis. On the complexity of Nash equilibria and other fixed points. *SIAM Journal on Computing*, 39(6):2531–2597, 2010.

12 Michael L. Littman, Michael J. Kearns, and Satinder P. Singh. An efficient, exact algorithm for solving tree-structured graphical games. In *Proc. of NIPS*, pages 817–823. MIT Press, 2001.

13 Ruta Mehta. Constant rank two-player games are PPAD-hard. *SIAM J. Comput.*, 47(5):1858–1887, 2018.

14 Luis E. Ortiz and Mohammad Tanvir Irfan. Tractable algorithms for approximate Nash equilibria in generalized graphical games with tree structure. In *Proc. of AAAI*, pages 635–641, 2017.

15 Christos H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *J. Comput. Syst. Sci.*, 48(3):498–532, 1994.

16 Aviad Rubinstein. Settling the complexity of computing approximate two-player Nash equilibria. In *Proc. of FOCS*, pages 258–265, 2016.

17 Aviad Rubinstein. Inapproximability of Nash equilibrium. *SIAM J. Comput.*, 47(3):917–959, 2018.

Spectral Sparsification via Bounded-Independence Sampling

Dean Doron
Department of Computer Science, Stanford University, CA, USA
ddoron@stanford.edu

Jack Murtagh
School of Engineering and Applied Sciences, Harvard University, Cambridge, MA, USA
jmurtagh@g.harvard.edu

Salil Vadhan
School of Engineering and Applied Sciences, Harvard University, Cambridge, MA, USA
salil_vadhan@harvard.edu

David Zuckerman
Department of Computer Science, University of Texas at Austin, TX, USA
diz@cs.utexas.edu

Abstract

We give a deterministic, nearly logarithmic-space algorithm for mild spectral sparsification of undirected graphs. Given a weighted, undirected graph G on n vertices described by a binary string of length N, an integer $k \leq \log n$ and an error parameter $\varepsilon > 0$, our algorithm runs in space $\widetilde{O}(k \log(N \cdot w_{\max}/w_{\min}))$ where $w_{\max}$ and $w_{\min}$ are the maximum and minimum edge weights in G, and produces a weighted graph H with $\widetilde{O}(n^{1+2/k}/\varepsilon^2)$ edges that spectrally approximates G, in the sense of Spielmen and Teng [52], up to an error of ε.

Our algorithm is based on a new bounded-independence analysis of Spielman and Srivastava's effective resistance based edge sampling algorithm [51] and uses results from recent work on space-bounded Laplacian solvers [41]. In particular, we demonstrate an inherent tradeoff (via upper and lower bounds) between the amount of (bounded) independence used in the edge sampling algorithm, denoted by k above, and the resulting sparsity that can be achieved.

2012 ACM Subject Classification Theory of computation → Pseudorandomness and derandomization

Keywords and phrases Spectral sparsification, Derandomization, Space complexity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.39

Category Track A: Algorithms, Complexity and Games

Related Version A full version of the paper is available at [19], `https://eccc.weizmann.ac.il/report/2020/026/`.

Funding *Dean Doron*: Research supported by a Motwani Postdoctoral Fellowship.
Jack Murtagh: Research supported by NSF grant CCF-1763299.
Salil Vadhan: Research supported by NSF grant CCF-1763299 and a Simons Investigator Award.
David Zuckerman: Research supported in part by NSF Grant CCF-1705028 and a Simons Investigator Award (#409864).

Acknowledgements We thank Jelani Nelson for his insights on spectral sparsification via k-wise independent sampling. We also thank Jarosław Błasiok for helpful discussions about random matrices. The first author would like to thank Tselil Schramm and Amnon Ta-Shma for interesting conversations.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 39; pp. 39:1–39:21
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

The graph sparsification problem is the following: given a weighted, undirected graph G, compute a graph H that has very few edges but is a close approximation to G for some definition of approximation. In general, graph sparsifiers are useful for developing more efficient graph-theoretic approximation algorithms. Algorithms whose complexity depend on the number of edges in the graph will be more efficient when run on the sparser graph H, and if H approximates G in an appropriate way, the result on H may be a good approximation to the desired result on G. In this work, we present an algorithm that can be implemented deterministically in small space and achieves sparsification in the *spectral sense* of Spielman and Teng [52]. (See Section 1.2 below for a more formal statement of our main result.)

1.1 Background

Motivated by network design and motion planning, Chew [10] studied *graph spanners*, which are sparse versions of graphs that approximately preserve the shortest distance between each pair of vertices. Benczúr and Karger [6] defined *cut sparsifiers* whose notion of approximation is that every cut of H has size within a $(1 \pm \varepsilon)$ factor of the size of the corresponding cut in G. They showed that every graph G on n vertices has a cut sparsifier H with $O(n \cdot \log n/\varepsilon^2)$ edges and gave a randomized algorithm for computing such cut sparsifiers. Their algorithm runs in nearly linear time (i.e., $\widetilde{O}(m)$ where m is the number of edges in G and the $\widetilde{O}(\cdot)$ notation hides polylogarithmic factors) and they used it to give a faster algorithm for approximating minimum s-t cuts.

Spielman and Teng introduced *spectral sparsifiers*, which define approximation between the graph and its sparsifier in terms of the quadratic forms of their *Laplacians* [52]. The Laplacian of an undirected graph is the matrix $L = D - A$ where A is the adjacency matrix of the graph and D is the diagonal matrix of vertex degrees (i.e. D_{ii} equals the weighted degree of vertex i). H is said to be an *ε-spectral approximation* of G if for all vectors $v \in \mathbb{R}^n$, we have that $v^\top \widetilde{L} v \in (1 \pm \varepsilon) \cdot v^\top L v$, where $\widetilde{L}$ and L are the Laplacians of H and G, respectively. Spectral sparsifiers generalize cut sparsifiers, which can be seen by observing that when $v \in \{0,1\}^n$, v is the characteristic vector of some set of vertices $S \subseteq [n]$ and $v^\top L v$ equals the sum of the weights of the edges cut by S.

Spielman and Teng showed that all graphs have spectral sparsifiers with $O(n \cdot \log^{O(1)} n/\varepsilon^2)$ edges and gave a nearly linear time randomized algorithm for computing them with high constant probability. Their spectral sparsifiers were a key ingredient that they used to develop the first nearly linear time algorithm for solving Laplacian systems. These fast Laplacian solvers spawned a flurry of improvements and simplifications [16, 26, 28, 27, 30, 31, 34, 44] as well as extensions to directed graphs [15, 14, 13] and to the space-bounded setting [18, 41, 1]. Spectral sparsification and the nearly linear time Laplacian solvers that use them have been critical primitives that have enabled the development of faster algorithms for a wide variety of problems including max flow [37, 11, 17, 24, 33], random generation of spanning trees, [25, 40, 49], and other problems in computer science [43, 29].

Spielman and Srivastava [51] gave a spectral sparsification algorithm that both simplified and improved upon the algorithm of Spielman and Teng. They show that randomly sampling edges, independently with probabilities proportional to their *effective resistances* produces a good spectral sparsifier with high probability. Viewing a graph as an electrical network, the effective resistance of an edge (a, b) is the potential difference induced between them when a unit of current is injected at a and extracted at b (or vice versa). More formally, the effective resistance of an edge (a, b) in a graph with Laplacian L is $R_{ab} = (e_a - e_b)^\top L^+ (e_a - e_b)$, where e_i denotes the ith standard basis vector and L^+ denotes the *Moore-Penrose pseudoinverse* of L[1]. Spielman and Srivastava proved the following theorem.

▶ **Theorem 1** (spectral sparsification via effective resistance sampling [51, 50]). *Let $G = (V, E, w)$ be a weighted graph on n vertices and for each edge $(a, b) \in E$ with weight w_{ab}, define $p_{ab} = \min\{1, 4 \cdot \log n \cdot w_{ab} \cdot R_{ab}/\varepsilon^2\}$, where R_{ab} is the effective resistance of (a, b) as defined above. Construct a sparsifier H by sampling edges from G independently such that each edge (a, b) in G is added to H with probability p_{ab}. For edges that get added to H, reweight them with weight w_{ab}/p_{ab}. Let L and $\widetilde{L}$ be the Laplacians of G and H, respectively. Then, with high probability,*

1. *H has $O(n \cdot (\log n)/\varepsilon^2)$ edges, and,*
2. *$\widetilde{L}$ ε-spectrally approximates L.*

Furthermore, this procedure can be implemented to run in time $\widetilde{O}(\frac{m}{\varepsilon^2} \cdot \log(w_{\max}/w_{\min}))$, where m is the number of edges in G and $w_{\max}, w_{\min}$ are the maximum and minimum edge weights of G, respectively.[2]

The sparsity achieved by the Spielman and Srivastava sparsifiers was improved by Batson, Spielman and Srivastava [5], who gave a deterministic algorithm for computing ε-spectral sparsifiers with $O(n/\varepsilon^2)$ edges, which is asymptotically optimal, however, their algorithm is less efficient, running in time $O(m \cdot n^3/\varepsilon^2)$. Work on these optimal sparsifiers continued with another slightly faster deterministic algorithm [53] followed by an $O(n^{2+\varepsilon})$-time randomized algorithm [2], and culminating in the randomized algorithms of Lee and Sun who achieved almost-linear time [35] and finally nearly-linear time [36].

1.2 Our Main Result

In this work we study the deterministic *space complexity* of computing spectral sparsifiers. Our main result is a deterministic, nearly-logarithmic space algorithm for computing *mild spectral sparsifiers*, that is, graphs with $O(n^{1+\alpha}/\varepsilon^2)$ edges for any constant $\alpha > 0$.

▶ **Theorem 2** (see also Theorem 19). *Let G be a connected, weighted, undirected graph on n vertices, $k \in \mathbb{N}$ an independence parameter and $\varepsilon > 0$ an error parameter. There is a deterministic algorithm that on input G, k, and ε, outputs a weighted graph H that is an ε-spectral sparsifier of G and has $O(n^{1+2/k} \cdot (\log n)/\varepsilon^2)$ edges. The algorithm runs in space $O(k \log(N \cdot w) + \log(N \cdot w) \log\log(N \cdot w))$, where $w = w_{\max}/w_{\min}$ is the ratio of the maximum and minimum edge weights in G and N is the length of the input.*

The closest analogue to spectral sparsifiers in the space-bounded derandomization literature is the *derandomized square* of Rozenman and Vadhan [47], a graph operation that produces a sparse approximation to the square of a graph.[3] The derandomized square was introduced to give an alternative proof to Reingold's celebrated result that Undirected S-T Connectivity can be solved in deterministic logspace [46]. Murtagh, Sidford, Reingold,

[1] L^+ is a matrix with the same kernel as L that acts as an inverse of L on the orthogonal complement of the kernel. See Section 2.2 for a formal definition.

[2] In their original paper, [51], they fix the number of edges in the sparsifier in advance resulting in a slightly different theorem statement and analysis. The version we cite here and what we model our algorithm after was presented later in [50].

[3] The *square* of a graph G is a graph on the same vertex set whose edges correspond to all walks of length 2 in G.

and Vadhan [41] showed that the derandomized square actually produces a *spectral sparsifier* of the square of a graph and this was a key observation they used to develop a deterministic, nearly logarithmic space algorithm for solving Laplacian systems. Later the sparsification benefits of the derandomized square were also used in nearly logarithmic space algorithms for deterministically approximating random walk probabilities and for solving Laplacian systems in Eulerian directed graphs [42, 1].

For a d-regular graph G on n vertices, its square G^2 has degree d^2 and the derandomized square computes an ε-spectral approximation to G^2 with degree $O(d/\varepsilon^2)$. On the other hand, applying our sparsification to G^2 results in an ε-spectral approximation with on average $O(n^\alpha/\varepsilon^2)$ edges adjacent to each vertex for any constant α, which is independent of d and much sparser when $d = \omega(n^\alpha)$. Also, our algorithm can sparsify any undirected graph, not just squares. Our algorithm does not replace the derandomized square, however, because the derandomized square can be iterated very space efficiently, a property that is used in all of its applications thus far. Nevertheless, given the success of spectral sparsification and Laplacian solvers in the nearly-linear time context and the fruit borne of porting these techniques to the logspace setting, we are hopeful that our spectral sparsifiers will have further applications in derandomization of space-bounded computation.

1.3 Techniques

Our deterministic space-efficient algorithm is modeled after the effective resistance based sampling algorithm of Spielman and Srivastava (Theorem 1). Although the Spielman and Srivastava procedure is randomized and does not achieve optimal sparsity, the known algorithms that do ([5, 53, 2, 35, 36]) are more involved and often sequential in nature so do not seem as amenable to small-space implementations.

To derandomize the Spielman-Srivastava algorithm, we follow the standard approach of first reducing the number of random bits used to logarithmic, and then enumerating over all random choices of the resulting algorithm. Following [39, 3], a natural way to reduce the number of random bits used is to do the edge sampling only k-wise independently for some $k \ll |E|$ rather than sampling every edge independently from all other edges.

Let k be our bounded-independence parameter. Namely, we are only guaranteed that every subset of k edges is chosen independently (with the right marginals), however there may be correlations between the choices in tuples of size $k+1$. It is well known that such a sampling can be performed using fewer random bits. By [51], we know that $k = |E|$ will, with high probability, produce an ε-spectral sparsifier with $O(n \cdot \log n/\varepsilon^2)$ edges in expectation. What about much smaller values of k? In Section 3, we prove the following:

▶ **Theorem 3** (informal; see Theorem 9). *Let G be a connected weighted undirected graph on n vertices with Laplacian L, $k \in \mathbb{N}$ an independence parameter and $\varepsilon > 0$ an error parameter. Let H be the graph which is the output of Spielman and Srivastava's sampling-based sparsification algorithm (Theorem 1), when the edge sampling is done in a k-wise independent manner, and let $\widetilde{L}$ be the Laplacian of H. Then, with high constant probability, $\widetilde{L}$ ε-approximates L and H has $O(n^{1+2/k} \cdot (\log n)/\varepsilon^2)$ edges.*

A first thing to observe is that $k = \log n$ gives the same result as in [51]. More importantly, the above shows that the result *interpolates*: Even for a constant k, Theorem 3 gives a *mild sparsification* that sparsifies dense graphs to $O(n^{1+\alpha})$ expected edges, where $\alpha > 0$ is an arbitrarily small constant.

We prove Theorem 3 by extending the arguments in [51, 50]. For every edge $(a, b) \in E$, we define a random matrix X_{ab} that corresponds to the choice made by the sparsification algorithm, in such a way that $X = \sum_{(a,b)\in E} X_{a,b}$ relates to the resulting Laplacian $\widetilde{L}$.[4] Let Π be the orthogonal projection onto the image of L. Following [51, 50], we show that $\widetilde{L}$ ε-spectrally approximates L (equivalently, that H is an ε-spectral sparsifier for G) with high probability if $X - \Pi$ has bounded moments. Deriving a tail bound that relies on the first k moments alone, we can proceed with the analysis as if the X_{ab}'s were *truly independent*. More specifically, we bound $\mathrm{Tr}(\mathbb{E}_X[(X - \Pi)^k])$ using a matrix concentration result due to Chen, Gittens and Tropp [9]. For the complete details, as well as how our argument differs from [51, 50], see Section 3.

Getting a Deterministic Algorithm

Theorem 3 readily gives a simple, randomness-efficient algorithm, as k-wise independent sampling of edges only requires $O(k \cdot \log(N \cdot w))$ random bits [23, 3] (See Lemma 8). However, more work is needed to obtain a space-efficient deterministic algorithm. First, we need to be able to compute the marginal sampling probabilities, which depend on the effective resistances R_{ab}. Fortunately, the recent work of Murtagh et al. [41] allows us to approximate the effective resistances using only $O(\log(N \cdot w) \log\log(N \cdot w))$ space and we show that the k-wise independent sampling procedure can tolerate the approximation.

Next, to obtain a deterministic algorithm, we can enumerate over all possible random choices of the algorithm in space $O(k \cdot \log(N \cdot w))$ and compute a candidate sparsifier H for each. We are guaranteed that at least one (indeed, most) of the resulting graphs H is a good sparsifier for G but how can we identify which one? To do this, it suffices for us, given Laplacians L and $\widetilde{L}$, to distinguish the case that $\widetilde{L}$ is an ε-spectral approximation of L from the case that $\widetilde{L}$ is not a $2 \cdot \varepsilon$-spectral approximation of L. We reduce that problem to that of approximating the spectral radius of $M = ((\widetilde{L} - L)L^+/\varepsilon)^2$, where L^+ is the pseudoinverse of L, which can be approximated in nearly logarithmic space by [41]. In fact, it will be sufficient to check whether the trace of a logarithmically high power of M is below a certain threshold to deduce that the spectral radius of M does not exceed 1. In Section 5.2, we show that the latter case implies that $\widetilde{L}$ indeed ε-approximates L.

The deterministic, nearly logarithmic space Laplacian solver of [41] only worked for *multigraphs*, i.e. graphs with integer edge weights. To get our result for arbitrary weighted graphs, we extend the work of [41] and give a deterministic, nearly logarithmic space Laplacian solver for arbitrary undirected weighted graphs. Combining this extension with the k-wise independent analysis of the edge sampling algorithm (Theorem 3) and the verification procedure described above lets us prove our main result Theorem 2.

1.4 Lower Bounds for Bounded-Independence Sampling

Having established an upper bound on the amount of independence required for the edge-sampling procedure (Theorem 3), a natural goal would be to come up with a corresponding lower bound. Theorem 3 tells us that in order to sparsify to $\widetilde{O}(n^{1+\alpha})$ expected edges, we can use k-wise independent sampling for $k = 2/\alpha$. Can a substantially smaller choice of k perform just as well? In Section 4, we show that our upper bound of $k = 2/\alpha$ is tight up to a small constant factor.

[4] Specifically, $X = L^{+/2}\widetilde{L}L^{+/2}$, where $L^{+/2}$ is the square-root of the pseudoinverse of L.

▶ **Theorem 4** (informal; see Theorem 11). *For every small enough $\alpha > 0$ there exist infinitely many connected graphs $G = (V = [n], E)$ with all effective resistances equal that are d-regular with $d = \Omega(n^{\alpha})$ and a distribution $\mathcal{D} \sim \{0,1\}^{|E|}$ that is k-wise independent for $k = \lfloor 4/3\alpha \rfloor$ with marginals $1/2$ that would fail to produce an ε-spectral sparsifier of G to within any $\varepsilon > 0$ with high probability.*

Our family of "bad graphs" will be dense graphs having large girth. Namely, given a girth g and an integer $d \geq 3$, we consider graphs $G = (V = [n], E)$ satisfying $d \geq n^{\gamma/g} + 1$ for some constant $0 < \gamma < 2$ [32]. Getting an infinite family of graphs with γ approaching 2 (and specifically attaining the *Moore bound*), even non-explicitly, has been the subject of extensive study (see [21] and references therein). See also Section 4.1 for a further discussion. Given a sparsification parameter $\alpha > 0$, we set $k \approx \gamma/\alpha$ and take a graph G on n vertices with girth $g = k + 1$ and degree $d > n^{\gamma/g} + 1$.

Our construction of the distribution $\mathcal{D}$ is inspired by Alon and Nussboim [4]: choose a partition of the vertices $V = V_0 \uplus V_1$ uniformly at random, and for every edge $e = (u, v) \in E$, include it in the sample if and only if either $u, v \in V_0$ or $u, v \in V_1$. Clearly, sampling edges according to $\mathcal{D}$ results in a disconnected graph almost surely. However, we show that $\mathcal{D}$ is indeed k-wise independent, relying on the fact that the girth of G is $k + 1$.

To obtain Theorem 4 we use the family of graphs given by Lazebnik et al. [32] who obtained $\gamma = 4/3$. Indeed, any improvement in γ would bring our upper bound of $k \approx 2/\alpha$ and lower bound of $k \approx \gamma/\alpha$ closer together.

1.5 Open Problems

An interesting open problem is to achieve improved sparsity, e.g. $O(n \cdot (\log n)/\varepsilon^2)$ matching [51]. Our algorithm would require space $\Omega(\log^2 n)$ to achieve this sparsity, due to setting $k = \Omega(\log n)$. We remark that previous work implies that this can be done in *randomized* logarithmic space. Indeed, Doron et al. [18] gave a randomized algorithm for solving Laplacian systems in logarithmic space (without $\log\log(\cdot)$ factors), and this implies that one can approximate effective resistances and hence implement the Spielman-Srivastava edge sampling with full independence in randomized logspace. It is also an interesting question whether there is a nearly logspace algorithm (even randomized) that produces spectral sparsifiers of optimal sparsity (i.e., $O(n/\varepsilon^2)$ edges).

Finally, while it is not known how to compute spectral sparsifiers of arbitrary *directed* graphs, there has been progress on sparsifying Eulerian digraphs in the nearly-linear time literature [15, 14, 13, 12]. Given the recent advance of a nearly-logarithmic space solver for Eulerian Laplacian systems [1], an interesting question is sparsifying Eulerian graphs in small space.

2 Preliminaries

We will work with undirected weighted graphs, $G = (V, E, w)$, where w is a vector of length $|E|$ and each edge $(a, b) \in E$ is associated with a positive weight $w_{ab} > 0$. At times we refer to undirected *multigraphs*, which are weighted graphs where all of the weights are integers. The adjacency matrix of G is a symmetric, real-valued matrix A in which $A_{ij} = w_{ij}$ if $(i, j) \in E$ and $A_{ij} = 0$ otherwise.

For any matrix A, its *spectral norm* $\|A\|$ is $\max_{\|x\|=1}\|Ax\|_2$, which is also the largest singular value of A. For any square matrix A, its *spectral radius*, denoted $\rho(A)$, is the largest absolute value of its eigenvalues. When A is real and symmetric, the spectral norm equals the spectral radius. The spectral norm is sub-multiplicative, i.e., $\|AB\| \le \|A\|\,\|B\|$. We denote by $A^\top$ the transpose of A. We denote by $\mathbf{1}$ the all-ones vector, by $\mathbf{0}$ the all-zeros vector, and e_a is the vector with 1 in the a-th coordinate and 0 elsewhere, where e_a's dimension will be understood from context (i.e., e_a is the a-th standard basis vector).

The *trace* of a matrix $A \in \mathbb{R}^{n\times n}$, is $\mathrm{Tr}(A) = \sum_{i\in[n]} A_{ii}$, which also equals the sum of its eigenvalues. The trace is invariant under cyclic permutations, i.e., $\mathrm{Tr}(AB) = \mathrm{Tr}(BA)$. The expectation of a *random matrix* is the matrix of the coordinate-wise expectations. More formally, if A is a random matrix, then $\mathbb{E}[A] = \widehat{A}$ where $\widehat{A}_{ij} = \mathbb{E}[A_{ij}]$ for all $i,j \in [n]$. The trace and the expectation are both linear functions of a matrix and they commute. That is, for all random matrices A, we have $\mathrm{Tr}(\mathbb{E}[A]) = \mathbb{E}[\mathrm{Tr}(A)]$ (see, e.g.,[45]).

2.1 PSD Matrices and Spectral Approximation

A symmetric matrix $A \in \mathbb{R}^{n\times n}$ is *positive semi-definite* (PSD), denoted $A \succeq 0$, if for every $x \in \mathbb{R}^n$ it holds that $x^\top Ax \ge 0$, or equivalently, if all its eigenvalues are non-negative. We write $A \succeq B$ if $A - B \succeq 0$.

▶ **Definition 5.** *Let A and B be $n \times n$ symmetric PSD matrices. For a real $\varepsilon > 0$, we say that A is an* ε-spectral approximation *of B, denoted $A \approx_\varepsilon B$, if*

$$(1-\varepsilon)B \preceq A \preceq (1+\varepsilon)B.$$

When A and B share an eigenvector basis $v_1,\ldots,v_n$, Definition 5 is equivalent to requiring $(1-\varepsilon)\mu_i \le \lambda_i \le (1+\varepsilon)\mu_i$, where $\lambda_1,\ldots,\lambda_n$ are the eigenvalues of A corresponding to $v_1,\ldots,v_n$ and $\mu_1,\ldots,\mu_n$ are the eigenvalues of B corresponding to $v_1,\ldots,v_n$.

2.2 The Moore-Penrose Pseudoinverse

Let A be any linear operator. The *Moore-Penrose pseudoinverse* of A, denoted A^+, is defined as follows. If $A = U\Sigma V^\top$ is the singular value decomposition (SVD) of A, the pseudoinverse is given by $A^+ = V\Sigma^+U^\top$ where Σ^+ is the matrix obtained by taking the reciprocal of each nonzero diagonal element of Σ, and leaving the zeros intact. When A is a symmetric PSD matrix, the SVD coincides with the eigen-decomposition and so if $\lambda_1,\ldots,\lambda_n$ are the eigenvalues of A then A^+ shares the same eigenvector basis and has eigenvalues $\lambda_1^+,\ldots,\lambda_n^+$, where $\lambda_i^+ = 1/\lambda_i$ if $\lambda_i \ne 0$ and $\lambda_i^+ = 0$ otherwise.

Also note that if A is real then A^+ is real-valued as well.

A *square root* of a matrix A is any matrix X that satisfies $X^2 = A$. When A is symmetric and PSD, it has a unique symmetric PSD square root, which we write as $A^{1/2}$. If $A = U\Sigma U^\top$ is the eigen-decomposition of A then $A^{1/2} = U\sqrt{\Sigma}U^\top$ where $\sqrt{\Sigma}$ is obtained by taking the square root of each diagonal element of Σ. We denote by $A^{+/2}$ the matrix $(A^+)^{1/2} = (A^{1/2})^+$.

2.3 The Graph Laplacian and Effective Resistance

Given a graph G on n vertices with an adjacency matrix A and degree matrix D (i.e., D is a diagonal matrix where $D_{ii} = \sum_{j=1}^n A_{ij}$ equals the weighted degree of vertex i in G), the *Laplacian* of G is the matrix

$$L = D - A.$$

For every undirected weighted graph $G = (V, E, w)$, its Laplacian L is symmetric and PSD, with smallest eigenvalue 0. The zero eigenvalue has multiplicity one if and only if G is connected. In this case, $\ker(L_G) = \operatorname{span}(\{\mathbf{1}\})$. For every edge $(a, b) \in E$, define the *edge Laplacian* of (a, b) to be

$$L_{ab} = (e_a - e_b)(e_a - e_b)^\top = (e_b - e_a)(e_b - e_a)^\top.$$

Note that $L = \sum_{(a,b)\in E} w_{ab} \cdot L_{ab}$.

It is often helpful to associate G with an electric circuit, where an edge $(a, b) \in E$ corresponds to a resistor of resistance $1/w_{ab}$. For each pair of vertices a and b, the *effective resistance* between them, denoted by R_{ab}, is the energy of the electrical flow that sends one unit of current from a to b. The effective resistance can be calculated using the pseudoinverse of the Laplacian:

$$R_{ab} = (e_a - e_b)^\top L^+(e_a - e_b).$$

(See [8] for more information on Laplacians and viewing graphs as electrical networks). A useful fact about effective resistances is Foster's Theorem:

▶ **Theorem 6** ([22]). *For every undirected weighted graph $G = (V, E, w)$ on n vertices it holds that*

$$\sum_{(a,b)\in E} w_{ab} \cdot R_{ab} = n - 1.$$

2.4 Bounded-Independence Sampling

Given a probability vector $p \in [0, 1]^m$, let $\operatorname{Bernoulli}(p)$ denote the distribution X over $\{0, 1\}^m$ where the bits are independent and for each $i \in [m]$, $\mathbb{E}[X_i] = p_i$. For a set $I \subseteq [m]$ and a string $z \in \{0, 1\}^m$, we let $z|_I \in \{0, 1\}^{|I|}$ be the restriction of z to the indices in I.

▶ **Definition 7.** *We say a distribution $X \sim \{0, 1\}^m$ is k-wise independent with marginals $p \in [0, 1]^m$ if for every set $I \subseteq [m]$ with $|I| \le k$, it holds that $X|_I = \operatorname{Bernoulli}(p|_I)$. We refer to X as a k-wise independent sample space with marginals p.*

Consider $G = (V, E, w)$ with $|E| = m$. Throughout, when we say *sampling edges in a k-wise independent manner*, we refer to the process of picking an element $x \in \{0, 1\}^m$ from a k-wise independent sample space uniformly at random and taking those edges $e \in E$ for which $x_e = 1$.

For $p \in [0, 1]^m$ and a positive integer t, we define $\lfloor p \rfloor_t$ to be the vector p' obtained by truncating every element of p after t bits. Thus, for each $i \in [m]$, $p'_i = 2^{-t}\lfloor 2^t p_i \rfloor$, and so $|p_i - p'_i| \le 2^{-t}$. The following lemma states that we can construct small k-wise independent sample spaces with any specified marginals.

▶ **Lemma 8** (following [23, 3]). *For every $m, k, t \in \mathbb{N}$ and $p \in [0, 1]^m$ there exists an explicit k-wise independent distribution $X \sim \{0, 1\}^m$ with marginals $\lfloor p \rfloor_t$, that can be sampled with $r = O(k \cdot \max\{t, \log m\})$ truly random bits. Furthermore, given $\rho \in \{0, 1\}^r$, the element $x \in \operatorname{Supp}(X)$ corresponding to the random bits ρ can be computed in $O(k \cdot \max\{t, \log m\})$ space.*

3 Sparsification via Bounded-Independence Sampling

In Section 1, we briefly introduced the Spielman-Srivastava sparsification algorithm [51] based on (truly) independent edge sampling, with probabilities proportional to the effective resistances of the edges. In this section, we explore the tradeoff between the amount of independence used in the edge sampling process and the resulting sparsity that can be achieved.

In particular, we analyze the algorithm Sparsify (see Algorithm 1). The algorithm gets as input an undirected, weighted, dense graph $G = (V, E, w)$ on n vertices, approximate effective resistances $\widetilde{R}_{ab}$ for each edge $(a, b) \in E$, a bounded independence parameter $k \le \log n$, a desired approximation error $\varepsilon > 0$, and a parameter $\delta > 0$ governing the success probability, and outputs a sparser graph H whose Laplacian ε-spectral approximates the Laplacian of G with probability at least $1 - 2\delta$.

Algorithm 1 Computing a spectral sparsifier via bounded independence sampling.

```
Sparsify($G = (V, E, w)$, $\{\widetilde{R}_{ab}\}_{(a,b)\in E}$, $k$, $\varepsilon$, $\delta$).
1. Initialize $H$ to be the empty graph on $n = |V(G)|$ vertices.
2. Set $s \leftarrow \frac{18e\log n}{\varepsilon^2} \cdot \left(\frac{n}{\delta}\right)^{2/k}$.
3. For every edge $(a, b) \in E$, set $p_{ab} \leftarrow \min\left\{1, w_{ab} \cdot \widetilde{R}_{ab} \cdot s\right\}$.
4. For every edge $(a, b) \in E$, add $(a, b)$ to $H$ with weight $w_{ab}/p_{ab}$ with probability $p_{ab}$.
   Do this sampling in a $k$-wise independent manner, following Lemma 8.
5. Return $H$.
```

First we will analyze Sparsify for the case where the effective resistances are given exactly, i.e. $\widetilde{R}_{ab} = R_{ab}$ for all $(a, b) \in E$. Then, in Section 3.2 we will analyze the more general case where we are given approximations to the effective resistances. This latter case is useful algorithmically because more efficient algorithms are known for estimating effective resistances than for computing them exactly, both in the time-bounded and space-bounded settings [51, 41].

3.1 Sparsification With Exact Effective Resistances

In this section we give our main theorem about Sparsify.

▶ **Theorem 9** (spectral sparsification via bounded independence). *Let $G = (V, E, w)$ be an undirected connected weighted graph on n vertices with Laplacian L and effective resistances $R = \{R_{ab}\}_{(a,b)\in E}$. Let $0 < \varepsilon < 1$, $0 < \delta < 1/2$ and let $k \le \log n$ be an even integer. Let H be the output of* Sparsify$(G, R, k, \varepsilon, \delta)$ *and let $\widetilde{L}$ be its Laplacian. Then, with probability at least $1 - 2\delta$ we have:*

1. *$\widetilde{L} \approx_\varepsilon L$, and,*
2. *H has $O\left(\frac{1}{\delta^{1+2/k}} \cdot \frac{\log n}{\varepsilon^2} \cdot n^{1+\frac{2}{k}}\right)$ edges.*

Spielman and Srivastava showed that by using truly independent sampling (i.e., $k = |E|$) in Sparsify, one can compute an ε-spectral sparsification of G with $O(n \cdot \log n/\varepsilon^2)$ edges, with high constant probability [51]. One immediate consequence of Theorem 9 is that $\log n$-wise independent sampling suffices to match the sparsity that truly independent sampling achieves. Another consequence of Theorem 9 is that for any constant $0 < \alpha < 1$ and any constant $\gamma < \alpha/2$, for $k \approx 2/(\alpha - 2\gamma)$, k-wise independent sampling achieves a spectral sparsifier with error $\varepsilon = n^{-\gamma}$ and $O(n^{1+\alpha})$ expected edges, with high constant probability.

The proof of Theorem 9 is modeled after Spielman and Srivastava's argument [51]. One difference is that the sparsification algorithm in [51] fixes the number of edges to be sampled in advance rather than having the number of edges be a random variable. They then prove spectral approximation by reducing the problem to a question about concentration of random matrices, which they resolve with a matrix Chernoff bound due to Rudelson and Vershynin [48]. We follow a variant of this argument for the case where the number of edges in the sparsifier is random and use a matrix concentration bound of Chen, Gittens, and Tropp [9]. This variant, for truly independent sampling, has appeared before in [50]. Our argument is modified to address the fact that we only use k-wise independent sampling, and the proof is given in the full version of the paper.

3.2 Sparsification With Approximate Effective Resistances

Spielman and Srivastava showed that the original version of spectral sparsification through effective resistance sampling (with fully independent sampling and fixing the number of edges in advance) is robust to small changes in the sampling probabilities. In this section we show the same is true of Sparsify. As said, this is useful because more efficient algorithms are known for estimating effective resistances than for computing them exactly, and we will also use this fact for our space-bounded algorithm for sparsification in Section 5.

The lemma below says that if we only have small multiplicative approximations to the effective resistances then the guarantees of Theorem 9 still hold with a small loss in the sparsity.

▶ **Lemma 10.** *Let $G = (V, E, w)$ be an undirected connected weighted graph on n vertices with Laplacian L. Let $0 < \varepsilon < 1$, $0 < \delta < 1/2$ and let $k \le \log n$ be an even integer. For each $(a, b) \in E$, let $\widetilde{R}_{ab}$ be such that*

$$(1-\alpha)\cdot R_{ab} \le \widetilde{R}_{ab} \le (1+\alpha)\cdot R_{ab},$$

where R_{ab} is the effective resistance of (a, b) and $0 < \alpha < 1$. Let $\widetilde{R} = \{\widetilde{R}_{ab}/(1-\alpha)\}_{(a,b)\in E}$. Let H be the output of Sparsify$(G, \widetilde{R}, k, \varepsilon, \delta)$ *and let $\widetilde{L}$ be its Laplacian. Then, with probability at least $1 - 2\delta$ we have:*

1. *$\widetilde{L} \approx_{\varepsilon} L$, and,*
2. *H has $O\left(\frac{1+\alpha}{1-\alpha} \cdot \frac{1}{\delta^{1+2/k}} \cdot \frac{\log n}{\varepsilon^2} \cdot n^{1+\frac{2}{k}}\right)$ edges.*

A proof of Lemma 10 can be found in the full version. Note that we could equivalently define Sparsify to take approximate sampling probabilities as input (i.e., $(1-\alpha)p_{ab} \le \widetilde{p}_{ab} \le (1+\alpha)p_{ab}$) rather than α-approximate effective resistances and the same lemma applies.

4 Lower Bounds for Bounded-Independence Sampling

In this section we give a lower bound for sampling-based bounded independence sparsification. Our lower bound will hold even for unweighted, simple, regular graphs in which all the effective resistances are the same, so for this section, assume $G = (V = [n], E)$ is such a graph. In Section 3 we measure sparsity in terms of the number of edges in the graph. We use this measure rather than average degree because in weighted graphs, the degree of a vertex v typically refers to the sum of the weights of the edges incident to v, whereas in sparsification algorithms we are trying to minimize the number of edges incident to v, regardless of their weight. In this section, we will sometimes refer to average degree rather than number of edges. When we refer to the average degree of a weighted graph, we mean the average number of edges incident to each vertex. For simple, unweighted graphs, these quantities are the same.

Fix some $\alpha > 0$. Theorem 9 tells us that if we want to sparsify G to within error ε and expected degree $s = O\left(n^{\alpha} \cdot \log n/\varepsilon^2\right)$, we can do so by sampling each edge with probability $p = s \cdot (n-1)/|E|$ in a k-wise independent manner, where $k = 2/\alpha$ (rounded to an even integer).[5] We now prove that $k \geq 4/3\alpha$ is essential for such a sampling procedure, at least for constant α.

▶ **Theorem 11** (lower bound for spectral sparsification via bounded independence)**.** *Fix $c > 0$. For every $\alpha \in (0, 4/15]$, there exist infinitely many n's for which the following holds.*

There exists a connected graph $G = (V = [n], E)$ whose effective resistances are all equal and a distribution $\mathcal{D} \sim \{0,1\}^{|E|}$ that is k-wise independent for $k = \lfloor 4/3\alpha \rfloor$ with marginals $1/2$ that would fail to sparsify G to within any error $\varepsilon > 0$ and expected degree $s = c \log n \cdot n^{\alpha_0}$, where $\alpha_0 \geq (1-2\alpha)\alpha$.

More specifically, sampling a subgraph of G according to $\mathcal{D}$ would result in a disconnected graph with probability at least $1 - 2/2^n$.

We note that a disconnected graph fails to be a good spectral sparsifier of a connected graph, which is implicit in Theorem 11. Formally:

▷ Claim 12. Let G and $\widetilde{G}$ be undirected graphs on n vertices with Laplacians L and $\widetilde{L}$, respectively. If G is connected and $\widetilde{G}$ is disconnected then $\widetilde{L} \not\approx_{\varepsilon} L$ for any $\varepsilon > 0$.

We give a proof of Claim 12 in the full version of the paper.

4.1 Moore-Like Graphs With a Given Girth

Toward proving Theorem 11, we will need, for every bounded-independence parameter k, an infinite family of graphs satisfying certain properties. Recall that the *girth* of a graph G is the length of the shortest cycle in G. We will need an infinite family of girth-g graphs having large degree. Formally:

▶ **Definition 13.** *Given $\gamma > 0$ and $g \colon \mathbb{N} \to \mathbb{N}$, an infinite family of graphs $\{G_i = (V_i = [n_i], E_i)\}_{i \in \mathbb{N}}$ is* (g, γ)-Moorish *if for every $i \in \mathbb{N}$, G_i is connected, has girth at least $g(n_i)$ and is d_i-regular for some $d_i \geq n_i^{\gamma/g(n_i)} + 1$.*

The problem of finding such families of graphs, or even proving their existence in some regime of parameters, has been widely studied in extremal graph theory. A simple counting argument ([20], see also [8]) shows that (g, γ)-Moorish families of graphs can only exist when $\gamma \leq 2$:

▶ **Lemma 14** (the Moore bound, see, e.g., [8])**.** *Every d-regular graph of girth g on n vertices satisfies $n \geq 2 \cdot ((d-1)^{g/2} - 1)/(d-2)$.*

Still, no families with γ approaching 2 for arbitrary girths are known. The Ramanujan graphs of Lubotzky, Phillips and Sarnak [38] were shown to obtain $\gamma \geq 4/3$ by Biggs and Boshier [7]. Lazebnik, Ustimenko and Woldar [32] slightly improved upon [38] in the lower-order terms, but more importantly for us, the family they construct consists of *edge-transitive graphs.*

[5] We used the fact that for every $(a,b) \in E$, $p_{ab} \leftarrow \min\{1, R_{ab}s\} = R_{ab}s = R \cdot s$, which can be argued as follows. When all effective resistances equal R, we have $R = (n-1)/|E|$ due to Theorem 6. Now, if G has $n \cdot s$ edges or fewer, then it already achieves the desired sparsity so without loss of generality we can assume that $|E| > n \cdot s$. Hence, $R \cdot s < (n-1)s/ns < 1$. Also, the resulting graph should indeed be a weighted one, however all its weights will be the same, $1/p$.

▶ **Theorem 15** ([32]). *For every prime power d and even integer $g \geq 6$ there exists a d-regular explicit simple, edge-transitive graph with $n \leq 2d^{g-\lfloor \frac{g-3}{4} \rfloor -4}$ vertices and girth g. In particular, for every prime power d there exists a $(g, \gamma = 4/3)$-Moorish family of edge-transitive graphs, where $\mathrm{Im}(g) = \{6, 8, \dots\}$.*

Intuitively, in an edge-transitive graph the local environment of every edge (i.e., the vertices and edges adjacent to it) looks the same. More formally, an edge-transitive graph is one in which any two edges are equivalent under some element of its automorphism group. As the computation of the effective resistance is not affected by an automorphism, we can conclude the following claim.

▷ Claim 16. Let $G = (V, E)$ be an unweighted edge-transitive graph. Then, for every two edges $e = (a, b)$ and $e' = (a', b')$ in E it holds that $R_{ab} = R_{a'b'}$.

4.2 The Lower Bound Proof

We next prove our main result for this section, showing that Moorish edge-transitive graphs cannot be sparsified via bounded-independence edge sampling when k is too small. Our proof can be seen as an extension of an argument by Alon and Nussboim [4], who studied the bounded independence relaxation of the usual Erdős-Rényi random graph model, where it is only required that the distribution of any subset of k edges is independent. They provide upper and lower bounds on the minimal k required to maintain properties that are satisfied by a truly random graph, and in particular they show that there exists a pairwise independent distribution $\mathcal{D}$ over edges with marginals $1/2$ such that a random graph sampled from $\mathcal{D}$ is disconnected almost surely.

As a warm-up, we extend the argument in [4] and show that *3-wise* independence also does not suffice, even for the special case of sparsifying the complete graph.

▶ **Lemma 17.** *Let $G = (V = [n], E)$ be the complete graph. There exists a distribution $\mathcal{D} \sim \{0,1\}^{|E|}$ that is 3-wise independent with marginals $1/4$ such that sampling a subgraph of G according to $\mathcal{D}$ would result in a disconnected graph with probability at least $1 - 2/2^n$.*

Proof. We first set some notations. Let $\mathcal{G}(A, p)$ be the usual Erdős-Rényi model, in which each edge between two vertices in A is included in the graph with probability p. Let $\mathcal{B}(A)$ be the natural distribution over complete bipartite graphs: Choose a partition $A = A_1 \uplus A_2$ uniformly at random and include all edges between A_1 and A_2.

We construct $\mathcal{D} \sim \{0,1\}^{|E|}$ as follows. Choose a partition $[n] = V_0 \uplus V_1$ uniformly at random. On V_0, draw a graph from $\mathcal{G}(V_0, 1/2)$ and on V_1, draw a graph from $\mathcal{B}(V_1)$. Clearly, sampling G' according to $\mathcal{D}$ would result in a disconnected graph unless $V_0 = \varnothing$ or $V_1 = \varnothing$, which occurs with probability at most $2/2^n$, so what is left to show is that $\mathcal{D}$ is 3-wise independent with marginals $1/4$.

This is equivalent to saying that for every $T \subseteq E$ with $|T| \leq 3$, $\Pr[\forall e \in T,\ \mathcal{D}(e) = 1] = 1/4^{|T|}$. Let us first consider the case $|T| = 1$, i.e. $T = e$ for a single edge $e \in E$. Notice that $\mathcal{D}(e) = 1$ only if both endpoints of e appear in the same side of the partition $V_0 \uplus V_1$, which occurs with probability $1/2$, and given that this occurs, e appears in $\mathcal{G}(V_0, 1/2)$ or $\mathcal{B}(V_1)$ with probability $1/2$. Thus, $\Pr[\mathcal{D}(e) = 1] = 1/4$, as desired.

Next, fix a set $T \subseteq E$ of $t \in \{2, 3\}$ edges and note that we can assume without loss of generality that these edges form either a path or a triangle (for $t = 3$), as disjoint paths will occur independently. If T forms a path, then similarly,

$$\Pr[T \in \mathcal{D}^{-1}(1)] = \Pr[V(T) \subseteq V_0] \cdot 2^{-t} + \Pr[V(T) \subseteq V_1] \cdot 2^{-t} = 2^{-(t+1)} \cdot 2^{-t} + 2^{-(t+1)} \cdot 2^{-t} = 4^{-t},$$

which is what we want. If T forms a triangle, then using the fact that a bipartite graph is triangle-free,

$$\Pr[T \in \mathcal{D}^{-1}(1)] = \Pr[V(T) \subseteq V_0] \cdot \frac{1}{8} = 4^{-3},$$

concluding the proof. ◀

The above lemma shows that one cannot sparsify the complete graph via ($k = 3$)-wise independent edge sampling. For a general k, we indeed need to resort to Moore-like graphs.

Proof of Theorem 11. Recalling that $k = \lfloor 4/3\alpha \rfloor$, let $g = k+1$ or $g = k+2$, whichever is even. Set d_0 to be the first prime power larger than

$$1 + \max\left\{2^{6^2/\alpha^8}, (2c)^{6/\alpha^2}\right\}.$$

By Theorem 15, for every prime power $d \geq d_0$ there exists $n = n(g,d)$ and a girth-g, edge-transitive, d-regular graph graph $G = (V = [n], E)$ (note that by our assumption that $\alpha \leq 4/15$, indeed $g \geq 6$). From here onwards, fix such a d and $n = n(g,d)$, observing that $\{n(g,d)\}_{d \geq d_0}$ is infinite.

Choose α_0 so that $c \log n \cdot n^{\alpha_0} = d/2$ so that marginals $1/2$ correspond to expected sparsity s. Using the fact that $d \geq n^{4/3g} + 1$, it can be verified that $\alpha_0 \geq (1-2\alpha) \cdot \alpha$. Using the fact that $d \geq n^{4/3g} + 1$,

$$\alpha_0 \geq \frac{4}{3g} - \frac{\log(2c \log n)}{\log n} \geq \frac{4}{\frac{4}{\alpha} + 6} - \frac{\log(2c \log n)}{\log n} \geq \left(1 - \frac{3\alpha}{2}\right)\alpha - \frac{\log(2c \log n)}{\log n}.$$

As $n \leq (d-1)^{\frac{3g}{4}}$ and $n \geq 2 \cdot \frac{(d-1)^{\frac{g}{2}-1}}{d-2} \geq (d-1)^{\frac{g}{2}-1}$, the latter being the Moore bound, we have

$$\frac{\log(2c \log n)}{\log n} \leq \frac{\log(2c) + \log \frac{3g}{4} + \log\log(d-1)}{\left(\frac{g}{2} - 1\right)\log(d-1)} \leq \frac{\log(2c)}{\log(d-1)} + 2 \cdot \frac{\log\log(d-1)}{\log(d-1)} \leq \frac{\alpha^2}{2},$$

where we used $\frac{\log(2c)}{\log(d-1)} \leq \frac{\alpha^2}{6}$, $\log \frac{3g}{4} \leq \frac{g}{2} - 1$ and $\frac{\log\log(d-1)}{\log(d-1)} \leq \frac{\alpha^2}{6}$. Thus, overall, $\alpha_0 \geq (1-2\alpha)\alpha$.

We now give a k-wise independent distribution with marginals $1/2$ that fails to yield a good spectral sparsifier for G, namely it will be disconnected with high probability.

To do so, construct $\mathcal{D} \sim \{0,1\}^{|E|}$ as follows. Choose a partition $[n] = V_0 \uplus V_1$ uniformly at random. Each random partition gives rise to an element $D \sim \mathcal{D}$ in which for $e = (u,v) \in E$, $D(e) = 1$ (i.e., the edge e is chosen to survive) if and only if either $u, v \in V_0$ or $u, v \in V_1$.

▷ Claim 18. The distribution $\mathcal{D}$ is k-wise independent with marginals $1/2$.

Proof. As in the proof of Lemma 17, it suffices to show that for every set $T \subseteq E$ of $t \leq k$ edges of G we have $\Pr[T \subseteq \mathcal{D}^{-1}(1)] = 2^{-t}$. First, similar to Lemma 17, note that we can assume without loss of generality that T is a connected component, since whenever T_1 and T_2 are over disjoint sets of vertices, $\Pr[T_1 \cup T_2 \subseteq \mathcal{D}^{-1}(1)] = \Pr[T_1 \subseteq \mathcal{D}^{-1}(1)] \cdot \Pr[T_2 \subseteq \mathcal{D}^{-1}(1)]$. As the girth of G is larger than t, it must be the case that A is a tree.

In such a case, where T contains no cycles, $\Pr[T \in \mathcal{D}]$ is equal to the probability that all $t+1$ vertices in T belong to the same partition, which is $2 \cdot 2^{-(t+1)} = 2^{-t}$. ◁

By the way $\mathcal{D}$ was constructed, it is clear that sampling G' according to $\mathcal{D}$ would result in a disconnected graph unless $V_0 = \varnothing$ or $V_1 = \varnothing$, which occurs with probability $1 - 2/2^n$, meaning that G' almost surely does not ε-approximate G, for *any* ε. ◀

We again stress that by the work in Section 3, we know that *any* k-wise independent distribution over the edges of G with marginals $s \cdot (n-1)/|E|$ for $k = \lceil 2/\alpha \rceil$ would produce an ε-spectral sparsifier with expected degree $O(s)$ with high constant probability.

The above also implies that any improvement upon Moorish families of edge-transitive graphs will improve our lower bound. Assuming the existence of a $(g, \gamma = 2)$-Moorish family of edge-transitive graphs we are able to show that the result of Section 3 is essentially tight.

5 Spectral Sparsifiers in Deterministic Small Space

In this section we show that Sparsify can be derandomized space efficiently.

▶ **Theorem 19** (deterministic small-space sparsification). *Let G be an undirected, connected, weighted graph on n vertices with Laplacian L. There is a deterministic algorithm that, when given G, an even integer k and $0 < \varepsilon < 1$ outputs a weighted graph H with Laplacian $\widetilde{L}$ satisfying:*

1. *$\widetilde{L} \approx_\varepsilon L$, and,*
2. *H has $O\left(\frac{\log n}{\varepsilon^2} n^{1+2/k}\right)$ edges.*

The algorithm runs in space $O(k \log(N \cdot w) + \log(N \cdot w) \log\log(N \cdot w))$, where $w = w_{\max}/w_{\min}$ is the ratio of the maximum and minimum edge weights in G and N is the bitlength of the input.

We use the standard model of space-bounded computation. The machine has a read-only input tape, a constant number of read/write work tapes, and a write-only output tape. We say the machine runs in space s if throughout the computation, it only uses s total tape cells on the work tapes. The machine may write outputs to the output tape that are larger than s (in fact as large as $2^{O(s)}$) but the output tape is write-only. We use the following fact about the composition of space-bounded algorithms.

▶ **Lemma 20.** *Let f_1 and f_2 be functions that can be computed in space $s_1(n), s_2(n) \geq \log n$, respectively, and f_2 has output of length $\ell_1(n)$ on inputs of size n. Then $f_2 \circ f_1$ can be computed in space*

$$O(s_2(\ell_1(n)) + s_1(n)).$$

The natural way to derandomize Sparsify would be to iterate over all elements of the corresponding k-wise independent sample space. More formally, given $\{p_{ab}\}_{(a,b)\in E}$, let I_{ab} be the indicator random variable that is 1 if and only if edge (a, b) is chosen. If the I_{ab}'s are k-wise independent so that $\Pr[I_{ab} = 1] = p_{ab}$ (or some good approximation of p_{ab}), we are guaranteed to succeed with nonzero probability. Hence, at least one assignment to the I_{ab}'s taken from the k-wise independent is guaranteed to work. From Section 2.4 we know the sample space is small enough that we can afford to enumerate over all elements in it. Towards proving Theorem 19, there are still three issues to consider:

1. Approximating the effective resistances R_{ab} for every $(a, b) \in E$, space efficiently. Fortunately, we can do this with high accuracy using the result of Murtagh, Reingold, Sidford, and Vadhan [41] for approximating the pseudoinverse of a Laplacian, which we state shortly.

2. Verifying that a given set of random choices in Sparsify provides a sparse and accurate approximation to the input graph. The sparsity requirement is easy to check. To check that $\widetilde{L} \approx_\varepsilon L$, we devise a verification algorithm that uses the algorithm of [41]. The details are given in Lemma 25.
3. The Laplacian solver of [41] only works for multigraphs (graphs with integer edge weights) and we want an algorithm that works for general weighted graphs. To fix this, we extend the work of [41] by giving a simple reduction from the weighted case to the multigraph case. The details can be found in the full version of the paper.

5.1 Algorithm for Approximating Effective Resistances

A key ingredient in our deterministic sparsification algorithm is a deterministic nearly logarithmic space algorithm for approximating the pseudoinverse of an undirected Laplacian.

▶ **Theorem 21** ([41]). *Given an undirected, connected multigraph G with Laplacian $L = D - A$ and $\varepsilon > 0$, there is a deterministic algorithm that computes a symmetric PSD matrix $\widetilde{L^+}$ such that $\widetilde{L^+} \approx_\varepsilon L^+$, and uses space $O(\log N \cdot \log\log \frac{N}{\varepsilon})$, where N is the bitlength of the input (as a list of edges).*

Note that the space complexity above assumes that the multigraph is given as a list of edges. If we instead think of parallel edges as integer edge weights, then N should be replaced by $N \cdot w_{\max}$, where $w_{\max}$ is the maximum edge weight in G since an edge of weight w gets repeated w times in the edge-list representation. To work with general weighted graphs, we extend the result of [41].

▶ **Lemma 22** (small space laplacian solver for weighted graphs). *Given an undirected connected weighted graph $G = (V, E, w)$ with Laplacian $L = D - A$, and $0 < \varepsilon < 1$, there exists a deterministic algorithm that computes a symmetric PSD matrix $\widetilde{L^+}$ such that $\widetilde{L^+} \approx_\varepsilon L^+$, and uses space $O(\log(N \cdot w) \log\log(N \cdot w/\varepsilon))$, where $w = w_{\max}/w_{\min}$ is the ratio of the maximum and minimum edge weights in G and N is the bitlength of the input.*

A proof of Lemma 22 can be found in the full version. Lemma 22 immediately gives an algorithm for computing strong multiplicative approximations to effective resistances.

▶ **Lemma 23.** *Let $G = (V, E, w)$ be an undirected, connected, weighted graph and let R_{ab} be the effective resistance of $(a, b) \in E$. There is an algorithm that computes a real number $\widetilde{R}_{ab}$ such that*

$$(1-\varepsilon) \cdot R_{ab} \leq \widetilde{R}_{ab} \leq (1+\varepsilon) \cdot R_{ab}$$

and uses space $O(\log(N \cdot w) \cdot \log\log \frac{N \cdot w}{\varepsilon})$, where $w = w_{\max}/w_{\min}$ is the ratio of the maximum and minimum edge weights in G and N is the bitlength of the input.

See the full version for a proof of Lemma 23.

Next we show how we test whether two matrices spectrally approximate each other. We will need the following claim about the space complexity of matrix multiplication.

▷ Claim 24. Given $n \times n$ matrices $M_1, \ldots, M_k$, their product $M_1 \cdot \ldots \cdot M_k$ can be computed using $O(\log N \cdot \log k)$ space, where N is the bitlength of $(M_1, \ldots, M_k)$.

The proof of Claim 24 uses the natural divide and conquer algorithm and the fact that two matrices can be multiplied in logarithmic space. A detailed proof can be found in [41].

5.2 Testing for Spectral Proximity

In this section we give our deterministic, small-space procedure for verifying that two Laplacians spectrally approximate one another.

▶ **Lemma 25.** *There exists a deterministic algorithm that, given undirected, connected, weighted graphs $\widetilde{G}$ and G with Laplacians $\widetilde{L}, L$, and $\varepsilon, \alpha > 0$, outputs* `YES` *or* `NO` *such that*

1. *If $\widetilde{L} \approx_\varepsilon L$, then the algorithm outputs* `YES`*, and,*
2. *If $\widetilde{L} \not\approx_{\varepsilon \cdot \sqrt{1+\alpha}} L$ then the algorithm outputs* `NO`*.*

The algorithm uses space $O(\log(N \cdot w) \cdot \log\log \frac{N \cdot w}{\alpha\varepsilon} + \log(N \cdot w) \cdot \log \frac{1}{\alpha})$, where $w = w_{\max}/w_{\min}$ is the ratio of the maximum and minimum edge weights in G and $\widetilde{G}$ and N is the bitlength of the input.

The high level idea for the proof is that testing whether two matrices L and $\widetilde{L}$ spectrally approximate each other can be reduced to approximating the spectral radius of a particular matrix

$$M = \left(\frac{(\widetilde{L} - L)L^+}{\varepsilon} \right)^2 .$$

In fact, it will be sufficient to check whether the trace of a sufficiently high power of M is below a certain threshold to deduce whether the spectral radius of M does not exceed 1. For intuition, replace the matrices with scalars m, ℓ, and $\widetilde{\ell}$ where

$$m = \frac{(\widetilde{\ell} - \ell)^2}{(\varepsilon \cdot \ell)^2}.$$

Then, $m \leq 1$ implies $\sqrt{m} \leq 1$, which implies $|\widetilde{\ell} - \ell| \leq \varepsilon \cdot \ell$ – the kind of relative closeness we want between the matrices $\widetilde{L}$ and L when aiming for spectral approximation. The proof of Lemma 25 can be found in the full version.

5.3 Completing the Proof of Theorem 19

We can now prove the main result of this section. As noted above, the algorithm proceeds by first approximating the sampling probabilities and then sparsifying G where the surviving edges are chosen from a small k-wise independent sample space whose marginals are set properly. Each potential sparsifier is checked using the algorithm given in Section 5.2.

Proof of Theorem 19. Set $\delta = \frac{1}{4}$, $\widehat{\varepsilon} = \frac{4\varepsilon}{5}$ and

$$s = \frac{18e \log n}{\widehat{\varepsilon}^2} \cdot \left(\frac{n}{\delta}\right)^{2/k},$$

for α soon to be determined. These parameters are chosen in accordance with the parameters required for Sparsify to succeed with probability 1/2 and approximation error $\widehat{\varepsilon}$ (see Lemma 10). Set $\alpha' = \alpha/(4+\alpha)$. We compute approximate effective resistances $\widetilde{R}_{ab}$ for each edge (a, b) in G using Lemma 23, so that

$$(1 - \alpha')R_{ab} \leq \widetilde{R}_{ab} \leq (1 + \alpha')R_{ab}.$$

This takes $O(\log(N \cdot w) \log\log((N \cdot w)/\alpha))$ space. Then, we compute approximate sampling probabilities as follows:

$$\widetilde{p}_{ab} = \alpha' \cdot \left\lfloor \frac{1}{\alpha'} \cdot \min\left\{1, w_{ab} \cdot \widetilde{R}_{ab} \cdot s/(1 - \alpha')\right\} \right\rfloor$$

That is, we truncate the required (approximate) sampling probabilities to $\log \frac{1}{\alpha'}$ bits of precision. In particular, denoting the precise sampling probabilities by $p^\star_{ab} = \min\{1, w_{ab} \cdot R_{ab} \cdot s\}$, we have

$$\begin{aligned}\min\{1, w_{ab} \cdot \widetilde{R}_{ab} \cdot s/(1-\alpha')\} - p^*_{ab} &\leq w_{ab} \cdot s \cdot R_{ab} \cdot \left(\frac{1+\alpha'}{1-\alpha'} - 1\right) \\ &= p^*_{ab} \cdot \frac{2\alpha'}{1-\alpha'} \leq \alpha/2\end{aligned}$$

Furthermore, we have an additional error of α' due to the truncation so $|\widetilde{p}_{ab} - p^\star_{ab}| \leq \alpha/2 + \alpha' \leq \alpha$.

We want to set α so that $\widetilde{p}_{ab}$ is a multiplicative approximation to p^*_{ab} for all $(a,b) \in E$, which requires α to be smaller than $\min_{(a,b)\in E)}\{p^*_{ab}\}$.

▷ Claim 26. Let $d_{\max}$ be the maximum weighted degree over all vertices in G. Then, for all $(a,b) \in E$, $p^\star_{ab} \geq 1/d_{\max}$.

Proof. Since $s > 1$ and $w_{ab} \geq 1$ (all edge weights are positive integers) we have $p^\star_{ab} \geq R_{ab}$. Let $\lambda_{\min}(C)$ denote the minimal nonzero eigenvalue of a matrix C. To lower bound R_{ab}, we use the variational characterization of eigenvalues and the definition of effective resistance to write

$$\begin{aligned}R_{ab} &= (e_a - e_b)^\top L^+ (e_a - e_b) \geq \lambda_{\min}(L^+) \cdot \|e_a - e_b\|^2 \\ &= \frac{2}{\|L\|} \geq \frac{1}{d_{\max}}.\end{aligned}$$

Note that we can indeed consider the minimal nonzero eigenvalue of L^+ because $e_a - e_b$ is perpendicular to the one-dimensional kernel of L (the all-ones vector). ◁

In light of the above, we can set α so that $1/\alpha = 2 \cdot d_{\max} = O(N \cdot w)$ and get a 1/2-multiplicative approximation to the sampling probabilities.

Now, consider the k-wise independent sample space $\mathcal{D} \subseteq \{0,1\}^{|E|}$ guaranteed to us by Lemma 8, substituting $t = \lceil \log(1/\alpha') \rceil$. By Lemma 8, each element of $\mathcal{D}$ can be sampled using

$$O(k \cdot \max\{\log(1/\alpha'), \log|E|\}) = O(k \cdot \log(N \cdot w))$$

space. For each element of $\mathcal{D}$, construct the corresponding sparse graph. Note that the space used to cycle through each element can be reused. Lemma 10 tells us that at least $1 - 2\delta = 1/2$ of the Laplacians of the resulting graphs $\widehat{\varepsilon}$-approximate the Laplacian of G and have

$$O\left(\frac{1+1/2}{1-1/2} \cdot \frac{1}{\delta^{1+2/k}} \cdot \frac{\log n}{\widehat{\varepsilon}^2} \cdot n^{1+\frac{2}{k}}\right) = O\left(\frac{\log n}{\varepsilon^2} \cdot n^{1+\frac{2}{k}}\right)$$

edges. For each of these graphs, we run the verification algorithm with accuracy parameter 9/16, which is guaranteed to find a graph with the above sparsity whose Laplacian approximates the Laplacian of G with error

$$\widehat{\varepsilon} \cdot \sqrt{1 + \frac{9}{16}} = \frac{4\varepsilon}{5} \cdot \sqrt{\frac{25}{16}} = \varepsilon$$

in space

$$O\left(\log(N \cdot w) \log\log \frac{16N \cdot w}{9\widehat{\varepsilon}} + \log(N \cdot w) \log \frac{16}{9}\right) = O\left(\log(N \cdot w) \log\log \frac{N \cdot w}{\varepsilon}\right).$$

Again, the space used for the verification process can be reused. Adding up the space complexities gives us a total of

$$O\left(k\log(N\cdot w)+\log(N\cdot w)\log\log\frac{N\cdot w}{\varepsilon}\right)$$

space. Note that the final result is vacuous when $\varepsilon \leq 1/n$ so we can without loss of generality assume that $\varepsilon \geq 1/n$. This gives a total space complexity of $O(k\log(N\cdot w)+\log(N\cdot w)\log\log(N\cdot w))$. ◀

References

1 AmirMahdi Ahmadinejad, Jonathan Kelner, Jack Murtagh, John Peebles, Aaron Sidford, and Salil Vadhan. High-precision estimation of random walks in small space. *arXiv preprint*, 2019. arXiv:1912.04524.

2 Zeyuan Allen-Zhu, Zhenyu Liao, and Lorenzo Orecchia. Spectral sparsification and regret minimization beyond matrix multiplicative updates. In *Proceedings of the 47th Annual ACM Symposium on Theory of Computing (STOC 2015)*, pages 237–245. ACM, 2015.

3 Noga Alon, László Babai, and Alon Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *Journal of algorithms*, 7(4):567–583, 1986.

4 Noga Alon and Asaf Nussboim. k-wise independent random graphs. In *Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2008)*, pages 813–822. IEEE, 2008.

5 Joshua Batson, Daniel A. Spielman, and Nikhil Srivastava. Twice-ramanujan sparsifiers. *SIAM Review*, 56(2):315–334, 2014.

6 András A. Benczúr and David R. Karger. Approximating s-t minimum cuts in $\widetilde{O}(n^2)$ time. In *STOC*, volume 96, pages 47–55. Citeseer, 1996.

7 Norman L. Biggs and Alan G. Boshier. Note on the girth of Ramanujan graphs. *Journal of Combinatorial Theory, Series B*, 49(2):190–194, 1990.

8 Béla Bollobás. *Modern graph theory*, volume 184. Springer Science & Business Media, 2013.

9 Richard Y. Chen, Alex Gittens, and Joel A. Tropp. The masked sample covariance estimator: An analysis using matrix concentration inequalities. *Information and Inference: A Journal of the IMA*, 1(1):2–20, 2012.

10 L. Paul Chew. There are planar graphs almost as good as the complete graph. *Journal of Computer and System Sciences*, 39(2):205–219, 1989.

11 Paul Christiano, Jonathan A. Kelner, Aleksander Madry, Daniel A. Spielman, and Shang-Hua Teng. Electrical flows, Laplacian systems, and faster approximation of maximum flow in undirected graphs. In *Proceedings of the 43rd Annual ACM Symposium on Theory of Computing (STOC 2011)*, pages 273–282. ACM, 2011.

12 Timothy Chu, Yu Gao, Richard Peng, Sushant Sachdeva, Saurabh Sawlani, and Junxing Wang. Graph sparsification, spectral sketches, and faster resistance computation, via short cycle decompositions. In *Proceedings of the 59th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2018)*, pages 361–372. IEEE, 2018.

13 Michael B. Cohen, Jonathan Kelner, Rasmus Kyng, John Peebles, Richard Peng, Anup B. Rao, and Aaron Sidford. Solving directed Laplacian systems in nearly-linear time through sparse LU factorizations. In *Proceedings of the 59th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2018)*, pages 898–909. IEEE, 2018.

14 Michael B. Cohen, Jonathan Kelner, John Peebles, Richard Peng, Anup B. Rao, Aaron Sidford, and Adrian Vladu. Almost-linear-time algorithms for Markov chains and new spectral primitives for directed graphs. In *Proceedings of the 49th Annual ACM Symposium on Theory of Computing (STOC 2017)*, pages 410–419. ACM, 2017.

15 Michael B. Cohen, Jonathan Kelner, John Peebles, Richard Peng, Aaron Sidford, and Adrian Vladu. Faster algorithms for computing the stationary distribution, simulating random walks, and more. In *Proceedings of the 57th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2016)*, pages 583–592. IEEE, 2016.

16 Michael B. Cohen, Rasmus Kyng, Gary L. Miller, Jakub W. Pachocki, Richard Peng, Anup B. Rao, and Shen Chen Xu. Solving SDD linear systems in nearly $m \log^{1/2} n$ time. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing (STOC 2014)*, pages 343–352. ACM, 2014.

17 Michael B Cohen, Aleksander Madry, Piotr Sankowski, and Adrian Vladu. Negative-weight shortest paths and unit capacity minimum cost flow in $\widetilde{O}(m^{10/7} \log W)$ time. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2017)*, pages 752–771. SIAM, 2017.

18 Dean Doron, François Le Gall, and Amnon Ta-Shma. Probabilistic logarithmic-space algorithms for Laplacian solvers. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2017)*, pages 41:1–41:20. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

19 Dean Doron, Jack Murtagh, Salil Vadhan, and David Zuckerman. Spectral sparsification via bounded-independence sampling. *Electronic Colloquium on Computational Complexity (ECCC)*, 2020.

20 Paul Erdős and Horst Sachs. Reguläre graphen gegebener taillenweite mit minimaler knotenzahl. *Wiss. Z. Martin-Luther-Univ. Halle-Wittenberg Math.-Natur. Reihe*, 12(251-257):22, 1963.

21 Geoffrey Exoo and Robert Jajcay. Dynamic cage survey. *Electron. J. Combin*, 15(16):4, 2008.

22 Ronald M. Foster. The average impedance of an electrical network. *Contributions to Applied Mechanics (Reissner Anniversary Volume)*, pages 333–340, 1949.

23 A. Joffe. On a set of almost deterministic k-independent random variables. *The Annals of Probability*, 2(1):161–162, 1974.

24 Jonathan A. Kelner, Yin Tat Lee, Lorenzo Orecchia, and Aaron Sidford. An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2014)*, pages 217–226. SIAM, 2014.

25 Jonathan A. Kelner and Aleksander Madry. Faster generation of random spanning trees. In *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2019)*, pages 13–21. IEEE, 2009.

26 Jonathan A. Kelner, Lorenzo Orecchia, Aaron Sidford, and Zeyuan Allen Zhu. A simple, combinatorial algorithm for solving SDD systems in nearly-linear time. In *Proceedings of the 45th Annual ACM Symposium on Theory of Computing (STOC 2013)*, pages 911–920. ACM, 2013.

27 Ioannis Koutis, Gary L. Miller, and Richard Peng. A nearly-$m \log n$ time solver for SDD linear systems. In *Proceedings of the 52nd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2011)*, pages 590–598. IEEE, 2011.

28 Ioannis Koutis, Gary L. Miller, and Richard Peng. Approaching optimality for solving SDD linear systems. *SIAM Journal on Computing*, 43(1):337–354, 2014.

29 Ioannis Koutis, Gary L. Miller, and David Tolliver. Combinatorial preconditioners and multilevel solvers for problems in computer vision and image processing. In *International Symposium on Visual Computing*, pages 1067–1078. Springer, 2009.

30 Rasmus Kyng, Yin Tat Lee, Richard Peng, Sushant Sachdeva, and Daniel A. Spielman. Sparsified Cholesky and multigrid solvers for connection Laplacians. In *Proceedings of the 48th Annual ACM Symposium on Theory of Computing (STOC 2016)*, pages 842–850. ACM, 2016.

31 Rasmus Kyng and Sushant Sachdeva. Approximate Gaussian elimination for Laplacians-fast, sparse, and simple. In *Proceedings of the 57th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2016)*, pages 573–582. IEEE, 2016.

32 Felix Lazebnik, Vasiliy A. Ustimenko, and Andrew J. Woldar. A new series of dense graphs of high girth. *Bulletin of the American mathematical society*, 32(1):73–79, 1995.

33 Yin Tat Lee, Satish Rao, and Nikhil Srivastava. A new approach to computing maximum flows using electrical flows. In *Proceedings of the 45th Annual ACM Symposium on Theory of Computing (STOC 2013)*, pages 755–764. ACM, 2013.

34 Yin Tat Lee and Aaron Sidford. Efficient accelerated coordinate descent methods and faster algorithms for solving linear systems. In *Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2013)*, pages 147–156. IEEE, 2013.

35 Yin Tat Lee and He Sun. Constructing linear-sized spectral sparsification in almost-linear time. In *Proceedings of the 56th IEEE Annual Symposium on Foundations of Computer Science (FOCS 2015)*, pages 250–269. IEEE, 2015.

36 Yin Tat Lee and He Sun. An SDP-based algorithm for linear-sized spectral sparsification. In *Proceedings of the 49th Annual ACM Symposium on Theory of Computing (STOC 2017)*, pages 678–687. ACM, 2017.

37 Yang P. Liu and Aaron Sidford. Faster energy maximization for faster maximum flow. *arXiv preprint*, 2019. `arXiv:1910.14276`.

38 Alexander Lubotzky, Ralph Phillips, and Peter Sarnak. Ramanujan graphs. *Combinatorica*, 8(3):261–277, 1988.

39 Michael Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM Journal on Computing*, 15(4):1036–1053, 1986.

40 Aleksander Madry, Damian Straszak, and Jakub Tarnawski. Fast generation of random spanning trees and the effective resistance metric. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2014)*, pages 2019–2036. SIAM, 2014.

41 Jack Murtagh, Omer Reingold, Aaron Sidford, and Salil P. Vadhan. Derandomization beyond connectivity: Undirected Laplacian systems in nearly logarithmic space. In *Proceedings of the 58th IEEE Annual Symposium on Foundations of Computer Science (FOCS 2017)*, pages 801–812. IEEE, 2017.

42 Jack Murtagh, Omer Reingold, Aaron Sidford, and Salil P. Vadhan. Deterministic approximation of random walks in small space. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2019)*, pages 42:1–42:22. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

43 Lorenzo Orecchia, Sushant Sachdeva, and Nisheeth K. Vishnoi. Approximating the exponential, the Lanczos method and an $\widetilde{O}(m)$-time spectral algorithm for balanced separator. In *Proceedings of the 44th Annual ACM Symposium on Theory of Computing (STOC 2012)*, pages 1141–1160. ACM, 2012.

44 Richard Peng and Daniel A. Spielman. An efficient parallel solver for SDD linear systems. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing (STOC 2014)*, pages 333–342. ACM, 2014.

45 Robert Qiu and Michael Wicks. *Cognitive networked sensing and big data*. Springer, 2014.

46 Omer Reingold. Undirected connectivity in log-space. *J. ACM*, 55(4), 2008.

47 Eyal Rozenman and Salil Vadhan. Derandomized squaring of graphs. In *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*, pages 436–447. Springer, 2005.

48 Mark Rudelson and Roman Vershynin. Sampling from large matrices: An approach through geometric functional analysis. *Journal of the ACM (JACM)*, 54(4):21, 2007.

49 Aaron Schild. An almost-linear time algorithm for uniform random spanning tree generation. In *Proceedings of the 50th Annual ACM Symposium on Theory of Computing (STOC 2018)*, pages 214–227. ACM, 2018.

50 Daniel A. Spielman. Yale Applied Mathematics 561/Computer Science 662, Lecture Notes: Spectral Graph Theory, November 2015. URL: `http://www.cs.yale.edu/homes/spielman/561/lect17-15.pdf`.

51 Daniel A. Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. *SIAM Journal on Computing*, 40(6):1913–1926, 2011.

52 Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC 2004)*, pages 81–90, 2004.

53 Anastasios Zouzias. A matrix hyperbolic cosine algorithm and applications. In *International Colloquium on Automata, Languages, and Programming (ICALP 2012)*, pages 846–858. Springer, 2012.

Hard Problems on Random Graphs

Jan Dreier
Department of Computer Science, RWTH Aachen University, Germany
https://tcs.rwth-aachen.de/~dreier
dreier@cs.rwth-aachen.de

Henri Lotze
Department of Computer Science, RWTH Aachen University, Germany
https://tcs.rwth-aachen.de
lotze@cs.rwth-aachen.de

Peter Rossmanith
Department of Computer Science, RWTH Aachen University, Germany
https://tcs.rwth-aachen.de
rossmani@cs.rwth-aachen.de

Abstract

Many graph properties are expressible in first order logic. Whether a graph contains a clique or a dominating set of size k are two examples. For the solution size as its parameter the first one is W[1]-complete and the second one W[2]-complete meaning that both of them are hard problems in the worst-case. If we look at both problem from the aspect of average-case complexity, the picture changes. Clique can be solved in expected FPT time on uniformly distributed graphs of size n, while this is not clear for Dominating Set. We show that it is indeed unlikely that Dominating Set can be solved efficiently on random graphs: If yes, then every first-order expressible graph property can be solved in expected FPT time, too. Furthermore, this remains true when we consider random graphs with an arbitrary constant edge probability. We identify a very simple problem on random matrices that is equally hard to solve on average: Given a square boolean matrix, are there k rows whose logical AND is the zero vector? The related Even Set problem on the other hand turns out to be efficiently solvable on random instances, while it is known to be hard in the worst-case.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Complexity theory and logic

Keywords and phrases random graphs, average-case complexity, first-order model checking

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.40

Category Track A: Algorithms, Complexity and Games

Funding Grant DFG-927/15-1 from the German Science Foundation (DFG).

Acknowledgements We thank the refeeres of ICALP for their very helpful comments that helped to improve the readability of the paper.

1 Introduction

The worst-case analysis of problems has a long tradition and has led to a complexity theory that allows us easily to classify many problems. Such complexity theories do not only exist for the time complexity, but also for other resources such as space. There are complexity classes for approximations, parallel computations, randomization, parameterized algorithms, and many more. Usually they come with complete problems under certain reductions.

The average-case analysis of problems is less developed, but Levin showed quite early that there exist problems that are hard to solve even on random inputs [19]. He considered problems together with input distributions and defined reductions that take the probability distribution into account. This also led to complete problems for an analogue of NP in the average-case

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 40; pp. 40:1–40:14
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

world. This theory has been constantly refined. For example, Gurevich [17] showed some inherent limitations of these techniques. Ben-David et al. showed a rare connection to the worst-case world: If all problems in NP with a "simple" probability distribution on the inputs can be solved in average polynomial time, then NEXPTIME = EXPTIME [1]. Such connections are extremely rare and the latter one together with all others suffer from a severe problem: They rely on quite unnatural probability distributions. Up to today no connection seems to exist that shows, e.g., that if some natural problem in NP is hard to solve on average under a uniform distribution, then P $\neq$ NP. Such a result would be a big breakthrough.

It has to be noted, however, that it is in general not easy even to find problems that are hard on average (with a uniform distribution). Having such problems is crucial in cryptography. The RSA system is based on the assumption that factoring the product of two primes is hard on average, but we cannot prove today that the existence of an algorithm that can factor in polynomial time on average would imply some unexpected collapse between worst-case complexity classes.

Many problems that are hard in the worst-case become easy on average. Take for example the three-coloring problem on graphs. While NP-complete this problem can be solved in constant time on average when drawing the graph from a uniform distribution of all graphs of size n: It is easy to see that you can find a triangle in expected constant time by just looking for one among the first three vertices, then the next three and so on. Each time you find a triangle with probability $\frac{1}{8}$ and you find a triangle on average with only eight tries. See for example [4] for a similar but more complicated example.

The same holds for finding any fixed size subgraph or induced subgraph. This means that p-CLIQUE, the problem of finding a clique of size k, can be solved in expected $f(k)\mathit{poly}(n)$ time for some function f if the input is a uniformly distributed graph of size n. Parameterized complexity shows that it is unlikely to solve the same problem in $f(k)\mathit{poly}(n)$ time in the worst-case [9]. Fountoulakis, Friedrich, and Hermelin showed that finding cliques is in FPT if the probability in the random graph is an almost arbitrary function of its size [14].

In this paper we look at first-order model checking on uniformly distributed random graphs and more generally on Erdős–Rényi graphs with a constant edge probability. In this model we assume that each possible edge in a graph with n vertices exists independently with a probability of p. While in the worst-case the FO model checking problem seems to become harder the more quantifier alternations we have, this hierarchy collapses when looking at the average time complexity. We will show that the dominating set problem is as hard as the whole model checking problem. We also identify a very natural problem on boolean matrices that has the same complexity: Does a random boolean matrix have k rows whose logical AND is the zero vector? We conclude that this matrix problem and the dominating set problem are hard on *average* (unless the very general model checking problem is easy, which would be unexpected). Finally we consider also the Even Set problem, which has been finally shown to be W[1]-hard in the worst-case [2] and is similar to the above mentioned matrix problem. Nevertheless it turns out that Even Set can be efficiently solved on random instances.

Among the techniques "half"-reductions play an important role. While in a reduction f from A to B you require that $w \in A$ iff $f(w) \in B$, we often need only one direction: Showing "if $w \in A$ then $f(w) \in B$" is sufficient if $f(w) \in B$ holds with a very small probability. Having an algorithm for B we can then solve A as follows. Compute $f(w)$ and find out whether $f(w) \in B$ holds. If not, conclude that $w \notin A$. If yes, then solve $w \in A$ with a very slow, but simple algorithm. As this happens with a small probability it does not spoil the expected running time.

2 Preliminaries

Parameterized complexity

Parameterized complexity was introduced by Downey and Fellows in a series of papers to investigate further what makes problems hard to solve (see, e.g., [5, 6, 7, 8, 9, 13, 22]). Instead of measuring the run time solely as a function on the input length, it may also depend on other parameters of the input. A parameterized problem has therefore a parameter k and the input length n and we classify a problem as *fixed parameter tractable* if it can be solved in time $f(k)\mathit{poly}(n)$ for some computable function f. If an NP-hard problem is fixed parameter tractable, then it runs in polynomial time for every fixed value of k and the degree of the polynomial does not depend on k. In particular this means that there exist efficient algorithms for scenarios where the parameter is small.

In this paper we will look at distributional problems on random graphs and boolean matrices. Here a distributional problem will be a parameterized problem together with a probability distribution of the inputs. Usually we will denote such a distributional problem by stating the problem and the probability distribution separately.

We use the notation of Flum and Grohe [13] for parameterized problems. The first two important problems we consider are the dominating set problem on undirected graphs and a simple problem on boolean matrices:

▶ **Definition 1.**

p-Dominating Set

Input:	A graph G and $k \in \mathbf{N}$.
Parameter:	k
Problem:	Is there a dominating set of size $\leq k$ for G?

p-Matrix($\wedge$)

Input:	A boolean matrix $M \in \{0,1\}^{n \times n}$ and $k \in \mathbf{N}$.
Parameter:	k
Problem:	Are there k rows in M whose logical AND is the zero vector?

Logic on graphs and the zero-one law

We use graphs as a structure (V, E) where V is the vertex set and E the binary edge relation. Instead of Euv we will write $u \sim v$, which expresses that there is an edge between u and v in an undirected graph. First-order (FO) formulas on graphs are atomic formulas of the form $x = y$ or $x \sim y$ or one of the following: $\phi \wedge \psi$, $\phi \vee \psi$, $\neg\phi$, $\forall x\phi$, $\exists x\phi$, where ϕ and ψ are already FO-formulas. The semantics are as expected. A sentence is a formula without free variables.

▶ **Definition 2.** We define the first-order model checking problems on graphs. The more general problem on relational structures can be reduced to this more special problem [13].

p-MC(FO)

Input:	A first-order sentence ϕ and a graph G
Parameter:	$\lvert\phi\rvert$, the length of ϕ
Problem:	Does ϕ hold in G, i.e., $G \models \phi$?

For example, the formula $\exists x_1 \exists x_2 \ldots \exists x_k \forall y \bigvee_i (x_i = y \vee x_i \sim y)$ expresses that a graph has a dominating set of size at most k. If a formula ϕ holds for a graph G we write $G \models \phi$. If a formula ϕ follows from a set of formulas Φ we write $\Phi \models \phi$. This is the case iff there is a formal derivation of ϕ from Φ, which we write as $\Phi \vdash \phi$. Sometimes we will use colored graphs, which we represent by a graph G and a coloring function χ mapping vertices to a set of colors. Formulas can speak about colors via atomic formulas of the form $\chi(x) = \mathit{red}$ and we write $(G, \chi) \models \phi$ if the formula ϕ is true for G with colors χ.

By $G(n,p)$ we denote an Erdős–Rényi-graph with n vertices and edge probability p, where edges exist independently from each other with a probability of exactly $p \in [0,1]$. Fagin [11] proved the zero-one law for first-order sentences, which states that for every sentence ϕ either $\lim_{n\to\infty} \Pr[G(n,1/2) \models \phi] = 0$ or $\lim_{n\to\infty} \Pr[G(n,1/2) \models \phi] = 1$. With other words, a graph property that is expressible in first-order logic either holds asymptotically almost surely or almost never. Given ϕ as an input, it can be decided whether the limit is 0 or 1 and Grandjean showed that it turns out to be a PSPACE-complete problem [15]. An important role in the proof of the zero-one law play the so-called *extension axioms* (not to be confused with the axiom of extension in Zermelo–Fraenkel set theory). They state that every constant-size set of vertices is connected in every possible way to other vertices. For a set or vector of variables $x_1, \ldots, x_k$ we will often write $\bar{x}$. With this notation an extension axiom can be written as

$$\forall \bar{x} \forall \bar{y} \exists z \Big(\bigwedge_{i,j} x_i \neq y_j \to \bigwedge_i (x_i \sim z \wedge y_i \not\sim z) \Big).$$

For an extension axiom it is easy to see that it holds almost surely, but if we look at the whole set Φ of all extension axioms it turns out that there is only one countable model up to isomorphisms, the so-called *Rado graph*, which contains every finite and countable infinite graph as an induced subgraph. Hence by the Łoś–Vaught Theorem [20, 23], $\Phi \models \phi$ or $\Phi \models \neg\phi$ for every first-order sentence ϕ. This means also that either $\Phi \vdash \phi$ or $\Phi \vdash \neg\phi$. To find out out which one is true we can just enumerate all proofs. Note that in a proof only a finite number of formulas from Φ are used and the proof itself is of course also finite. The result by Grandjean states that this can be done in polynomial space.

All these observations suggest that FO-model checking should be easy on random graphs: Just find out from ϕ alone whether it holds almost surely or almost never. Then verify that this is indeed the case for the given G. The strategy of using an abundance of witnesses suggests itself, just as the triangle finding described in the introduction. While this intuition is correct for purely existential formulas, life becomes much harder when considering formulas with quantifier alternations.

In worst-case complexity FO-model checking with a fixed number of quantifier alternations form the complete problems for the A-hierarchy [12]. Among known results about the relationship to other complexity classes are W[t] $\subseteq$ A[t] and W[1] = A[1]. To today's knowledge this hierarchy appears to be proper, see e.g. [3]. A collapse of the A-hierarchy implies a collapse of both the W-hierarchy and of the polynomial hierarchy.

In this paper we investigate how hard FO-model checking is on average, which could be interpreted as looking at the average-case analogue to the A-hierarchy. In the worst-case model checking purely existential formulas is already W[1] = A[1]-complete, while it is easy to see that you can achieve expected FPT time (because of abundance of witnesses). If we turn to edge probabilities apart from $\frac{1}{2}$ and look at Erdős–Rényi graphs $G(n,p)$ with a constant p the problem stays in expected FPT time. Grohe showed that for sparse Erdős–Rényi graphs $G(n, d(n)/n)$ with $d(n) = n^{o(1)}$ the whole p-MC(FO) can be solved in expected FPT time [16]. The latter result also holds for graphs with vertex colors and it turns out that for colored random graphs Grohe's result is optimal with regard to the edge density: For no $\epsilon > 0$ and $G(n, n^\epsilon/n)$ is it possible to solve colored-p-MC(FO) in expected FPT time unless AW[$*$] $\subseteq$ FPT/poly [10].

Universal sets, bisectors, and colorings

▶ **Definition 3.** Let $n \in \mathbf{N}$ and $k \in \mathbf{N}$ with $n \geq 2k$.

1. A family $\mathcal{U}$ of functions $[n] \mapsto \{0, 1\}$ is called an (n, k)-*universal set* if for every subset $M \subseteq [n]$ of size $|M| = k$ and every $M' \subseteq M$ there is a function $f \in \mathcal{U}$ such that $f(t) = 0$ for every $t \in M'$ and $f(t) = 1$ for every $t \in M - M'$.
2. A family $\mathcal{B}$ of functions $[n] \mapsto \{0, 1\}$ is called a k-*universal bisector family* if for every subset $M \subseteq [n]$ of size $|M| = k$ there is a function $f \in \mathcal{B}$ such that $f(t) = 0$ for every $t \in M$ and every function f bisects $[n]$ in two sets of almost the same size: $|f^{-1}(0)| = \lceil n/2 \rceil$.
3. A family $\mathcal{C}$ of functions $[n] \mapsto \{\mathit{black}, \mathit{white}, \mathit{gray}\}$ is called a k-*universal coloring* if for every subset $M \subseteq [n]$ of size $|M| = k$ and every $M' \subseteq M$ there is a function $f \in \mathcal{C}$ such that $f(t) = \mathit{black}$ for every $t \in M'$ and $f(t) = \mathit{white}$ for every $t \in M - M'$. Moreover, $|f^{-1}|(\mathit{gray}) = \lceil n/2 \rceil$ for every $f \in \mathcal{C}$.

(n, k)-universal sets are a well-known concept that has been used, e.g., in the derandomization of algorithms. Naor, Schulman, and Srinivasan designed such a family that can be constructed in linear time and has size $2^k k^{O(\log k)} \log n$ [21].

The concept of a universal bisector is somehow orthogonal to (n, k)-universal sets. Combining both concepts leads to (n, k)-universal colorings that will always color half of the nodes gray.

A simple idea to build a k-universal bisector family of small size is this: For every size k subset M of $[2k]$ we define the function

$$b_M \colon [n] \to \{0, 1\},\ b_M(t) = \begin{cases} 0 & \text{if } t \bmod 2k \in M \\ 1 & \text{otherwise,} \end{cases} \tag{1}$$

where $a \bmod b$ is the remainder when dividing a by b.

Assume that $S \subseteq [n]$ with $|S| \leq k$. If we choose M such that it contains $t \bmod 2k$ for every $t \in S$ (and some more elements), then clearly $b_M(i) = 0$ for every $i \in S$. Hence, we have a k-universal bisector family. If n is a multiple of $2k$ then $|b_M^{-1}(0)| = n/2$ because every group of $2k$ elements is split equally. The last group, however, can be split unevenly leading to an error of $O(k)$. Algorithmically such a family of functions that bisects with a small error would be sufficient for our purposes. It is, however, possible to achieve perfect balance at a small cost in the size of the family, which makes their application slightly easier. The next lemma shows that universal bisector families exist.

We leave the smallest possible size of such families as an open question as it is not a critical issue for the results of this paper, but give two comments on the issue: If $n = 2k$ you have to use all $\binom{2k}{k} = \Theta(k^{-1/2} 4^k)$ possible balanced bipartitions and it seems that a size of $O^*(4^k)$ is already optimal. If n is much bigger, however, a random perfect bipartition works with a relatively high probability of $\Omega^*(2^{-k})$ and suggests that families of size $O^*(2^k)$ exist, although it is not immediately clear how to construct families of that size.

▶ **Lemma 4.** *For every $k \in \boldsymbol{N}$ and $n \geq 2k(2k + 1)$ there is a k-universal bisector family of size $O(4^k k)$. A table of all functions in the family can be computed in time $O(4^k kn)$, which is linear in the size of the table.*

Proof. We use a slight modification of the construction in (1). Let $n = 2kd + r$ with $r < 2k$. Let us call the last r elements of $[n]$ the *jokers*. We define

$$b_M(t) = \begin{cases} 0 & \text{if } t < 2kd \text{ and } t \bmod 2k \in M, \\ 1 & \text{if } t < 2kd \text{ and } t \bmod 2k \notin M, \\ t \bmod 2 & \text{if } t \geq 2kd. \end{cases}$$

Such a function $b_M(t)$ maps exactly $\lceil n/2 \rceil$ elements to 0 and is therefore perfectly balanced, but those function do not give us a universal family of bisectors. If S contains odd jokers, not all of S is mapped to 0. We can overcome this problem by using $2k + 1$ families build in this way, but where each family uses a different interval in $[n]$ to place the jokers. In that way for each $S \subseteq [n]$ of size up to $2k$ there will be one family where S does not contain a joker (by the pigeon-hole principle). There is enough space for these intervals if $n \geq 2k(2k + 1)$ and the resulting universal family of bisectors has size $(2k + 1)4^k$. ◀

Combining k-universal bisectors and (n, k)-universal sets leads easily to (n, k)-universal colorings.

▶ **Lemma 5.** *For every $k \in \boldsymbol{N}$ and $n \geq 2k(2k+1)$ there is an (n, k)-universal coloring family of size $8^k k^{O(\log k)} \log n$. A table of all functions in the family can be computed in linear time.*

Proof. We use a k-universal bisector family $\mathcal{B}$ of size $O(4^k k)$ from Lemma 4 and an (n, k)-universal set $\mathcal{U}$ of size $2^k k^{O(\log k)} \log n$ according to [21]. For $f \in \mathcal{U}$ and $g \in \mathcal{B}$ we define a function $h\colon [n] \to \{\mathit{black}, \mathit{white}, \mathit{gray}\}$ as follows:

$$h(v) = \begin{cases} \mathit{gray} & \text{if } g(v) = 1 \\ \mathit{black} & \text{if } g(v) = 0 \text{ and } f(v) = 0 \\ \mathit{white} & \text{if } g(v) = 0 \text{ and } f(v) = 1 \end{cases}$$

It is easy to see that the set of all such h's forms an (n, k)-universal family of colorings. ◀

3 Results

We define the following three formulas:

$$\phi \equiv \forall \bar{x} \forall \bar{y} \exists z \Big(\bigwedge_{i,j=1}^{k} x_i \neq y_j \to \bigwedge_{i=1}^{k} (x_i \sim z \wedge y_i \nsim z) \Big)$$

$$\phi' \equiv \forall \bar{x} \forall \bar{y} \exists z \Big(\bigwedge_{i=1}^{k} \big(\chi(x_i) = \mathit{black} \wedge \chi(y_i) = \mathit{white} \big) \to \chi(z) = \mathit{gray} \wedge \bigwedge_{i=1}^{k} (x_i \sim z \wedge y_i \nsim z) \Big)$$

$$\phi'' \equiv \forall \bar{x} \forall \bar{y} \exists z \Big(\bigwedge_{i=1}^{k} \big(\chi(x_i) = \mathit{black} \wedge \chi(y_i) = \mathit{white} \big) \to \chi(z) = \mathit{gray} \wedge \bigwedge_{i=1}^{k} (x_i \nsim z \wedge y_i \nsim z) \Big)$$

The next lemma uses k in the formulas ϕ and ϕ' as the parameter.

▶ **Lemma 6.** *If there is an algorithm that, given a graph $G = G(n, 1/2)$ and a coloring $\chi\colon \mathcal{G} \to \{\mathit{gray}, \mathit{black}, \mathit{white}\}$, can decide in expected FPT time whether $(G, \chi) \models \phi'$, then there is an algorithm that can decide for $G = G(n, 1/2)$ in expected FPT time whether $G \models \phi$.*

Proof. Let X be a $2k$-universal family of colorings. Given a graph G first solve $(G,\chi) \models \phi'$ for every $\chi \in X$ in expected FPT time. If the answer is *yes* for every χ, then we can conclude $G \models \phi$, as by using a universal family of colorings, we have covered all distinguishable ways to color the nodes of G: Assume that $(G,\chi) \models \phi'$ for every $\chi \in X$, but $G \not\models \phi$. Then there exist $\bar{x}, \bar{y}$ where $\bar{x}$ and $\bar{y}$ are distinct such that for all z, $\bigwedge_{i=1}^{k}(x_i \sim z \wedge y_i \not\sim z)$ is unsatisfied. However, as $|\bar{x}| = |\bar{y}| = k$ and X is a $2k$-universal family of colorings, there is in particular a coloring $\chi \in X$ such that $\chi(\bar{x}) = \mathit{black}$ and $\chi(\bar{y}) = \mathit{white}$. If we assume that $G \not\models \phi$ for these particular $\bar{x}, \bar{y}$, then clearly also $(G,\chi) \not\models \phi'$ for these choices of χ and $\bar{x}, \bar{y}$.

We cannot tell whether $G \models \phi$ holds or not if $(G,\chi) \not\models \phi'$ for at least one coloring $\chi \in X$. The probability that this happens is exponentially small in n as we will show in the following. The negation of ϕ' reads

$$\exists\bar{x}\exists\bar{y}\forall z\Big(\bigwedge_{i=1}^{k}\big(\chi(x_i) = \mathit{black} \wedge \chi(y_i) = \mathit{white}\big) \wedge \Big(\chi(z) = \mathit{gray} \rightarrow \bigvee_{i=1}^{k}(x_i \not\sim z \vee y_i \sim z)\Big)\Big).$$

Once the coloring and $\bar{x}, \bar{y}$ are fixed, the probability that a gray vertex z is not connected to some x_i or connected to some y_i, is exactly $1 - 2^{-2k}$. This happens with all $\frac{n}{2} + O(k)$ many gray vertices with a probability of $(1 - 2^{-2k})^{\frac{n}{2}+O(k)}$. Altogether the probability of $G \not\models \phi'$ is

$$\sum_{\chi \in X} P[(G,\chi) \not\models \phi'] \leq |X| \binom{n}{k}\binom{n-k}{k}\Big(1 - \frac{1}{2^{2k}}\Big)^{\frac{n}{2}+O(k)} =$$

$$= 8^k n^{O(k)}\Big(1 - \frac{1}{2^{2k}}\Big)^{\frac{n}{2}+O(k)} = e^{-n2^{-2k-1}+O(k\log n)},$$

because there are $|X| \leq 8^k \mathit{poly}(n)$ (Lemma 5) many colorings and at most $\binom{n}{k}\binom{n-k}{k} = n^{O(k)}$ ways to choose $\bar{x}$ and $\bar{y}$. In that case, we can solve $G \models \phi$ by brute force in $n^{O(k)}$ time. For big enough n, the expected running time of the complete algorithm then remains in expected FPT time as long as $(G,\chi) \models \phi'$ is decidable in expected FPT time. The tradeoff works as long as n^k is subexponential and for large k the problem is automatically in FPT even in the worst-case. ◀

Let $\bar{G}_\chi = (V, E')$ be defined as follows: If $\chi(u) = \mathit{gray}$ and $\chi(v) = \mathit{black}$ or $\chi(u) = \mathit{black}$ and $\chi(v) = \mathit{gray}$ then $uv \in E'$ iff $uv \notin E$. Otherwise $uv \in E'$ iff $uv \in E$.

Informally speaking, $\bar{G}_\chi$ is the same graph as G, but edges are replaced by non-edges and vice versa between a vertex pair that is colored black and gray. It is important to note that if G is random then $\bar{G}_\chi$ is also random, although the edge probability of the flipped edges of $\bar{G}_\chi$ are inverted. This poses no problem if the edge probability is 1/2, but wrecks havoc when it is not. Edge probabilities different from 1/2 are discussed in Section 4.

▶ **Lemma 7.** *Let G be a graph and χ a coloring. Then $(G,\chi) \models \phi'$ iff $(\bar{G}_\chi,\chi) \models \phi''$.*

Proof. The only difference between ϕ' and ϕ'' is $x_i \sim z$ versus $x_i \not\sim z$. This subformula is only relevant when x_i is black and z is gray, as it is guarded by that condition. ◀

A coloring $\chi \colon [n] \to \{\mathit{black}, \mathit{white}, \mathit{gray}\}$ is *balanced* if $\lceil n/2 \rceil$ numbers are mapped to *gray*.

▶ **Lemma 8.** *If we can solve p-*MATRIX*$(\wedge)$ on random $n \times n$-matrices in expected FPT time, then we can solve $(G,\chi) \models \phi''$ for every balanced coloring χ on $G(n,1/2)$ in expected FPT time.*

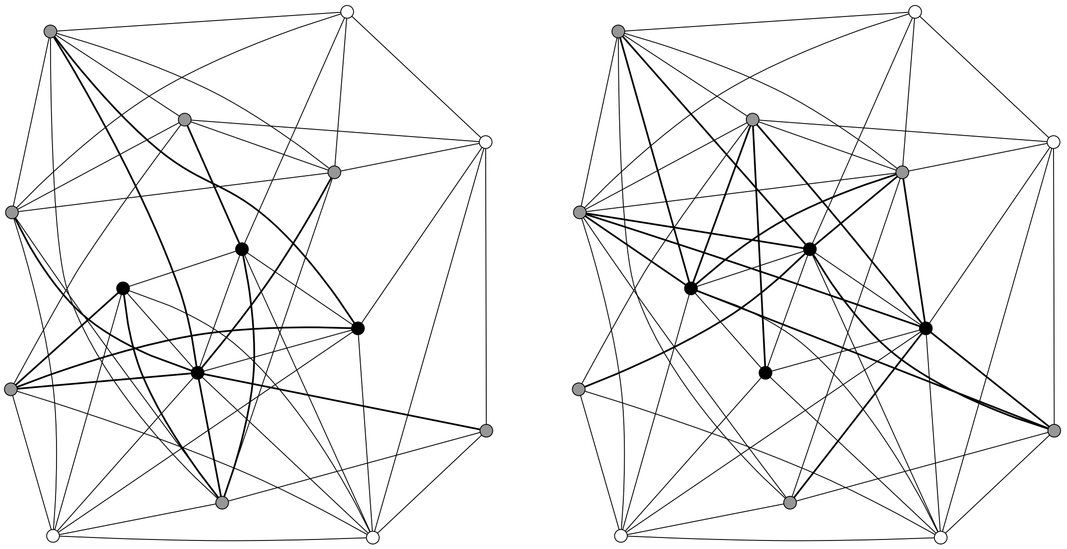


Figure 1 A graph G (left) with a node coloring χ and the corresponding graph $\bar{G}_\chi$ (right) with gray-black edges flipped.

Proof. Let $G = (V, E)$ be a random graph with n vertices and $\chi\colon \mathcal{G} \to \{\mathit{gray}, \mathit{black}, \mathit{white}\}$ be a balanced coloring. Assume first that n is even and that $V = \{v_1, \dots, v_{n/2}, u_1, \dots, u_{n/2}\}$ where v_i are not gray and u_i are gray. We construct a matrix $M \in \mathbf{F}_2^{n/2 \times n/2}$ such that $M_{ij} = 0$ iff $v_i u_j \in E$. This matrix is random and therefore we can find out in expected FPT time whether there are $2k$ rows $i_1, \dots, i_{2k}$ whose logical AND is 0. If this is not the case then $(G, \chi) \models \phi''$: Choosing $\bar{x}$ and $\bar{y}$ corresponds to picking $2k$ rows $i_1, \dots, i_{2k}$ of M. Choosing z corresponds to picking a column j of M. As M is a no-instance regardless what $i_1, \dots, i_{2k}$ are, the AND of the corresponding rows is never 0. So there is a column j with $M_{ij} = 1$ for all $i \in \{i_1, \dots, i_{2k}\}$ and correspondingly in G there is a gray z for every black $\bar{x}$, white $\bar{y}$ that is non-adjacent to all of $\bar{x}$ or $\bar{y}$ and we can conclude $(G, \chi) \models \phi''$.

Otherwise (if there are no such $2k$ rows) we can test in time $n^{O(k)}$ whether $(G, \chi) \models \phi''$. The probability for this to happen is at most $\binom{n}{2k}(1 - \frac{1}{2^{2k}})^{n/2}$ because there are $\binom{n}{2k}$ ways to choose $2k$ rows and for each column the probability is 2^{-2k} that it contains ones in all selected rows.

It remains to consider an odd n. In that case χ colors $\lceil n/2 \rceil$ vertices gray. Using the above construction leads to a matrix M with one more column than rows. The premise of the lemma allows us, however, only to assume that p-MATRIX$(\wedge)$ is solvable on square matrices. If we just remove the last column of M we are left with a random square matrix. The probability is still exponentially small that the truncated matrix is a yes-instance of p-MATRIX$(\wedge)$, but if it is a no-instance we can conclude that M is also a no-instance and then $(G, \chi) \models \phi''$ follows. Otherwise, we can again use a brute-force solution in time $n^{O(k)}$. ◀

▶ **Lemma 9.** *If we can solve p-DOMINATING SET on $G(n, 1/2)$ in expected FPT time, then we can solve p-MATRIX$(\wedge)$ on random matrices in expected FPT time.*

Proof. (Sketch) Let $M \in \mathbf{F}_2^{n \times n}$ be a random matrix. Let $\bar{M} = M \oplus 1$ (pointwise negation of M). Then M contains k rows whose AND is zero iff the directed graph H with $\bar{M}$ as its adjacency matrix has a dominating set of size k, which corresponds to the logical OR of k rows in $\bar{M}$ to be equal to the one vector.

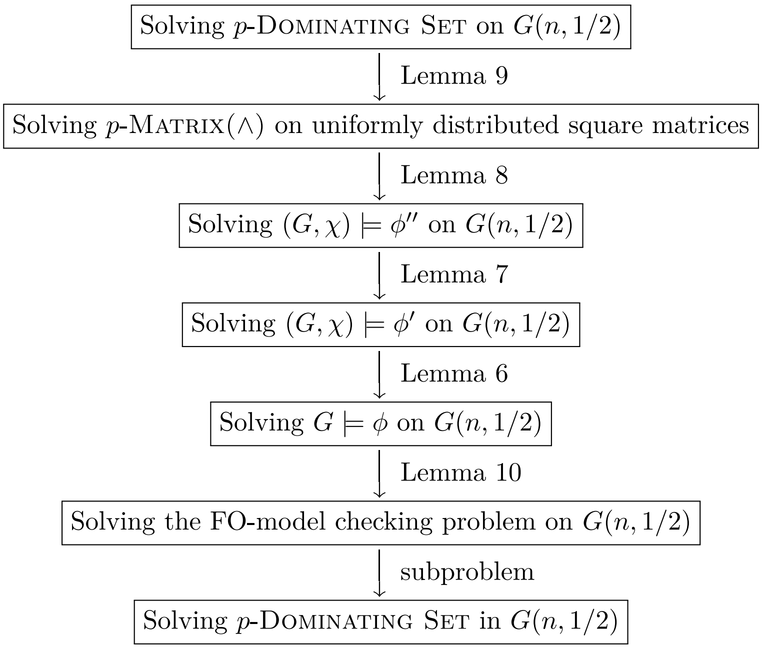

Solving p-Dominating Set on $G(n, 1/2)$
Lemma 9
Solving p-Matrix($\wedge$) on uniformly distributed square matrices
Lemma 8
Solving $(G, \chi) \models \phi''$ on $G(n, 1/2)$
Lemma 7
Solving $(G, \chi) \models \phi'$ on $G(n, 1/2)$
Lemma 6
Solving $G \models \phi$ on $G(n, 1/2)$
Lemma 10
Solving the FO-model checking problem on $G(n, 1/2)$
subproblem
Solving p-Dominating Set in $G(n, 1/2)$

Figure 2 Structure of the main proof. “Solving” means that the problem can be solved in expected FPT time. $A \longrightarrow B$ means that if A is in expected FPT time then so is B.

By using k-universal bisector families we can assume that a dominating set is among the first third of the vertices and that the k rows of a solution can be found in the upper third of $\bar{M}$. Decompose $\bar{M}$ into nine blocks and rearrange them as follows:

$$\bar{M} = \begin{pmatrix} A & B & C \\ D & E & F \\ G & H & I \end{pmatrix} \quad \rightarrow \quad M' = \begin{pmatrix} D' & B & A^T \\ B^T & E' & C \\ A & C^T & F' \end{pmatrix}$$

Here B^T is the transposed matrix B and A' is a symmetric matrix built from the upper triangular part of A. It is easy to see that M' is symmetric and random, if M is random. This means that M' is the adjacency matrix of a random, undirected graph G. It is also clear that if $\bar{M}$ contains k rows whose logical OR is the one vector, then M' contains $3k$ such rows and G has a dominating set of size $3k$. We can construct G and find out whether it has a dominating set of size $3k$. If not, then we know that M does not contain k rows whose logical AND is the zero vector. Otherwise, we can run a brute-force $n^{O(k)}$ algorithm, because the probability that we have to do this is at most $\binom{n}{3k}(1 - \frac{1}{2^{3k}})^{n-3k}$. ◀

▶ **Lemma 10.** *If there is an algorithm that can decide $G \models \phi$ on $G(n, 1/2)$ in expected FPT time, then there is an algorithm that solves p-MC(FO) on $G(n, 1/2)$ in expected FPT time.*

Proof. Let Φ be the set of extension axioms and ψ be a first-order formula on the logic of graphs. Remember that then either $\Phi \vdash \psi$ or $\Phi \vdash \neg\psi$ [11]. We can enumerate all proofs that use axioms in Φ in ascending order of length. Eventually we will find a proof $\Phi' \vdash \psi$ or $\Phi' \vdash \neg\psi$ for a finite subset $\Phi' \subseteq \Phi$. In our formulation of extension axioms (where $\bar{x}$ and $\bar{y}$ have the same size and only $x_i \neq y_j$, but not $x_i \neq x_j$, $y_i \neq y_j$ is required), a longer axiom implies all shorter ones. Therefore there is a single extension axiom ϕ with $2k + 1$ variables such that $\phi \vdash \psi$ or $\phi \vdash \neg\psi$. The length of ϕ and k are bounded by a function of the length of ψ because the whole proof that we found depends only on ψ. We decide in expected FPT time on the parameter $|\phi|$ and therefore also on the parameter $|\psi|$ whether $G \models \phi$. If the answer is *yes* we can conclude whether $G \models \psi$ holds and we are done. The answer is *no* only with an exponentially small probability in the number n of vertices if $|\phi|$ is small (for example if $|\phi| = O(\log n)$): There are $n - 2k$ possibilities to choose z outside of $\bar{x}$ and $\bar{y}$ and then the probability that z is correctly connected to them is exactly 2^{-2k}. Hence the probability of *no* is at most $n^{2k}(1-2^{-2k})^{n-2k} = \Omega(e^{-n/2k+k\log n})$ and we can use a brute-force $n^{O(k)}$ algorithm in that case. ◀

▶ **Theorem 11.** *p-Dominating Set can be solved on $G(n,1/2)$ in expected FPT time iff p-MC(FO) can be solved on $G(n,1/2)$ in expected FPT time.*

Proof. Assume p-Dominating Set can be solved on $G(n,1/2)$ in expected FPT time. By Lemmas 9, 8, 7, 6, and 10 we can conclude that p-MC(FO) can be solved on $G(n,1/2)$ in expected FPT time. The other direction is trivial as p-Dominating Set is a special case of p-MC(FO), where the length of the formula is linear in the size of the sought-after dominating set. See Figure 2 for an overview. ◀

4 Playing with the probability

Up to now we were looking at the uniform distribution of graphs, which corresponds to an edge probability of 1/2. It seems at first glance that all the proof techniques should also work for an arbitrary constant probability of $0 < p < 1$. A close look at the proofs, however, shows that Lemma 7 uses the fact that $p = 1/2$ in a crucial way: We flip gray–black edges, which changes their probability from p to $1-p$, which is only harmless when $p = 1/2$. The next lemma shows that we can prove a variant for an arbitrary edge probability.

▶ **Lemma 12.** *If we can solve $G \models \phi''$ on $G(n,p)$ in expected FPT time for some rational number $0 < p \le 1/2$, then we can also solve $G \models \phi'$ on $G(n,p)$ in expected FPT time.*

Proof. Let G' be a graph and χ a coloring and $\bar{G}'_\chi$ be the same graph with edges between a black and gray vertex flipped. Then $(\bar{G}'_\chi, \chi) \models \phi''$ iff $(G', \chi) \models \phi'$. However, if the edge probability in G' is p then the edge probability in $\bar{G}'_\chi$ is different: $1-p$ for gray–black edges and p for the others. Assume we can change that by turning every gray–black edge that is present in $\bar{G}'_\chi$ into a non-edge with probability $\frac{p}{1-p}$ such that the resulting probability for each edge is exactly p. Let us call the resulting graph $\bar{G}''_\chi$, which is distributed as $G(n,p)$ if G' is distributed as $G(n,p)$.

It is easy to see that $(G', \chi) \models \phi'$ follows from $(\bar{G}''_\chi) \models \phi''$ because ϕ'' remains true when removing edges. Nevertheless, the probability of $(\bar{G}''_\chi) \models \phi''$ is exponentially close to one. Therefore we can solve $(G', \chi) \models \phi'$ as follows: First find out whether $(\bar{G}''_\chi, \chi) \models \phi''$ in expected FPT time. If the answer is yes, we can conclude that $(G', \chi) \models \phi'$. Otherwise, we solve $(G', \chi) \models \phi'$ in $n^{O(k)}$ time.

We have, however, assumed that we can delete an edge with probability $p/(1-p)$, which would be true for a randomized algorithm. Using bisector families iteratively by applying the next family on the vertices that were mapped to 0 we can assume that the $\bar{x}$, $\bar{y}$ that are witnesses for $(\bar{G}''_\chi, \chi) \not\models \phi''$ are among the first ϵn vertices for any $\epsilon > 0$. Instead of testing whether ϕ'' holds on the whole graph we can now test only the subgraph induced by the first ϵn black and white vertices and all gray vertices. There are at most ϵn^2 edges between a relevant black and some gray vertex. Hence, we need only to simulate ϵn^2 coin tosses with a heads probability of $p(1-p)$, which is a rational number. Using von Neumann's trick we can simulate such a coin toss using expected $O(1)$ random bits [24]. Chernoff bounds show that the total number of such bits needed remains $O(\epsilon n^2)$ with a probability exponentially close to one. We can thus use the edges between gray vertices, there are $\binom{n/2}{2}$ many, as a coin toss with probability p. If we run out of simulated random bits, we can use a brute-force algorithm. ◀

Finally, we show that changing the probability does not make the problems harder or easier. This shows a certain robustness of the average-case complexity of p-MC(FO), p-Matrix($\wedge$), and p-Dominating Set.

▶ **Theorem 13.** *Let $0 < p, q < 1$, $p, q \in \mathbf{Q}$. p-MC(FO) can be solved on $G(n,p)$ in expected FPT time iff dominating set can be solved on $G(n,q)$ in expected FPT time.*

Proof. You can check that all steps in Figure 2 work for a probability other than $1/2$ except Lemma 7 (Lemmas 8 and 9 change p to $1-p$ and together leave it untouched). Plugging in Lemma 12 instead makes the whole chain work for any $0 < p \leq 1/2$. If we can solve p-MC(FO) on $G(n,p)$ we can solve it on $G(n,1-p)$, too, by complementing the graph and interchanging $\sim$ and $\not\sim$ in the formula. So far this shows that all problems in Figure 2 are in expected FPT or none of them on inputs distributed with an edge probability p for the graph problems and a probability of p that an entry is 1 for p-Matrix($\wedge$), but we still have to show that we can change the probability to q without making the problem harder or easier.

For this purpose we use p-Matrix($\wedge$). Let M be a matrix with probabilty p and we assume that $p < q$, otherwise we can invert the matrix and use $1-p$ and $1-q$. Similar to the proof of Lemma 12 we use iterated bisectors to reduce the problem of finding k rows to finding them in the first ϵn rows. We can then take a square submatrix consisting of the first ϵn rows and columns. Using the entries outside of this submatrix as coin tosses with heads probability p, we simulate coin tosses with a head probability of $1-q/p$. As before the probability is exponentially small that we do not succeed in the simulation (and have to use a brute-force algorithm). We use the $1-q/p$-coins to change a 1 in the submatrix to a 0 with probability $1-q/p$. Then the probability of a 1 in the submatrix becomes q. Now we can use the postulated algorithm to solve p-Matrix($\wedge$) on the transformed submatrix. If the answer is no for all bisectors then we can answer no. If the answer is at least once yes, which happens with an exponentially small probability, we can use a brute-force algorithm. ◀

The technique used above relies on p and q being rational numbers as we need to simulate a coin toss with head probability $1-q/p$, which is not necessarily possible for uncomputable probabilies (or even for very inefficiently computable ones).

5 Average Case Complexity of Even Set

The problem p-Matrix($\wedge$) turned out to be as hard as p-MC(FO) on random graphs. Instead of looking for k rows whose logical AND is the zero vector, we can also consider the related problem where we look for k rows whose exclusive or is zero. This variant is actually a problem well-known under the name Even Set and has many other equivalent definitions.

▶ **Definition 14.**

p-Even Set
Input: A matrix $A \in \mathbf{F}_2^{n\times n}$ and a number $k > 0$
Parameter: k
Problem: Are there k rows in A whose exclusive or is $\mathbf{0}$?

This problem was one of the original open problems in Downey and Fellows' book on parameterized complexity [8] and one of the few that has not been solved for a long time. Bhattacharyya et al. recently succeeded to classify the problem as W[1]-hard [2] under randomized fpt-reductions. As the problem is very similar to p-Matrix($\wedge$) we could expect that it is similarly hard on random matrices. It turns out, however, that we can solve it on uniformly distributed boolean square matrices in expected FPT time.

▶ **Theorem 15.** *p-Even Set can be solved in expected FPT time on uniformly distributed random square matrices.*

Proof (Sketch). Assume that M is a boolean $n/2 \times n$ matrix for an even n and $\Pr[M_{ij}] = 1/2$ independently for every $1 \leq i \leq n/2$, $1 \leq j \leq n$. It is easy to see and well-known (see, e.g., [18]) that the probability that M has not full rank is exponentially small in n.

Hence, we can proceed as follows to solve p-Even Set: First use a universal bisector family on a square matrix to select half the rows to form an $n/2 \times n$ matrix. If the original matrix was a yes-instance of p-Even Set, then it contained k rows whose exclusive or is the zero vector. Then the same holds for at least one of the transformed $n/2 \times n$ matrices. We check all of them for full rank. If all have full rank, then we conclude that the original matrix was a no-instance. Otherwise we use a brute-force $n^{O(k)}$ algorithm. This happens with small probability: As the $n/2 \times n$-matrices are random, all of them have full rank with high probability. The proof can easily be adjusted for odd n. ◀

It has to be noted that p-Even Set might behave quite differently on rectangular matrices. If a matrix has n columns, but n^2 rows it seems impossible to reduce the problem back to a square matrix and it is quite possible that the problem is then hard on average.

6 Conclusion

We have shown that the dominating set problem, which is with one quantifier alternation fairly low in the hierarchy of first-order definable properties, is nevertheless as hard to solve on average as any other problem that is first-order expressible. Stronger evidence for hardness under random instances would be a theorem that bridges the worlds of average-case and worst-case complexities. For example, some collapse in the W- or A-hierarchies implied by an efficient algorithm for dominating set on average instances would be a breakthrough.

A detail that is missing in this paper is the generalization of Theorem 13 to non-rational probability. We believe that more complicated techniques beyond the scope of this paper prove a generalization to even non-computable numbers, but we leave it as an open question to find a simple argument, which probably exists.

References

1 Shai Ben-David, Benny Chor, Oded Goldreich, and Michael Luby. On the theory of average case complexity. *J. Comput. Syst. Sci.*, 44(2):193–219, 1992. doi:10.1016/0022-0000(92)90019-F.

2 Arnab Bhattacharyya, Édouard Bonnet, László Egri, Suprovat Ghoshal, Karthik C. S., Bingkai Lin, Pasin Manurangsi, and Dániel Marx. Parameterized intractability of even set and shortest vector problem. *CoRR*, abs/1909.01986, 2019. arXiv:1909.01986.

3 Ralph Christian Bottesch. On W[1]-hardness as evidence for intractability. In Igor Potapov, Paul G. Spirakis, and James Worrell, editors, *43rd International Symposium on Mathematical Foundations of Computer Science, MFCS 2018, August 27-31, 2018, Liverpool, UK*, volume 117 of *LIPIcs*, pages 73:1–73:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.MFCS.2018.73.

4 Amin Coja-Oghlan and Anusch Taraz. Colouring random graphs in expected polynomial time. In Helmut Alt and Michel Habib, editors, *STACS 2003, 20th Annual Symposium on Theoretical Aspects of Computer Science, Berlin, Germany, February 27 - March 1, 2003, Proceedings*, volume 2607 of *Lecture Notes in Computer Science*, pages 487–498. Springer, 2003. doi:10.1007/3-540-36494-3_43.

5 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

6 Rodney G. Downey and Michael R. Fellows. Fixed-parameter intractability. In *Proceedings of the Seventh Annual Structure in Complexity Theory Conference, Boston, Massachusetts, USA, June 22-25, 1992*, pages 36–49. IEEE Computer Society, 1992. doi:10.1109/SCT.1992.215379.

7 Rodney G. Downey and Michael R. Fellows. Fixed parameter tractability and completeness. In Klaus Ambos-Spies, Steven Homer, and Uwe Schöning, editors, *Complexity Theory: Current Research, Dagstuhl Workshop, February 2-8, 1992*, pages 191–225. Cambridge University Press, 1992.

8 Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer, 1999. doi:10.1007/978-1-4612-0515-9.

9 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. doi:10.1007/978-1-4471-5559-1.

10 Jan Dreier and Peter Rossmanith. Hardness of FO model-checking on random graphs. In Bart M. P. Jansen and Jan Arne Telle, editors, *14th International Symposium on Parameterized and Exact Computation, IPEC 2019, September 11-13, 2019, Munich, Germany*, volume 148 of *LIPIcs*, pages 11:1–11:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.IPEC.2019.11.

11 Ronald Fagin. Probabilities on finite models. *J. Symb. Log.*, 41(1):50–58, 1976. doi:10.1017/S0022481200051756.

12 Jörg Flum and Martin Grohe. Model-checking problems as a basis for parameterized intractability. *Logical Methods in Computer Science*, 1(1), 2005. doi:10.2168/LMCS-1(1:2)2005.

13 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006. doi:10.1007/3-540-29953-X.

14 Nikolaos Fountoulakis, Tobias Friedrich, and Danny Hermelin. On the average-case complexity of parameterized clique. *Theor. Comput. Sci.*, 576:18–29, 2015. doi:10.1016/j.tcs.2015.01.042.

15 Etienne Grandjean. Complexity of the first-order theory of almost all finite structures. *Information and Control*, 57(2/3):180–204, 1983. doi:10.1016/S0019-9958(83)80043-6.

16 Martin Grohe. Generalized model-checking problems for first-order logic. In Afonso Ferreira and Horst Reichel, editors, *STACS 2001, 18th Annual Symposium on Theoretical Aspects of Computer Science, Dresden, Germany, February 15-17, 2001, Proceedings*, volume 2010 of *Lecture Notes in Computer Science*, pages 12–26. Springer, 2001. doi:10.1007/3-540-44693-1_2.

17 Yuri Gurevich. Complete and incomplete randomized NP problems. In *28th Annual Symposium on Foundations of Computer Science, Los Angeles, California, USA, 27-29 October 1987*, pages 111–117. IEEE Computer Society, 1987. doi:10.1109/SFCS.1987.14.

18 Valentin F. Kolchin. *Random graphs*, volume 53 of *Encyclopedia of mathematics and its applications*. Cambridge University Press, 1999.

19 Leonid A. Levin. Problems, complete in “average” instance. In Richard A. DeMillo, editor, *Proceedings of the 16th Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1984, Washington, DC, USA*, page 465. ACM, 1984. doi:10.1145/800057.808713.

20 Jerzy Loś. On the categoricity in power of elementary deductive systems and some related problems. *Colloq. Math.*, 3:58–62, 1954.

21 Moni Naor, Leonard J. Schulman, and Aravind Srinivasan. Splitters and near-optimal derandomization. In *36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin, USA, 23-25 October 1995*, pages 182–191. IEEE Computer Society, 1995. doi:10.1109/SFCS.1995.492475.

22 Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms.* Oxford University Press, 2006. doi:10.1093/ACPROF:OSO/9780198566076.001.0001.

23 Robert L. Vaught. Applications to the Löwenheim–Skolem–Tarski theorem to problems of completeness and decidability. *Indagationes Mathematicae*, 16:467–472, 1954.

24 John von Neumann. Various techniques used in connection with random digits. In Alston S. Householder, George E. Forsythe, and Hallett-Hunt Germond, editors, *Monte Carlo method. Proceedings of a Symposium Held June 29, 30 and July 1, 1949 in Los Angeles, California*, volume 12, pages 36–38, 1951. URL: https://dornsifecms.usc.edu/assets/sites/520/docs/VonNeumann-ams12p36-38.pdf.

A Scaling Algorithm for Weighted f-Factors in General Graphs

Ran Duan
Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China
duanran@mail.tsinghua.edu.cn

Haoqing He
Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China
hehq13@mails.tsinghua.edu.cn

Tianyi Zhang
Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China
tianyi-z16@mails.tsinghua.edu.cn

Abstract

We study the maximum weight perfect f-factor problem on any general simple graph $G = (V, E, \omega)$ with positive integral edge weights w, and $n = |V|$,$m = |E|$. When we have a function $f : V \to \mathbb{N}_+$ on vertices, a perfect f-factor is a generalized matching so that every vertex u is matched to exactly $f(u)$ different edges. The previous best results on this problem have running time $O(mf(V))$ [Gabow 2018] or $\tilde{O}(W(f(V))^{2.373}))$ [Gabow and Sankowski 2013], where W is the maximum edge weight, and $f(V) = \sum_{u\in V} f(u)$. In this paper, we present a scaling algorithm for this problem with running time $\tilde{O}(mn^{2/3} \log W)$. Previously this bound is only known for bipartite graphs [Gabow and Tarjan 1989]. The advantage is that the running time is independent of $f(V)$, and consequently it breaks the $\Omega(mn)$ barrier for large $f(V)$ even for the unweighted f-factor problem in general graphs.

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

Keywords and phrases Scaling Algorithm, f-Factors, General Graphs

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.41

Category Track A: Algorithms, Complexity and Games

Related Version A full version of the paper is available at `https://arxiv.org/abs/2003.07589`.

1 Introduction

Suppose we are given an undirected simple graph $G = (V, E, \omega)$ on n vertices and m edges, with positive integer edge weights $\omega : E \to \{1, 2, \cdots, W\}$. Let $f : V \to \mathbb{N}_+$ be a function that maps vertices to positive integers. An f-factor is a subset of edges $F \subseteq E$ such that $\deg_F(u) \le f(u)$ for all $u \in V$, and F is a *perfect* f-factor if $\deg_F(u) = f(u)$ for all $u \in V$. In this paper we are concerned with computing a perfect f-factor with maximum edge weights[1]

For polynomial running time algorithms, the previous best result on this problem has running time [2]$\tilde{O}(mf(V))$ [8], where conventionally $f(V) = \sum_{v\in V} f(v)$. When edge weights are small integers, a pseudo-polynomial running time of $\tilde{O}(W\,(f(V))^{2.373})$ was obtained using algebraic approaches by [9]. For unweighted graphs, one can achieve $\tilde{O}(m\sqrt{f(V)})$ running time using algorithms from [13, 6]. Faster algorithms with running time independent of $f(V)$ were obtained previously but only in bipartite graphs: [11] gave a scaling algorithm that runs in time $\tilde{O}(m^{2/3}n^{5/3} \log W)$ solving the more general min-cost unit-capacity max-flow problem,

[1] Of course, original definition of f-factors means perfect ones (e.g. in [16]), but we follow the definition from [13] which does not require perfectness for convenience.

[2] In this paper $\tilde{O}(\cdot)$ hides $\log n$ factors.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 41; pp. 41:1–41:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

and the time bound for bipartite f-factor was later improved to $\tilde{O}(m \min\{n^{2/3}, m^{1/2}\} \cdot \log W)$ in [10], and the time bound for min-cost flow was further improved to $\tilde{O}(mn^{1/2})$ and $\tilde{O}(m^{10/7} \log W)$ using algebraic approaches [15, 1]. For the maximum weight f-factor problem, if one is willing to settle for approximate solutions instead of the exact maximum, linear time algorithms can be found from [13, 2]. A closely related problem is the min-cost perfect b-matching, in which every edge can be matched multiple times. There are several classical results for b-matchings [10, 4, 7, 8].

In this paper we prove the following result, which is the first one to break the $\Omega(mn)$ barrier for perfect f-factors in general graphs.

▶ **Theorem 1.** *There is a deterministic algorithm that computes a maximum weight perfect f-factor in $\tilde{O}(mn^{2/3} \log W)$ time.*

1.1 Technical overview

Our algorithm is based on the scaling approach for maximum weight matching in general graphs that runs in time $\tilde{O}(m\sqrt{n} \log W)$ from [3] and the blocking flow method in [5, 14, 12]. Here we begin with a sketch of our idea on finding a perfect f-factor in an unweighted graph. To generalize it to weighted graphs, we will adapt the scaling algorithmic framework for maximum weight perfect matching from [3].

The algorithm for the unweighted case uses primal-dual approach for f-factors which was presented in [8]. It maintains a set of dual variables $y : V \to \mathbb{Z}$ and $z : 2^V \to \mathbb{N}$, as well as a laminar family of blossoms $\Omega \subseteq 2^V$ and a compatible f-factor F, which are initialized as $y = 0, z = 0, F = \Omega = \emptyset$. Basically, the algorithm invokes for $Cn^{2/3}$ times the Edmonds search procedure under an approximate complementary slackness constraint on F, y, z, Ω, where C is a sufficiently large constant. The key idea is that when G is a simple graph, after that we wish to prove that the total deficiency of the current f-factor F is bounded by $O(n^{2/3})$, namely $\sum_{v \in V}(f(v) - \deg_F(v)) \leq O(n^{2/3})$. If this is true, then we only need extra $O(n^{2/3})$ rounds of Edmonds searches to reach a perfect f-factor.

Let F^* be an arbitrary perfect f-factor. To upper bound the total deficiency $\sum_{v \in V}(f(v) - \deg_F(v)) \leq O(n^{2/3})$, we need to bound the total number of edge-disjoint augmenting walks in $F^* \oplus F$. Consider any augmenting walk which is specified by a sequence of consecutive edges $(u_1, u_2), (u_2, u_3), \cdots, (u_{2s-1}, u_{2s})$, where $(u_{2i-1}, u_{2i}) \in F^*, (u_{2i}, u_{2i+1}) \in F$, and all u_i's but u_1, u_{2s} are saturated vertices ($\deg_F(u_i) = f(u_i)$). If we start the search for y-values of all vertices equal to some positive constant, then y-values of unsaturated vertices remain equal. Since u_1, u_{2s} are both unsaturated vertices, we have $y(u_1) = y(u_{2s}) = -Cn^{2/3}$.

No blossoms. For bipartite graphs, we do not need to consider blossoms, so we can use the idea from [12, 5]. By approximate complementary slackness we know: $y(u_{2i-1}) + y(u_{2i}) \geq -2, y(u_{2i}) + y(u_{2i+1}) \leq 0$. Then we have $y(u_{2i+1}) - y(u_{2i-1}) \leq 2$, $y(u_{2s-1}) \geq Cn^{2/3}$. Consider the sequence of duals: $y(u_1), y(u_3), \cdots, y(u_{2s-1})$. This sequence starts with a small value $y(u_1) = -Cn^{2/3}$ but ends with a large value $y(u_{2s-1}) \geq Cn^{2/3}$, and so intuitively many of the differences $y(u_{2i+1}) - y(u_{2i-1})$ should be positive. However, given the upper bound $y(u_{2i+1}) - y(u_{2i-1}) \leq 2$, we would know many differences $y(u_{2i+1}) - y(u_{2i-1})$ can only belong to a very narrow range $\{1, 2\}$. In this case, since $y(u_{2i-1}) + y(u_{2i}) \geq -2, y(u_{2i}) + y(u_{2i+1}) \leq 0$, it must be $-1 - y(u_{2i+1}) \leq y(u_{2i}) \leq -y(u_{2i+1})$. In words, this augmenting walk contains an edge in $V_q \times V_{-q}$, where $V_x = \{|y(u) - x| \leq 1 \mid u \in V\}$, $q = y(u_{2i})$.

Since there are many different such pairs $y(u_{2i-1}), y(u_{2i+1})$, intuitively we can imagine this augmenting walk contains edges in $V_q \times V_{-q}$ for $\Omega(n^{2/3})$ different integer q's. If the number of augmenting walks is $\omega(n^{2/3})$, there will be $\Omega(n^{2/3})$ different $V_q \times V_{-q}$'s intersecting

$\omega(n^{2/3})$ augmenting walks each. By the pigeon-hole principle, there exists one such q such that $|V_q \cup V_{-q}| \leq O(n^{1/3})$. As G is a simple graph, the total number of edge-disjoint augmenting walks that contains an edge in $V_q \times V_{-q}$ is at most $|V_q \cup V_{-q}|^2 = O(n^{2/3})$, which comes to a contradiction.

Handling blossoms. The major difficulty for general graphs comes from the blossoms. We use the generalized blossoms introduced in [8], and utilize the blossom dissolution technique from [3], but it will become much more complicated for f-factors. To analyze the influence of blossoms, let us divide Ω into two categories: large and small. A blossom $B \in \Omega$ is large if $|B| \geq n^{1/3}$. For small blossoms, we know by definition, the total number of edges contained in any small blossoms is bounded by $n^{4/3}$. So if $F^* \oplus F$ contains $\geq Cn^{2/3}$ augmenting walks, then most augmenting walks contain $O(n^{2/3})$ small blossom edges. To restore the argument we discussed in previous paragraphs, we could safely remove those vertices incident to any edges belonging to small blossoms from the sequence $u_1, u_3, u_5, \cdots, u_{2s-1}$, and we could still work with a very long sequence of vertices that are not removed (if C is large).

As for large blossoms, we could prove that $\sum_{\text{large } B\in\Omega} z(B) \leq O(n^{4/3})$. Basically, this is because the total number of root large blossoms is always bounded by $n^{2/3}$, and so each round of Edmonds search could increase this sum by at most $n^{2/3}$, and therefore the algorithm could raise $\sum_{\text{large } B\in\Omega} z(B)$ to at most $Cn^{4/3}$ during $Cn^{2/3}$ executions of Edmonds search. Once we have a good handle of the total sum $\sum_{\text{large } B\in\Omega} z(B) \leq O(n^{4/3})$, we could argue that the "average influence" of large blossoms on each augmenting walk is bounded by $O(n^{2/3})$, if $F^* \oplus F$ has more than $Cn^{2/3}$ augmenting walks.

1.2 Structure of our paper

In Section 2 we define the notations and basic concepts we will use in the paper, while in Section 3 the algorithm is given. A brief analysis of the running time of the algorithm is given in Section 4. Due to page limit, many details of the proofs are omitted, which can be found in the full version of this paper.

2 Preliminaries

2.1 Notations

Our input is a weighted simple graph $G = (V, E, \omega)$ without self-loops and a function $f : V \to \mathbb{N}_+$. For $S \subseteq V$, define $f(S) = \sum_{v\in S} f(v)$, and let $\delta(S)$ and $\gamma(S)$ be sets of edges with exactly one endpoint and both endpoints in S, respectively, and $\delta(v)$ is an abbreviation for $\delta(\{v\})$. For any edge subset $F \subseteq E$, define $\delta_F(S) = \delta(S) \cap F$, $\omega(F) = \sum_{e\in F} \omega(e)$, and $\deg_F(u) = |F \cap \{(u, v) \in E\}|$. $F \subseteq E$ is called an f-factor if $\deg_F(u) \leq f(u)$ for all $u \in V$. For an f-factor F, the **deficiency** of u in F is defined as $f(u) - \deg_F(u)$ and u is **saturated** by F if $f(u) - \deg_F(u) = 0$. When all vertices are saturated, F is called a perfect f-factor. Edges in F are called **matching edges**. For a graph G and a subset of vertices U, $G[U]$ denotes the subgraph of G induced by U.

2.2 Blowup graphs

Instead of running on the original graph, our algorithm will be operating on an auxiliary weighted graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mu)$ which is called the *blowup graph.* $\mathcal{V}$ contains all original vertices in V. For each edge $e = (u, v)$ in the original graph, add two vertices e_u, e_v to $\mathcal{V}$ and add three edges $(u, e_u), (e_u, e_v), (e_v, v)$ to $\mathcal{E}$. All vertices in V are called *original* vertices, and the newly added vertices are called *auxiliary* vertices. Then assign $\mu(u, e_u) = \mu(v, e_v) = \omega(u, v)$, $\mu(e_u, e_v) = 0$ and $f(e_u) = f(e_v) = 1$.

The purpose of transferring original graph G to $\mathcal{G}$ is mainly to avoid edges contained in the sets $I(B)$ for disjoint blossoms B. (See subsection Blossom.) Note that the number of vertices in $\mathcal{G}$ is $n + 2m$, so we need to carefully analyze the running time. It is easy to see the following, whose proof is in the full version of this paper.

▶ **Lemma 2.** *Computing maximum weight perfect f-factor in G and $\mathcal{G}$ are equivalent.*

2.3 LP formulation

Computing maximum weight perfect f-factor on the blowup graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mu)$ can be expressed as a linear program [8]:

$$\begin{aligned}
\text{maximize} \quad & \sum_{e \in \mathcal{E}} \mu(e)x(e) \\
\text{subject to} \quad & \sum_{e \in \delta(v)} x(e) = f(v), \forall v \in \mathcal{V} \\
& \sum_{e \in \gamma(B) \cup I} x(e) \leq \left\lfloor \frac{f(B) + |I|}{2} \right\rfloor, \forall B \subseteq \mathcal{V}, I \subseteq \delta(B) \\
& 0 \leq x(e) \leq 1, \forall e \in \mathcal{E}
\end{aligned}$$

Its dual LP is written as the following.

$$\begin{aligned}
\text{minimize} \quad & \sum_{v \in \mathcal{V}} f(v)y(v) + \sum_{B \subseteq \mathcal{V}, I \subseteq \delta(B)} \left\lfloor \frac{f(B) + |I|}{2} \right\rfloor z(B, I) + \sum_{e \in \mathcal{E}} u(e) \\
\text{subject to} \quad & yz(e) + u(e) \geq \mu(e), \forall e \in \mathcal{E} \\
& z(B, I) \geq 0, u(e) \geq 0
\end{aligned}$$

Here $yz(u, v)$ is defined as: $yz(u, v) = y(u) + y(v) + \sum_{B, I : (u,v) \in \gamma(B) \cup I, I \subseteq \delta(B)} z(B, I)$.

2.4 Blossoms

We follow the definitions and the terminology of [8, 13] for f-factor blossoms. A blossom is specified by a tuple $(B, \mathcal{E}_B, \beta(B), \eta(B))$, where $B \subseteq \mathcal{V}$ is a subset of vertices, $\mathcal{E}_B \subseteq \mathcal{E}$ a subset of edges, $\beta(B) \in B$ a special vertex which is called the **base**, and $\eta(B)$ is either null or an edge from $\delta(\beta(B)) \cap \delta(B)$. Blossoms follow an inductive definition below.

▶ **Definition 3** (Blossom, [8, 13]). *A single vertex v forms a trivial blossom, also called a singleton. Here $B = \{v\}$, $\mathcal{E}_B = \emptyset$, $\beta(B) = v$, and $\eta(B)$ is null. Inductively, let $B_0, B_1, \cdots, B_{l-1}$ be a sequence of disjoint singletons or non-trivial blossoms. Suppose there exists a closed walk $C_B = \{e_0, e_1, \cdots, e_{l-1}\}$ starting and ending with B_0 such that $e_i \in B_i \times B_{i+1}$, ($B_l = B_0$). The vertex set $B = \bigcup_{i=0}^{l-1} B_i$ is identified as a blossom if the following are satisfied.*

1. *Base. If B_0 is a singleton, the two edges incident to B_0 on C_B, i.e., e_0 and e_{l-1}, must both be matched or both be unmatched.*
2. *Alternation. Fix a $B_i, i \neq 0$. If B_i is a singleton, exactly one of e_{i-1} and e_i is matched. If B_i is a non-trivial blossom, $\eta(B_i) = e_{i-1}$ or e_i.*

The edge set of the blossom B is $\mathcal{E}_B = C_B \cup (\cup_{i=0}^{l-1}\mathcal{E}_{B_i})$ and its base is $\beta(B) = \beta(B_0)$. If B_0 is not a singleton, $\eta(B) = \eta(B_0)$. Otherwise, $\eta(B)$ may either be null or one edge in $\delta(B) \cap \delta(B_0)$ that is the opposite type of e_0 and e_{l-1}.

A blossom is called **root blossom** if it is not contained in any other blossom. Blossoms have two different types: **light** and **heavy**. If B_0 is a singleton, B is light/heavy if e_0 and e_{l-1} are both unmatched/matched. Otherwise, B is light/heavy if B_0 is light/heavy.

▶ **Definition 4.** *Given an f-factor F, an alternating walk on $\mathcal{G}$ is a sequence of consecutive edges $(u_1, u_2), (u_2, u_3), \cdots, (u_{l-1}, u_l)$ such that:*

- *$(u_i, u_{i+1}) \in \mathcal{E}$ are different edges $1 \leq i < l$.*
- *exactly one of $(u_{i-1}, u_i), (u_i, u_{i+1})$ belongs to F, $1 < i < l$.*

This walk is called an augmenting walk if both $(u_1, u_2), (u_{l-1}, u_l) \notin F$.

When searching for an augmenting walk, a blossom behaves as a unit in the graph. These properties are formally stated by the following lemma.

▶ **Lemma 5** ([8, 13])**.** *Let v be an arbitrary vertex in B. There exists an even length alternating walk $P_0(v)$ and an odd length alternating walk $P_1(v)$ from $\beta(B)$ to v using edges in $\mathcal{E}_B$. Moreover, the terminal edge of $P_{0,1}(v)$ incident to $\beta(B)$ must have a different type than $\eta(B)$, if $\eta(B)$ is defined.*

We also introduce the notion of **maturity** of blossoms below.

▶ **Definition 6** (Mature Blossom, [8, 13])**.** *A blossom is mature with respect to an f-factor F if the following requirements are satisfied.*

1. *Every vertex $v \in B \setminus \{\beta(B)\}$ is saturated, namely $\deg_F(v) = f(v)$.*
2. *The deficiency of $\beta(B)$ is at most 1. Furthermore, if it is 1, then B must be a light blossom and $\eta(B)$ is null; otherwise, $\eta(B)$ is defined.*

Our algorithm always keeps a set Ω of mature blossoms and maintains a non-negative value $z(B)$ for each $B \in \Omega$. For each blossom B, define a set $I(B) \subseteq \delta(B)$ as $I(B) = \delta_F(B) \oplus \{\eta(B)\}$.

2.5 Augmenting path

To find augmentations, we need to work with the contraction graph $\widehat{\mathcal{G}}$ where every root blossom is contracted to a single node.

▶ **Definition 7** ([8, 13])**.** *Let F, Ω and $\widehat{\mathcal{G}}$ be an f-factor, a set of blossoms and the graph obtained by contracting every root blossom in the Ω, respectively. $\widehat{P} = \langle B_0, e_0, B_1, e_1, \cdots, B_l \rangle \in \widehat{\mathcal{G}}$ is called an augmenting path if the following requirements are satisfied.*

1. *The terminals B_0 and B_l must be unsaturated singletons or unsaturated light blossoms. If $\widehat{P}$ is a closed path ($B_0 = B_l$), B_0 must be a singleton and the deficiency of $\beta(B_0)$ is at least 2. Otherwise B_0 and B_l can be either singletons or blossoms and their deficiency must be positive.*
2. *If the terminal vertex B_0 (B_l) is a singleton, then the incident terminal edges e_0 (e_{l-1}) must be unmatched. Otherwise, they can be either matched or unmatched.*
3. *Let B_i, $0 < i < l$ be an internal singleton or blossom. If B_i is a singleton, then exactly one of e_{i-1} and e_i is matched. If B_i is a non-trivial blossom, then $\eta(B_i) = e_{i-1}$ or e_i.*

To avoid misunderstanding, we emphasize the difference between the augmenting path and the augmenting walk. First they are defined on $\widehat{\mathcal{G}}$ and $\mathcal{G}$ respectively. Second, an augmenting walk can pass through a vertex in $\mathcal{G}$ several times but an augmenting path can pass through a vertex in $\widehat{\mathcal{G}}$ (except the endpoint) only once. In the following parts, these two concepts are used in different scenarios.

Next we define a concept of the alternating path, which is weaker than the concept of the augmenting path.

▶ **Definition 8.** *Let F, Ω and $\widehat{\mathcal{G}}$ be an f-factor, a set of blossoms and the graph obtained by contracting every root blossom in the Ω. A simple path $\widehat{P} = \langle B_0, e_0, B_1, e_1, \cdots, B_l\rangle$ is called an alternating path if it satisfies the following requirements.*

1. *The terminals B_0 must be unsaturated singletons or unsaturated light blossoms.*
2. *If the terminal vertex B_0 is a singleton, then the incident terminal edges e_0 must be non-matching. Otherwise, they can be either matching or non-matching.*
3. *For each $1 \leq i < l$, if B_i is a singleton, then exactly one of e_{i-1}, e_i is matched. Otherwise, $\eta(B_i) = e_{i-1}$ or e_i.*

2.6 Complementary slackness

Throughout the algorithm, we will be maintaining an f-factor F, a set of mature blossoms Ω, dual functions $y : \mathcal{V} \to \mathbb{N}$, $z : \Omega \to \mathbb{N}_{\geq 0}$ and $yz : \mathcal{E} \to \mathbb{N}$. For an f-factor F, we define two kinds of complementary slackness: complementary slackness and approximate complementary slackness.

▶ **Definition 9** (Complementary Slackness). *In the blowup graph $\mathcal{G}$, an f-factor F, duals y, z, as well as a laminar family of blossoms Ω satisfy* complementary slackness *if the following requirements hold.*

1. *Dominance. For each $e \in \mathcal{E}$, $yz(e) \geq \mu(e)$.*
2. *Tightness. For each $e \in F$, $yz(e) = \mu(e)$.*

▶ **Definition 10** (Approximate Complementary Slackness). *In the blowup graph $\mathcal{G}$, an f-factor F, duals y, z, as well as a laminar family of blossoms Ω satisfy* approximate complementary slackness *if the following requirements hold.*

1. *Dominance. For each $e \in \mathcal{E}$, $yz(e) \geq \mu(e) - 2$.*
2. *Tightness. For each $e \in F$, $yz(e) \leq \mu(e)$.*

▶ **Lemma 11** ([13]). *Let F be a perfect f-factor associated with duals y, z and blossoms Ω, and define F^* to be a maximum weight perfect f-factor. Suppose F, Ω, y, z satisfy approximate complementary slackness, then*

$$\mu(F) \geq \mu(F^*) - f(\mathcal{V})$$

Proof. We first define $u : \mathcal{E} \to \mathbb{N}$ as

$$u(e) = \begin{cases} \mu(e) - yz(e), & \text{if } e \in F \\ 0, & \text{otherwise} \end{cases}$$

According to the approximate domination and tightness properties, we have $u(e) \geq 0$ for all $e \in \mathcal{E}$. Moreover, $yz(e) + u(e) \geq \mu(e) - 2$ for all $e \in \mathcal{E}$. This gives the following:

$$\begin{aligned}
\mu(F) &= \sum_{e\in F}(yz(e) + u(e)) = \sum_{v\in V} deg_F(v)y(v) + \sum_{B\in\Omega} |F \cap (\gamma(B) \cup I(B))|z(B) + \sum_{e\in F} u(e) \\
&= \sum_{v\in V} f(v)y(v) + \sum_{B\in\Omega} \left\lfloor \frac{f(B) + |I(B)|}{2} \right\rfloor z(B) + \sum_{e\in\mathcal{E}} u(e) \\
&\geq \sum_{v\in V} deg_{F^*}(v)y(v) + \sum_{B\in\Omega} |F^* \cap (\gamma(B) \cup I(B))|z(B) + \sum_{e\in F^*} u(e) \\
&\geq \sum_{e\in F^*} (\mu(e) - 2) \geq \mu(F^*) - f(\mathcal{V})
\end{aligned}$$ ◀

2.7 Edmonds search

In this subsection, we introduce two different implementations of Edmonds search. Suppose we have an f-factor F, a set of blossoms Ω, and duals y, z satisfying some kind of slackness condition. The purpose of Edmonds search is to reduce total deficiency of F by eligible augmenting paths. We need two different notions of eligibility, namely eligibility and approximate eligibility, compatible with Definition 9 or Definition 10.

▶ **Definition 12** (Eligibility, [8]). *An edge $e \in \mathcal{E}$ is eligible if $yz(e) = \mu(e)$.*

▶ **Definition 13** (Approximate Eligibility, [13]). *An edge $e \in E$ is approximately eligible if it satisfies one of the following.*

1. *$e \in \mathcal{E}_B$ for some $B \in \Omega$.*
2. *$e \notin F$ and $yz(e) = \mu(e) - 2$.*
3. *$e \in F$ and $yz(e) = \mu(e)$.*

Let $\widehat{\mathcal{G}_{\text{elig}}}$ be the subgraph of $\widehat{\mathcal{G}}$ consisting of eligible edges. A root blossom $B' \in \Omega$ is called reachable from an unsaturated root blossom B via an alternating path in $\widehat{\mathcal{G}_{\text{elig}}}$, if there is an alternating path that starts at B and ends at B'. To find augmenting paths and blossoms in $\widehat{\mathcal{G}_{\text{elig}}}$, we start from any unsaturated node u_0 in the contraction graph $\widehat{\mathcal{G}}$ and grow a search tree $\widehat{T}$ rooted at u_0; this method was also described in [8, 13]. All nodes in $\widehat{T}$ are classified as **outer/inner**. Initially the root is outer. Next we use a DFS-like approach to build the entire $\widehat{T}$. During the process, we keep track of a tree path $\langle u_0, e_0, u_1, \cdots, e_{l-1}, u_l\rangle$ from the root, which is guaranteed to be an alternating path. According to the type of u_l, the next edge e_l and node u_{l+1} are selected by the rules below:

1. u_l is outer. If u_l is a singleton, then scan the next non-matching edge e_l and find the other endpoint u_{l+1}. If u_l is a nontrivial blossom, then scan the next edge e_l and find the other endpoint u_{l+1}.
2. u_l is inner. If u_l is a singleton, then scan the next matching edge e_l and find the other endpoint u_{l+1}. If u_l is nontrivial blossom, then assign $e_l = \eta(u_l)$ (if it was not scanned before) and find the other endpoint u_{l+1}.

After finding u_{l+1}, we try to classify it as outer or inner: if u_{l+1} is a singleton, then u_{l+1} is outer if e_l is matched; otherwise, u_{l+1} is outer if $e_l = \eta(u_{l+1})$. Issues may arise if (1) u_{l+1} was already classified by previous tree searches and there is a conflict between the new label and the old label; or (2) u_{l+1} is an unsaturated then the tree search has found a new augmenting path. In either case we can construct a new blossom or reduce the total deficiency.

In the end, when all reachable singletons or root blossoms are classified as outer or inner, let $\widehat{\mathcal{V}}_{\text{out}}$ be the set of all outer singletons or root blossoms, and let $\widehat{\mathcal{V}}_{\text{in}}$ be the set of all inner singletons or root blossoms. Define $\mathcal{V}_{\text{out}}, \mathcal{V}_{\text{in}}$ to be the set of all vertices in $\mathcal{V}$ contained in outer and inner root blossoms, respectively. Next we introduce a meta procedure that will be a basic building block, which is dual adjustment. A dual adjustment performs the following step: decrement $y(v)$ for all $v \in \mathcal{V}_{\text{out}}$, and increment $y(v)$ for all $v \in \mathcal{V}_{\text{in}}$; after that, increment by 2 all $z(B)$ for all $B \in \widehat{\mathcal{V}}_{\text{out}}$, and decrement by 2 all $z(B)$ for all $B \in \widehat{\mathcal{V}}_{\text{in}}$. This is summarized as the AdjustDuals algorithm 1.

We introduce two different implementations of Edmonds search: the EdmondsSearch algorithm 2 and the PQ-Edmonds algorithm 3, which both rely on the AdjustDuals subroutine 1. The EdmondsSearch algorithm searches from all unsaturated root blossoms, and it requires that the y-values of all unsaturated vertices have the same parity. It reserves approximate complementary slackness under the approximate eligibility, so it only needs to perform one step of augmentation before dual-adjustment:

Algorithm 1 AdjustDuals(F, Ω, y, z).

```
classify every root blossom in $\Omega$ as outer or inner;
let $\widehat{\mathcal{V}_{\text{out}}}/\widehat{\mathcal{V}_{\text{in}}}$ be the set of all outer/inner root blossoms in $\widehat{\mathcal{G}_{\text{elig}}}$ including singletons;
  let $\mathcal{V}_{\text{out}}/\mathcal{V}_{\text{in}}$ be the set of all vertices in $\mathcal{V}$ contained in outer/inner root blossoms;
adjust the duals $y, z$ as follows:

    $y(v) \leftarrow y(v) - 1, v \in \mathcal{V}_{\text{out}}$
    $y(v) \leftarrow y(v) + 1, v \in \mathcal{V}_{\text{in}}$
    $z(B) \leftarrow z(B) + 2, \text{for non-singleton } B \in \widehat{\mathcal{V}_{\text{out}}}$
    $z(B) \leftarrow z(B) - 2, \text{for non-singleton } B \in \widehat{\mathcal{V}_{\text{in}}}$
```

Algorithm 2 EdmondsSearch(F, Ω, y, z).

```
/* Precondition:  y-values of unsaturated vertices must all be of the
   same parity                                                        */
find a maximal set $\widehat{\Psi}$ of a vertex-disjoint augmenting paths in $\widehat{\mathcal{G}_{\text{elig}}}$ and extend $\widehat{\Psi}$ to a
  set $\Psi$ of vertex-disjoint augmenting walks in $\mathcal{G}_{\text{elig}}$;
update $F \leftarrow F \oplus \bigcup_{P \in \Psi} P$;
find a maximal set $\Omega'$ of mature blossoms reachable from unsaturated vertices in $\widehat{\mathcal{G}_{\text{elig}}}$;
update $\Omega \leftarrow \Omega \cup \Omega'$ and $\widehat{\mathcal{G}_{\text{elig}}}$;
run AdjustDuals($F, \Omega, y, z$);
for every matching edge $(u, v)$ that does not satisfy the dominance condition, choose
  an auxiliary node $u$, $y(u) \leftarrow \mu(u, v) - y(v) - \sum_B z(B)$;
recursively remove all root blossoms whose dual values are zero;
```

▶ **Lemma 14** ([13]). *In the EdmondsSearch algorithm, after augmentation and blossom formation, $\widehat{\mathcal{G}_{elig}}$ does not contain any augmenting paths.*

Proof. Suppose that, after the augmentation and blossom formation, there is an augmenting path P in $\widehat{\mathcal{G}_{\text{elig}}}$. Since $\widehat{\Psi}$ is maximal, P must intersect some augmenting path $P' \in \widehat{\Psi}$ at a vertex v. However, after the augmentation and blossom formation every edge in P' will become ineligible, so the matching edge $(v, v') \in P$ is no longer in $\widehat{\mathcal{G}_{\text{elig}}}$, contradicting the fact that P consists of eligible edges. ◀

The PQ-Edmonds algorithm searches for augmenting paths only from a set U of unsaturated vertices whose y-values share the same parity, halting after finding an augmenting path from vertices in U or making D dual adjustments.

The following two lemmas describe the properties of the EdmondsSearch algorithm and the PQ-Edmonds algorithm[3] which are from [8, 13]. Their proofs are presented in the full version of this paper.

[3] In the original paper [8], their algorithm actually searches from all unsaturated root blossoms. This slack can be remedied by the following reduction. For each unsaturated vertex $v \notin U$, match v to $f(v) - \deg_F(v)$ new temporary vertices whose duals are equal to $-y(v)$ and the matching edges have zero weight. The proof is described in the full version of this paper.

Algorithm 3 PQ-Edmonds(F, Ω, y, z, U, D).

```
/* Precondition:  $\{y(u)|u \in U\}$ must all be of the same parity   */
while less than $D$ dual adjustments have been made so far do
    if an augmenting path $P$ from vertices in $U$ is found then
        update $F \leftarrow F \oplus P$;
        break;
    end
    find a maximal set $\Omega'$ of mature blossoms reachable from $U$ in $\widehat{\mathcal{G}_{\text{elig}}}$;
    update $\Omega \leftarrow \Omega \cup \Omega'$ and $\widehat{\mathcal{G}_{\text{elig}}}$;
    run AdjustDuals$(F, \Omega, y, z)$;
    recursively remove all root blossoms whose dual values are zero;
end
for every matching edge $(u, v)$ that does not satisfy the dominance condition, choose
  an auxiliary node $u$, $y(u) \leftarrow \mu(u, v) - y(v) - \sum_B z(B)$;
```

▶ **Lemma 15.** *The EdmondsSearch algorithm preserves* ***approximate complementary slackness*** *under the* ***approximate eligibility*** *definition. Furthermore, one execution can be implemented in $O(m)$ time.*

▶ **Lemma 16.** *The PQ-Edmonds algorithm preserves the* ***complementary slackness*** *under the* ***eligibility*** *definition. Furthermore, one execution can be implemented in $O(m \log n)$ time. Moreover, the $y(u)$ for unsaturated vertex u not in U will not be increased during the algorithm.*

3 The Scaling Algorithm

Our algorithm follows the idea of the scaling algorithm in [3] for maximum weight perfect matching. The scaling algorithm maintains an f-factor F, a family of blossoms Ω, as well as duals y, z, and it is divided into $\lceil \log(2f(\mathcal{V})W) \rceil$ iterations. Edge weights $\mu(e)$ are rescaled to $2f(\mathcal{V})\mu(e)$ and they all have $\lceil \log(2f(\mathcal{V})W) \rceil$ bits. Throughout the algorithm we assume y always assigns integer values and z always assigns even non-negative integers. For any $B \in \Omega$, B is called a *large blossom* if $|B \cap V| \geq n^{1/3}$, i.e., the number of original vertices in B is at least $n^{1/3}$; otherwise it is deemed a *small blossom*.

Let $\bar{\mu}$ be the edge weight function that keeps track of the scaled edge weights in each iteration. Initially before the first iteration, assign $F, \Omega = \emptyset$, $y, z, \bar{\mu} = 0$. At the beginning of each iteration, define F_0 to be the f-factor from the previous iteration. Empty the matching $F \leftarrow \emptyset$, and update weights and duals as following.

$$\bar{\mu}(e) \leftarrow 2\left(\bar{\mu}(e) + \text{the next bit of } 2f(\mathcal{V})\mu(e)\right)$$
$$y(u) \leftarrow 2y(u) + 3$$
$$z(B) \leftarrow 2z(B)$$

The whole procedure is shown in the Scaling algorithm 5, involving an important subroutine: the Dissolve algorithm 4. Note that here we only provide a stretch of the proof ideas, while the whole proofs can be found in the full version of this paper.

Algorithm 4 Dissolve(B, y, z, Ω).

```
for $u \in B$ or there exists $v \in B$ such that $(u, v) \in I(B)$ do
    $y(u) \leftarrow y(u) + z(B)/2$;
end
$z(B) \leftarrow 0$ and remove it from $\Omega$;
```

3.1 Correctness

In this subsection, we will show that the Scaling algorithm indeed returns the maximum weight perfect f-factor in G, and in the next subsection we will analyze its time complexity. Some proofs of the statements here are omitted and can be found in the full version of this paper.

Define $\zeta(B) = (e_u, e_v)$, if $(u, e_u) = \eta(B)$ and e_v is the auxiliary vertex adjacent to e_u. It is not hard to see that:

▶ **Lemma 17.** *For any blossom $B \in \Omega$ in $\mathcal{G}$, the edge $e \in \delta(B)$ has the form of (u, e_u) where $u \in B$ is an original vertex and e_u is an auxiliary vertex.*

▶ **Lemma 18.** *There are two properties right after the scaling step (Line 3-6):*
1. *For each $e \in \mathcal{E}$, $\bar{\mu}(e) \leq yz(e)$.*
2. *For each $e \in F_0$, $\bar{\mu}(e) \geq yz(e) - 6$.*

▶ **Lemma 19.** *There are two properties right after the blossom dissolution step (Line 7-15):*
1. *For each $(u, v) \notin F_0 \cup \bigcup_{i=1}^{l} \gamma(B_i) \cup I(B_i)$, $\bar{\mu}(u, v) \leq 0$.*
2. *For each $(u, v) \in \mathcal{E}$, $\bar{\mu}(u, v) \leq 2\min\{y(u), y(v)\}$.*

Proof. Let $\bar{\mu}_1, y_1, z_1, \Omega_1$ be the edge weights, duals and blossoms at the beginning of the step of blossom dissolution, respectively. Let $\Omega_1' \subseteq \Omega_1$ be the set of all blossoms that are dissolved in Line 7-9 before the step of reweighting. By Lemma 18: (Note that for auxiliary edges adjacent to edges in $I(B)$, blossom dissolution can only cause $\bar{\mu}(u, v)$ to become smaller.)

$$\begin{aligned}\bar{\mu}(u, v) &\leq \bar{\mu}_1(u, v) - y_1(u) - y_1(v) - \sum_{\substack{B \in \Omega_1' \\ (u,v) \in \gamma(B) \cup I(B)}} z_1(B) \\ &\leq yz_1(u, v) - y_1(u) - y_1(v) - \sum_{\substack{B \in \Omega_1' \\ (u,v) \in \gamma(B) \cup I(B)}} z_1(B) \\ &= \sum_{\substack{B \in \Omega_1 \setminus \Omega_1' \\ (u,v) \in \gamma(B) \cup I(B)}} z_1(B)\end{aligned}$$

The last term is zero when $(u, v) \notin \bigcup_{i=1}^{l} \gamma(B_i) \cup I(B_i)$.

Hence, by the end of the step of blossom dissolution,

$$y(u) = \frac{1}{2} \sum_{\substack{B \in \Omega_1 \setminus \Omega_1' \\ \exists (u,w) \in \gamma(B) \cup I(B)}} z_1(B) \geq \frac{1}{2} \sum_{\substack{B \in \Omega_1 \setminus \Omega_1' \\ (u,v) \in \gamma(B) \cup I(B)}} z_1(B) \geq \frac{1}{2}\bar{\mu}(u, v)$$

By symmetry, we can also prove $y(v) \geq \frac{1}{2}\bar{\mu}(u, v)$. Then $\bar{\mu}(u, v) \leq 2\min\{y(u), y(v)\}$. ◀

Next we study what happens during the step of augmentation within small blossoms. If a matching edge (u, e_u) is newly added to F in line 17, for some small blossom B_i, $(u, e_u) \in I_{F_0}(B_i) \setminus \{\eta(B_i)\}$. The following statements will be proved in the full version.

Algorithm 5 Scaling($\mathcal{V}, \mathcal{E}, \mu, f$).

```
1  $y, z \leftarrow 0,\ F, \Omega \leftarrow \emptyset$;
2  for $iter = 1, \cdots, \lceil \log(2f(\mathcal{V})W) \rceil$ do
       /* scaling                                                        */
3      $\bar{\mu}(e) \leftarrow 2\,(\bar{\mu}(e) + \text{the next bit of } 2f(\mathcal{V})\mu(e))$;
4      $y(u) \leftarrow 2y(u) + 3$;
5      $z(B) \leftarrow 2z(B)$;
6      $F_0 \leftarrow F,\ F \leftarrow \emptyset$;
       /* blossom dissolution (Line 7-15)                                */
7      while exists a large blossom $B \in \Omega$, or a root blossom $B$ with $z(B) \le 12$ do
8          run Dissolve($B, y, z, \Omega$);
9      end
10     $\bar{\mu}(u,v) \leftarrow \bar{\mu}(u,v) - y(u) - y(v), \forall (u,v) \in \mathcal{E}$;
11     $y(u) \leftarrow 0, \forall u \in \mathcal{V}$;
12     let $B_1, B_2, \cdots, B_l$ be all the root small blossoms not dissolved yet;
13     while exists a blossom $B \in \Omega$ do
14         run Dissolve($B, y, z, \Omega$);
15     end
       /* $\Omega$ now becomes empty.                                     */
       /* augmentation within small blossoms (Line 16-27)                */
16     if $(u,v) \in I_{F_0}(B_j) \setminus \{\eta(B_j)\}$ for some previous root small blossom $B_j$ then
17         $F \leftarrow F \cup \{(u,v)\}$;
18         if $u \in B_j, v \notin B_j$, $y(v) \leftarrow \bar{\mu}(u,v) - y(u)$;
19     end
20     for $i = 1, 2, \cdots, l$ do
21         while $\max\{y(u) \mid \deg_F(u) < f(u), u \in B_i\} > 6$ do
22             let $Y_1, Y_2$ be the largest and second largest $y$ values of unsaturated vertices
                in $B_i$;
23             define $U \subseteq B_i$ to be the set of unsaturated vertices whose $y$ values equal to
                $Y_1$;
24             define $H_i = \mathcal{G}[B_i \cup$ all the endpoints of $I_{F_0}(B_i) \setminus \{\eta(B_i)\}]$;
25             run PQ-Edmonds($F, \Omega, y, z, U, Y_1 - Y_2$) in subgraph $H_i$;
26         end
27     end
       /* deficiency reduction                                           */
28     run EdmondsSearch($F, \Omega, y, z$) on the entire graph $G$ for $\lceil Cn^{2/3} \rceil + 6$ times;
29 end
   /* weight adjustment                                                  */
30 for an edge $(u,v) \in \mathcal{E}$ such that $yz(u,v) < \mu(u,v)$ do
31     $\mu(u,v) \leftarrow yz(u,v)$;
32 end
   /* PQ-deficiency reduction                                            */
33 repeat PQ-Edmonds($F, \Omega, y, z, \{u \mid u \in \mathcal{V}$ is unsaturated$\}, \infty$) on the entire graph $G$
     until the total deficiency becomes zero;
```

- At the beginning of the step of augmentation within small blossoms, $\bar{\mu}(u, e_u) \geq y(u) + y(e_u) - 6$.
- After we have added (u, e_u) to F and reassigned $y(e_u) \leftarrow \bar{\mu}(u, e_u) - y(u)$, the complementary slackness is preserved. Plus, $y(e_u) \geq \frac{1}{2}\bar{\mu}(u, e_u) - 6$.
- Within the while-loop, for any $u \in B_i$ reachable via alternating paths from U, $y(u) \geq Y_1$. Plus, $y(e_u) \geq 0$ at any moment. Also from Lemma 16, the y-values of unsaturated vertices outside current U cannot increase.

Then we can conclude the correctness of the algorithm:

▶ **Lemma 20.** *The Scaling algorithm 5 returns a maximum weight perfect f-factor in G.*

Proof. First we claim that approximate complementary slackness is maintained until the step of weight adjustment (Line 30-32). By Lemma 19, the tightness of complementary slackness is satisfied after the step of blossom dissolution. For each edge e newly added to F, $yz(e) = \mu(e)$ and the dominance of complementary slackness is satisfied. For the edges in H_i, the PQ-Edmonds preserve complementary slackness by Lemma 16. For the edge (u, v) where u and v does not belong to any H_i, the duals does not change. For the edge (u, v) where $u \in H_i$ and v does not belong to any H_j, we have $\mu(u, v) \leq 0$, $\mu(u, v) \leq 2y(v)$ and $y(u) \geq 0$ by Lemma 19. The complementary slackness is maintained after the step of augmentation within small blossoms. Since complementary slackness is stronger than approximate complementary slackness and the EdmondsSearch algorithm preserves approximate complementary slackness by Lemma 15, approximate complementary slackness is maintained at the end of each iteration.

Let μ, μ' be the edge weights respectively before and after the step of weight adjustment on line 30-32. As y, z, μ, Ω satisfy approximate complementary slackness, y, z, μ', Ω satisfy complementary slackness, we know for each edge e, $\mu(e) - \mu'(e) \in [0, 2]$. Since PQ-Edmonds algorithm preserves complementary slackness by Lemma 16, complementary slackness is maintained with respect to edge weights μ' after Algorithm 5. Now, again by $\mu(e) - \mu'(e) \in [0, 2], \forall e \in \mathcal{E}$, we know, y, z, F, Ω still satisfy approximate complementary slackness with respect to μ after Algorithm 5 is completed.

After the step of PQ-deficiency reduction, the total deficiency becomes zero. Then, according to Lemma 11, $\mu(F^*) - \mu(F) \leq f(\mathcal{V})$. Since for every edge $e \in \mathcal{E}$, $\mu(e)$ is an integral multiple of $2f(\mathcal{V})$, therefore it must be $\mu(F) = \mu(F^*)$. Hence F is a maximum weight perfect f-factor of $\mathcal{G}$. ◀

4 Running Time Analysis

Recall that an alternating walk on $\mathcal{G}$ is a sequence of edges $(u_1, u_2), (u_2, u_3), \cdots, (u_{l-1}, u_l)$ such that: (1) $(u_i, u_{i+1}) \in \mathcal{E}$ are different edges; (2) exactly one of $(u_{i-1}, u_i), (u_i, u_{i+1})$ belongs to F, $1 < i < l$. And this walk is called an augmenting walk if both $(u_1, u_2), (u_{l-1}, u_l) \notin F$.

▶ **Lemma 21.** *The running time of each iteration is $\tilde{O}(mn^{2/3})$, thus $\tilde{O}(mn^{2/3} \log W)$ for all iterations.*

Proof. We analyze the running time of the t-th iteration, where $t \geq 1$. Clearly the scaling step and the blossom dissolution step only take linear time. By Lemma 15, the deficiency reduction step takes $\tilde{O}(mn^{2/3})$ in total. So the only technical part is the running time of the augmentations within small blossoms.

For each small blossom B_i, $|B_i \cap V|$, i.e. the number of original vertices in B_i, is less than $n^{1/3}$. According to the properties of the blowup graph, the number of vertices in B_i is $O(n^{2/3})$. When we add edges in $I_{F_0}(B_i) \setminus \{\eta(B_i)\}$ to F, the overall deficiency of vertices in

B_i is at most $1 + 3\binom{|B_i \cap V|}{2} < 1.5n^{2/3}$. After one execution of the PQ-Edmonds algorithm, the overall deficiency is reduced by one, or the largest y value of unsaturated vertices in B_i is equal to Y_2; the former case could happen at most $1.5n^{2/3}$ times, while the latter case could happen at most $|B_i| = O(n^{2/3})$ times since every time this case happens we add at least one more unsaturated vertex to U. Thus the PQ-Edmonds algorithm is invoked for at most $O(n^{2/3})$ times. By Lemma 16, for each small blossom B_i, each instance of the PQ-Edmonds algorithm takes $O(m(B_i) \log n)$ time, where $m(B_i)$ denotes the number of edges in B_i. Thus, the total running time of the augmentations within small blossoms is $\tilde{O}(mn^{2/3})$. ◀

Now we only need to analyze the running time for the PQ-deficiency reduction step at the end of the Scaling algorithm 5.

▶ **Lemma 22.** *Let F_t denote the f-factor at the end of the t-th scaling iteration. For any $t \geq 1$, $F_{t-1} \oplus F_t$ contains at most $O(n^{2/3})$ edge-disjoint augmenting walks in $\mathcal{G}$, where augmenting walks are w.r.t. F_t. (For the F_1, we can imagine an arbitrary perfect f-factor F_0, but do not need to compute it explicitly.)*

Then we can see the total deficiency of F_t is at most $O(n^{2/3}t)$ for any $t \geq 1$, and the overall running time of our algorithm is bounded by $\tilde{O}(mn^{2/3} \log W)$ by Lemma 16.

The rest of this subsection is devoted to the proof of Lemma 22 in the t-th iteration. With a slight abuse of notations, let $F_0 = F_{t-1}$ and $F = F_t$ and when talking about augmenting walks, we always mean augmenting walks in $F_0 \oplus F$ w.r.t. F. (F-edges are considered as matching edges and F_0-edges are considered as non-matching edges.) Let $\bar{\mu}_{\text{old}}, y_{\text{old}}, z_{\text{old}}, \Omega_{\text{old}}$ denote the edge weights, duals, and blossoms at the beginning of the blossom dissolution step, respectively; and let $\Omega_{\text{old}}^{\text{large}}$ be the set of all blossoms in Ω_{old} that were dissolved in the blossom dissolution phase before the reweighting step. Similarly, $\bar{\mu}, y, z, \Omega$ denote the edge weights, duals, and blossoms at the end of the t-th iteration, respectively; and Ω^{large} denotes the set of all large blossoms in Ω.

Instead of directly working with duals y, define variables $\widehat{y}$ for vertices as follows:

$$\widehat{y}(u) = y(u) + \frac{1}{2} \sum_{\substack{X \in \Omega^{\text{large}} \text{ s.t.} \\ \exists (u,v) \in \gamma(X) \cup I(X)}} z(X)$$

When $\eta(B) \in I(B)$, $\eta(B)$ is not a matching edge. Then $\zeta(B)$ may be a matching edge and its yz-value will increase by $z(B)/2$ after the dissolution of blossom B. (Recall that $\zeta(B) = (e_u, e_v)$, if $(u, e_u) = \eta(B)$.)

Consider any subwalk $\rho = \langle u_1, u_2, \cdots, u_{2s+1} \rangle$ of an augmenting walk in $F_0 \oplus F$ starting with an edge not in F. Then, for $1 \leq i \leq s$, since $(u_{2i-1}, u_{2i}) \notin F$ and $(u_{2i}, u_{2i+1}) \in F$, by approximate complementary slackness we have

$$y(u_{2i-1}) + y(u_{2i}) + \sum_{\substack{X \in \Omega \\ (u_{2i-1}, u_{2i}) \in \gamma(X) \cup I(X)}} z(X) \quad \geq \quad \bar{\mu}(u_{2i-1}, u_{2i}) - 2 \tag{1}$$

$$y(u_{2i}) + y(u_{2i+1}) + \sum_{\substack{X \in \Omega \\ (u_{2i}, u_{2i+1}) \in \gamma(X) \cup I(X)}} z(X) \quad \leq \quad \bar{\mu}(u_{2i}, u_{2i+1}) \tag{2}$$

Plugging in the definition of $\widehat{y}$, we get:

$$\widehat{y}(u_{2i-1}) + \widehat{y}(u_{2i}) + \sum_{\substack{X \in \Omega \setminus \Omega^{\text{large}} \\ (u_{2i-1}, u_{2i}) \in \gamma(X) \cup I(X)}} z(X) \geq \bar{\mu}(u_{2i-1}, u_{2i}) - 2 \tag{3}$$

and since $(u_{2i}, u_{2i+1}) \in F$, we have:

$$\widehat{y}(u_{2i}) + \widehat{y}(u_{2i+1}) = y(u_{2i}) + y(u_{2i+1}) + \sum_{\substack{X\in\Omega^{\text{large}}\\(u_{2i},u_{2i+1})\in\gamma(X)\cup I(X)}} z(X) + \frac{1}{2}\sum_{\substack{X\in\Omega^{\text{large}}\\(u_{2i},u_{2i+1})\in\{\eta(X),\zeta(X)\}}} z(X)$$

$$\widehat{y}(u_{2i}) + \widehat{y}(u_{2i+1}) + \sum_{\substack{X\in\Omega\setminus\Omega^{\text{large}}\\(u_{2i},u_{2i+1})\in\gamma(X)\cup I(X)}} z(X) \le \bar{\mu}(u_{2i}, u_{2i+1}) + \frac{1}{2}\sum_{\substack{X\in\Omega^{\text{large}}\\(u_{2i},u_{2i+1})\in\{\eta(X),\zeta(X)\}}} z(X)$$

Taking a subtraction we have

$$\begin{aligned}\widehat{y}(u_{2i+1}) - \widehat{y}(u_{2i-1}) \le 2 + \bar{\mu}(u_{2i}, u_{2i+1}) - \bar{\mu}(u_{2i-1}, u_{2i}) + \frac{1}{2}\sum_{\substack{X\in\Omega^{\text{large}}\\(u_{2i},u_{2i+1})\in\{\eta(X),\zeta(X)\}}} z(X)\\ + \sum_{\substack{X\in\Omega\setminus\Omega^{\text{large}}\\(u_{2i-1},u_{2i})\in\gamma(X)\cup I(X)}} z(X) - \sum_{\substack{X\in\Omega\setminus\Omega^{\text{large}}\\(u_{2i},u_{2i+1})\in\gamma(X)\cup I(X)}} z(X)\end{aligned} \tag{4}$$

By the derivation presented in Lemma 19,

$$\begin{aligned}\bar{\mu}(u_{2i}, u_{2i+1}) \le yz_{\text{old}}(u_{2i}, u_{2i+1}) - y_{\text{old}}(u_{2i}) - y_{\text{old}}(u_{2i+1}) - \sum_{\substack{X\in\Omega^{\text{large}}_{\text{old}}\\(u_{2i},u_{2i+1})\in\gamma(X)\cup I(X)}} z_{\text{old}}(X)\\ = \sum_{\substack{X\in\Omega_{\text{old}}\setminus\Omega^{\text{large}}_{\text{old}}\\(u_{2i},u_{2i+1})\in\gamma(X)\cup I(X)}} z_{\text{old}}(X)\end{aligned} \tag{5}$$

Now, as $(u_{2i-1}, u_{2i}) \in F_0$, again by Lemma 18 we are able to prove:

$$\bar{\mu}(u_{2i-1}, u_{2i}) \ge -6 + \sum_{\substack{X\in\Omega_{\text{old}}\setminus\Omega^{\text{large}}_{\text{old}}\\(u_{2i-1},u_{2i})\in\gamma(X)\cup I(X)}} z_{\text{old}}(X) - \frac{1}{2}\sum_{\substack{X\in\Omega^{\text{large}}_{\text{old}}\\(u_{2i-1},u_{2i})\in\{\eta(X),\zeta(X)\}}} z_{\text{old}}(X) \tag{6}$$

So for (u_{2i-1}, u_{2i}) not an η-edge or ζ-edge for old large blossoms,

$$\bar{\mu}(u_{2i-1}, u_{2i}) \ge -6 \tag{7}$$

Also define the function Z for an augmenting walk ρ as:

$$\mathrm{Z}(\rho) = \frac{1}{2}\sum_{\substack{e\in\rho, X\in\Omega\\e\in\{\eta(X),\zeta(X)\}}} z(X) + \frac{1}{2}\sum_{\substack{e\in\rho, X\in\Omega_{\text{old}}\\e\in\{\eta(X),\zeta(X)\}}} z_{\text{old}}(X)$$

Let $\widehat{F}, \widehat{\Omega}, \widehat{z}$ denote any f-factor together with a compatible set of blossoms as well as their duals, and let ρ be an arbitrary alternating walk. For any blossom $X \in \widehat{\Omega}$, define the following quantity:

$$\mathrm{Diff}(\rho, X, \widehat{F}) \overset{\text{def}}{=} |\rho \cap \widehat{F} \cap (\gamma(X) \cup I(X))| - |\rho \cap (\gamma(X) \cup I(X)) \setminus \widehat{F}|$$

By a summation of (4) plugging in (5) and (6) over all $1 \le i \le s$,

$$\begin{aligned}\widehat{y}(u_{2s+1}) - \widehat{y}(u_1) \le & 8s + \mathrm{Z}(\rho) - \sum_{X\in\Omega\setminus\Omega^{\text{large}}} z(X)\cdot \mathrm{Diff}(\rho, X, F)\\ & - \sum_{X\in\Omega_{\text{old}}\setminus\Omega^{\text{large}}_{\text{old}}} z_{\text{old}}(X)\cdot \mathrm{Diff}(\rho, X, F_0)\end{aligned} \tag{8}$$

We consider augmenting walks $\rho = \langle u_1, u_2, \cdots, u_{2s}\rangle$ in $F_0 \oplus F$ starting and ending with edges not equal to $\eta(X)$ or $\zeta(X)$ for all $X \in \Omega^{\text{large}} \cup \Omega^{\text{large}}_{\text{old}}$, and u_1, u_{2s} are not equal to $\beta(X)$ for all $X \in \Omega^{\text{large}} \cup \Omega^{\text{large}}_{\text{old}}$. This only excludes $O(n^{2/3})$ augmenting walks. Thus, $\widehat{y}(u_1) = \widehat{y}(u_{2s}) = -Cn^{2/3}$. For convenience, we only consider augmenting walks starting and ending with non-matching edges ($\notin F$) not in $\gamma(X) \cup I(X)$ for current small blossoms X, otherwise we can just choose the first and last such edges on every augmenting walk and consider the subwalk between them, and we can get similar results.

In an augmenting walk satisfying those conditions, since its first and last edges are in F_0 and not equal to η or ζ edges of old large blossoms, $\bar{\mu}(u_{2s-1}, u_{2s}) \geq -6$, and also (u_{2s-1}, u_{2s}) cannot be in a large blossom, by (1) we have $\widehat{y}(u_{2s-1}) \geq -\widehat{y}(u_{2s}) - 8 = Cn^{2/3} - 8$. Then by (8), if $\mathrm{Z}(\rho)$ is less than $n^{2/3}$, and $\mathrm{Diff}(\rho, X, F)$ and $\mathrm{Diff}(\rho, X, F_0)$ are non-negative, then we can see the length of such an augmenting walk is $\Omega(n^{2/3})$ when C is a large constant. We have the following statement for $\mathrm{Diff}(\rho, X, F)$ and $\mathrm{Diff}(\rho, X, F_0)$, which is proven in the full version.

▶ **Lemma 23.** *Consider any blossom $X \in \Omega \cup \Omega_{old}$ and any augmenting walk ρ in $F_0 \oplus F$, in any maximal consecutive subwalk of $\rho \cap (\gamma(X) \cup I(X))$, the number of matching edges is at least the number of non-matching edges. (For $X \in \Omega$, edges in F are matching edges, and for $X \in \Omega_{old}$, edges in F_0 are matching edges.)*

So $\mathrm{Diff}(\rho, X, F)$ and $\mathrm{Diff}(\rho, X, F_0)$ are both nonnegative. It is also not hard to see the following observations:

- For old and new large blossoms, $\sum_{B \in \Omega^{\text{large}}_{\text{old}}} z_{\text{old}}(B)$ and $\sum_{B \in \Omega^{\text{large}}} z(B)$ are both $O(n^{4/3})$. This is because the number of root large blossoms at a given time are bounded by $O(n^{2/3})$ and large blossoms can only be formed and dual-adjusted in the $O(n^{2/3})$ EdmondsSearch steps in the deficiency reduction step.
- The total number of non-matching edges in $\gamma(B) \cup I(B)$ for all small blossoms B is bounded by $O(n^{4/3})$, that is, $\sum_{B \in \Omega \setminus \Omega^{\text{large}}} |(\gamma(B) \cup I(B)) \setminus F|$ and $\sum_{B \in \Omega_{\text{old}} \setminus \Omega^{\text{large}}_{\text{old}}} |(\gamma(B) \cup I(B)) \setminus F_0|$ are both $O(n^{4/3})$. This is because the number of edges in $\gamma(B)$ for every blossom B is bounded by $O(|B|^2)$, the size of root small blossoms is less than $n^{1/3}$, and every root blossom has at most one non-matching edge in $I(B) \setminus \gamma(B)$,

Therefore, if we assume the number of augmenting walks in $F_0 \oplus F$ is larger than $K \cdot n^{2/3}$ for a large constant K, then we still have $\Omega(n^{2/3})$ augmenting walks ρ satisfying the following conditions:

(a) Starting and ending with edges not equal to η-edge or ζ-edge for all old large blossoms.
(b) Starting and ending with vertices not equal to the base for all old large blossoms.
(c) $\mathrm{Z}(\rho) < n^{2/3}$
(d) The total number of non-matching edges on ρ in $\gamma(X) \cup I(X)$ for all old and new small blossoms X is less than $n^{2/3}$. As in Lemma 23, the number of maximal consecutive subwalks in $\rho \cap (\gamma(X) \cup I(X))$ containing those edges is also bounded by $n^{2/3}$. We call all the edges in such subwalks “skip edges”.

Among vertices $u_1, u_2, \cdots, u_{2s-1}$, we pick the vertices with odd subscript which are original vertices in G, plus the two endpoints, and obtain the list: $u_1 (= v_1), u_p (= v_2), u_{p+6} (= v_3), \cdots, u_{p+6q} (= v_{q+2}), u_{2s-1} (= v_{q+3})$, where p is 3,5 or 7. Consider all the differences $\widehat{y}(v_{i+1}) - \widehat{y}(v_i)$ for $i = 1, \cdots, q+2$. For the subwalk $[v_i, v_{i+1}]$ in ρ containing skip edges or the η or ζ-edges of old or new large blossoms, the sum of the differences $\widehat{y}(v_{i+1}) - \widehat{y}(v_i)$ is at most $O(n^{2/3})$, by (c),(d) and Lemma 23. For subwalk $[v_i, v_{i+1}]$ in ρ which does not contain skip edges or η or ζ-edges, from (4), $\widehat{y}(u_{2j+1}) - \widehat{y}(u_{2j-1}) \leq 8$ for (u_{2j+1}, u_{2j-1}) in the subwalk, so $\widehat{y}(v_{i+1}) - \widehat{y}(v_i) \le 24$, Since the sum of all $\widehat{y}(v_{i+1}) - \widehat{y}(v_i)$ is $\widehat{y}(u_{2s-1}) - \widehat{y}(u_1)$, which is at least $2Cn^{2/3}$, if C is a large constant, the number of subwalks $[v_i, v_{i+1}]$ in ρ which does not contain skip edges or η, ζ-edges and satisfies $0 < \widehat{y}(v_{i+1}) - \widehat{y}(v_i) \le 24$ is $\Omega(n^{2/3})$. Moreover, there are $\Omega(n^{2/3})$ different values of $\widehat{y}(v_i)$ in ρ. When v_i is not u_1 and v_{i+1} is not u_{2s-1}, then $v_i = u_{p+6k}$ and $v_{i+1} = u_{p+6k+6}$. Then we can see $\widehat{y}(u_{p+6k+2}) - \widehat{y}(u_{p+6k})$, $\widehat{y}(u_{p+6k+4}) - \widehat{y}(u_{p+6k+2})$ and $\widehat{y}(u_{p+6k+6}) - \widehat{y}(u_{p+6k+4})$ are at most 8, and also by (1),(2),(7) and Lemma 19, (remember p is odd)

$$\widehat{y}(u_{p+6k+3}) + \widehat{y}(u_{p+6k+4}) \le 0$$
$$\widehat{y}(u_{p+6k+2}) + \widehat{y}(u_{p+6k+3}) \ge -8$$

So we have $\widehat{y}(u_{p+6k+3}) \ge -\widehat{y}(u_{p+6k}) - 16$ and $\widehat{y}(u_{p+6k+3}) \le -\widehat{y}(u_{p+6k+6}) + 8 < -\widehat{y}(u_{p+6k}) + 8$. Note that u_{p+6k} and u_{p+6k+3} are original vertices in G, and their $\widehat{y}$-values are $\ge -Cn^{2/3}$, so their $\widehat{y}$-values are also $< Cn^{2/3} + 8$.

Given an interval $[a, b]$ of integers, define $V_{[a,b]} = \{v \in V | \widehat{y}(v) \in [a, b]\}$. For an edge $(u, v) \in E$ such that $u \in V_{[a,b]}$ and $v \in V_{[a',b']}$, we say the auxiliary edges $(u, e_u), (e_u, e_v), (e_v, v)$ are "between" the pair of intervals $[a, b]$ and $[a', b']$. If we divide the original vertices in G by their $\widehat{y}$-values into intervals of length 48, then every augmenting walk we consider will go through auxiliary edges between $\Omega(n^{2/3})$ pairs of intervals of the form $[a, a+48], [-a, -a+48]$. Any constant fraction of those $\Theta(n^{2/3})$ pairs of intervals contains $O(n^{1/3})$ vertices each, so there are at most $O(n^{2/3})$ auxiliary edges between any of such pair of intervals. Thus the number of augmenting walks satisfying (a),(b),(c),(d) is bounded by $O(n^{2/3})$.

References

1 Michael B Cohen, Aleksander Madry, Piotr Sankowski, and Adrian Vladu. Negative-weight shortest paths and unit capacity minimum cost flow in $\tilde{O}(m^{10/7} \log w)$. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 752–771. SIAM, 2017.

2 Ran Duan and Seth Pettie. Linear-time approximation for maximum weight matching. *Journal of the ACM (JACM)*, 61(1):1, 2014.

3 Ran Duan, Seth Pettie, and Hsin-Hao Su. Scaling algorithms for weighted matching in general graphs. *ACM Transactions on Algorithms (TALG)*, 14(1):8, 2018.

4 Jack Edmonds and Richard M Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM (JACM)*, 19(2):248–264, 1972.

5 Shimon Even and R Endre Tarjan. Network flow and testing graph connectivity. *SIAM journal on computing*, 4(4):507–518, 1975.

6 Harold N Gabow. An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems. In *Proceedings of the fifteenth annual ACM symposium on Theory of computing*, pages 448–456. ACM, 1983.

7 Harold N Gabow. Scaling algorithms for network problems. In *24th Annual Symposium on Foundations of Computer Science (SFCS 1983)*, pages 248–258. IEEE, 1983.

8 Harold N Gabow. Data structures for weighted matching and extensions to b-matching and f-factors. *ACM Transactions on Algorithms (TALG)*, 14(3):39, 2018.

9 Harold N Gabow and Piotr Sankowski. Algebraic algorithms for b-matching, shortest undirected paths, and f-factors. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 137–146. IEEE, 2013.

10 Harold N Gabow and Robert E Tarjan. Faster scaling algorithms for network problems. *SIAM Journal on Computing*, 18(5):1013–1036, 1989.

11 Andrew Goldberg and Robert Tarjan. Solving minimum-cost flow problems by successive approximation. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 7–18. ACM, 1987.

12 Andrew V Goldberg and Satish Rao. Beyond the flow decomposition barrier. *Journal of the ACM (JACM)*, 45(5):783–797, 1998.

13 Dawei Huang and Seth Pettie. Approximate generalized matching: f-factors and f-edge covers. *arXiv preprint*, 2017. arXiv:1706.05761.

14 Alexander V Karzanov. On finding maximum flows in networks with special structure and some applications. *Matematicheskie Voprosy Upravleniya Proizvodstvom*, 5:81–94, 1973.

15 Yin Tat Lee and Aaron Sidford. Path finding methods for linear programming: Solving linear programs in $\tilde{O}(\sqrt{rank})$ iterations and faster algorithms for maximum flow. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 424–433. IEEE, 2014.

16 A. Schrijver. *Combinatorial Optimization - Polyhedra and Efficiency*. Springer, 2003.

The Outer Limits of Contention Resolution on Matroids and Connections to the Secretary Problem

Shaddin Dughmi
Department of Computer Science, University of Southern California, Los Angeles, CA, USA
shaddin@usc.edu

Abstract

Contention resolution schemes have proven to be a useful and unifying abstraction for a variety of constrained optimization problems, in both offline and online arrival models. Much of prior work restricts attention to product distributions for the input set of elements, and studies contention resolution for increasingly general packing constraints, both offline and online. In this paper, we instead focus on generalizing the input distribution, restricting attention to matroid constraints in both the offline and online random arrival models. In particular, we study contention resolution when the input set is arbitrarily distributed, and may exhibit positive and/or negative correlations between elements. We characterize the distributions for which offline contention resolution is possible, and establish some of their basic closure properties. Our characterization can be interpreted as a distributional generalization of the matroid covering theorem. For the online random arrival model, we show that contention resolution is intimately tied to the secretary problem via two results. First, we show that a competitive algorithm for the matroid secretary problem implies that online contention resolution is essentially as powerful as offline contention resolution for matroids, so long as the algorithm is given the input distribution. Second, we reduce the matroid secretary problem to the design of an online contention resolution scheme of a particular form.

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

Keywords and phrases Contention Resolution, Secretary Problems, Matroids

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.42

Category Track A: Algorithms, Complexity and Games

Related Version `https://arxiv.org/abs/1909.04268`

Funding *Shaddin Dughmi*: This work was supported by NSF CAREER Award CCF-1350900.

1 Introduction

The notion of a contention resolution scheme (CRS) abstracts a familiar task in constrained optimization: converting a (random) set-valued solution which is *ex-ante* (i.e., on average) feasible for a packing problem to one which is *ex-post* (i.e., always) feasible. Unlike randomized rounding algorithms more broadly, which in general may be catered to both the constraint and objective function at hand, a contention resolution scheme is specific only to the constraints of the problem, and preserves solution quality in a manner which is largely agnostic to the objective function[1] – element by element. Since they were formalized by Chekuri et al [10], CRSs have been connected to a variety of online and offline computational tasks, including rounding the solutions of mathematical programs [10], online mechanism design and stochastic probing [17, 1], and prophet inequalities [17, 25].

[1] In its most general form, a CRS approximately preserves all linear objective functions simultaneously, whereas a *monotone* CRS approximately preserves all submodular objectives [10].

Starting with [10], prior work defines an (offline) contention resolution scheme for a set system $(\mathcal{E}, \mathcal{I})$ – where $\mathcal{E}$ is a *ground set* of *elements* and $\mathcal{I} \subseteq 2^{\mathcal{E}}$ is a downwards-closed family of *feasible sets* – as an algorithm which takes as input the marginal probabilities $x \in [0,1]^{\mathcal{E}}$ of a product distribution $\mathcal{D}$ supported on $2^{\mathcal{E}}$ as well as a random set $R \sim \mathcal{D}$ of *active elements*, and must output a feasible subset S of R. The contention resolution scheme is α-competitive if $\mathbf{Pr}[i \in S] \geq \frac{1}{\alpha} \mathbf{Pr}[i \in R]$ holds for all product distributions of interest – typically those with marginals x in the convex hull of indicator vectors of $\mathcal{I}$ (ex-ante feasibility). In *online* contention resolution schemes, first explored by Feldman et al [17] and subsequently by Adamczyk and Włodarczyk [1] and Lee and Singla [25], the active elements R arrive sequentially and the decision to include an element in S must irrevocably be made online.

The existing literature on (offline and online) contention resolution has mostly restricted attention to ex-ante-feasible and given product distributions, and varied the set system (e.g. matroids, knapsacks, and their intersections), all the while drawing connections to applications such as stochastic online problems, approximation algorithms, mechanism design, and prophet inequalities. In this paper, we restrict our attention to matroid constraints,[2] and instead focus on generalizing the class of input distributions. Our main goal is to understand the power and limitations of contention resolution, offline and online, in the presence of correlations in the input distribution and without regard to ex-ante feasibility. A secondary goal is to understand how knowledge of the distribution influences contention resolution. In pursuit of both goals, we draw connections between contention resolution and the secretary problem on matroids, shedding light on challenges posed by the matroid secretary conjecture in the process.

Results

Our first set of results develops an understanding of offline contention resolution on matroids. We begin with a characterization of the class of *α-uncontentious distributions*: those distributions $\mathcal{D} \in \Delta(2^{\mathcal{E}})$ permitting α-competitive offline contention resolution for a given matroid. Most notably, we show that a distribution is α-uncontentious if and only if it satisfies a family of $2^{|\mathcal{E}|}$ inequalities, one for each subset of the ground set. Moreover, we observe that our inequality characterization is the natural generalization of the *matroid base covering theorem* (see e.g. [31]) from covering a set of elements to covering a distribution over sets of elements. In other words, we show that *contention resolution is the natural distributional generalization of base covering.* Leveraging our characterization, we establish some basic closure properties of the class of uncontentious distributions, and present some examples of uncontentious distributions exhibiting negative and positive correlation between elements. Finally, we examine whether knowledge of the distribution $\mathcal{D}$ is essential to contention resolution, and exhibit an impossibility result: any contention resolution scheme which has nontrivial guarantees for all α-uncontentious distributions cannot be *prior-independent*, in that it cannot make do with a finite number of samples from the distribution, even for very simple matroids.

Our second set of results concerns online contention resolution on matroids in the random arrival model, and in particular its connection to the matroid secretary problem. First, we show that a competitive secretary algorithm for a matroid implies that online contention resolution is essentially as powerful as offline contention resolution for that same matroid: a γ-competitive secretary algorithm implies that any α-uncontentious distribution permits $\gamma\alpha$-competitive online contention resolution.

[2] Though some of our results hold beyond matroids; we discuss those in the conclusion section.

Second, we provide evidence that contention resolution might hold the key to resolving the matroid secretary conjecture. As our most technically-involved result, we leverage our characterization of uncontentious distributions to show that the random set of *improving elements* in a weighted matroid – as originally defined by Karger [20] – is $O(1)$-uncontentious. Since the improving elements can be recognized online, and moreover hold a constant fraction of the weighted rank of the matroid in expectation, our result can be loosely interpreted as a reduction from the matroid secretary problem to online contention resolution for a particular uncontentious distribution. There is one major caveat to this interpretation of our result, however: not only does the set of active (improving) elements arrive online, but so does the description of the uncontentious distribution from which that set is drawn. Though we present our proof of this result in an elementary form, the underlying arguments are reminiscent of – and inspired by – those often encountered in the analysis of martingales: we condition on carefully-chosen random variables, and employ a delicate charging argument between different probability events.

Third, in response to feedback on the previous version of this manuscript, we show that our aforementioned result – that improving elements are uncontentious – cannot be derived as a consequence of prior work.

Additional Discussion of Related Work

Contention Resolution Schemes

Contention resolution schemes were introduced by Chekuri et al [10], motivated by the problem of maximizing a submodular function subject to packing constraints. In particular, offline CRS were used to transform a randomized rounding algorithm which respects the packing constraints ex-ante to one which respects them ex-post, at the cost of the competitive ratio of the CRS. Their focus – like that of all related work prior to ours – was on product input distributions, in which case the optimal competitive ratio of an offline CRS was shown to equal the worst-case *correlation gap* (first studied by [2, 7]) of the weighted rank function associated with the packing constraint. The characterization result of [10] result forms the basis for ours.

Online contention resolution was first studied by Feldman et al [17], and applied to a number of online selection problems. They show that simple packing constraints – such as matroids, knapsacks, and matchings – permit constant competitive online contention resolution schemes even when elements arrive in an unknown and adversarial order. Moreover, they show how to combine competitive online schemes for different constraints in order to yield competitive online schemes for their intersection. Lee and Singla [25] obtain optimal online CRS in both the known adversarial-order model as well as the random-arrival model. Adamczyk and Włodarczyk [1] restrict attention to the random-arrival model, and obtain a particularly elegant algorithm and argument based on martingales, as well as improved competitive ratios for intersections of matroids and knapsacks.

Prophet Inequalities

Contention resolution is intimately tied to *prophet inequality* problems, also known as *Bayesian online selection* problems. In the traditional model for these problems, independent real-valued random variables with known distributions arrive online in a known but adversarial order, and the goal is to select a subset of the variables with maximum sum, subject to a packing constraint. An α-competitive algorithm for a Bayesian online selection problem is also referred to as a prophet inequality with ratio α, for historical reasons. Krengel, Sucheston, and Garling [22, 23] proved the first (classical) single-choice prophet inequality with ratio 1/2 for selecting a single variable (i.e., a 1-uniform matroid packing constraint). Motivated by applications in algorithmic mechanism design, more recent work (e.g. [18, 3, 8, 32]) pursued prophet inequalities for more general packing constraints. Of particular note is the work of Kleinberg and Weinberg [21], who proved an optimal prophet inequality with ratio 1/2 for matroids. Also notable is polylogarithmic prophet inequality for general packing constraints due to Rubinstein [28]. The (easier) variant of Bayesian online selection problems in which the variables arrive in a uniformly random order has also received recent interest, resulting in improved prophet inequalities for various packing constraints [15, 14, 5].

It was shown by Feldman et al [17] that an online CRS yields a prophet inequality with the same competitive ratio, and in the same arrival model. A weak converse is also true, as shown by Lee and Singla [25]: a stronger form of prophet inequality – in particular one which competes against the *ex-ante relaxation* of the Bayesian online selection problem – yields an online CRS with the same competitive ratio and in the same arrival model.

Beyond Known Product Distributions

The vast majority of work on contention resolution or prophet inequalities, and all such work discussed thus far, restricts attention to known product distributions, and crucially exploits the product structure and knowledge of the distribution. We note the few exceptions next.

Rinott et al [27] and Samuel-Cahn [29] show that the single-choice prophet inequality, and some slight generalizations, continue to hold for negatively dependent random variables. It is known [19] that there is no single-choice prophet inequality with ratio better than the number of variables in the presence of arbitrary positive correlation. Moreover, we are unaware of any nontrivial positive results for a class of distributions exhibiting positive correlation, in either prophet inequality or contention resolution models. We note that whereas [25] and [1] use specially-crafted correlated distributions as benchmarks, their results and techniques do not appear to shed light on contention resolution or prophet inequalities in the presence of correlation more generally.

Some work has relaxed the requirement that the distributions be known in prophet inequality problems. Azar et al [4] study prophet inequality problems when only a single sample is given from each distribution, and obtain constant competitive ratios for a variety of constraints. Wang [30] obtains an optimal algorithm for the single-choice prophet inequality, with ratio 1/2, in the same single-sample model. Correa et al [11] study the single-choice prophet inequality with i.i.d. variables drawn from an unknown distribution, and characterize the relationship between the competitive ratio and the number of samples available from the distribution.

Secretary Problems

In a *generalized secretary problem*, a set of adversarially chosen variables arrive online in a random order, and the goal is to select a subset of the variables with maximum sum subject to a packing constraint. The (classical) single-choice *secretary problem*, corresponding to a 1-uniform matroid constraint, was introduced and solved by Dynkin [13]. The *matroid secretary problem* was introduced by Babaioff et al [6], and has since spawned a long line of work. Constant-competitive algorithms have been discovered for most natural matroids and for some alternative models – see [12] for a semi-recent survey – though the general conjecture remains open. The state of the art for the general matroid secretary problem is a $O(\log\log \mathbf{rank})$-competitive algorithm due to Lachish [24], which was henceforth simplified by Feldman et al [16].

2 Preliminaries

2.1 Matroid Theory Basics

We use standard definitions from matroid theory; for details see [26, 31]. A matroid $\mathcal{M} = (\mathcal{E}, \mathcal{I})$ consists of a *ground set* $\mathcal{E}$ of *elements*, and a family $\mathcal{I} \subseteq 2^{\mathcal{E}}$ of *independent sets*, satisfying the three *matroid axioms*. A *weighted matroid* $(\mathcal{M}, w)$ consists of a matroid $\mathcal{M} = (\mathcal{E}, \mathcal{I})$ together with weights $w \in \mathbb{R}^{\mathcal{E}}$ on the elements. We use the standard notions of a *dependent set*, *circuit*, *flat*, and *minor* in a matroid. We denote the *rank* of a matroid $\mathcal{M}$ as $\mathbf{rank}(\mathcal{M})$, and the rank of a set of elements A in $\mathcal{M}$ as $\mathbf{rank}_{\mathcal{M}}(A)$, or $\mathbf{rank}(A)$ when $\mathcal{M}$ is clear from context. Overloading notation, we use $\mathbf{rank}_w^{\mathcal{M}}(A)$ to denote the *weighted rank* of a set A – the maximum weight of an independent subset of A – in the weighted matroid $(\mathcal{M}, w)$, though we omit the superscript $\mathcal{M}$ when the matroid is clear from context. We note that both rank and weighted rank are submodular set functions on the ground set of the matroid. For $\mathcal{M} = (\mathcal{E}, \mathcal{I})$ and $A \subseteq \mathcal{E}$, we denote the *restriction* of $\mathcal{M}$ to A as $\mathcal{M}|A$, *deletion* of A as $\mathcal{M} \setminus A$, and *contraction* by A as $\mathcal{M}/A$.

When $\mathcal{E}$ is clear from context, and $S \subseteq \mathcal{E}$, we use $\mathbf{1}_S \in \{0,1\}^{\mathcal{E}}$ to denote the vector indicating membership in S. We often reference the *matroid polytope* $\mathcal{P}(\mathcal{M})$ of a matroid $\mathcal{M} = (\mathcal{E}, \mathcal{I})$, defined as the convex hull of $\{\mathbf{1}_S : S \in \mathcal{I}\}$, or equivalently as the family of $x \in [0,1]^{\mathcal{E}}$ satisfying $\sum_{i \in S} x_i \leq \mathbf{rank}_{\mathcal{M}}(S)$ for all $S \subseteq \mathcal{E}$.

Throughout this paper we assume that any weighted matroid has distinct weights. This assumption is made merely to simplify some of our proofs, and – using standard tie-breaking arguments – can be shown to be without loss of generality in as much as our results are concerned. Under this assumption, we define $\mathbf{OPT}_w^{\mathcal{M}}(A)$ as the (unique) maximum-weight independent subset of A of minimum cardinality (excluding zero-weight elements), and we omit the superscript when the matroid is clear from context.

2.2 The Matroid Secretary Problem

In the *matroid secretary problem*, originally defined by [6] there is matroid $\mathcal{M} = (\mathcal{E}, \mathcal{I})$ with nonnegative weights $w : \mathcal{E} \to \mathbb{R}_+$ on the elements. Elements $\mathcal{E}$ arrive online in a uniformly random order Π, and an online algorithm must irrevocably accept or reject an element when it arrives, subject to accepting an independent set of $\mathcal{M}$. The algorithm is given $\mathcal{M}$ at the outset (as an independence oracle), but the weights w are chosen adversarially before the order Π is drawn and then are revealed online. The goal of the online algorithm is to maximize the expected weight of the accepted set of elements. We say that an algorithm is *α-competitive* for a class of matroids if for every matroid $\mathcal{M}$ in that class and every adversarial choice of w, the expected weight of the accepted set (over the random choice of Π and any internal randomness of the algorithm) is at least an α fraction of the maximum weight of an independent set of $(\mathcal{M}, w)$.

The *matroid secretary conjecture*, posed by [6], postulates that the matroid secretary problem admits an (online) algorithm which is constant-competitive for all matroids.

2.3 Miscellaneous Notation and Terminology

We denote the natural numbers by $\mathbb{N}$, and the nonnegative real numbers by $\mathbb{R}_+$. Given a set $\mathcal{A}$ with weights $w \in \mathbb{R}^{\mathcal{A}}$, and a subset $B \subseteq \mathcal{A}$, we use the shorthand $w(B) = \sum_{i \in B} w_i$. We use $[n]$ as shorthand for the set $1, \ldots, n$. For a set A, we use $\Delta(A)$ to denote the family of distributions over A, and 2^A to denote the family of subsets of A.

Let $\mathcal{A}$ be a finite ground set. For a distribution $\mathcal{D}$ supported on $2^{\mathcal{A}}$, we define the vector $x \in [0,1]^{\mathcal{A}}$ of *marginals* of $\mathcal{D}$ by $x_i = \mathbf{Pr}_{B\sim\mathcal{D}}[i \in B]$, and refer to x_i as the *marginal probability* of i in $\mathcal{D}$. When marginals $x \in [0,1]^{\mathcal{A}}$ are given, we use $\mathbf{Ind}(x)$ to denote the distribution of the random set $B \subseteq \mathcal{A}$ which includes each element $i \in \mathcal{A}$ independently with probability x_i. We also use $\mathbf{Ind}_p(\mathcal{A})$ as shorthand for $\mathbf{Ind}(x)$ when $x_i = p$ for all $i \in \mathcal{A}$.

3 Understanding Contention Resolution

3.1 The Basics of Contention Resolution

The definitions below are parametrized by a given matroid $\mathcal{M} = (\mathcal{E}, \mathcal{I})$.

▶ **Definition 3.1.** *A* contention resolution map (CRM) ϕ *is a randomized function from* $2^{\mathcal{E}}$ *to* $\mathcal{I}$ *with the property that* $\phi(R) \subseteq R$ *for all* $R \subseteq \mathcal{E}$*. Such a map is* α-competitive *for a distribution* $\mathcal{D} \in \Delta(2^{\mathcal{E}})$ *if, for* $R \sim \mathcal{D}$*, we have* $\mathbf{Pr}[i \in R] \leq \alpha\,\mathbf{Pr}[i \in \phi(R)]$ *for all* $i \in \mathcal{E}$*.*

The following is known from Chekuri et al [10].

▶ **Theorem 3.2** ([10]). *Every product distribution with marginals in* $\mathcal{P}(\mathcal{M})$ *admits an* $\frac{e}{e-1}$*-competitive CRM.*

▶ **Definition 3.3.** *An* online random-order contention resolution map *(henceforth* online CRM *for short) is a contention resolution map* ϕ *which can be implemented as an algorithm in the online random-arrival model. In the online random-arrival model,* $\mathcal{E}$ *is presented to the algorithm in a uniformly random order* $(e_1, \ldots, e_n)$*, and at the ith step the algorithm learns whether* e_i *is* active *– i.e.,* $e_i \in R \sim \mathcal{D}$ *– and if so must make an irrevocable decision on whether to include* e_i *in* $\phi(R)$*.*

The following is known from Lee and Singla [25].

▶ **Theorem 3.4** ([25]). *Every product distribution with marginals in* $\mathcal{P}(\mathcal{M})$ *admits a* $\frac{e}{e-1}$*-competitive online CRM.*

3.2 Uncontentious Distributions and their Characterization

As shorthand, we refer to distributions which permit competitive (offline) CRMs as *uncontentious.*

▶ **Definition 3.5.** *Fix a matroid* $\mathcal{M} = (\mathcal{E}, \mathcal{I})$*. For* $\alpha \geq 1$*, we say that a distribution* $\mathcal{D} \in \Delta(2^{\mathcal{E}})$ *is* α-uncontentious *if it admits an* α*-competitive contention resolution map.*

For convenience, we also refer to a random set $R \sim \mathcal{D}$ as α-uncontentious if its distribution $\mathcal{D}$ is α-uncontentious. We prove the following characterization of uncontentious distributions.

▶ **Theorem 3.6.** *Fix a matroid* $\mathcal{M} = (\mathcal{E}, \mathcal{I})$*, and let* $\mathcal{D}$ *be a distribution supported on* $2^{\mathcal{E}}$*. The following are equivalent for every* $\alpha \geq 1$*.*

(a) *$\mathcal{D}$ is α-uncontentious (i.e., admits an α-competitive contention resolution map).*

(b) *For every weight vector $w \in \mathbb{R}_+^{\mathcal{E}}$, the following holds for $R \sim \mathcal{D}$:* $\mathbf{E}[\mathbf{rank}_w(R)] \geq \frac{\mathbf{E}[w(R)]}{\alpha}$.

(c) *For every $\mathcal{F} \subseteq \mathcal{E}$, the following holds for $R \sim \mathcal{D}$:* $\mathbf{E}[|R \cap \mathcal{F}|] \leq \alpha\,\mathbf{E}[\mathbf{rank}(R \cap \mathcal{F})]$.

Proof. Property (a) implies property (c) by applying an α-CRM ϕ to R, noting that $\phi(R) \cap \mathcal{F}$ is necessarily an independent subset of $R \cap \mathcal{F}$.

$$\begin{aligned}\mathbf{E}[\mathbf{rank}(R \cap \mathcal{F})] &\geq \mathbf{E}[|\phi(R) \cap \mathcal{F}|] \\ &= \sum_{i \in \mathcal{F}} \mathbf{Pr}[i \in \phi(R)] \\ &\geq \frac{1}{\alpha} \sum_{i \in \mathcal{F}} \mathbf{Pr}[i \in R] \\ &= \frac{1}{\alpha} \mathbf{E}[|R \cap \mathcal{F}|].\end{aligned}$$

Property (c) implies property (b) by a summation argument. Sort and number the elements $\mathcal{E} = (e_1, \ldots, e_n)$ in decreasing order of weights $w_1 \geq w_2 \geq \ldots \geq w_n \geq 0$, where w_i denotes the weight of e_i. Denote $\mathcal{E}_i = \{e_1, \ldots, e_i\}$, and let $\mathcal{E}_0 = \emptyset$, and $w_{n+1} = 0$. Recalling that the greedy algorithm computes the maximum weight independent subset of a matroid:

$$\begin{aligned}\mathbf{E}[\mathbf{rank}_w(R)] &= \mathbf{E}\left[\sum_{i=1}^{n} w_i \left(\mathbf{rank}(R \cap \mathcal{E}_i) - \mathbf{rank}(R \cap \mathcal{E}_{i-1})\right)\right] && \text{(Greedy alg. on } \mathcal{M}|R) \\ &= \mathbf{E}\left[\sum_{i=1}^{n} (w_i - w_{i+1}) \mathbf{rank}(R \cap \mathcal{E}_i)\right] && \text{(Reverse summations)} \\ &\geq \frac{1}{\alpha} \mathbf{E}\left[\sum_{i=1}^{n} (w_i - w_{i+1}) |R \cap \mathcal{E}_i|\right] && \text{((c) and linearity of exp.)} \\ &= \frac{1}{\alpha} \mathbf{E}\left[\sum_{i=1}^{n} w_i \left(|R \cap \mathcal{E}_i| - |R \cap \mathcal{E}_{i-1}|\right)\right] && \text{(Reverse summations)} \\ &= \frac{1}{\alpha} \mathbf{E}[w(R)].\end{aligned}$$

Property (b) implies property (a) by a duality argument identical to that presented in [10]. We present a self-contained proof here. Let $x = x(\mathcal{D}) \in [0,1]^{\mathcal{E}}$ denote the marginals of $\mathcal{D}$. The distribution $\mathcal{D}$ is α-uncontentious if the optimal value of the following LP, with variables β and λ_ϕ for each deterministic CRM ϕ, is at least $\frac{1}{\alpha}$.

$$\begin{array}{ll} \text{maximize} & \beta \\ \text{subject to} & \sum_\phi \lambda_\phi \, \mathbf{Pr}_{R \sim \mathcal{D}}[i \in \phi(R)] \geq \beta x_i, \quad \text{for } i \in \mathcal{E}. \\ & \sum_\phi \lambda_\phi = 1 \\ & \lambda \succeq 0 \end{array}$$

The dual of the preceding LP is the following

$$\begin{array}{ll} \text{minimize} & \mu \\ \text{subject to} & \sum_{i \in \mathcal{E}} w_i \, \mathbf{Pr}_{R \sim D}[i \in \phi(R)] \leq \mu, \quad \text{for all CRM } \phi. \\ & \sum_{i \in \mathcal{E}} w_i x_i = 1 \\ & w \succeq 0 \end{array}$$

It is not hard to see that, at optimality, the binding constraint on μ corresponds to the CRM ϕ which maps each set R to its maximum weight independent subset according to weights w. It follows that the optimal value of the dual, and hence the primal, equals the minimum over all weight vectors $w \succeq 0$ of the ratio $\frac{\mathbf{E}[\mathbf{rank}_w(R)]}{\sum_i w_i x_i}$. (b) implies that this quantity is at least $\frac{1}{\alpha}$, as needed. ◀

We note that the equivalence between (a) and (b) is essentially implicit in the arguments of [10]. Condition (c) is the most notable part of Theorem 3.6, in no small part because it is reminiscent of the *matroid base covering theorem* (see e.g., [31]). This theorem can equivalently be stated as follows: a (deterministic) set $T \subseteq \mathcal{E}$ in a matroid $\mathcal{M} = (\mathcal{E}, \mathcal{I})$ can be *covered by* (i.e., expressed as a union of) $\alpha \in \mathbb{N}$ independent sets if and only if $|S| \leq \alpha\, \mathbf{rank}_{\mathcal{M}}(S)$ for all $S \subseteq T$. In light of part (c) of Theorem 3.6, a set T of elements can be covered by α independent sets if and only if the point distribution on T is α-uncontentious. Therefore, *we can interpret contention resolution as a distributional generalization of base covering.*

3.3 Elementary Properties of Uncontentious Distributions

We state some elementary, yet quite useful, properties of uncontentious distributions.

▶ **Proposition 3.7.** *Fix a matroid $\mathcal{M}$. Every α-uncontentious distribution $\mathcal{D}$ for $\alpha \geq 1$ has marginals $x(\mathcal{D}) \in \alpha\mathcal{P}(\mathcal{M})$.*

Proof. Let $x = x(\mathcal{D})$ and $R \sim \mathcal{D}$. From Theorem 3.6 (c), for every set of ground set elements $\mathcal{F}$ we have

$$\sum_{i \in \mathcal{F}} x_i = \mathbf{E}[|R \cap \mathcal{F}|] \leq \alpha\, \mathbf{E}[\mathbf{rank}_{\mathcal{M}}(R \cap \mathcal{F})] \leq \alpha\, \mathbf{rank}_{\mathcal{M}}(\mathcal{F}).$$

These are the inequalities describing $\alpha\mathcal{P}(\mathcal{M})$. ◀

▶ **Proposition 3.8.** *Fix a matroid. A mixture of α-uncontentious distributions is α-uncontentious.*

Proof. Follows directly from Theorem 3.6 (b) and linearity of expectations. ◀

▶ **Proposition 3.9.** *Fix a matroid $\mathcal{M} = (\mathcal{E}, \mathcal{I})$, and let $\mathcal{M} = (\mathcal{E}', \mathcal{I}')$ be a minor of $\mathcal{M}$, with $\mathcal{E}' \subseteq \mathcal{E}$. If a random set $R \subseteq \mathcal{E}'$ is α-uncontentious in $\mathcal{M}'$, then R is also α-uncontentious in $\mathcal{M}$.*

Proof. An independent set of $\mathcal{M}'$ is also independent in $\mathcal{M}$. Therefore, the proposition follows by simply applying the same CRM in the context of the larger matroid $\mathcal{M}$. ◀

▶ **Proposition 3.10.** *Fix a matroid. Let R be an α-uncontentious random set, and let $R' \sim \mathbf{Ind}_p(R)$ for some $p \in [0, 1]$. The random set R' is α-uncontentious as well.*

Proof. We use Theorem 3.6 (b). For any weight vector w, submodularity of the weighted rank function implies that $\mathbf{E}[\mathbf{rank}_w(R')] \geq p\, \mathbf{E}[\mathbf{rank}_w(R)]$. It follows that $\mathbf{E}[w(R')] = p\, \mathbf{E}[w(R)] \leq p\alpha\, \mathbf{E}[\mathbf{rank}_w(R)] \leq \alpha\, \mathbf{E}[\mathbf{rank}_w(R')]$. ◀

We note that Proposition 3.10 is tight when both p and α are absolute constants. In particular, the random set R' cannot be guaranteed to be α'-uncontentious for a constant $\alpha' < \alpha$, even if p is a very small constant. To see this, consider the a 1-uniform matroid with elements $[n]$, and the following 2-uncontentious random set R: For every singleton $i \in [n]$ we have $\mathbf{Pr}[R = \{i\}] = \frac{1}{n+1}$, and $\mathbf{Pr}[R = [n]] = \frac{1}{n+1}$.

3.4 Examples of Uncontentious Distributions

We now present some examples of uncontentious distributions in order to develop a feel for them. As mentioned previously, and shown in [10], every product distribution with marginals in the matroid polytope is $\frac{e}{e-1}$-uncontentious. Combined with Proposition 3.8, this extends to mixtures of product distributions.

▶ **Proposition 3.11.** *Fix a matroid $\mathcal{M} = (\mathcal{E}, \mathcal{I})$, and let $\mathcal{D} \in \Delta(2^{\mathcal{E}})$ be a mixture of product distributions, each with marginals in $\mathcal{P}(\mathcal{M})$. It follows that $\mathcal{D}$ is $\frac{e}{e-1}$-uncontentious.*

Going beyond product distributions and their mixtures, if a distribution satisfies a certain strong notion of negative correlation, defined in [9], then it also is $\frac{e}{e-1}$-uncontentious.

▶ **Proposition 3.12.** *Fix a matroid $\mathcal{M} = (\mathcal{E}, \mathcal{I})$, and let $\mathcal{D} \in \Delta(2^{\mathcal{E}})$ be a distribution with marginals $x = x(D) \in \mathcal{P}(\mathcal{M})$. Assume that $\mathcal{D}$ satisfies the* property of increasing submodular expectations*: for every submodular function f we have $\mathbf{E}_{R \sim \mathcal{D}}[f(R)] \geq \mathbf{E}_{S \sim \mathbf{Ind}(x)}[f(S)]$.*[3] *It follows that $\mathcal{D}$ is $\frac{e}{e-1}$-uncontentious.*

Proof. This is immediate by combining Theorem 3.6 (b) with the property of increasing submodular expectations and the fact that $\mathbf{Ind}(x)$ is $\frac{e}{e-1}$-uncontentious. ◀

As shown in [9], the property of increasing submodular expectations is stronger than the following standard notion of negative correlation for $R \sim \mathcal{D}$: For all sets T, $\mathbf{Pr}[T \subseteq R] \leq \prod_{i \in T} \mathbf{Pr}[i \in R]$ and $\mathbf{Pr}[T \subseteq \overline{R}] \leq \prod_{i \in T}(1 - \mathbf{Pr}[i \in R])$.[4] However, we can show that there are distributions exhibiting positive correlation which are also uncontentious for specific matroids. We now list some examples of uncontentious distributions exhibiting positive correlation.

▶ **Example 3.13.** Let $\mathcal{M}$ be a k-uniform matroid with n elements where $2 \leq k \leq n$. Let the random set R be empty with probability $1/2$, and a uniformly random base of $\mathcal{M}$ otherwise.

It is clear that R is 1-uncontentious, since it is supported on the family of independent sets. However, for each distinct pair of elements i and j, we have $\mathbf{Pr}[i \in R] = \mathbf{Pr}[j \in R] = \frac{k}{2n}$, yet $\mathbf{Pr}[i \in R | j \in R] = \mathbf{Pr}[j \in R | i \in R] = \frac{k-1}{n-1} > \frac{k}{2n}$.

The next example will feature repeatedly in this paper, since it is the random set of improving elements for the rank 1 matroid.

▶ **Example 3.14.** Consider the 1-uniform matroid with elements $[n] = \{1, \ldots, n\}$. For $k = 0, 1, \ldots, n-1$, let $R = \{1, \ldots, k\}$ with probability $2^{-(k+1)}$, and let $R = [n]$ with remaining probability 2^{-n}. The random set R is 2-uncontentious, as evidenced by the CRM ϕ with $\phi(\{1, \ldots, k\}) = \{k\}$ and $\phi(\emptyset) = \emptyset$, and a simple calculation. Note the positive correlation between elements $i < j$: $\mathbf{Pr}[j \in R] = 2^{-j}$, and $\mathbf{Pr}[j \in R | i \in R] = 2^{i-j} > \mathbf{Pr}[i \in R]$.

As a generalization of the previous example, we get the following.

▶ **Example 3.15.** Let $\mathcal{M}$ be a matroid with m pairwise-disjoint bases $B_1, \ldots, B_m$. For each $k = 1, \ldots, m-1$, let $R = \cup_{i=1}^{k} B_i$ with probability 2^{-k}, and let $R = \cup_{i=1}^{m} B_m$ with the remaining probability 2^{1-m}. The set R is 2-uncontentious, as evidenced by the CRM $\phi(\cup_{i=1}^{k} B_i) = B_k$. However, for $e_i \in B_i$ and $e_j \in B_j$ with $i < j$, we have $\mathbf{Pr}[e_j \in R] = 2^{1-j}$ and $\mathbf{Pr}[e_j \in R | e_i \in R] = 2^{i-j} > \mathbf{Pr}[e_j \in R]$.

[3] In fact, it suffices for $\mathcal{D}$ to satisfy the (weaker) property of increasing expectations for matroid rank functions (or, equivalently, their weighted sums).

[4] A natural question is whether negative correlation suffices for the distribution to be $\frac{e}{e-1}$-uncontentious. This is open as far as we know.

3.5 Contention Resolution Schemes, Universality, and Prior Dependence

A *contention resolution scheme (CRS)* Φ for a matroid $\mathcal{M} = (\mathcal{E}, \mathcal{I})$ and class of distributions $\mathbb{D} \subseteq \Delta(2^{\mathcal{E}})$ is an algorithm which takes as input a (possibly partial) description of a distribution $\mathcal{D} \in \mathbb{D}$ and a sample $R \sim \mathcal{D}$, and outputs $S \in \mathcal{I}$ satisfying $S \subseteq R$. In effect, Φ is a collection of contention resolution maps $\phi_{\mathcal{D}}$, one for each $\mathcal{D} \in \mathbb{D}$. In much of the prior work on contention resolution, $\mathbb{D}$ was taken to be the class of product distributions with marginals in $\mathcal{P}(\mathcal{M})$, and each $\mathcal{D} \in \mathbb{D}$ is described completely via its marginals $x \in \mathcal{P}(\mathcal{M})$. In such a setting, the notion of a CRS offers little beyond the notion of a CRM, as each distribution gets its own dedicated CRM. More generally, however, we allow $\mathbb{D}$ to be an arbitrary class of distributions, and we allow the description to be partial and/or random; for example, $\mathcal{D}$ may be described by m independent samples from $\mathcal{D}$.

Next, we set the stage by defining some desirable contention resolution schemes, and establish some limitations on their existence.

▶ **Definition 3.16.** *Fix a matroid. For $\beta \geq \alpha > 1$, an* α-universal β-competitive CRS *is a CRS which is β-competitive for the class of α-uncontentious distributions.*

By definition, there exists an (offline) α-universal α-competitive CRS for every α and every matroid. The notion of a universal scheme becomes more interesting when we restrict dependence on the prior, as per the following definitions.

▶ **Definition 3.17.** *Fix a matroid. A contention resolution scheme Φ is said to be* prior-independent *if it is not given a complete description of $\mathcal{D}$ as input, but rather is given a set of independent samples from $\mathcal{D}$. When the number of samples is m, we say Φ is a* prior-independent m-sample scheme*. The number of samples may be function of the size of the matroid. If $m = 0$, we say the scheme is* oblivious*: the scheme consists of a single contention resolution map.*

We now show that, if a scheme is universal, it cannot be prior-independent with any finite number of samples, even for very simple matroids.

▶ **Theorem 3.18.** *Let $\mathcal{M}$ be the 1-uniform matroid on n elements. For every finite m, and every $1 < \alpha \leq \beta < n$, there does not exist a β-competitive α-universal CRS for $\mathcal{M}$ which is prior independent with m samples.*

To prove Theorem 3.18, we first show that a prior-independent universal scheme implies the existence of an oblivious universal scheme; then we show that an oblivious universal scheme does not exist for the uniform matroid. This is captured in the two following lemmas.

▶ **Lemma 3.19.** *Fix a matroid $\mathcal{M}$. If there exists a β-competitive α-universal CRS Φ which is prior-independent with m samples, then there exists an oblivious β-competitive α-universal scheme Φ'.*

Proof. Let $\mathcal{D}$ be any α-uncontentious distribution. Let $\epsilon \in (0, 1)$, and let $\mathcal{D}' = \mathcal{D}'(\epsilon)$ be the mixture of $\mathcal{D}$ with the point distribution on the empty set with proportions ϵ and $1 - \epsilon$ respectively. By Proposition 3.8 and the fact that the point distribution on the empty set is 1-uncontentious, it follows that $\mathcal{D}'$ is α-uncontentious.

The CRS Φ induces a CRM $\phi_{\mathcal{D}'}$ on the distribution $\mathcal{D}'$, and by assumption $\phi_{\mathcal{D}'}$ is β-competitive for $\mathcal{D}'$. Since Φ is prior-independent with m samples, its induced CRM $\phi_{\mathcal{D}'}$ is a mixture over CRMs ϕ_S, where $S = (S_1, \ldots, S_m)$ is a random vector of m samples from $\mathcal{D}'$.

With probability at least $(1-\epsilon)^m$, we have $S = \emptyset^m := (\emptyset, \ldots, \emptyset)$. For $\phi_{\mathcal{D}'}$ to be β-competitive, in particular when with probability ϵ it is queried with a draw $R \sim D$, a simple calculation shows that $\phi_{\emptyset^m}$ must be β'-competitive for $\mathcal{D}$ for $\beta' = \frac{(1-\epsilon)^m}{1/\beta + (1-\epsilon)^m - 1}$. As ϵ tends to 0, β' tends to β, and a basic analytic argument implies that $\phi_{\emptyset^m}$ is β-competitive for $\mathcal{D}$. Since $\mathcal{D}$ was chosen arbitrarily among α-uncontentious distributions, and $\phi_{\emptyset^m}$ does not depend on $\mathcal{D}$, it follows that the oblivious scheme Φ' with $\phi'_{\mathcal{D}} = \phi_{\emptyset^m}$ for every $\mathcal{D}$ is β-competitive and α-universal. ◀

▶ **Lemma 3.20.** *The 1-uniform matroid with n elements does not admit an oblivious β-competitive α-universal CRS for any $1 < \alpha \leq \beta < n$.*

Proof. Let $[n]$ be the ground set of the matroid, and fix α such that $1 < \alpha < n$. An oblivious CRS consists of a single CRM ϕ. There exists at least one element $i \in [n]$ such that $\mathbf{Pr}[i \in \phi([n])] \leq 1/n$. Let $\epsilon = \alpha - 1 > 0$, and consider the following random set R: For each $j \in [n] \setminus i$ we have $R = \{j\}$ with probability $\frac{1}{n-1+\epsilon}$, and $R = [n]$ with the remaining probability $\frac{\epsilon}{n-1+\epsilon}$. The random set R is α-uncontentious: consider the CRM ϕ' with $\phi'(\{j\}) = j$ for $j \neq i$, and $\phi'([n]) = i$. However, our original CRM ϕ is no better than n-competitive for R, since its probability of selecting i is no more than $\frac{1}{n}\mathbf{Pr}[R = [n]] = \frac{1}{n}\mathbf{Pr}[i \in R]$. ◀

4 An Online Universal CRS from a Secretary Algorithm

We show that competitive matroid secretary algorithms imply that every contention resolution scheme can be made online, in the random arrival model, without much loss.

▶ **Theorem 4.1.** *Suppose that there is a γ-competitive online algorithm for the secretary problem on matroid $\mathcal{M}$. It follows that every α-uncontentious distribution admits an online $\gamma\alpha$-competitive contention resolution map. In other words, for every α there exists an online $\gamma\alpha$-competitive α-universal contention resolution scheme for $\mathcal{M}$.*

We interpret the above theorem as follows: the design of competitive universal online schemes is a necessary technical hurdle towards resolving the matroid secretary conjecture.

We now proceed with proving Theorem 4.1. Let $\mathcal{M} = (\mathcal{E}, \mathcal{I})$, and let $\mathcal{D} \in \Delta(2^{\mathcal{E}})$. Recall that an online CRM operates in the following model: a set of active elements $R \sim \mathcal{D}$ and a random permutation Π are (independently) sampled by nature, then $\mathcal{E}$ arrive online in order Π. When $i \in \mathcal{E}$ arrives, it is revealed whether $i \in R$, and if so the online CRS must determine whether to select i. The online CRM must only select an independent subset of R.

Suppose we are given a secretary algorithm $\mathcal{A}$ for $\mathcal{M}$ with competitive ratio γ. Without loss of generality, we assume that $\mathcal{A}$ selects only non-zero weight elements. Consider the following online CRM ϕ_w for $\mathcal{M}$, parametrized by a weight vector $w \in \mathbb{R}_+^{\mathcal{E}}$. When element i arrives, if $i \in R$ then it is presented to $\mathcal{A}$ with weight w_i, and if $i \notin R$ then it is presented to $\mathcal{A}$ with weight 0. ϕ_w selects precisely the elements selected by $\mathcal{A}$.

▶ **Lemma 4.2.** *For every distribution $\mathcal{D}$, we have $\mathbf{E}_{R\sim\mathcal{D}}[w(\phi_w(R))] \geq \frac{1}{\gamma}\mathbf{E}_{R\sim\mathcal{D}}[\mathbf{rank}_w(R)]$.*

Proof. Condition on the choice of R, and let $w'_i = w_i$ if $i \in R$ and $w'_i = 0$ otherwise. $\mathcal{E}$ are presented to $\mathcal{A}$ in a uniformly random order, with weights w'_i, and $\phi_w(R) \subseteq R$ is the set of elements selected by $\mathcal{A}$. Since $\mathcal{A}$ is γ-competitive, it follows that $\mathbf{E}[w'(\phi_w(R))] \geq \frac{1}{\gamma}\mathbf{rank}_{w'}(\mathcal{M})$. Since $w'(\phi_w(R)) = w(\phi_w(R))$ and $\mathbf{rank}_{w'}(\mathcal{M}) = \mathbf{rank}_w(R)$, we are done. ◀

▶ **Lemma 4.3.** *If $\mathcal{D}$ is α-uncontentious, then $\mathbf{E}_{R\sim\mathcal{D}}[w(\phi_w(R))] \geq \frac{1}{\gamma\alpha}\mathbf{E}_{R\sim\mathcal{D}}[w(R)]$.*

Proof. Combining the previous lemma with Theorem 3.6 (b). ◀

Recall that we are assuming for now that we know the α-uncontentious distribution $\mathcal{D}$, and we can design an online CRM $\phi_{\mathcal{D}}$ accordingly. $\phi_{\mathcal{D}}$ will be a random mixture of the maps ϕ_w described above; in particular, we will show that there exists a distribution $\mathcal{W} = \mathcal{W}(\mathcal{D})$ over weight vectors such that the (randomized) online CRM $\phi_{\mathcal{W}}$ which samples $w \sim \mathcal{W}$ upfront then invokes ϕ_w is an online $\gamma\alpha$-CRM for $\mathcal{D}$.

For each element $i \in \mathcal{E}$, let $x_i = \mathbf{Pr}_{R\sim\mathcal{D}}[i \in R]$. For each weight vector w and $i \in \mathcal{E}$, let $y_i(w) = \mathbf{Pr}_{R\sim\mathcal{D}}[i \in \phi_w(R)]$. For each distribution $\mathcal{W}$ over weight vectors and element $i \in \mathcal{E}$, let $y_i(\mathcal{W}) = \mathbf{Pr}_{R\sim\mathcal{D}}[i \in \phi_{\mathcal{W}}(R)] = \mathbf{Pr}_{R\sim\mathcal{D},w\sim\mathcal{W}}[i \in \phi_w(R)]$. Let $\mathcal{Y} = \left\{y(\mathcal{W}) : \mathcal{W} \in \Delta(\mathbb{R}_+^{\mathcal{E}})\right\} \subseteq [0,1]^{\mathcal{E}}$ be the family of all inclusion probabilities achievable by some online CRM of the form $\phi_{\mathcal{W}}$. It is immediate that $\mathcal{Y} = \mathbf{convexhull}(\left\{y(w) : w \in \mathbb{R}_+^{\mathcal{E}}\right\})$, and hence $\mathcal{Y}$ is a convex subset of $[0,1]^{\mathcal{E}}$.

An online $\alpha\gamma$-CRM for $\mathcal{D}$ of the form $\phi_{\mathcal{W}}$ exists if and only if $\mathcal{Y}$ intersects with the upwards closed convex set $\frac{x}{\alpha\gamma} + \mathbb{R}_+^{\mathcal{E}}$. Suppose for a contradiction that this intersection is empty; by the separating hyperplane theorem, this implies that there exists $w \in \mathbb{R}_+^{\mathcal{E}}$ such that $\frac{1}{\alpha\gamma}\sum_i w_i x_i > \sum_i w_i y_i$ for all $y \in \mathcal{Y}$. In particular, $\frac{1}{\alpha\gamma}\sum_i w_i x_i > \sum_i w_i y_i(w)$. Since $\sum_i w_i x_i = \mathbf{E}_{R\sim\mathcal{D}}\, w(R)$ and $\sum_i w_i y_i(w) = \mathbf{E}_{R\sim\mathcal{D}}[w(\phi_w(R))]$, we get a contradiction with Lemma 4.3. This concludes the proof of the theorem.

5 From Contention Resolution to a Secretary Algorithm?

One might hope that online contention resolution is equivalent to the secretary problem on matroids. In particular, does a competitive universal online CRS imply a competitive secretary algorithm? We make partial progress towards this question. In particular, we reduce the secretary problem to online contention resolution on a particular uncontentious distribution derived from the matroid and sample of its elements: the distribution of “improving elements”, as originally defined by Karger [20] for purposes different from ours.

▶ **Definition 5.1.** *Fix a matroid $\mathcal{M} = (\mathcal{E}, \mathcal{I})$ with weights $w \in \mathbb{R}_+^{\mathcal{E}}$, and let $p \in (0,1)$. The random set R of* improving elements *with parameter p is sampled as follows: Let $S \sim \mathbf{Ind}_p(\mathcal{E})$, and let $R = R(S) = \{i \in \mathcal{E} : \mathbf{rank}_w(S \cup i) > \mathbf{rank}_w(S)\}$. Equivalently, R is the set of elements in $\mathcal{E} \setminus S$ which are not spanned by higher weight elements in S. Another equivalent definition is $R = \{i \in \mathcal{E} \setminus S : i \in \mathbf{OPT}_w(S \cup i)\}$.*

The maximum-weight independent subset of the improving elements is $(1-p)$-approximately optimal in expectation:

▶ **Fact 5.2.** *Fix a weighted matroid $(\mathcal{M}, w)$, and let R be the random set of improving elements with parameter p. Each element of $\mathbf{OPT}_w(\mathcal{M})$ is in R with probability $1-p$. It follows that $\mathbf{E}[w(R)] \geq \mathbf{E}[\mathbf{rank}_w(R)] \geq (1-p)\mathbf{rank}_w(\mathcal{M})$.*

Note that the random set R of improving elements does not follow a product distribution. In fact, elements are (weakly) positively correlated in general. This is illustrated by the special case of the 1-uniform matroid on $[n]$ with weights $w_1 > w_2 > \ldots > w_n$, and $p = 1/2$: the distribution of R is as described in Example 3.14. As our main result in this section, we nevertheless show that the random set of improving elements is uncontentious.

▶ **Theorem 5.3.** *Let $\mathcal{M} = (\mathcal{E}, \mathcal{I})$ be a matroid with weights $w \in \mathbb{R}_+^{\mathcal{E}}$, and let $p \in (0,1)$. The random set of improving elements with parameter p is $\frac{1}{p}$-uncontentious.*

Theorem 5.3 and Fact 5.2, taken together, essentially reduce the matroid secretary problem to online contention resolution for the distribution of the random set of improving elements, with one caveat we will discuss shortly. In particular, consider the following blueprint for a secretary algorithm:

1. Let S be the first $\mathbf{Binom}(|\mathcal{E}|, p)$ elements arriving online.
2. Let $R = R(S) \subseteq \mathcal{E} \setminus S$ be a sample of the set of improving elements with parameter p.
3. After observing S, the elements of $\mathcal{E} \setminus S$ arrive online in random order and are presented as such to an online contention resolution algorithm, along with their membership status in R. Note that membership in R can be determined "on the spot" as required for online contention resolution.[5]

Now given a β-competitive α-universal online CRS, we set $p = \frac{1}{\alpha}$ and obtain a $\frac{\beta}{1-p}$-competitive secretary algorithm. However, the following caveat prevents us from proving a formal theorem of this form: we cannot provide the online CRS with a complete description of the prior distribution. In particular, the distribution $\mathcal{D}$ of improving elements – while fully described by the weighted matroid $(\mathcal{M}, w)$ and the parameter p – can not be fully described to the contention resolution algorithm prior to its invocation, since entries of w are revealed online. As such, we learn both the sample $R \sim \mathcal{D}$ and the distribution $\mathcal{D}$ gradually as elements arrive. An oblivious universal online CRS would resolve this difficulty, but unfortunately we proved in Theorem 3.18 that such a CRS can not exist even for simple matroids and even offline. A reduction from the matroid secretary problem to contention resolution must therefore require a CRS which can make do with only partial knowledge of the prior. We leave exploration of these possibilities for future work, and discuss them further in the Conclusion section.

5.1 Proof of Theorem 5.3

Let p, S, and R be as in Definition 5.1. We prove that R is uncontentious by leveraging (c) from Theorem 3.6. In particular we will show that, for arbitrary $F \subseteq \mathcal{E}$, we have $\mathbf{E}[\mathbf{rank}(R \cap F)] \geq p\,\mathbf{E}[|R \cap F|]$. We break this up into the following three lemmas.

▶ **Lemma 5.4.** $\mathbf{E}[\mathbf{rank}(R \cap F)] \geq (1-p)\,\mathbf{E}[|F \cap \mathbf{OPT}_w(S \cup F)|]$

Proof. Let $T = S \setminus F$, and note that $S \cup F = T \uplus F$. We condition on the random variable T and show conditionally that $\mathbf{E}[\mathbf{rank}(R \cap F)] \geq (1-p)|F \cap \mathbf{OPT}_w(T \uplus F)|]$.

Take $i \in F \cap \mathbf{OPT}_w(T \uplus F)$. We will show that i is in R, and hence is in $R \cap F$, with probability $1-p$. Since $i \in S \cup i \subseteq T \uplus F$ and $i \in \mathbf{OPT}_w(T \uplus F)$, it follows from the matroid axioms that $i \in \mathbf{OPT}_w(S \cup i)$. With probability $1-p$ we also have $i \notin S$, in which case $i \in R$ by definition. Since $F \cap \mathbf{OPT}_w(T \uplus F)$ is an independent set, the claim follows. ◀

▶ **Lemma 5.5.** $|F \cap \mathbf{OPT}_w(S \cup F)| \geq |F \cap \mathbf{OPT}_w(S)|$

Proof. We prove this by induction on a set T with $S \subseteq T \subseteq S \cup F$, initialized to $T = S$ at the base case. Consider how the value of $|F \cap \mathbf{OPT}_w(T)|$ changes as we add elements of $F \setminus S$ to T one by one. When adding an element $i \in F \setminus T$ to T, there are three cases:

- $i \notin \mathbf{OPT}_w(T \cup i)$: In this case, $\mathbf{OPT}_w(T \cup i) = \mathbf{OPT}_w(T)$ and $|F \cap \mathbf{OPT}_w(T \cup i)| = |F \cap \mathbf{OPT}_w(T)|$.
- i is not spanned by T, and $i \in \mathbf{OPT}_w(T \cup i)$: In this case, $\mathbf{OPT}_w(T \cup i) = \mathbf{OPT}_w(T) \cup \{i\}$, and therefore $|F \cap \mathbf{OPT}_w(T \cup i)| = 1 + |F \cap \mathbf{OPT}_w(T)|$.
- i is spanned by T, and $i \in \mathbf{OPT}_w(T \cup i)$: In this case, elementary application of the matroid axioms implies that $\mathbf{OPT}_w(T \cup i) = \mathbf{OPT}_w(T) \cup \{i\} \setminus \{j\}$ for some $j \in T$. Since $i \in F$, it follows that $|F \cap \mathbf{OPT}_w(T \cup i)|$ is either equal to $|F \cap \mathbf{OPT}_w(T)|$ or exceeds it by 1, depending on whether $j \in F$. ◀

[5] Technically, a CRM requires that elements of $\mathcal{E}$ – rather than merely $\mathcal{E} \setminus S$ – be presented in uniform random order along with their membership status in R. This is easily accomplished by appropriately interleaving the elements of S – none of which are in R – among the elements of $\mathcal{E} \setminus S$.

▶ **Lemma 5.6.** $\mathbf{E}[|\mathbf{OPT}_w(S) \cap F|] \geq \frac{p}{1-p} \mathbf{E}[|R \cap F|]$

Proof. For each $i \in F$, we will show that $\mathbf{Pr}[i \in \mathbf{OPT}_w(S)] \geq \frac{p}{1-p} \mathbf{Pr}[i \in R]$, which suffices.

Take $i \in F$, and let $S_{>i} = \{j \in S : w_j > w_i\}$. Conditioning on $S_{>i}$, there are two cases:

- $i \in \mathbf{span}(S_{>i})$: It follows that $i \notin \mathbf{OPT}_w(S)$ and $i \notin R$, with certainty.
- $i \notin \mathbf{span}(S_{>i})$: With probability p we have $i \in S$ and therefore $i \in \mathbf{OPT}_w(S)$ and $i \notin R$. With the remaining probability $(1-p)$ we have $i \notin S$ and therefore $i \in R$ and $i \notin \mathbf{OPT}_w(S)$.

In both cases, the conditional probability that $i \in \mathbf{OPT}_w(S)$ is at least $\frac{p}{1-p}$ times the conditional probability that $i \in R$. The lemma follows. ◀

5.2 Where Prior Work Fails

There has been speculation in the community that contention resolution for improving element distributions can be accomplished online using the ideas of Feldman et al [17]. If this were true, then a stronger (online) form of our Theorem 5.3 would follow. We show that such conjecture is fatally flawed: there exists no $o(n)$-competitive online CRS in the worst-case arrival model, even when both the order and the distribution of improving elements are known to the algorithm. In other words, any competitive online CRS for improving element distributions must make and exploit assumptions on the arrival order. This rules out direct application of the arguments and techniques of Feldman et al [17], which – in holding for an (unknown) worst-case arrivals – cannot exploit the uniform arrival order. The same can be said for the work of Lee and Singla [25], which operates in the known worst-case arrival model.

We prove the following theorem, then elaborate on how algorithms from prior work tend to fail on simple examples.

▶ **Theorem 5.7.** *Let $\mathcal{M}$ be a matroid on n elements. There is no $o(n)$-competitive online CRS for (known) improving element distributions on M in the worst-case arrival model. This holds even for the 1-uniform matroid, for every constant parameter p of the distribution of improving elements, and even when the arrival order is known to the algorithm.*

Proof. Let $\{1, \ldots, n\}$ denote the ground set of of a 1-uniform matroid, listed in decreasing order of weight. Let R be the random set of improving elements with parameter p. Note that R is supported on sets of the form $\{1, \ldots, k\}$ for $k = 0, \ldots, n$. In the special case of $p = 1/2$, the distribution of R is as described in Example 3.14. In general, $\mathbf{Pr}[R = \{1, \ldots, k\}] = p(1-p)^k$. The random set R is $1/p$ uncontentious, as shown by Theorem 5.3. Concretely, the offline CRM $\phi(\{1, \ldots, k\}) = k$ is $\frac{1}{p}$-competitive.

Now suppose that elements are known to arrive online in the order $1, 2, 3, \ldots, n$, and consider an α-competitive online CRM for some $\alpha \geq 1$. Let $T \subseteq R$ be the (random) set of elements selected by the CRM. Conditioned on $i \in R$, the CRM must select i with probability at least $\frac{1}{\alpha}$. Formally, $\mathbf{Pr}[i \in T | i \in R] \geq \frac{1}{\alpha}$.

When element i arrives, the CRM learns whether $i \in R$, and if so must decide whether to select i. Since the online CRM has only observed elements $1, \ldots, i$, and must make its decision on the spot, it cannot distinguish between different sets of the form $R = \{1, \ldots, k\}$ for $k \geq i$. In other words, it cannot distinguish between the different realizations of R which include i, and must therefore select i with probability at least $\frac{1}{\alpha}$ in every realization of R which includes i. Formally, $\frac{1}{\alpha} \leq \mathbf{Pr}[i \in T | i \in R] = \mathbf{Pr}[i \in T | R = \{1, \ldots, k\}]$ for every $k \geq i$.

Since i was chosen arbitrarily, we can take $k = n$ and conclude that $\mathbf{Pr}[i \in T | R = \{1, \ldots, n\}] \geq \frac{1}{\alpha}$ for all i. Feasibility requires that $\sum_{i=1}^{n} \mathbf{Pr}[i \in T | R = \{1, \ldots, n\}] \leq 1$. Therefore, $\alpha \geq n$. ◀

It is instructive to examine where the algorithm of Feldman et al [17] fails in the special case of the 1-uniform matroid on n elements, even when improving elements are presented in a uniformly random order. Indeed, we will argue that no "simple tricks" seem to save the day. Recall that the algorithm of [17] defines a sequence of nested flats $\emptyset \subset F_1 \subset F_2 \subset \ldots \subset F_k$, and runs the greedy online algorithm on each contracted submatroid F_i/F_{i-1}. The 1-uniform matroid contains only a single non-empty flat, containing all elements. Therefore, the algorithm of [17] reduces merely to the naive greedy online algorithm which simply selects the first element it encounters, which in the case of a uniform arrival order is a uniformly random improving element.

Now, let $[n] = \{1, \ldots, n\}$ denote the ground set of the 1-uniform matroid listed in decreasing order of weight, and consider the distribution of improving elements R with parameter $p = 1/2$ as described in Example 3.14. Element k is improving with probability 2^{-k}, yet is selected by the algorithm with probability $\sum_{i=k}^{n-1} 2^{-(i+1)} \cdot \frac{1}{i} + 2^{-n} \cdot \frac{1}{n} < \frac{2^{-k}}{k} = \frac{\mathbf{Pr}[k \in R]}{k}$. Intuitively, when k is improving, so are elements $1, \ldots, k-1$, which easily span k and are not distinguished from k by the algorithm. It is easy to show that the algorithm suffers the same fate for any other choice of p.

One might be tempted to employ other tricks, such as for example "canceling" each element in R with independent constant probability in order to reduce contention and place the marginal probability vector deep in the matroid polytope. Such tricks are doomed to fail all the same: the algorithm groups all elements into the same (unique) flat, and in doing so does not distinguish between the "uncanceled" elements of R, so cannot select element k with probability exceeding $\frac{\mathbf{Pr}[k \in R]}{k}$.

It is hopefully now clear that any online CRS for improving element distributions must make and exploit assumptions on the arrival order. Whereas this rules out obvious extensions of Feldman et al [17] and Lee and Singla [25], one might hope that the algorithm of Adamczyk and Włodarczyk [1] might fare better, since they do exploit the random ordering assumption. Sadly, their algorithm also fails for the 1-uniform matroid: it also does not distinguish between different improving elements in this special case, and therefore also selects element k with probability no more than $\frac{\mathbf{Pr}[k \in R]}{k}$. That being said, we are more hopeful that the techniques of [1], if combined with significant new ideas, might yield progress on online contention resolution for positively correlated distributions.

6 Conclusions and Open Problems

In this paper, we begin an exploration of the power and limitations of contention resolution beyond known product distributions, as well as its connections to secretary problems. We hope that our results are a first step towards broader application of the techniques behind contention resolution and online selection. Most notably, our results highlight approaches to resolving the matroid secretary conjecture. We identify several intriguing open questions in pursuit of these agendas.

- Can the result of Theorem 4.1 be shown unconditionally; i.e., can we show a competitive universal online CRS for matroids without assuming the matroid secretary conjecture? We believe this to be a reasonable first step towards proving the matroid secretary conjecture. As we show in Section 5.2, prior work on online contention resolution fails in the presence of even the modest positive correlation exhibited by (uncontentious) improving element distributions on simple matroids. Therefore, we believe significant new ideas are required.

- Recalling the caveat to our results from Section 5, can a tighter connection be made between the secretary problem and contention resolution? Is there a natural model of contention resolution on matroids which permits a reduction both from and to the matroid secretary problem? The knee-jerk approaches using duality-like arguments fail to establish such an equivalence, so new ideas appear to be required.
- The caveat to our results from Section 5 suggests that resolving contention with limited knowledge of the prior is closely related to the matroid secretary conjecture. Recalling our impossibility result of Theorem 3.18, we can start by examining prior-independent contention resolution for interesting classes of distributions. For example, is there a competitive prior-independent (or even oblivious) CRS for ex-ante-feasible product distributions?
- Can Theorem 4.1 be made computationally efficient? Given only oracle access to an arbitrary uncontentious distribution and an arbitrary algorithm for the matroid secretary problem, this is unclear.
- Is there an analogue of our characterization of uncontentious distributions for prophet inequality problems? In particular, can we characterize joint distributions of random variables which permit competitive prophet inequalities with respect to a given matroid?
- Do more general set systems permit a characterization of uncontentious distributions with a finite set of inequalities, a-la Theorem 3.6?

We restricted our attention to matroids in the paper, though some notes are in order on extensions of our results to more general constraints. In the characterization of Theorem 3.6, the equivalence of (a) and (b) holds for a general downwards-closed set systems, and is implicit in the arguments of [10]. The equivalence with (c) exploits the matroid structure, however. Theorem 4.1 also holds for general downwards-closed set systems, and our proof does not invoke the matroid assumption. The results and arguments of Section 5, in particular Theorem 5.3, heavily rely on the matroid structure and do not appear to be easily extensible beyond matroids. We leave further extensions of our results beyond matroids for future work.

References

1 Marek Adamczyk and Michał Włodarczyk. Random order contention resolution schemes. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 790–801. IEEE, 2018.

2 Shipra Agrawal, Yichuan Ding, Amin Saberi, and Yinyu Ye. Price of correlations in stochastic optimization. *Operations Research*, 60(1):150–162, 2012.

3 Saeed Alaei. Bayesian combinatorial auctions: Expanding single buyer mechanisms to many buyers. *SIAM Journal on Computing*, 43(2):930–972, 2014.

4 Pablo D Azar, Robert Kleinberg, and S Matthew Weinberg. Prophet inequalities with limited information. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 1358–1377. Society for Industrial and Applied Mathematics, 2014.

5 Yossi Azar, Ashish Chiplunkar, and Haim Kaplan. Prophet secretary: Surpassing the 1-1/e barrier. In *Proceedings of the 2018 ACM Conference on Economics and Computation*, pages 303–318. ACM, 2018.

6 Moshe Babaioff, Nicole Immorlica, and Robert Kleinberg. Matroids, secretary problems, and online mechanisms. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 434–443. Society for Industrial and Applied Mathematics, 2007.

7 Gruia Calinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a monotone submodular function subject to a matroid constraint. *SIAM Journal on Computing*, 40(6):1740–1766, 2011.

8 Shuchi Chawla, Jason D Hartline, David L Malec, and Balasubramanian Sivan. Multi-parameter mechanism design and sequential posted pricing. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 311–320. ACM, 2010.

9 Chandra Chekuri, Jan Vondrák, and Rico Zenklusen. Dependent randomized rounding via exchange properties of combinatorial structures. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 575–584. IEEE, 2010.

10 Chandra Chekuri, Jan Vondrák, and Rico Zenklusen. Submodular function maximization via the multilinear relaxation and contention resolution schemes. *SIAM Journal on Computing*, 43(6):1831–1879, 2014.

11 José Correa, Paul Dütting, Felix Fischer, and Kevin Schewior. Prophet inequalities for iid random variables from an unknown distribution. In *Proceedings of the 2019 ACM Conference on Economics and Computation*, pages 3–17. ACM, 2019.

12 Michael Dinitz. Recent advances on the matroid secretary problem. *ACM SIGACT News*, 44(2):126–142, 2013.

13 Evgenii Borisovich Dynkin. The optimum choice of the instant for stopping a markov process. *Soviet Mathematics*, 4:627–629, 1963.

14 Soheil Ehsani, MohammadTaghi Hajiaghayi, Thomas Kesselheim, and Sahil Singla. Prophet secretary for combinatorial auctions and matroids. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 700–714. SIAM, 2018.

15 Hossein Esfandiari, MohammadTaghi Hajiaghayi, Vahid Liaghat, and Morteza Monemizadeh. Prophet secretary. *SIAM Journal on Discrete Mathematics*, 31(3):1685–1701, 2017.

16 Moran Feldman, Ola Svensson, and Rico Zenklusen. A simple o (log log (rank))-competitive algorithm for the matroid secretary problem. In *Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms*, pages 1189–1201. SIAM, 2014.

17 Moran Feldman, Ola Svensson, and Rico Zenklusen. Online contention resolution schemes. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 1014–1033. Society for Industrial and Applied Mathematics, 2016.

18 Mohammad Taghi Hajiaghayi, Robert Kleinberg, and Tuomas Sandholm. Automated online mechanism design and prophet inequalities. In *AAAI*, volume 7, pages 58–65, 2007.

19 Theodore P Hill and Robert P Kertz. A survey of prophet inequalities in optimal stopping theory. *Contemp. Math*, 125:191–207, 1992.

20 David R Karger. Random sampling and greedy sparsification for matroid optimization problems. *Mathematical Programming*, 82(1-2):41–81, 1998.

21 Robert Kleinberg and Seth Matthew Weinberg. Matroid prophet inequalities. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 123–136. ACM, 2012.

22 Ulrich Krengel and Louis Sucheston. Semiamarts and finite values. *Bulletin of the American Mathematical Society*, 83(4):745–747, 1977.

23 Ulrich Krengel and Louis Sucheston. On semiamarts, amarts, and processes with finite value. *Probability on Banach spaces*, 4:197–266, 1978.

24 Oded Lachish. O (log log rank) competitive ratio for the matroid secretary problem. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 326–335. IEEE, 2014.

25 Euiwoong Lee and Sahil Singla. Optimal online contention resolution schemes via ex-ante prophet inequalities. In *26th Annual European Symposium on Algorithms (ESA 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

26 J. G. Oxley. *Matroid Theory*. Oxford University Press, 1992.

27 Yosef Rinott, Ester Samuel-Cahn, et al. Comparisons of optimal stopping values and prophet inequalities for negatively dependent random variables. *The Annals of Statistics*, 15(4):1482–1490, 1987.

28 Aviad Rubinstein. Beyond matroids: Secretary problem and prophet inequality with general constraints. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 324–332. ACM, 2016.

29 Ester Samuel-Cahn. Prophet inequalities for bounded negatively dependent random variables. *Statistics & probability letters*, 12(3):213–216, 1991.

30 Jack Wang. The prophet inequality can be solved optimally with a single set of samples. *arXiv preprint*, 2018. `arXiv:1812.10563`.

31 Dominic JA Welsh. *Matroid theory.* Courier Corporation, 2010.

32 Qiqi Yan. Mechanism design via correlation gap. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, pages 710–719. Society for Industrial and Applied Mathematics, 2011.

Extending Partial 1-Planar Drawings

Eduard Eiben
Department of Computer Science, Royal Holloway, University of London, Egham, United Kingdom
eduard.eiben@rhul.ac.uk

Robert Ganian
Algorithms and Complexity Group, TU Wien, Austria
rganian@gmail.com

Thekla Hamm
Algorithms and Complexity Group, TU Wien, Austria
thamm@ac.tuwien.ac.at

Fabian Klute
Algorithms and Complexity Group, TU Wien, Austria
fklute@ac.tuwien.ac.at

Martin Nöllenburg
Algorithms and Complexity Group, TU Wien, Austria
noellenburg@ac.tuwien.ac.at

Abstract

Algorithmic extension problems of partial graph representations such as planar graph drawings or geometric intersection representations are of growing interest in topological graph theory and graph drawing. In such an extension problem, we are given a tuple $(G, H, \mathcal{H})$ consisting of a graph G, a connected subgraph H of G and a drawing $\mathcal{H}$ of H, and the task is to extend $\mathcal{H}$ into a drawing of G while maintaining some desired property of the drawing, such as planarity.

In this paper we study the problem of extending partial 1-planar drawings, which are drawings in the plane that allow each edge to have at most one crossing. In addition we consider the subclass of IC-planar drawings, which are 1-planar drawings with independent crossings. Recognizing 1-planar graphs as well as IC-planar graphs is NP-complete and the NP-completeness easily carries over to the extension problem. Therefore, our focus lies on establishing the tractability of such extension problems in a weaker sense than polynomial-time tractability. Here, we show that both problems are fixed-parameter tractable when parameterized by the number of edges missing from H, i.e., the edge deletion distance between H and G. The second part of the paper then turns to a more powerful parameterization which is based on measuring the vertex+edge deletion distance between the partial and complete drawing, i.e., the minimum number of vertices and edges that need to be deleted to obtain H from G.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Computational geometry

Keywords and phrases Extension problems, 1-planarity, parameterized algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.43

Category Track A: Algorithms, Complexity and Games

Related Version A full version of this article is available at `https://arxiv.org/abs/2004.12222`.

Funding *Robert Ganian*: Robert Ganian acknowledges support by the Austrian Science Fund (FWF, project P31336).
Thekla Hamm: Thekla Hamm acknowledges support by the Austrian Science Fund (FWF, projects P31336 and W1255-N23).

1 Introduction

In the last decade, algorithmic extension problems of partial *planar* graph drawings have received a lot of attention in the fields of graph algorithms and graph theory as well as in graph drawing and computational geometry. In this problem setting, the input consists of a planar graph G, a connected subgraph H of G, and a planar drawing $\mathcal{H}$ of H; the question is then whether $\mathcal{H}$ can be extended to a planar drawing of G. This extension problem is motivated from applications in network visualization, where important patterns (subgraphs) are required to have a special layout, or where new vertices and edges in a dynamic graph must be inserted into an existing (partial) connected drawing, which must remain stable to preserve its mental map [31]. A major result on the extension of partial planar drawings is the linear-time algorithm of Angelini et al. [2] which can answer the above question as well as provide the desired planar drawing of G (if it exists). The result of Angelini et al. contrasts other extension problems in the context of computational geometry and graph drawing, which are typically NP-complete [7, 9–13, 16, 23–26, 30, 32], even in settings where recognition is polynomial-time solvable. On the other end of the planarity spectrum, Arroyo et al. [3, 4] studied drawing extension problems, where the number of crossings per edge is not restricted, yet the drawing must be *simple*, i.e., any pair of edges can intersect in at most one point. They showed that the simple drawing extension problem is NP-complete [3], even if just one edge is to be added [4].

In this paper, we study the algorithmic extension problem of partial drawings of 1-planar graphs, one of the most natural and most studied generalizations of planarity [17, 27, 33], and of partial drawings of IC-planar graphs, a natural restriction of 1-planarity [1, 5, 29, 35]. A 1-planar graph is a graph that admits a drawing in the plane with at most one crossing per edge; for IC-planarity, we additionally require that no two crossed edges are adjacent. Unlike planarity testing, recognizing 1-planar graphs is NP-complete [20, 28], even if the graph is a planar graph plus a single edge [8]. Recognition of IC-planar graphs also remains NP-complete [6].

Contributions. Given a graph G, a connected subgraph H, and a 1-planar drawing $\mathcal{H}$ of H, the 1-Planar Extension problem asks whether $\mathcal{H}$ can be extended by inserting the remaining vertices $V_{\text{add}} = V(G) \setminus V(H)$ and edges $E_{\text{add}} = E(G) \setminus E(H)$ of G into $\mathcal{H}$ while maintaining the property of being 1-planar. The IC-Planar Drawing Extension problem is then defined analogously, but for IC-planarity.

The NP-completeness of these extension problems is a simple consequence of the NP-completeness of the recognition problem [6, 20, 28] (see also Section 3). With this in mind, the aim of this paper is to establish the tractability of the problems when $\mathcal{H}$ is almost a complete 1-planar drawing of G. To capture this setting, we turn to the notion of *fixed-parameter tractability* [15, 19] and consider two natural parameters which capture how complete $\mathcal{H}$ is: the *edge deletion distance* between H and G (denoted by k), and the *vertex+edge deletion distance* between H and G (denoted by κ). More precisely, k is equal to $|E(G) \setminus E(H)|$ and κ is equal to $|V(G) \setminus V(H)| + |E(G[V(H)]) \setminus E(H)|$. We refer to Section 3 for formal definitions and a discussion of the parameters.

After introducing necessary notation in Section 2 and introducing the problem formally in Section 3, we consider the edge deletion distance k in **Section 4**. Our first result is:

▶ **Theorem 1.** *1-Planar Drawing Extension is* FPT *when parameterized by k.*

The proof of Theorem 1 involves the use of several ingredients:

1. Introducing and developing a notion of *patterns*, which are combinatorial objects that capture critical information about the potential interaction of newly added edges with $\mathcal{H}$;
2. a pruning procedure that reduces our instance to an equivalent sub-instance where H has treewidth bounded in k;
3. an embedding graph, which carries information about the drawing $\mathcal{H}$; and finally
4. completing the proof by constructing a formula Φ in Monadic Second Order Logic to check whether a pattern can "fit" in the embedding graph, using Courcelle's Theorem [14].

Next, we turn towards the question of whether one can obtain an *efficient* fixed-parameter algorithm for the extension problem. In particular, due to the use of Courcelle's Theorem [14] to model-check Φ, the algorithm obtained in the proof of Theorem 1 will have a prohibitive dependency on the parameter k. In this direction, we note that it is not immediately obvious how one can design an efficient and "formally clean" purely combinatorial algorithm for the pattern-fitting task (i.e., the task we relegate to model checking Φ in the embedding graph). At the very least, using a direct translation of the model-checking procedure would come at a significant cost in terms of presentation clarity.

That being said, one can observe that the main reason for the use of patterns is that it is not at all obvious where (i.e., in which cell of the drawing) one should place the vertices used to extend $\mathcal{H}$. Indeed, our second result for parameter k assumes that $V_{\text{add}} = \emptyset$ and avoids using Courcelle's Theorem.

▶ **Theorem 2.** *1-Planar Drawing Extension parameterized by k can be solved in time $\mathcal{O}(k^{2k} \cdot n^{\mathcal{O}(1)})$ if $V(G) = V(H)$.*

This algorithm uses entirely different techniques – notably, it prunes the search space for inserting each individual edge via a combination of geometric and combinatorial arguments, and then applies exhaustive branching. We note that the techniques used to prove Theorem 1 and 2 can be directly translated to also obtain analogous results for the IC-planarity setting.

In **Section 5**, we turn our attention to the vertex+edge deletion distance κ as a parameter, which represents a more relaxed way of measuring how complete $\mathcal{H}$ is than k – indeed, while $\kappa \leq k$, it is easy to construct instances where $\kappa = 1$ but k can be arbitrarily large. For our third result, we start with IC-planar drawings.

▶ **Theorem 3.** *IC-Planar Drawing Extension is* FPT *parameterized by κ.*

The proof of Theorem 3 requires a significant extension of the toolkit developed for Theorem 1. The main additional complication lies in the fact that the number of edges that are missing from H is no longer bounded by the parameter. To deal with this, we show that the added vertices can only connect to the boundary of a cell in a bounded number of "ways" (formalized via a notion we call *regions*), and we use this fact to develop a more general notion of patterns and embedding graphs than those used for Theorem 1.

Finally, in **Section 6**, we present a first step towards the tractability of 1-Planar Drawing Extension parameterized by κ. We note that the techniques developed for the other parameterizations and problem variants cannot be applied to solve this case – the main difference compared to the setting of Theorem 3 is that the "missing" vertices can be incident to many edges with crossings, which prevents the use of our bounded-size patterns to capture the behavior of new edges. As our final contribution, we investigate the special case of $\kappa = 2$, i.e., when adding two new vertices.

▶ **Theorem 4.** *1-Planar Drawing Extension is polynomial-time tractable if $\kappa \leq 2$.*

We note that even this, seemingly very restricted, subcase of 1-Planar Drawing Extension was non-trivial and required the combination of several algorithmic techniques (this contrasts to the case of $|V_{\text{add}}| = 1$, whose polynomial-time tractability is a simple corollary of one of our lemmas). In particular, the algorithm uses a new two-step “delimit-and-sweep” approach: first, we apply branching to find a curve with specific properties that bounds the instance by a left and right “delimiter”. The second step is then a left-to-right sweep of the instance that iteratively pushes the left delimiter towards the right one while performing dynamic programming combined with branching and network-flow subroutines.

Albeit being a special case, we believe these delimited instances with two added vertices can play a role in a potential XP algorithm parameterized by κ – the existence of which we leave open for future work.

Further Related Work. In addition to the given related work on extension problems, it is also worth noting that identifying a substructure of bounded treewidth and applying Courcelle’s Theorem to decide an MSO formula on it has been preciously used for a graph drawing problem by Grohe [21], namely to identify graph drawings of bounded crossing number. Both the way in which one arrives at bounded treewidth and the nature of the employed MSO formula are substantially different from our approach, which is not surprising as the problem of generating drawings from scratch and the problem of extending partial drawings are in general fundamentally different. Specifically in the case of generating drawings, the MSO formula could essentially encode the existence of a drawing with bounded crossing number by inductively planarizing crossings of pairs of edges; here the planarity of the planarization can of course be captured via excluded $K_{3,3}$ and K_5 minors by MSO. This approach is not possible in our setting. There are examples of 1-planar graphs which have partial drawings which cannot be extended to a 1-plane drawing. Thus a planarization with respect to the added parts of a solution needs to be compatible with the partial drawing and cannot be encoded by an MSO formula straightforwardly.

2 Preliminaries

Let G be a simple graph, $V(G)$ its vertices, and $E(G)$ its edges. We use standard graph terminology [18]. For $r \in \mathbb{N}$, we write $[r]$ as shorthand for the set $\{1, \ldots, r\}$. We also assume a basic understanding of parameterized complexity theory [15, 19], Monadic Second Order (MSO) Logic and Courcelle’s Theorem [14].

A *drawing* $\mathcal{G}$ of G in the plane $\mathbb{R}^2$ is a function that maps each vertex $v \in V(G)$ to a distinct point $\mathcal{G}(v) \in \mathbb{R}^2$ and each edge $e = uv \in E(G)$ to a simple open curve $\mathcal{G}(e) \subset \mathbb{R}^2$ with endpoints $\mathcal{G}(u)$ and $\mathcal{G}(v)$. In a slight abuse of notation we often identify a vertex v and its drawing $\mathcal{G}(v)$ as well as an edge e and its drawing $\mathcal{G}(e)$. Throughout the paper we will assume that: (i) no edge passes through a vertex other than its endpoints, (ii) any two edges intersect in at most one point, which is either a common endpoint or a proper *crossing* (i.e., edges cannot touch), and (iii) no three edges cross in a single point. For a drawing $\mathcal{G}$ of G and $e \in E(G)$, we use $\mathcal{G} - e$ to denote the drawing of $G - e$ obtained by removing the drawing of e from $\mathcal{G}$, and for $J \subseteq E(G)$ we define $\mathcal{G} - J$ analogously.

We assume that readers are familiar with the notion of *planarity* and *faces*. The *boundary* of a face is the set of edges and vertices whose drawing delimits the face. Further, $\mathcal{G}$ induces for each vertex $v \in V(G)$ a cyclic order of its neighbors by using the clockwise order of its incident edges. This set of cyclic orders is called a *rotation scheme*. Two planar drawings $\mathcal{G}_1$ and $\mathcal{G}_2$ of the same graph G are *equivalent* if they have the same rotation scheme and the same outer face; equivalence classes of planar drawings are also called *embeddings*. A *plane* graph is a planar graph with a fixed embedding.

A drawing $\mathcal{G}$ is *1-planar* if each edge has at most one crossing and a graph G is *1-planar* if it admits a 1-planar drawing. Similarly to planar drawings, 1-planar drawings subdivide the plane into connected regions, which we call *cells* in order to distinguish them from the faces of a planar drawing. The *planarization* $G^\times$ of a 1-planar drawing $\mathcal{G}$ of G is a graph $G^\times$ with $V(G) \subseteq V(G^\times)$ that introduces for each crossing c of $\mathcal{G}$ a *dummy vertex* $c \in V(G^\times)$ and that replaces each pair of crossing edges uv, wx in $E(G)$ by the four *half-edges* uc, vc, wc, xc in $E(G^\times)$, where c is the crossing of uv and wx. In addition all crossing-free edges of $E(G)$ belong to $E(G^\times)$. Obviously, $G^\times$ is planar and the drawing $\mathcal{G}^\times$ of $G^\times$ corresponds to $\mathcal{G}$ with the crossings replaced by the dummy vertices.

3 Extending 1-Planar Drawings

Given a graph G and a subgraph H of G with a 1-planar drawing $\mathcal{H}$ of H, we say that a drawing $\mathcal{G}$ of G is an *extension* of $\mathcal{H}$ if the planarization $H^\times$ of $\mathcal{H}$ and the planarization $\mathcal{G}^\times$ of $\mathcal{G}$ restricted to $\mathcal{H}^\times$ have the same embedding. We formalize our problem of interest as:

1-Planar Drawing Extension
Instance: A graph G, a connected subgraph H of G, and a 1-planar drawing $\mathcal{H}$ of H.
Task: Find an 1-planar extension of $\mathcal{H}$ to G, or correctly identify that there is none.

The IC-Planar Drawing Extension problem is then defined analogously. Both problem definitions follow previously considered drawing extension problems, where the connectivity of H is considered a well-motivated and standard assumption [22, 30, 31].

Given an instance $(G, H, \mathcal{H})$ of 1-Planar Drawing Extension, a *solution* is a 1-planar drawing $\mathcal{G}$ of G that is an extension of $\mathcal{H}$. We refer to $V_{\text{add}} := V(G) \setminus V(H)$ as the *added vertices* and to $E_{\text{add}} := E(G) \setminus E(H)$ as the *added edges*. Let $V_{\text{inc}} = \{v \in V(H) \mid \exists vw \in E_{\text{add}}\}$, i.e., V_{inc} is the set of vertices of H that are incident to at least one added edge. We also distinguish added edges whose endpoints are already part of the drawing, and added edges with at least one endpoint yet to be added into the drawing – notably, we let $E^H_{\text{add}} := \{vw \in E_{\text{add}} \mid v, w \in V(H)\}$ and $E^{\neg H}_{\text{add}} := E_{\text{add}} \setminus E^H_{\text{add}}$. This distinction will become important later, since it opens up two options for how to quantify how “complete” the drawing of $\mathcal{H}$ is. It is worth noting that, without loss of generality, we may assume each vertex in V_{add} to be incident to at least one edge in E_{add} and hence $|V_{\text{add}} \cup V_{\text{inc}}| \leq 2|E_{\text{add}}|$.

Given the NP-completeness of recognizing 1-planar [20, 28] and IC-planar [6] graphs we get as an immediate consequence that also the corresponding extension problems are NP-complete. In view of the NP-completeness of the problem, it is natural to ask about its complexity when H is nearly “complete”, i.e., we only need to extend the drawing $\mathcal{H}$ by a small part of G. In this sense, deletion distance represents the most immediate way of quantifying how far H is from G, and the parameterized complexity paradigm [15, 19] offers complexity classes that provide a more refined view on “tractability” in this setting.

The most immediate way of capturing the completeness of H in this way is to parameterize the problem via the *edge deletion distance* to G – formalized by setting $k = |E_{\text{add}}|$. The aim of Section 4 is to establish the fixed-parameter tractability of 1-Planar Drawing Extension parameterized by k. A second parameter that we consider is the *vertex+edge deletion distance* to G, i.e., the minimum number of vertices and edges that need to be deleted from G to obtain H. We call this parameter κ and set $\kappa = |V_{\text{add}}| + |E^H_{\text{add}}|$. The parameterization by κ is the topic of Section 5 and 6. Since we can always assume that each added vertex is incident to at least one added edge, $|V_{\text{add}}| + |E^H_{\text{add}}| \leq |E_{\text{add}}|$ and so parameterizing by κ leads to a more general (and difficult) parameterized problem.

4 Using Edge Deletion Distance for Drawing Extensions

The main goal of this section is to establish the fixed-parameter tractability of 1-Planar Drawing Extension parameterized by the edge deletion distance k.

We note that one major obstacle faced by a fixed-parameter algorithm is that it is not at all obvious how to decide where the vertices in V_{add} should be drawn in an augmented drawing of H. As a follow-up, we will show that when $V_{\text{add}} = \emptyset$ (i.e., $V(H) = V(G)$), it is possible to obtain a more self-contained combinatorial algorithm with a significantly better runtime; this is presented in Subsection 4.2.

4.1 A Fixed-Parameter Algorithm for 1-Planar Drawing Extension

Our first step towards a proof of the desired tractability result is the definition of a *pattern*, which is a combinatorial object capturing essential information about a potential 1-planar extension of $\mathcal{H}$. The formal definition of pattern is given in Definition 1. Definition 2 then defines the notion of *derived patterns*, which create a link between solutions to an instance of 1-Planar Drawing Extension and patterns.

To given an intuition of the patterns, assume that a pattern consists of a tuple (S, Q, C) and let $(G, H, \mathcal{H})$ be a 1-Planar Drawing Extension instance. Then, the general intuition is that S represents the set of faces in $\mathcal{H}^\times$ which contain at least a part of the drawing of an edge in E_{add} in a hypothetical 1-planar extension $\mathcal{G}$ of $\mathcal{H}$. Crucially, our aim is to keep the size of patterns bounded in k, and so we only “anchor” S to $\mathcal{H}^\times$ by storing information about which faces will contain individual edges in E_{add}, vertices from V_{add}, and be adjacent to individual vertices in V_{inc}; this is captured by the mapping Q. The third piece of information we store is C, which represents the cyclic order of how edges in E_{add} exit or enter the boundary of each face (including the case where an edge crosses through an edge into the same face, i.e., occurs twice when traversing the boundary of that face).

▶ **Definition 1.** *A* pattern *for an instance $(G, H, \mathcal{H})$ is a tuple (S, Q, C) where*

1. *S is a set of at most $2k$ elements;*
2. *Q is a mapping from $V_{add} \cup E_{add} \cup V_{inc}$ which maps:* **(a)** *vertices in V_{add} to elements of S,* **(b)** *edges in E_{add} to ordered pairs of elements of S, and* **(c)** *vertices in V_{inc} to subsets of S.*
3. *C is a mapping from S that maps each $s \in S$ to a cyclically ordered multiset of pairs $((e_1, q_1), (e_2, q_2), \dots, (e_\ell, q_\ell))$, where each e_i is in E_{add} and each q_i is in $V_{inc} \cup \{\text{crossing}\}$. Moreover, C must satisfy the following conditions:* **(a)** *for each $s \in S$ and each tuple $(e, q) \in C(s)$ such that $q \in V_{inc}$, it must hold that $s \in Q(q)$ and e is incident to q in G;* **(b)** *for each $e \in E_{add}$ and $s \in S$, if e occurs in at least one tuple in $C(s)$, then $s \in Q(e)$ and $C(s)$ contains at most two tuples of the form $(e, *)$, where $*$ is an arbitrary element;* **(c)** *for each $s \in S$, each tuple occurs at most once in $C(s)$ with the exception of tuples containing* “crossing”*, which may occur twice.*

Let $\mathcal{P}$ be the set of all patterns for our considered instance $(G, H, \mathcal{H})$. Let $\#\text{pat}(k) = 2k \cdot (2^{2k})^{3k} \cdot ((2k)! \cdot 2^{3k})^{2k}$ and note that $|\mathcal{P}| \leq \#\text{pat}(k) \in 2^{\mathcal{O}(k^2 \log k)}$. In particular, the number of possible patterns can be bounded by first considering $2k$ options for $|S|$, multiplying this by the at most $(2^{2k})^{3k}$-many ways of choosing Q, and finally multiply this by the number of choices for C which can be bounded as follows: for each $s \in S$, $C(s)$ is a set that forms a subset (of size at most $2k$) of the $3k$-cardinality set of tuples (note that $e = \{a, b\} \in E_{\text{add}}$ can only occur in the tuples (e, a), (e, b) and $(e, \text{crossing})$).

The intuition behind patterns will be formalized in the next definition, which creates a link between solutions to our instance and patterns.

▶ **Definition 2.** *Let $(G, H, \mathcal{H})$ be a* 1-Planar Drawing Extension *instance. For each solution $\mathcal{G}$ of $(G, H, \mathcal{H})$ we define a derived pattern $P = (S, Q, C)$ as follows:*

- *S is the set of faces of $\mathcal{H}^\times$ which have a non-empty intersection with $\mathcal{G}(e)$ for some $e \in E_{add}$.*
- *For $v \in V_{add}$ we set $Q(v)$ to the face f of $\mathcal{H}^\times$ for which $\mathcal{G}(v)$ lies inside f, for $e \in E_{add}$ we set $Q(e)$ to the set of at most two faces which have a non-empty intersection with $\mathcal{G}(e)$, and for $w \in V_{inc}$ we set $Q(w)$ to all faces in S incident to w in $\mathcal{G}$.*
- *For a face $s \in S$ we consider all edges $e = uv \in E_{add}$ with a non-empty intersection between $\mathcal{G}(e)$ and s. It follows that there is an edge $e' \in E(H)$ on the boundary of s such that $\mathcal{G}(e)$ crosses $\mathcal{G}(e')$, or $u \in V_{inc}$ and u is on the boundary of s, or both. We set $C(s)$ as the ordered set of these crossing points or vertices when traversing s in clockwise fashion.*

Our next task is to define *valid patterns*; generally speaking, these are patterns which are not malformed and could serve as derived patterns for a hypothetical solution. One notable property that every valid pattern must satisfy is that all vertices and edges mapped by Q to some $s \in S$ can be drawn in a 1-planar way while respecting $C(s)$.

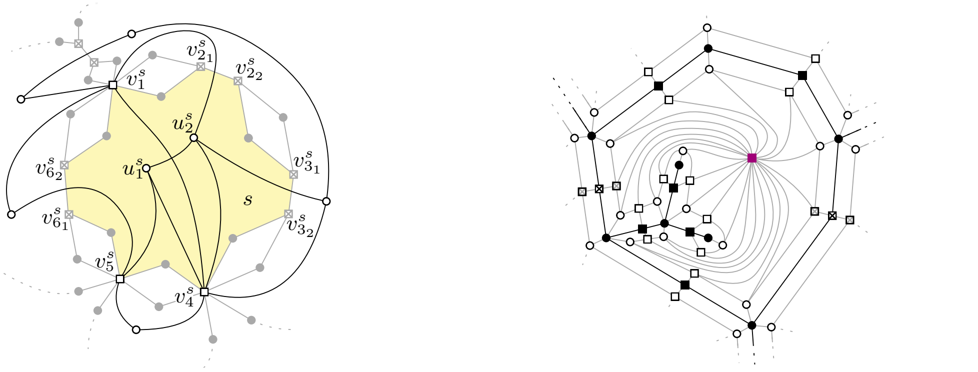
▶ **Definition 3.** *For an instance $(G, H, \mathcal{H})$, a pattern $P = (S, Q, C)$ is* valid *if there exists a* pattern graph *G_P with a 1-planar drawing $\mathcal{G}_P$ satisfying the following properties:*

- *$V_{add} \cup V_{inc} \subseteq V(G_P)$ and $E_{add} \subseteq E(G_P)$.*
- *$\mathcal{G}_P - E_{add}$ is a planar drawing.*
- *S is a subset of the inner faces of $\mathcal{G}_P - E_{add}$.*
- *Each $v \in V_{add}$ is contained in the face $Q(v)$ of $\mathcal{G}_P - V_{add} - E_{add}$.*
- *Each $e \in E_{add}$ is contained in the face(s) $Q(e)$ of $\mathcal{G}_P - E_{add}$.*
- *Each $v \in V_{inc}$ is incident to the faces $Q(v)$ of $\mathcal{G}_P - E_{add}$.*
- *When traversing the inner side of the boundary of each face s of $\mathcal{G}_P - E_{add}$ in clockwise fashion, the order in which each edge $e \in E_{add}$ is seen in $\mathcal{G}_P$ together with the information whether e crosses here or ends in its endpoint in V_{inc}, is precisely $C(s)$.*

Note that the instance $(G, H, \mathcal{H})$ in the definition of a valid pattern is only important to define V_{add}, V_{inc}, and E_{add}. Moreover, observe that for each solution $\mathcal{G}$ of an instance $(G, H, \mathcal{H})$, the derived pattern is valid by definition. An illustration of a pattern graph is provided in Part (a) of Figure 1. We also remark that, when comparing a pattern graph to a hypothetical solution which draws an edge into the outer face of $\mathcal{H}$, we will map the outer face to an inner face of the pattern graph.

▶ **Lemma 4.** *Given pattern $P = (S, Q, C)$, in time $\mathcal{O}((k!)^k \cdot k^{2k+1})$ we can either construct a pattern graph G_P together with the drawing $\mathcal{G}_P$ satisfying all the properties of Definition 3 or decide that P is not valid.*

Proof Sketch. The idea is to build a planarized version G'_P of the pattern graph with size bounded in $\mathcal{O}(k)$. To build the graph we introduce for every $s \in S$ a cycle containing the vertices in $C(s)$. We further subdivide each edge on a cycle by dummy vertices and identify the vertices corresponding to the same vertex in V_{inc} or the same crossing. Finally we add each vertex in V_{add} and connect it to the necessary vertices on the face's boundary. Crossings between these in-face edges can then be guessed since there are only $O(k)$ such crossings. Finding the drawing $\mathcal{G}'_P$ that adheres to Definition 3 can then be done by iterating all possible rotation schemes. To turn $\mathcal{G}'_P$ into a 1-plane drawing replace all crossing vertices by crossing



(a) A pattern graph as constructed in Lemma 4. The face representing $s \in S$ is yellow, gray disks are dummy vertices. Black circles are in V_{add}. Squares are either in V_{inc} or represent crossings.

(b) An example of an embedding graph. The white vertices are shadow-vertices, the purple one the face vertex, and gray edges got added.

Figure 1 Examples for the definition of a pattern and the embedding graph.

edges (see also Part (a) of Figure 1a). For the reverse direction, simply observe that from a given pattern graph we can delete and contract all "unecessary" vertices and edges. Then all that remains are the cycles of vertices representing a set $C(s)$ and vertices in V_{add}. ◀

Next, we will define an annotated ("labeled") graph representation of $\mathcal{H}$ and $\mathcal{H}^{\times}$'s faces. The *embedding graph* H^* of $\mathcal{H}$ is obtained from $\mathcal{H}^{\times}$ by:

1. subdividing each uncrossed edge e (resulting in vertex v_e);
2. creating a vertex for each face f of $\mathcal{H}^{\times}$ (resulting in vertex v_f);
3. traversing the boundary of each face f[1] and whenever we see a vertex v (including the vertices created in Step 1) we create a shadow copy of v and place it right next to v in the direction we saw v from. Add a cycle connecting the shadow vertices we created in f in the order they were created, and direct it in clockwise fashion[2];
4. connecting v_f to all shadow-vertices created by traversing f, and all shadow copies of a vertex v to the original v.

Observe that the embedding graph is a connected plane graph. We label the vertices of the embedding graph to distinguish original vertices, edge-vertices, face-vertices, crossing-vertices and shadow-vertices, and use at most $2k$ special labels to identify vertices in V_{inc}. An illustration of the embedding graph is provided in Part (b) of Figure 1. Next, we show that it suffices to restrict our attention to the parts of H^* which are "close" to vertices in V_{inc}.

▶ **Lemma 5.** *Let $I = (G, H, \mathcal{H})$ be an instance of 1-Planar Drawing Extension. Let Z be the set of all vertices in H^* of distance at least $4k + 7$ from each vertex in V_{inc}. Let G', H', and $\mathcal{H}'$ be obtained by deleting all vertices in Z from G, H, and $\mathcal{H}$ respectively. Then:*

1. *If I is a YES-instance, then each connected component of G' contains at most one connected component of H'*[3]*;*

[1] formally, we draw a curve in f that closely follows the boundary until it forms a closed curve.

[2] Note that this may create multiple shadow copies of a vertex. The reason we use shadow copies of vertices instead of using the original vertices is that when traversing the inner boundary of a face, a vertex may be seen multiple times, and such shadow-vertices allow us to pinpoint from which part of the face we are visiting the given vertex.

[3] This can be seen not to hold in general if we allow H to be disconnected.

2. *I is a YES-instance if and only if for each connected component A of H' the restriction of $\mathcal{H}'$ to $H'[A]$ can be extended to a drawing of the connected component of G' containing A. Moreover, given such a 1-planar extension for every connected component of G', we can output a solution for I in linear time.*

We split the proof of Lemma 5 into proofs for the two individual points.

Proof of Point 1. For the sake of contradiction let J be a connected component of G' that contains two distinct connected components H'_1 and H'_2 of H'. Since J is a connected component, there must be a path P from a vertex $v_1 \in H'_1$ to a vertex $v_2 \in H'_2$ in $J - (H'_1 \cup H'_2)$, and moreover P must have length at most k. By definition, both v_1 and v_2 are in V_{inc}. To complete the proof, it suffices to show that in any solution $\mathcal{G}$, v_1 and v_2 have distance at most $4k+4$ in H^*.

Moreover, in any solution $\mathcal{G}$, two consecutive vertices of P are either drawn in the same face of $\mathcal{H}^\times$ or in two adjacent faces of $\mathcal{H}^\times$. Observe that the distance in H^* between two face-vertices for the faces that share an edge is 4, and that the distance from an original vertex v to a face-vertex of a face incident to v is 2. Therefore, if $(G, H, \mathcal{H})$ is a YES-instance, then the distance between v_1 and v_2 in H^* must be at most $4k+4$. ◀

Proof of Point 2. The forward direction is obvious. For the backward direction, let $G_1, \dots, G_r$ be the connected components of G' and for $i \in [r]$ let H_i and $\mathcal{H}_i$ be the restriction of H' and $\mathcal{H}'$, respectively, to G_i. Moreover, let $\mathcal{H}_i^\times$ be the planarization derived from $\mathcal{H}_i$ and note that H_i is connected for all $i \in [r]$ by Point 1. Now let us fix an arbitrary $i \in [r]$ such that H_i is not empty and let $\mathcal{G}_i$ be a 1-planar extension of $\mathcal{H}_i$ to G_i.

Observe that each face of $\mathcal{H}^\times$ is completely contained in precisely one face of $\mathcal{H}_i^\times$. Moreover, if a face f of $\mathcal{H}_i^\times$ contains at least two faces f_1 and f_2 of $\mathcal{H}^\times$, then both v_{f_1} and v_{f_2} are at distance at least $4k+4$ of any vertex in $V_{\text{inc}} \cap V(H_i)$ in H^*. Indeed, if this were not the case, then w.l.o.g. the vertices on the boundary of v_{f_1} would have distance at most $4k+6$ from some $w \in V_{\text{inc}} \cap V(H_i)$ in H^*, which would mean that f_1 is also a face in $\mathcal{H}_i^\times$. By the same distance-counting argument introduced at the end of the Proof of Point 1, This implies that no edge in a path P of G from a vertex $v \in H_i$ whose internal vertices all lie in V_{add} can be drawn in any face of $\mathcal{H}^\times$ contained in f.

To complete the proof, let $G_1, \dots, G_p$, $p \le r$ be the connected components of G' that contain a vertex in H and $G_{p+1}, \dots, G_r$ the remaining connected components of G'. We obtain a solution $\mathcal{G}$ to the instance I by simply taking the union of $\mathcal{H}$ and $\mathcal{G}_i$ for $i \in [p]$ and then for $i \in \{p+1, \dots, r\}$ shifting $\mathcal{G}_i$ so that $\mathcal{G}_i$ do not intersect any other part of the drawing. ◀

Since $|V_{\text{inc}}| \le 2k$, Lemma 5 allows us to restrict our attention to a subgraph of diameter at most $(4k+7) \cdot 2 \cdot 2k = 16k^2 + 28k$. This will be especially useful in view of the following known fact, that allows us to assume that the treewidth of our instances is bounded.

▶ **Proposition 6** ([34]). *A planar graph G with radius at most r has treewidth at most $3r+1$.*

▶ **Lemma 7.** *1-Planar Drawing Extension is* FPT *parameterized by $k + \text{tw}(H^*)$ if and only if it is* FPT *parameterized by k, where H^* is the embedding graph of $\mathcal{H}$.*

Proof. The backward direction is trivial. For the forward direction, assume that that there exists an algorithm $\mathcal{B}$ which solves 1-Planar Drawing Extension in time $f(k + \text{tw}(H^*)) \cdot |V(G)|^c$ for some constant c and computable function f. Now, consider the following algorithm $\mathcal{A}$ for 1-Planar Drawing Extension: $\mathcal{A}$ takes an instance $(G_0, H_0, \mathcal{H}_0)$ and constructs

$(G_1, H_1, \mathcal{H}_1)$ by applying Lemma 5. Recall that by Point 1 of Lemma 5, $(G_0, H_0, \mathcal{H}_0)$ is either NO-instance, in which case $\mathcal{A}$ correctly outputs "NO", or each connected component of G_1 contains at most one connected component of H_1.

Now let us consider a connected component $\mathcal{C}$ of G_1 and the embedding graph $H_1^*[\mathcal{C}]$ of $\mathcal{H}_1[\mathcal{C}]$ and let v_f be a face-vertex in $H_1^*[\mathcal{C}]$. If v_f is at distance at least $4k+9$ from every vertex in $V_{\text{inc}} \cap \mathcal{C}$ in $H_1^*[\mathcal{C}]$, then every vertex on the boundary of f is at distance at least $4k+7$ from every vertex $w \in V_{\text{inc}} \cap \mathcal{C}$ in $H_1^*[\mathcal{C}]$. Let v be an arbitrary vertex incident to f in $\mathcal{H}_1[\mathcal{C}]$. Since each face of $\mathcal{H}_0^\times$ is completely contained in precisely one face of $\mathcal{H}_1^\times[\mathcal{C}]$, it follows that v is at distance at least $4k+7$ from each vertex $w \in V_{\text{inc}} \cap \mathcal{C}$ in H^*. Because $v \in V(H_1[\mathcal{C}])$, this contradicts the fact that every vertex in $V(H_1)$ is at distance at most $4k+6$ from a vertex $w \in V_{\text{inc}}$ in H^*. Hence, every face-vertex in $H_1^*[\mathcal{C}]$ is at distance at most $4k+8$ from a vertex in $V_{\text{inc}} \cap \mathcal{C}$. Moreover, every vertex in $H_1^*[\mathcal{C}]$ is at distance at most 2 from some face-vertex and there are at most $2k$ vertices in $V_{\text{inc}} \cap \mathcal{C}$. Therefore, the radius, and by Proposition 6 the treewidth, of $H_1^*[\mathcal{C}]$ is bounded by $\mathcal{O}(k^2)$.

Now, for each connected component $\mathcal{C}$ of G_1, we solve the instance $(G_1[\mathcal{C}], H_1[\mathcal{C}], \mathcal{H}_1[\mathcal{C}])$ using algorithm $\mathcal{B}$. If $\mathcal{B}$ determines that at least one such (sub)-instance is a NO-instance, then $\mathcal{A}$ correctly outputs "NO". Otherwise, $\mathcal{A}$ outputs a solution for $(G_0, H_0, \mathcal{H}_0)$ that it computes by invoking the algorithm given by Point 2 of Lemma 5. To conclude, we observe that $\mathcal{A}$ is a fixed-parameter algorithm parameterized by k and its correctness follows from Lemma 5. ◀

We now have all the ingredients we need to establish our tractability result.

▶ **Theorem 1.** *1-Planar Drawing Extension is* FPT *when parameterized by k.*

Proof Sketch. We prove the theorem by showing that 1-Planar Drawing Extension is fixed-parameter tractable parameterized by $k + \text{tw}(H^*)$, which suffices thanks to Lemma 7.

To this end, consider the following algorithm $\mathcal{A}$. Initially, $\mathcal{A}$ loops over all of the at most $\#\text{pat}(k)$ many patterns, tests whether each pattern is valid or not using Lemma 4, and stores all valid patterns in a set $\mathcal{P}$. Next, it branches over all valid patterns in $\mathcal{P}$, and for each such pattern $P = (S = \{s_1, \ldots, s_\ell\}, Q, C)$ it constructs an MSO formula $\Phi_P(\mathcal{F})$, where $\mathcal{F}$ is a set of at most $7k$ free variables specified later, the purpose of which is to find a suitable "placement" for P in $\mathcal{H}$ by finding an interpretation in the embedding graph H^*. In particular, Φ_P uses the free variables in $\mathcal{F}$ to find a suitable face-vertex x_i for each $s_i \in S$ and a suitable crossing point for each edge mapped to two elements of S, while also guaranteeing that the cyclic orders specified by C are adhered to. Once we find a suitable placement for P in $\mathcal{H}$, the algorithm constructs an extension by topologically "inserting" the pattern graph G_P into the identified faces of $\mathcal{H}^\times$ and using the crossing points as well as vertices in V_{inc} as "anchors". ◀

4.2 A More Efficient Algorithm for Extending by Edges Only

In this subsection we obtain a more explicit and efficient algorithm than in Theorem 1 for the case where $V(G) = V(H)$. The idea underlying the algorithm is to iteratively identify sufficiently many 1-planar drawings of each added edge into $\mathcal{H}$ that can either all be extended to a 1-planar drawing of G, or none of them can, which allows us to branch over a small number of possible drawings for that edge.

Let $X = \bigcup_{uv \in E_{\text{add}}} \{u, v\}$ be the set of all endpoints of edges in E_{add}, and let us fix an order of the added edges by enumerating $E_{\text{add}} = \{e_1, \ldots, e_k\}$. Now, consider a 1-planar drawing $\mathcal{H}_i$ of $H_i := H + \{e_1, \ldots, e_{i-1}\}$ and assume that we want to add e_i as a curve

$\gamma(e_i)$. For a cell f in $\mathcal{H}_i + \gamma(e_i)$ and vertices x_1, x_2 on the boundary of f, we denote by $b_{\gamma(e_i)}(f, x_1, x_2) \subseteq E(H_{i+1})$ the edges on the x_1-x_2-path along the boundary of f which traverses this boundary in counterclockwise direction. We explicitly note that $b_{\gamma(e_i)}(f, x_1, x_2)$ does not contain any half-edges. In this way $b_{\gamma(e_i)}(f, x_1, x_2)$ is the set of edges of H_{i+1} on the x_1-x_2-path along the boundary of f that are not crossed in $\mathcal{H}_i + \gamma(e_i)$, and hence may still be crossed by drawings of $e_{i+1}, \dots, e_k$ in a 1-planar extension of $\mathcal{H}_i + \gamma(e_i)$ to G.

Let $\gamma_1(e_i)$ and $\gamma_2(e_i)$ be two possible curves for e_i to be drawn into $\mathcal{H}_i$. Then we call $\gamma_1(e_i)$ and $\gamma_2(e_i)$ $\{e_{i+1}, \dots, e_k\}$*-partition equivalent* if there is a bijection π from the cells of $\mathcal{H}_i + \gamma_1(e_i)$ to the cells of $\mathcal{H}_i + \gamma_2(e_i)$ such that

- the vertices in X on the boundaries of the cells are invariant under π, i.e., for each cell f whose boundary intersects X precisely in X' it must hold that $\pi(f)$ intersects X precisely in X' as well; and
- for each pair of cells f, f' of $\mathcal{H}_i + \gamma_1(e_i)$ and ordered pairs of (not necessarily pairwise distinct) vertices $(x_1, x_2), (x_1', x_2') \in X^2$ that
  - are on the boundary of f and f', respectively, and
  - the counterclockwise x_1-x_2-path and the counterclockwise x_1'-x_2'-path along the boundaries of f and f', respectively, does not contain any inner vertices in X,

  the following must hold:

$$\begin{cases} \text{if } |b_{\gamma_1(e_i)}(f, x_1, x_2) \cap b_{\gamma_1(e_i)}(f', x_1', x_2')| \le k, \text{ then} \\ \quad |b_{\gamma_1(e_i)}(f, x_1, x_2) \cap b_{\gamma_1(e_i)}(f', x_1', x_2')| = |b_{\gamma_2(e_i)}(\pi(f), x_1, x_2) \cap b_{\gamma_2(e_i)}(\pi(f'), x_1', x_2')| \\ \text{otherwise} \\ \quad \text{also } |b_{\gamma_2(e_i)}(\pi(f), x_1, x_2) \cap b_{\gamma_2(e_i)}(\pi(f'), x_1', x_2')| > k. \end{cases}$$

Roughly speaking, the first condition guarantees that when extending $\mathcal{H}_i$ by $\{e_{i+1}, \dots, e_k\}$-partition equivalent drawings of e_i, the topological separation of all vertices that might be important when drawing $e_{i+1}, \dots, e_k$ is the same. The second condition ensures that when extending $\mathcal{H}_i$ by $\{e_{i+1}, \dots, e_k\}$-partition equivalent drawings of e_i, the number of edges whose drawings might be crossed by drawings of $\{e_{i+1} \dots, e_k\}$ is the same, or so large that they cannot all be crossed by drawings of $\{e_{i+1} \dots, e_k\}$.

▶ **Lemma 8.** *For any $1 \le i \le k$, if two drawings $\gamma_1(e_i), \gamma_2(e_i)$ of e_i into a drawing $\mathcal{H}_i$ of H_i are $\{e_{i+1}, \dots, e_k\}$-partition-equivalent, they either both can be extended to a 1-planar drawing of G, or none of them can.*

Proof. We show that we can obtain a 1-planar drawing extension of $\mathcal{H}_i + \gamma_2(e_i)$ to $G = H_i + \{e_{i+1}, \dots, e_k\}$ from a 1-planar drawing extension of $\mathcal{H}_i + \gamma_1(e_i)$ to G. Then the claim immediately follows by a symmetric argument when $\gamma_1(e_i)$ and $\gamma_2(e_i)$ are interchanged.

Let π be a bijection between the cells of $\mathcal{H}_i + \gamma_1(e_i)$ and the cells of $\mathcal{H}_i + \gamma_2(e_i)$ that witnesses $\{e_{i+1}, \dots, e_k\}$-partition equivalence of $\gamma_1(e_i)$ and $\gamma_2(e_i)$. Assume we are given a 1-planar drawing extension $\mathcal{G}_1$ of $\mathcal{H}_i + \gamma_1(e_i)$ to G. From this, we will define a 1-planar drawing extension $\mathcal{G}_2$ of $\mathcal{H}_i + \gamma_2(e_i)$ to G. For $e \in E(H_i)$ set $\mathcal{G}_2(e) = \mathcal{H}_i(e)$ and set $\mathcal{G}_2(e_i) = \gamma_2(e_i)$. In this way, $\mathcal{G}_2$ is an extension of $\mathcal{H}_i$.

Note that for any cell f of $\mathcal{H}_i + \gamma_1(e_i)$ the order in which the vertices of X occur on the boundary of f is the same (up to possibly reversal) in which they occur on the boundary of $\pi(f)$ (exactly the same such vertices occur because of $\{e_{i+1}, \dots, e_k\}$-partition equivalence). This is due to the fact that $\mathcal{H}_i + \gamma_1(e_i)$ and $\mathcal{H}_i + \gamma_2(e_i)$ are obtained from the same drawing $\mathcal{H}_i$ and drawing edges into $\mathcal{H}_i$ merely subdivides cells and cannot permute the order on their boundaries.

Now we can define $\mathcal{G}_2(e_j)$ for $j \in \{i+1,\dots,k\}$ as follows: For $J \subseteq \{i+1,\dots,k\}$ such that $\mathcal{G}_1(e_j)$ intersects two cells f and g of $\mathcal{H}_i + \gamma_1(e_i)$ for every $j \in J$, it holds that each $\mathcal{G}_1(e_j)$ crosses the drawing $(\mathcal{H}_i + \gamma_1(e_i))(c_j)$ of an edge $c_j \in E(H_i) \cup \{e_i\}$. In particular, c_j lies on the shared boundary of f and g. Both f and g contain a vertex in X in their boundary, as each of them contain at least one endpoint of e_j. Hence there are $x_1, x_2 \in X$ that are consecutive on the boundary of f neglecting everything but X, and $y_1, y_2 \in X$ that are consecutive on the boundary of g neglecting everything but X such that $c_j \in b_{\gamma_1(e_i)}(f, x_1, x_2) \cap b_{\gamma_1(e_i)}(g, y_1, y_2)$. By partition-equivalence the boundaries of $\pi(f)$ and $\pi(g)$ each contain an endpoint of each e_j, and because $|J| \leq k$, we find distinct $c'_j \in b_{\gamma_2(e_i)}(\pi(f), x_1, x_2) \cap b_{\gamma_2(e_i)}(\pi(g), y_1, y_2)$ (or possibly $c'_j \in b_{\gamma_2(e_i)}(\pi(f), x_2, x_1) \cap b_{\gamma_2(e_i)}(\pi(g), y_2, y_1)$) for each $j \in J$. Without loss of generality the c_j are indexed in the order in which they occur on the counterclockwise x_1-x_2-path along the boundary of f. We re-index the c'_j to conform to the same order (up to reversal), also taking x_1 and x_2 into account, on $\pi(f)$. ◀

The next lemma shows that the number of non-equivalent drawings is bounded by a function of k, which in turn allows us to apply exhaustive branching to prove the theorem.

▶ **Lemma 9.** *For any $1 \leq i \leq k$, the number of ways to draw e_i into a drawing $\mathcal{H}_i$ of H_i that are pairwise not $\{e_{i+1},\dots,e_k\}$-partition-equivalent is at most $4(2k+1) \cdot 2(k+1) \in \mathcal{O}(k^2)$.*

▶ **Theorem 2.** *1-Planar Drawing Extension parameterized by k can be solved in time $\mathcal{O}(k^{2k} \cdot n^{\mathcal{O}(1)})$ if $V(G) = V(H)$.*

Proof. We can pre-compute the intersection of the boundary of each cell of $\mathcal{H}$ with X and for each pair of cells f, f' of $\mathcal{H}$ and ordered pairs of vertices $x_1, x_2 \in X$ and x'_1, x'_2 that are consecutive on the boundaries of f and f' respectively if one neglects everything but X, the cardinality of the set of edges that are on the clockwise x_1-x_2-path along the boundary of f and at the same time on the clockwise x'_1-x'_2-path along the boundary of f' in polynomial time.

As described in the proof of Lemma 9, at any stage, for $1 \leq i \leq k$, we can branch on $\{e_{i+1},\dots,e_k\}$-partition-equivalent drawings $\gamma(e)$ of e using the pre-computed information. This information can be modified within each branch according to the choice of $\gamma(e)$ in constant time because, as described in the proof of Lemma 9 the impact of $\gamma(e)$ involves only few values whose modifications can correctly be computed from the updated pre-computed information up to this stage and the chosen values determining $\gamma(e)$. Correctness of this branching follows from Lemma 8. ◀

5 Using Vertex+Edge Deletion Distance for IC-Planar Drawing Extension

In this section, we show that IC-Planar Drawing Extension parameterized by κ is fixed-parameter tractable. We note that an immediate consequence of this is the fixed-parameter tractability of IC-Planar Drawing Extension parameterized by k.

On a high level, our strategy is similar to the one used to prove Theorem 1, in the sense that we also use a (more complicated) variant of the patterns along with Courcelle's Theorem. However, obtaining the result requires us to extend the previous proof technique to accommodate the fact that the number of edges incident to V_{add}, and hence the size of a pattern, is no longer bounded by κ. This is achieved by identifying so-called *difficult vertices* and *regions* that split up the neighborhood of each face-vertex in the embedding graph into a small number of sections (a situation which can then be handled by a formula in Monadic

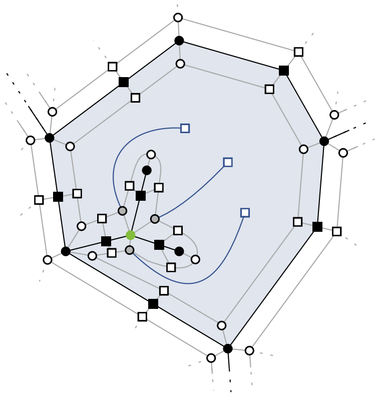


Figure 2 An example of H^* where a vertex $v' \in V_{\text{inc}}$ has several non-difficult shadow copies. Blue vertices are in V_{add}. The green vertex has no difficult shadow copy w.r.t. the blue face in $\mathcal{H}^\times$.

Second Order logic). Less significant complications are that we need a stronger version of Lemma 5 to ensure that the diameter of the resulting graph is bounded, and need to be more careful when using MSO logic in the proof of the main theorem.

Let f be a face of $\mathcal{H}^\times$ and let $\mathcal{G}$ be a solution (i.e., an IC-planar drawing of G) for the instance $(G, H, \mathcal{H})$. Let H^* be the embedding graph of $\mathcal{H}$, and without loss of generality let us assume (via topological shifting) that each edge between a vertex a' on the boundary of f and a vertex $b \in V_{\text{add}}$ placed by $\mathcal{G}$ in f is routed "through" one shadow copy of a'[4]. Let V_{add}^f be the subset of V_{add} drawn by $\mathcal{G}$ in the face f.

Observe that, since shadow vertices are not part of the original instance and instead merely mark possible "parts" of the face that can be used to access a given vertex, it may happen that a solution routes several edges through one shadow vertex. We say that a shadow vertex $v \in N_{H^*}(v_f)$ (where $N_{H^*}(v_f)$ denotes the neighborhood of v_f in H^*) is *difficult* w.r.t. f if $\mathcal{G}$ routes at least two edges through v. Note that it may happen that a vertex $v' \in V_{\text{inc}}$ with more than one neighbor in V_{add} has several shadow copies, none of which are difficult (see Figure 2).

▶ **Lemma 10.** *There are at most $3\kappa^2$ difficult vertices w.r.t. a face f of $\mathcal{H}^\times$.*

Proof. We show that any two of the ℓ added vertices drawn into f in $\mathcal{G}$ are both connected to at most 3 vertices in $N_{H^*}(v_f)$. Then the claim follows. Assume for contradiction that $v_1, v_2 \in V_{\text{add}}$ are drawn into f in $\mathcal{G}$ and $w_1, w_2, w_3, w_4 \in N_{H^*}(v_f)$ are shadow vertices that each route two edges, one of which is incident to v_1 and one of which is incident to v_2. Since H is connected, the boundary of f is connected, and by construction of H^*, w_1, w_2, w_3, w_4 all lie on a cycle in H^* that does not involve any of $\{v_{f'} \mid f' \text{ face in } \mathcal{H}^\times\}$. Hence the following graph H' is a minor of $H^* - \{v_{f'} \mid f' \text{ face in } \mathcal{H}^\times\} + V_{\text{add}} + E_{\text{add}}$: $H' = (\{v_1, v_2, w_1, \dots, w_4, v_f\}, \{v_i w_j \mid i \in \{1, 2, f\}, j \in \{1, \dots, 4\} \cup \{w_i w_{(i \bmod 4)+1} \mid i \in \{1, \dots, 4\})$. H' does not admit a 1-planar drawing in which both v_1 and v_2 lie on the same side of the drawing of the w_1-w_2-w_3-w_4-cycle and v_1 and v_2 are each incident to at most one edge whose drawing is crossed. However the existence of $\mathcal{G}$ implies that exactly such a drawing of H' exists. ◀

[4] The reason one distinguishes which shadow copy of a' the edge is routed through is because this unambiguously identifies which part of the face the edge uses to access a'.

A region R of a vertex $x \in V_{\mathrm{add}}^f$ (or, equivalently, of a face f) is a maximal path $(r_1, \ldots, r_p)$ in $N_{H^*}(v_f)$ with the following properties: (1) $\mathcal{G}$ does not route through any shadow copy of an edge in R; (2) for each vertex r in R, a uncrossed curve can be drawn in $\mathcal{G}$ inside f between r and x; (3) none of the vertices in R are adjacent to $V_{\mathrm{add}}^f \setminus \{x\}$; and (4) r_1 and r_p are adjacent to x.

▶ **Lemma 11.** *There are at most 3κ regions of a face f.*

Proof. Consider a path P in $H^*[N_{H^*}(v_f)]$ that traverses all ℓ regions of a vertex $v \in V_{\mathrm{add}}^f$. It contains at least $\ell - 1$ pairwise disjoint subpaths $P_1, \ldots, P_{\ell-1}$ of paths connecting regions of v that are consecutive in P.

For every P_i ($i \in \{1, \ldots, \ell\}$), by the property that regions are inclusion maximal paths of vertices with certain properties, we find some vertex x in P_i that has to violate one of these properties. This can happen in three ways:

(1) x is a shadow copy of an edge and $\mathcal{G}$ routes through x,
(2) (1) is not the case and the drawing of some edge $e \in E_{\mathrm{add}}$ separates x from v in $\mathcal{H}$, or
(3) (1) and (2) are not the case and x is adjacent to another vertex $w \in V_{\mathrm{add}}^f \setminus \{v\}$.

In case (1) there is an edge $e \in E_{\mathrm{add}}$ such that the drawing of e crosses the boundary of f in $\mathcal{H}$ and routes through x.

In case (2) the edge in question has both endpoints on P, thus $e \in E_{\mathrm{add}}^H$ or the drawing of e crosses either the boundary of f in $\mathcal{H}$ or a drawing of another added edge.

There are at most $|E_{\mathrm{add}}^H|$ edges in E_{add}^H and at most $|V_{\mathrm{add}}|$ edges in $E_{\mathrm{add}} \setminus E_{\mathrm{add}}^H$ that can cross another edge in an IC-planar drawing. Moreover, in both cases (1) and (2), the endpoints of e cannot occur in any P_j with $j \in \{1, \ldots, \ell\} \setminus \{i\}$.

In case (3) either x is contained in a region of w or the drawing of xw in $\mathcal{H}$ crosses an edge and is the only drawing of an edge incident to w that does so. If x is in a region of w, w is separated from $N_{H^*}(v_f) \setminus P_i$ by the edges from v to the outermost vertices of the regions that P_i connects and hence can have no region outside of P_i. There are at most $|V_{\mathrm{add}}|$ many such w.

Thus we find at most $|E_{\mathrm{add}}^H| + |V_{\mathrm{add}}| + 2|V_{\mathrm{add}}|$ such x on P in total and thus $\ell \leq |E_{\mathrm{add}}^H| + |V_{\mathrm{add}}| + 2|V_{\mathrm{add}}| \leq 3\kappa$. This concludes the proof. ◀

The underlying intuition one should keep about regions and difficult vertices is that a solution $\mathcal{G}$ partitions the shadow vertices into those which **(a)** have no edges routed through them, **(b)** have precisely one edge routed through them (in which case they must be part of the respective region), and **(c)** have at least two edges routed through them (in which case they form a difficult vertex).

In the remainder of this section we give some intuition on how the techniques used to prove Theorem 1 need to be extended to obtain Theorem 3. Unlike in the proof of Theorem 1 the number of vertices in V_{inc} is not bounded by κ. Consequently, we cannot track their exact placement through patterns as defined in Definition 1. To circumvent this, we define *extended patterns* that store the necessary details about the cyclic orders in which individual regions, difficult vertices together with crossings, and endpoints of edges in E_{add}^H as well as the single edge per vertex in V_{add} that is allowed to cross, are supposed to appear inside a face. Then, as for patterns, we define how an extended pattern is *derived* from potential solutions.

The proof then proceeds by following the strategy laid down in Subsection 4.1. In particular, we define a notion of validity along with *pattern graphs* for extended patterns (cf. Definition 3). We show that validity can be checked and pattern graphs can be constructed efficiently (cf. Lemma 4). We use analogues to Lemma 5 and Lemma 7 to prune our instances so that they have bounded treewidth. At this point, we have all the ingredients we need to prove our final Theorem 3 – the main complication here compared to the proof of Theorem 1 is that we cannot have explicit labels identifying individual vertices in V_{inc}.

▶ **Theorem 3.** *IC-Planar Drawing Extension is* FPT *parameterized by* κ.

6 Inserting Two Vertices into a 1-Plane Drawing

In this section we show that 1-Planar Drawing Extension is polynomial-time tractable in the case where we are only adding 2 vertices to the graph along with their incident edges (i.e., when $|V_{\text{add}}| = 2$ and $E^H_{\text{add}} = \emptyset$[5]). Already solving this, at first glance simple, case seems to require non-trivial insight into the problem. In the following we call the two vertices in V_{add} the *red* and *blue* vertex, denoted by r and b, respectively.

On a high level, our algorithm employs a "delimit-and-sweep" approach. First, it employs exhaustive branching to place the vertices and identify a so-called "initial delimiter" – a Jordan curve that isolates a part of our instance that we need to focus on. In the second step, it uses such an initial delimiter to solve the instance via a careful dynamic programming subroutine. As our very first step, we exhaustively branch to determine which cells r and b should be drawn in, in $\mathcal{O}(n^2)$ time, and in each branch we add r and b into the selected cell(s) (from now on, we consider these embeddings part of $\mathcal{H}$).

The Flow Subroutine. Throughout this section, we will employ a generic network-flow subroutine that allows us to immediately solve certain restricted instances of 1-Planar Drawing Extension. In particular, assuming we are in the setting where r and b have already been inserted into $\mathcal{H}$, consider the situation where:

- There is a partial mapping λ from the faces of $\mathcal{H}^\times$ to $\{R, B\}$; and
- r and b are in different cells of $\mathcal{H}$.

We say a 1-planar extension of $\mathcal{H}$ to G is λ-*consistent* if the drawing of any edge in $E(G) \setminus E(H)$ which is incident to r intersects the interior of face F of $\mathcal{H}^\times$ only if $\lambda(F) = R$, and correspondingly the drawing of any edge in $E(G) \setminus E(H)$ which is incident to b intersects the interior of face F of $\mathcal{H}^\times$ only if $\lambda(F) = B$ (i.e., λ specifies precisely which kind of edges may enter which face). We use a reduction to network flows to show:

▶ **Lemma 12.** *Given λ as above, it is possible to determine whether there exists a λ-consistent 1-planar extension of $\mathcal{H}$ to G in polynomial time.*

Proof Sketch. Consider the max flow instance θ_1 constructed as follows. θ_1 contains a universal sink t and a universal source s. We add one vertex for each vertex in $N_{E(G)\setminus E(H)}(r)$, and a capacity-1 edge from each such "R-vertex" to t. We add one "f-vertex" for each face f in $\mathcal{H}^\times$ that λ maps to R, and a capacity-1 edge from each such vertex to every R-vertex that lies on the boundary of f. We add an (unlimited-capacity) edge from s to every f-vertex whose face contains r (possibly on its boundary). Finally, we add an edge from every f-vertex whose face contains r to each other f'-vertex of capacity equal to the number of crossable edges that lie on the shared boundary of f and f'. The instance θ_2 is constructed in an analogous fashion for B and b. To conclude the proof, it suffices to show that the drawings of the edges in E_{add} incident to r in a λ-consistent extension $\mathcal{G}$ of $\mathcal{H}$ to G correspond to s-t-flows of values $|N_{E(G)\setminus E(H)}(r)|$ and $|N_{E(G)\setminus E(H)}(b)|$ for θ_1 and θ_2, respectively. ◀

[5] We note that it is trivial to extend the result to the case where the number of added edges is bounded by a fixed constant, via simple exhaustive branching.

▶ **Corollary 13.** *1-Planar Drawing Extension for $|V_{add}| = 1$ and $E^H_{add} = \emptyset$ can be solved in polynomial time.*

Finding an Initial Delimiter. We begin by formally defining the following notion:

▶ **Definition 14.** *A Jordan curve ω in the plane is an* initial delimiter *for $\mathcal{H}$ if:*

1. *ω passes through both r and b but through no other vertex of $\mathcal{H}$,*
2. *whenever ω shares at most one point with the interior of an edge, this point is a proper crossing between ω and that edge,*
3. *the intersection between $\mathcal{H}$ and the exterior of ω (including ω itself) contains a single cell c_r whose boundary contains r, and a single cell c_b whose boundary contains b, and*
4. *the intersection of the boundary of c_r (resp. c_b) and ω is a single simple curve containing r (resp. b) as an interior point.*

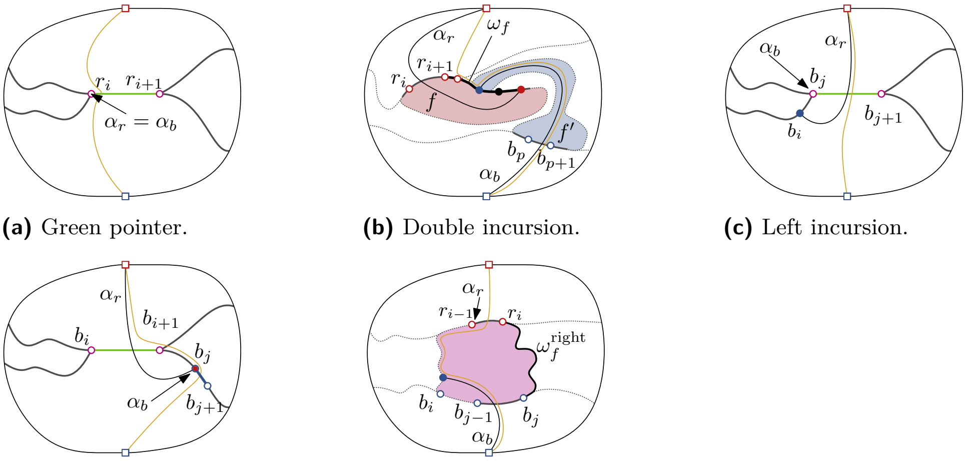
Intuitively, the third condition means that if we add ω onto $\mathcal{H}$, then there are unique cells in the exterior of ω for r and b. A solution for our instance, i.e., a drawing $\mathcal{G}$ of G, is ω-*compatible* if every edge from E_{add} is drawn in the exterior region defined by ω. We state the main result of this subsection below – intuitively, it provides us with a set of initial delimiters that we can exhaustively branch over, and in each branch we can restrict our attention to solutions that are compatible with the chosen initial delimiter. To avoid confusion, we note that the lemma covers the case where r and b are placed in the same cell f (by branching on and placing a constant number of new edges that "separate" the boundary of f).

▶ **Lemma 15.** *For every instance $(G, H, \mathcal{H})$ where $|V_{add}| \leq 2$ and $E^H_{add} = \emptyset$, we can in polynomial time either solve $(G, H, \mathcal{H})$ or construct a set Q of initial delimiters with the following property (or both): if $(G, H, \mathcal{H})$ admits a solution, then Q contains at least one ω such that $(G, H, \mathcal{H})$ also admits an ω-compatible solution.*

Dynamic Programming. We can now proceed with a very high-level sketch of how the algorithm proceeds once we have pre-selected (via branching) an initial delimiter. The general idea is to perform a left-to-right sweep of $\mathcal{H}$ by starting with the boundaries provided by the initial delimiter. The runtime of the algorithm is upper-bounded by the fact that it relies on dynamic programming where the maximum size of the records (representing possible "positions" on our sweep) is polynomial in the input size, and where the possibility of transitioning from one record to the next can be checked in polynomial time. In particular, the "steps" we use to move from one record to the next relies on a situational combination of exhaustive branching and the network-flow subroutine described in Lemma 12.

The intuitive reason we need to combine both of these techniques is that in some parts of our sweep, we will encounter faces where edges from both r and b may enter – there the interactions between these edges are too complicated to be modeled as a simple flow problem, but (as we will show) we can identify separating curves that cut our instance into parts for which we have a mapping λ that can be applied in conjunction with Lemma 12.

We now formalize the records used by our algorithm: a *record* is a tuple (α_r, α_b, T), where α_r is either a vertex in $\mathcal{H}$ or an edge $e \in E_{\text{add}}$ incident to r. α_b is defined symmetrically, and we describe such edges by specifying their endpoint and potential crossing point. T is then an auxiliary element that specifies the "type" of a given record. Finally, we associate each record with a *delimiter* that iteratively pushes our "left" initial delimiter boundary towards the "right" one; everything to the left of the *delimiter* can be ignored, since the assumptions



(a) Green pointer.

(b) Double incursion.

(c) Left incursion.

(d) Right incursion.

(e) Slice.

Figure 3 The five types of records for the dynamic program. The computed delimiters are the orange curves. r and b are the rectangular vertices, colored disks are in R and B, white disks are r_i and b_i depending on their border-color.

we use to select our records guarantee that it cannot be crossed by an edge of the targeted solution.

While we are forced to omit the majority of the details of the algorithm due to space constraints, below we at least provide a brief, intuitive summary of the 5 types of records used (see also Figure 3):

1. **Green Pointer.** $\alpha_r = \alpha_b$ is a vertex incident to an edge on the boundary of c_r and c_b.
2. **Double Incursion.** One edge "covers" a part of one face that is accessed by the other edge from the other side.
3. **Left Incursion.** α_r (or α_b) crosses into c_b (or c_a) and heads "left".
4. **Right Incursion.** α_r (or α_b) crosses into c_b (or c_a) and heads "right".
5. **Slice.** One or both edges cross into a face other than c_r and c_b.

Altogether, by using these records and carefully analyzing the cases that allow us to transition from one record to the other, we obtain a proof of:

▶ **Theorem 4.** *1-Planar Drawing Extension is polynomial-time tractable if $\kappa \leq 2$.*

7 Concluding Remarks

In this paper, we initiated the study of the problem of extending partial 1-planar and IC-planar drawings by providing several parameterized algorithms that target cases where only a few edges and/or vertices are missing from the graph. Our results follow up on previous seminal work on extending planar drawings, but the techniques introduced and used here are fundamentally different [2]. The by far most prominent question left open in our work concerns the (not only parameterized, but also classical) complexity of 1-Planar Extension w.r.t. κ. In particular, can one show that the problem is, at least, polynomial-time tractable for fixed values of κ? While the results presented in Section 6 are a promising start in this direction, it seems that new ideas are needed to push beyond the two-vertex case.

Follow-up work may also focus on extending other types of beyond planar drawings [17].

References

1 Michael Albertson. Chromatic number, independence ratio, and crossing number. *ARS MATHEMATICA CONTEMPORANEA*, 1(1):1–6, 2008. doi:10.26493/1855-3974.10.2d0.

2 Patrizio Angelini, Giuseppe Di Battista, Fabrizio Frati, Vít Jelínek, Jan Kratochvíl, Maurizio Patrignani, and Ignaz Rutter. Testing planarity of partially embedded graphs. *ACM Trans. Algorithms*, 11(4):32:1–32:42, 2015. doi:10.1145/2629341.

3 Alan Arroyo, Martin Derka, and Irene Parada. Extending simple drawings. In Daniel Archambault and Csaba D. Tóth, editors, *27th International Symposium on Graph Drawing and Network Visualization (GD'19)*, volume 11904 of *LNCS*, pages 230–243. Springer, 2019. doi:10.1007/978-3-030-35802-0_18.

4 Alan Arroyo, Fabian Klute, Irene Parada, Raimund Seidel, Birgit Vogtenhuber, and Tilo Wiedera. Extending simple drawings with one edge is hard. *CoRR*, abs/1909.07347, 2019. Accepted for presentation at WG'20. arXiv:1909.07347.

5 Franz J. Brandenburg. Recognizing IC-planar and NIC-planar graphs. *J. Graph Algorithms Appl.*, 22(2):239–271, 2018. doi:10.7155/jgaa.00466.

6 Franz J. Brandenburg, Walter Didimo, William S. Evans, Philipp Kindermann, Giuseppe Liotta, and Fabrizio Montecchiani. Recognizing and drawing IC-planar graphs. *Theor. Comput. Sci.*, 636:1–16, 2016. doi:10.1016/j.tcs.2016.04.026.

7 Guido Brückner and Ignaz Rutter. Partial and constrained level planarity. In Philip N. Klein, editor, *Discrete Algorithms (SODA'17)*, pages 2000–2011. SIAM, 2017. doi:10.1137/1.9781611974782.130.

8 Sergio Cabello and Bojan Mohar. Adding one edge to planar graphs makes crossing number and 1-planarity hard. *SIAM J. Comput.*, 42(5):1803–1829, 2013. doi:10.1137/120872310.

9 Erin W. Chambers, David Eppstein, Michael T. Goodrich, and Maarten Löffler. Drawing graphs in the plane with a prescribed outer face and polynomial area. *J. Graph Algorithms Appl.*, 16(2):243–259, 2012. doi:10.7155/jgaa.00257.

10 Timothy M. Chan, Fabrizio Frati, Carsten Gutwenger, Anna Lubiw, Petra Mutzel, and Marcus Schaefer. Drawing partially embedded and simultaneously planar graphs. *J. Graph Algorithms Appl.*, 19(2):681–706, 2015. doi:10.7155/jgaa.00375.

11 Steven Chaplick, Paul Dorbec, Jan Kratochvíl, Mickaël Montassier, and Juraj Stacho. Contact representations of planar graphs: Extending a partial representation is hard. In Dieter Kratsch and Ioan Todinca, editors, *40th International Workshop on Graph-Theoretic Concepts in Computer Science (WG'14)*, volume 8747 of *LNCS*, pages 139–151. Springer, 2014. doi:10.1007/978-3-319-12340-0_12.

12 Steven Chaplick, Radoslav Fulek, and Pavel Klavík. Extending partial representations of circle graphs. In Stephen K. Wismath and Alexander Wolff, editors, *21st International Symposium on Graph Drawing (GD'13)*, volume 8242 of *LNCS*, pages 131–142. Springer, 2013. doi:10.1007/978-3-319-03841-4_12.

13 Steven Chaplick, Grzegorz Guspiel, Grzegorz Gutowski, Tomasz Krawczyk, and Giuseppe Liotta. The partial visibility representation extension problem. *Algorithmica*, 80(8):2286–2323, 2018. doi:10.1007/s00453-017-0322-4.

14 Bruno Courcelle. The monadic second-order logic of graphs. I. recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

15 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms.* Springer, 2015. doi:10.1007/978-3-319-21275-3.

16 Giordano Da Lozzo, Giuseppe Di Battista, and Fabrizio Frati. Extending upward planar graph drawings. In Zachary Friggstad, Jörg-Rüdiger Sack, and Mohammad R. Salavatipour, editors, *16th International Symposium on Algorithms and Data Structures (WADS'19)*, volume 11646 of *LNCS*, pages 339–352. Springer, 2019. doi:10.1007/978-3-030-24766-9_25.

17 Walter Didimo, Giuseppe Liotta, and Fabrizio Montecchiani. A survey on graph drawing beyond planarity. *ACM Comput. Surv.*, 52(1):4:1–4:37, 2019. doi:10.1145/3301281.

18 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

19 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. doi:10.1007/978-1-4471-5559-1.

20 Alexander Grigoriev and Hans L. Bodlaender. Algorithms for graphs embeddable with few crossings per edge. *Algorithmica*, 49(1):1–11, 2007. doi:10.1007/s00453-007-0010-x.

21 Martin Grohe. Computing crossing numbers in quadratic time. *Journal of Computer and System Sciences*, 68(2):285–302, 2004. Special Issue on STOC 2001. doi:10.1016/j.jcss.2003.07.008.

22 Seok-Hee Hong and Hiroshi Nagamochi. Convex drawings of graphs with non-convex boundary constraints. *Discret. Appl. Math.*, 156(12):2368–2380, 2008.

23 Pavel Klavík, Jan Kratochvíl, Tomasz Krawczyk, and Bartosz Walczak. Extending partial representations of function graphs and permutation graphs. In Leah Epstein and Paolo Ferragina, editors, *20th Annual European Symposium on Algorithms (ESA'12)*, volume 7501 of *LNCS*, pages 671–682. Springer, 2012. doi:10.1007/978-3-642-33090-2_58.

24 Pavel Klavík, Jan Kratochvíl, Yota Otachi, Ignaz Rutter, Toshiki Saitoh, Maria Saumell, and Tomás Vyskocil. Extending partial representations of proper and unit interval graphs. *Algorithmica*, 77(4):1071–1104, 2017. doi:10.1007/s00453-016-0133-z.

25 Pavel Klavík, Jan Kratochvíl, Yota Otachi, and Toshiki Saitoh. Extending partial representations of subclasses of chordal graphs. *Theor. Comput. Sci.*, 576:85–101, 2015. doi:10.1016/j.tcs.2015.02.007.

26 Pavel Klavík, Jan Kratochvíl, Yota Otachi, Toshiki Saitoh, and Tomás Vyskocil. Extending partial representations of interval graphs. *Algorithmica*, 78(3):945–967, 2017. doi:10.1007/s00453-016-0186-z.

27 Stephen G. Kobourov, Giuseppe Liotta, and Fabrizio Montecchiani. An annotated bibliography on 1-planarity. *Computer Science Review*, 25:49–67, 2017. doi:10.1016/j.cosrev.2017.06.002.

28 Vladimir P. Korzhik and Bojan Mohar. Minimal obstructions for 1-immersions and hardness of 1-planarity testing. *Journal of Graph Theory*, 72(1):30–71, 2013. doi:10.1002/jgt.21630.

29 Giuseppe Liotta and Fabrizio Montecchiani. L-visibility drawings of IC-planar graphs. *Inf. Process. Lett.*, 116(3):217–222, 2016. doi:10.1016/j.ipl.2015.11.011.

30 Tamara Mchedlidze, Martin Nöllenburg, and Ignaz Rutter. Extending convex partial drawings of graphs. *Algorithmica*, 76(1):47–67, 2016. doi:10.1007/s00453-015-0018-6.

31 Kazuo Misue, Peter Eades, Wei Lai, and Kozo Sugiyama. Layout adjustment and the mental map. *J. Visual Languages and Computing*, 6(2):183–210, 1995. doi:10.1006/jvlc.1995.1010.

32 Maurizio Patrignani. On extending a partial straight-line drawing. *Int. J. Found. Comput. Sci.*, 17(5):1061–1070, 2006. doi:10.1142/S0129054106004261.

33 Gerhard Ringel. Ein Sechsfarbenproblem auf der Kugel. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, 29(1):107–117, 1965. doi:10.1007/BF02996313.

34 Neil Robertson and Paul D. Seymour. Graph minors. III. planar tree-width. *J. Comb. Theory, Ser. B*, 36(1):49–64, 1984. doi:10.1016/0095-8956(84)90013-3.

35 Wanshun Yang, Weifan Wang, and Yiqiao Wang. Acyclic coloring of IC-planar graphs. *Discret. Math.*, 342(12), 2019. doi:10.1016/j.disc.2019.111623.

How to Hide a Clique?

Uriel Feige
The Weizmann Institute, Rehovot, Israel
uriel.feige@weizmann.ac.il

Vadim Grinberg
Toyota Technological Institute at Chicago, IL, USA
vgm@ttic.edu

Abstract

In the well known planted clique problem, a clique (or alternatively, an independent set) of size k is planted at random in an Erdos-Renyi random $G(n, p)$ graph, and the goal is to design an algorithm that finds the maximum clique (or independent set) in the resulting graph. We introduce a variation on this problem, where instead of planting the clique at random, the clique is planted by an adversary who attempts to make it difficult to find the maximum clique in the resulting graph. We show that for the standard setting of the parameters of the problem, namely, a clique of size $k = \sqrt{n}$ planted in a random $G(n, \frac{1}{2})$ graph, the known polynomial time algorithms can be extended (in a non-trivial way) to work also in the adversarial setting. In contrast, we show that for other natural settings of the parameters, such as planting an independent set of size $k = \frac{n}{2}$ in a $G(n, p)$ graph with $p = n^{-\frac{1}{2}}$, there is no polynomial time algorithm that finds an independent set of size k, unless NP has randomized polynomial time algorithms.

2012 ACM Subject Classification Theory of computation

Keywords and phrases planted clique, semi-random model, Lovasz theta function, random graphs

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.44

Category Track A: Algorithms, Complexity and Games

Related Version A full version of the paper is available at `https://arxiv.org/abs/2004.12258`.

Funding *Uriel Feige*: Supported in part by the Israel Science Foundation (grant No. 1388/16).

1 Introduction

The planted clique problem, also referred to as hidden clique, is a problem of central importance in the design of algorithms. We introduce a variation of this problem where instead of planting the clique at random, an adversary plants the clique. Our main results are that in certain regimes of the parameters of the problem, the known polynomial time algorithms can be extended to work also in the adversarial settings, whereas for other regimes, the adversarial planting version becomes NP-hard. We find the results interesting for three reasons. One is that they concern an extensively studied problem (planted clique), but from a new direction, and we find that the results lead to a better understanding of what aspects of the planted clique problem are made use of by the known algorithms. Another is that extending the known algorithms (based on semidefinite programming) to the adversarial planted setting involves some new techniques regarding how semidefinite programming can be used and analysed. Finally, the NP-hardness results are interesting as they are proven in a semi-random model in which most of the input instance is random, and the adversary controls only a relatively small aspect of the input instance. One may hope that this brings us closer to proving NP-hardness results for purely random models, a task whose achievement would be a breakthrough in complexity theory.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 44; pp. 44:1–44:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1.1 The random planted clique model

Our starting point is the Erdos-Renyi $G(n,p)$ random graph model, which generates graphs on n vertices, and every two vertices are connected by an edge independently with probability p. We start our discussion with the special case in which $p = \frac{1}{2}$, and other values of p will be considered later. Given a graph G, let $\omega(G)$ denote the size of the maximum clique in G, and let $\alpha(G)$ denote the size of the maximum independent set. Given a distribution D over graphs, we use the notation $G \sim D$ for denoting a graph sampled at random according to D. The (edge) complement of a graph $G \sim G(n, \frac{1}{2})$ is by itself a graph sampled from $G(n, \frac{1}{2})$, and the complement of a clique is an independent set, and hence the discussion concerning cliques in $G(n, \frac{1}{2})$ extends without change to independent sets (and vice versa).

It is well known (proved by computing the expectation and variance of the number of cliques of the appropriate size) that for $G \sim G(n, \frac{1}{2})$, w.h.p. $\omega(G) \simeq 2\log n$ (the logarithm is in base 2). However, there is no known polynomial time algorithm that can find cliques of size $2\log n$ in such graphs. A polynomial time greedy algorithm can find a clique of size $(1 + o(1))\log n$. The existence of $\rho > 1$ for which polynomial time algorithms can find cliques of size $\rho \log n$ is a longstanding open problem.

In the classical planted clique problem, one starts with a graph $G' \sim G(n, \frac{1}{2})$ and a parameter k. In G' one chooses at random a set K of k vertices, and makes this set into a clique by inserting all missing edges between pairs of vertices with K. We refer to K as the planted clique, and say that the resulting graph G is distributed according to $G(n, \frac{1}{2}, k)$. Given $G \sim G(n, \frac{1}{2}, k)$, the algorithmic goal can be one of the following three: find K, find a clique of maximum size, or find any clique of size at least k. It is not difficult to show that when k is sufficiently large (say, $k > 3\log n$), then with high probability K is the unique maximum size clique in $G \sim G(n, \frac{1}{2}, k)$, and hence all three goals coincide. Hence in the planted clique problem, the goal is simply to design polynomial time algorithms that (with high probability over the choice of $G \sim G(n, \frac{1}{2}, k)$) find the planted clique K. The question is how large should k be (as a function of n) so as to make this task feasible.

For some sufficiently large constant $c > 0$ (throughout, we use c to denote a sufficiently large constant), if $k > c\sqrt{n \log n}$, with high probability the the vertices of K are simply the k vertices of highest degree in G (see [13]), and hence K can easily be recovered. Alon, Krivelevich and Sudakov [1] managed to shave the $\sqrt{\log n}$ factor, designing a spectral algorithm that recovers K when $k > c\sqrt{n}$. They also showed that c can be made an arbitrarily small constant, by increased the running time by a factor of $n^{O(\log(\frac{1}{c}))}$ (this is done by "guessing" a set K' of $O(\log(\frac{1}{c}))$ vertices of K, and finding the maximum clique in the subgraph induced on their common neighbors). Subsequently, additional algorithms were developed that find the planted clique when $k > c\sqrt{n}$. They include algorithms based on the Lovasz theta function, which is a form of semi-definite programming [7], algorithms based on a "reverse-greedy" principle [10, 3], and message passing algorithms [4]. There have been many attempts to find polynomial time algorithms that succeed when $k = o(\sqrt{n})$, but so far all of them failed (see for example [11, 8, 16]). It is a major open problem whether there is any such polynomial time algorithm.

Planted clique when $p \neq \frac{1}{2}$ was not studied as extensively, but it is quite well understood how results from the $G(n, \frac{1}{2}, k)$ model transfer to the $G(n, p, k)$ model. For p much smaller that $\frac{1}{2}$, say $p = n^{\delta - 1}$ for some $0 < \delta < 1$ (hence average degree n^{δ}), the problem changes completely. Even without planting, with high probability over the choice of $G \sim G(n,p)$ (with $p = n^{\delta-1}$) we have that $\omega(G) = O(\frac{1}{1-\delta})$, and the maximum clique can be found in polynomial time. This also extends to finding maximum cliques in the planted setting, regardless of the value of k. (We are not aware of such results being previously published, but they are not difficult. See Section 2.2.) For $p > \frac{1}{2}$, it is more convenient to instead look at the equivalent problem in which $p < \frac{1}{2}$, but with the goal of finding a planted independent set instead of a planted clique. We refer to this model as $\bar{G}(n,p,k)$. For $G \sim G(n,p)$ (with $p = n^{\delta-1}$) we have that with high probability $\alpha(G) = \Theta(n^{1-\delta}\log n)$. For $G \sim \bar{G}(n,p,k)$ the known algorithms extend to finding planted independent sets of size $k = cn^{1-\frac{\delta}{2}}$ in polynomial time. We remark that the approach of [1] of making c arbitrarily small does not work for such sparse graphs.

1.2 The adversarial planted clique model

In this paper we introduce a variation on the planted clique model (and planted independent set model) that we refer to as the adversarial planted clique model. As in the random planted clique model, we start with a graph $G' \sim G(n,p)$ and a parameter k. However, now a computationally unbounded adversary may inspect G', select within it a subset K of k vertices of its choice, and make this set into a clique by inserting all missing edges between pairs of vertices with K. We refer to this model as $AG(n,p,k)$ (and the corresponding model for planted independent set as $A\bar{G}(n,p,k)$). As shorthand notation shall use $G \sim AG(n,p,k)$ to denote a graph generated by this process. Let us clarify that $AG(n,p,k)$ is not a distribution over graphs, but rather a family of distributions, where each adversarial strategy (where a strategy of an adversary is a mapping from G' to a choice of K) gives rise to a different distribution.

In the adversarial planted model, it is no longer true that the planted clique is the one of maximum size in the resulting graph G. Moreover, finding K itself may be information theoretically impossible, as K might be statistically indistinguishable from some other clique of size k (that differs from K by a small number of vertices). The three goals, that of finding K, finding a clique of maximum size, or finding any clique of size at least k, are no longer equivalent. Consequently, for our algorithmic results we shall aim at the more demanding goal of finding a clique of maximum size, whereas for our hardness results, we shall want them to hold even for the less demanding goal of finding an arbitrary clique of size k.

1.3 Our results

Our results cover a wide range of values of $0 < p < 1$, where p may be a function of n. For simplicity of the presentation and to convey the main insights of our results, we present here the results for three representative regimes: $p = \frac{1}{2}$, $p = n^{\delta-1}$ for $0 < \delta < 1$, and $p = 1 - n^{\delta-1}$. For the latter regime, it will be more convenient to replace it by the equivalent problem of finding adversarially planted independent sets when $p = n^{\delta-1}$.

Informally, our results show the following phenomenon. We consider only the case that $p \le \frac{1}{2}$, but consider both the planted clique and the planted independent set problems, and hence the results can be translated to $p > \frac{1}{2}$ as well. For clique, we show (Theorem 1 and Theorem 2) how to extend the algorithmic results known for the random planted clique setting to the adversarial planted clique setting. However, for independent set, we show that this is no longer possible. Specifically, when p is sufficiently small, we prove (Theorem 3) that finding an independent set of size k (any independent set, not necessarily the planted one) in the adversarial planted independent set setting is NP-hard. Moreover, the NP-hardness result holds even for large values of k for which finding a random planted independent set is trivial.

▶ **Theorem 1.** *For every fixed $\varepsilon > 0$ and for every $k \geq \varepsilon\sqrt{n}$, there is an (explicitly described) algorithm running in time $n^{O(\log(\frac{1}{\varepsilon}))}$ which almost surely finds the maximum clique in a graph $G \sim AG(n, \frac{1}{2}, k)$. The statement holds for every adversarial planting strategy (choice of k vertices as a function of $G' \sim G(n, \frac{1}{2})$), and the probability of success is taken over the choice of $G' \sim G(n, \frac{1}{2})$.*

▶ **Theorem 2.** *Let $p = n^{\delta-1}$ for $0 < \delta < 1$. Then for every k, there is an (explicitly described) algorithm running in time $n^{O(\frac{1}{1-\delta})}$ which almost surely finds the maximum clique in a graph $G \sim AG(n,p,k)$. The statement holds for every adversarial planting strategy, and the probability of success is taken over the choice of $G' \sim G(n,p)$.*

▶ **Theorem 3.** *For $p = n^{\delta-1}$ with $0 < \delta < 1$, $0 < \gamma < 1$, and $cn^{1-\delta}\log n \leq k \leq \frac{2}{3}n$ (where c is a sufficiently large constant, and the constant $\frac{2}{3}$ was chosen for concreteness – any other constant smaller than 1 will work as well) the following holds. There is no polynomial time algorithm that has probability at least γ of finding an independent set of size k in $G \sim A\bar{G}(n,p,k)$, unless NP has randomized polynomial time algorithms (NP=RP). (The algorithm is required to succeed against every adversarial planting strategy, and the probability of success is taken over the choice of $G' \sim G(n,p)$.)*

1.4 Related work

Some related work was already mentioned in Section 1.1.

Our algorithm for Theorem 1 is based on an adaptation of the algorithm of [7] that applied to the random planted clique setting. In turn, that algorithm is based on the theta function of Lovasz [14].

A work that is closely related to ours and served as an inspiration both to the model that we study, and to the techniques that are used in the proof of the NP-hardness result (Theorem 3) is the work of David and Feige [2] on adversarially planted 3-colorings. That work uncovers a phenomenon similar to the one displayed in the current work. Specifically, for the problem of 3-coloring (rather than clique or independent set) it shows that for certain values of p, algorithms that work in the random planted setting can be extended to the adversarial planted setting, and for other values of p, finding a 3-coloring in the adversarial planted setting becomes NP-hard. However, there are large gaps left open in the picture that emerges from the work of [2]. For large ranges of the values of p, specifically, $n^{-1/2} < p < n^{-1/3}$ and $p < n^{-2/3}$, there are neither algorithmic results nor hardness results in the work of [2]. Unfortunately, the most interesting values of p for the 3-coloring problem, which are $p \leq \frac{c\log n}{n}$, lie within these gaps, and hence the results of [2] do not apply to them. Our work addresses a different problem (planted clique instead of planted 3-coloring), and for our problem, our analysis leaves almost no such gaps. We are able to determine for which values of p the problem is polynomial time solvable, and for which values it is NP-hard. See Section 3 for more details.

Our model is an example of a *semi-random* model, in which part of the input is determined at random and part is determined by an adversary. There are many other semi-random models, both for the clique problem and for other problems. Describing all these models is beyond the scope of this paper, and the interested reader is referred to [5] and references therein for additional information.

2 Overview of the proofs

In this section we provide an overview of the proofs for our three main theorems. Further details, as well as extensions to the results, appear in the full version of our paper [6].

The term *almost surely* denotes a probability that tends to 1 as n grows. The term *extremely high probability* denotes a probability of the form $1 - e^{-n^r}$ for some $r > 0$. By $\exp(x)$ for some expression x we mean e^x.

2.1 Finding cliques using the theta function

In this section we provide an overview of the proof of Theorem 1. Our algorithm is an adaptation of the algorithm of [7] that finds the maximum clique in the random planted model. We shall first review that algorithm, then describe why it does not apply in our setting in which an adversary plants the clique, and finally explain how we modify that algorithm and its analysis so as to apply it in the adversarial planted setting.

The key ingredient in the algorithm of [7] is the theta function of Lovasz, denoted by ϑ. Given a graph G, $\vartheta(G)$ can be computed in polynomial time (up to arbitrary precision, using semidefinite programming (SDP)), and satisfies $\vartheta(G) \geq \alpha(G)$. As we are interested here in cliques and not in independent sets, we shall consider $\bar{G}$, the edge complement of G, and then $\vartheta(\bar{G}) \geq \omega(G)$. The theta function has several equivalent definitions, and the one that we shall use here (referred to as ϑ_4 in [14]) is the following.

Given a graph $G = G(V, E)$, a collection of unit vectors $s_i \in \mathbb{R}^n$ (one vector for every vertex $i \in V$) is an *orthonormal representation* of G, if s_i and s_j are orthogonal ($s_i \cdot s_j = 0$) whenever $(i, j) \in E$. The theta function is the maximum value of the following expression, where maximization is over all orthonormal representations $\{s_i\}$ of G and over all unit vectors h (h is referred to as the *handle*):

$$\vartheta(G) = \max_{h,\{s_i\}} \sum_{i \in V} (h \cdot s_i)^2 \tag{1}$$

The optimal orthonormal representation and the associated handle that maximize the above formulation for ϑ can be found (up to arbitrary precision) in polynomial time by formulating the problem as an SDP (details omitted). Observe that for any independent set S the following is a feasible solution for the SDP: choose $s_i = h$ for all $i \in S$, and choose all remaining vectors s_j for $j \notin S$ to be orthogonal to h and to each other. Consequently, $\vartheta(G) \geq \alpha(G)$, as claimed.

The main content of the algorithm of [7] is summarized in the following theorem. We phrased it in a way that addresses cliques rather than independent sets, implicitly using $\alpha(\bar{G}) = \omega(G)$. We also remind the reader that in the random planted model, the planted clique K is almost surely the unique maximum clique.

▶ **Theorem 4** (Results of [7]). *Consider $G \sim G(n, \frac{1}{2}, k)$, a graph selected in the random planted clique model, with $k \geq c\sqrt{n}$ for some sufficiently large constant c. Then with extremely high probability (over choice of G) it holds that $\vartheta(\bar{G}) = \omega(G)$.*

Moreover, for every vertex i that belongs to the planted clique K, the corresponding vector s_i has inner product larger than $1 - \frac{1}{n}$ with the handle h, and for every other vertex, the corresponding inner product is at most $\frac{1}{n}$.

Given Theorem 4, the following algorithm finds the planted clique when $G \sim G(n, \frac{1}{2}, k)$, and $k \geq c\sqrt{n}$ for some sufficiently large constant c. Solve the optimization problem (1) (on $\bar{G}$) to sufficiently high precision, and output all vertices whose corresponding inner product with h is at least $\frac{1}{2}$.

The algorithm above does not apply to $G \sim AG(n, \frac{1}{2}, k)$, a graph selected in the adversarial planted clique model, for the simple reason that Theorem 4 is incorrect in that model. The following example illustrates what might go wrong,

▶ **Example 5.** Consider a graph $G' \sim G(n, \frac{1}{2})$. In G' first select a random vertex set T of size slightly smaller than $\frac{1}{2}\log n$. Observe that the number of vertices in G' that are in the common neighborhood of all vertices of T is roughly $2^{-|T|}n > \sqrt{n}$. Plant a clique K of size k in the common neighborhood of T. In this construction, K is no longer the largest clique in G. This is because T (being a random graph) is expected to have a clique K' of size $2\log|T| \simeq 2\log\log n$, and $K' \cup K$ forms a clique of size roughly $k + 2\log\log n$ in G. Moreover, as T itself is a random graph with edge probability $\frac{1}{2}$, the value of the theta function on T is roughly $\sqrt{|T|}$ (see [12]), and consequently one would expect the value of $\vartheta(\bar{G})$ to be roughly $k + \sqrt{\log n}$.

Summarizing, it is not difficult to come up with strategies for planting cliques of size k that result in the maximum clique having size strictly larger than k, and the value of $\vartheta(\bar{G})$ being even larger. Consequently, the solution of the optimization problem (1) by itself is not expected to correspond to the maximum clique in G.

We now explain how we overcome the above difficulty. A relatively simple, yet important, observation is the following.

▶ **Proposition 6.** *Let $G \sim AG(n,p,k)$ with $p = 1/2$ and $k > \sqrt{n}$, and let K' be the maximum clique in G (which may differ from the planted clique K). Then with extremely high probability over the choice of $G' \sim G(n, \frac{1}{2})$, for every possible choice of k vertices by the adversary, K' contains at least $k - O(\log n)$ vertices from K, and at most $O(\log n)$ additional vertices.*

Proof. Standard probabilistic arguments show that with extremely high probability, the largest clique in G' (prior to planting a clique of size k) is of size at most $\frac{k}{2}$. When this holds, K' contains at least $\frac{k}{2}$ vertices from K. Each of the remaining vertices of K' needs to be connected to all vertices in $K' \cap K$. Consequently, with extremely high probability, K' contains at most $2\log n$ vertices not from K. This is because a $G' \sim G(n, \frac{1}{2})$ graph, with extremely high probability, does not contain two sets of vertices A and B, with $|A| = 2\log n$, $|B| = \Omega(\sqrt{n})$, such that all pairs of vertices in $A \times B$ induce edges in G.

As $|K'| \geq k$, we conclude that all but $O(\log n)$ vertices of K must be members of K'. ◀

A key theorem that we prove is:

▶ **Theorem 7.** *Let $G \sim AG(n,p,k)$ with $p = 1/2$ and $k = k(n) \geq 10\sqrt{n}$. Then $k \leq \vartheta(\bar{G}) \leq k + O(\log n)$ with extremely high probability over the choice of $G' \sim G(n, \frac{1}{2})$, for every possible choice of k vertices by the adversary.*

We now explain how Theorem 7 is proved. The bound $\vartheta(\bar{G}) \geq k$ was already explained above. Hence it remains to show that $\vartheta(\bar{G}) \leq k + O(\log n)$. In general, to bound $\vartheta(G)$ from above for a graph $G(V,E)$, one considers the following dual formulation of ϑ, as a minimization problem.

$$\vartheta(G) = \min_M[\lambda_1(M)] \tag{2}$$

Here M ranges over all n by n symmetric matrices in which $M_{ij} = 1$ whenever $(i,j) \notin E$, and $\lambda_1(M)$ denotes the largest eigenvalue of M. (Observe that if G has an independent set S of size k, then M contains a k by k block of 1 entries. A Rayleigh quotient argument then implies that $\lambda_1(M) \geq k$, thus verifying the inequality $\vartheta(G) \geq \alpha(G)$.) To prove Theorem 7 we exhibit a matrix M as above (for the graph $\bar{G}$) for which we prove that $\lambda_1(M) \leq k + O(\log n)$.

We first review how a matrix M was chosen by [7] in the proof of Theorem 4. First, recall that we consider $\bar{G}$, and let E be the set of edges of $\bar{G}$ (non-edges of G). We need to associate values with the entries M_{ij} for $(i,j) \in E$ (as other entries are 1). The matrix block corresponding to the planted clique K (planted independent set in $\bar{G}$) is all 1 (by necessity). For every $(i,j) \in E$ where both vertices are not in K one sets $M_{ij} = -1$. For every other pair $(i,j) \in E$ (say, $i \in K$ and $j \notin K$) one sets $M_{i,j} = -\frac{k-d_{i,K}}{d_{i,K}}$, where $d_{i,K}$ is the number of neighbors that vertex i has in the set K. In order to show that $\lambda_1(M) = k$, one first observes that the vector x_K (with value 1 at entries that correspond to vertices of K, and value 0 elsewhere) is an eigenvector of M with eigenvalue k. Then one proves that $\lambda_2(M)$, the second largest eigenvalue of M, has value smaller than k. This is done by decomposing M into a sum of several matrices, bounding the second largest eigenvalue for one of these matrices, and the largest eigenvalue for the other matrices. By Weyl's inequality, the sum of these eigenvalues is an upper bound on $\lambda_2(M)$. This upper bound is not tight, but it does show that $\lambda_2(M) < k$. It follows that the eigenvalue k associated with x_K is indeed $\lambda_1(M)$. Further details are omitted.

We now explain how to choose a matrix M so as to prove the bound $\vartheta(\bar{G}) \le k + O(\log n)$ in Theorem 7. Recall (see Example 5) that we might be in a situation in which $\vartheta(\bar{G}) > \alpha(\bar{G}) > k$ (with all inequalities being strict). In this case, let K' denote the largest independent set in $\bar{G}$, and note that K' is larger than K. In M, the matrix block corresponding to K' is all 1. One may attempt to complete the construction of M as described above for the random planting case, but replacing K by K' everywhere in that construction. If one does so, the vector $x_{K'}$ (with value 1 at entries that correspond to vertices of K', and value 0 elsewhere) is an eigenvector of M with eigenvalue $\alpha(\bar{G}) > k$. However, M would necessarily have another eigenvector with a larger eigenvalue, because $\vartheta(\bar{G}) > \alpha(\bar{G})$. Hence we are still left with the problem of bounding $\lambda_1(M)$, rather than bounding $\lambda_2(M)$. Having failed to identify an eigenvector for $\lambda_1(M)$, we may still obtain an upper bound on $\lambda_1(M)$ by using approaches based on Weyl's inequality (or other approaches). However, these upper bounds are not tight, and it seems difficult to limit the error that they introduce to be as small as $O(\log n)$, which is needed for proving the inequality $\lambda_1(M) \le k + O(\log n)$.

For the above reason, we choose M differently. For some constant $\frac{1}{2} < \rho < 1$, we extend the clique K to a possibly larger clique Q, by adding to it every vertex that has ρk neighbors in K. (In Example 5, the corresponding clique Q will include all vertices of $K \cup T$. In contrast, if K is planted at random and not adversarially, then we will simply have $Q = K$.) Importantly, we prove that if $G' \sim G(n, \frac{1}{2})$, then with high probability $|Q| < k + O(\log n)$ (for every possible choice of planting a clique of size k by the adversary). For the resulting graph G_Q, we choose the corresponding matrix M in the same way as it was chosen for the random planting case. Now we do manage to show that the eigenvector x_Q (with eigenvalue $|Q|$) associated with this M indeed has the largest eigenvalue. This part is highly technical, and significantly more difficult than the corresponding proof for the random planting case. The reason for the added level of difficulty is that, unlike the random planting case in which we are dealing with only one random graph, here the adversary can plant the clique in any one of $\binom{n}{k}$ locations, and our analysis needs to hold simultaneously for all $\binom{n}{k}$ graphs that may result from such plantings. Further details can be found in [6].

Having established that $\vartheta(\bar{G}_Q) = |Q| \le k + O(\log n)$, we use monotonicity of the theta function to conclude that $\vartheta(\bar{G}) \le k + O(\log n)$. This concludes our overview for the proof of Theorem 7.

Given Theorem 7, let us now explain our algorithm for finding a maximum clique in $G \sim AG(n, \frac{1}{2}, k)$.

Given a graph $G \sim AG(n, \frac{1}{2}, k)$, the first step in our algorithm is to solve the optimization problem (1) on the complement graph $\bar{G}$. By Theorem 7, we will have $\vartheta(\bar{G}) \leq k + c \log n$ for some constant $c > 0$. Let $\{s_i\}$ denote the orthonormal representation found by our solution, and let h be the corresponding handle.

The second step of our algorithm it to extract from G a set of vertices that we shall refer to as H, that contains all those vertices i for which $(h \cdot s_i)^2 \geq \frac{3}{4}$.

▶ **Lemma 8.** *For H as defined above, with extremely high probability, at least $k - O(\log n)$ vertices of K are in H, and most $O(\log n)$ vertices not from K are in H.*

Proof. Let T denote the set of those vertices in K for which $(h \cdot s_i)^2 < \frac{3}{4}$. Remove T from G, thus obtaining the graph G_T. This graph can be thought of as a subgraph with $n - |T|$ vertices of the random graph $G' \sim G(n, \frac{1}{2})$, in which an adversary planted a clique of size $k - |T|$. We also have that $\vartheta(\bar{G}_T) \geq \vartheta(\bar{G}) - \sum_{i \in T}(h \cdot s_i)^2 \geq k - \frac{3}{4}|T|$. If $|T|$ is large (larger than $c' \log n$ for some sufficiently large constant $c' > 0$), the gap of $\frac{|T|}{4}$ between the size of the planted clique and the value of the theta function contradicts Theorem 7 for the graph G_T. (Technical remark: this last argument uses the fact that Theorem 7 holds with extremely high probability, as we take a union bound over all choices of T.)

Having established that T is small, let R be the set of vertices not in K for which $(h \cdot s_i)^2 \geq \frac{3}{4}$. We claim that every such vertex $i \in R$ is a neighbor of every vertex $j \in K \setminus T$. This is because in the orthogonal representation (for $\bar{G}$), if i and j are not neighbors we have that $s_i \cdot s_j = 0$, and then the fact that s_i,s_j and h are unit vectors implies that $(h \cdot s_i)^2 < 1 - (h \cdot s_j)^2 \leq \frac{1}{4}$. Having this claim and using the fact that $|K \setminus T| > \sqrt{n}$, it follows that $|R| \leq 2 \log n$. This is because a $G' \sim G(n, \frac{1}{2})$ graph, with extremely high probability, does not contain two sets of vertices A and B, with $|A| = 2 \log n$, $|B| = \sqrt{n}$, such that all pairs of vertices in $A \times B$ induce edges in G. ◀

The third step of our algorithm constructs a set F that contains all those vertices that have at least $\frac{3k}{4}$ neighbors in H.

▶ **Lemma 9.** *With extremely high probability, the set F described above contains the maximum clique in G, and at most $O(\log n)$ additional vertices.*

Proof. We may assume that H satisfies the properties of Lemma 8. Proposition 6 then implies that with extremely high probability, every vertex of the maximum clique in G has at least $\frac{3k}{4}$ neighbors in H, and hence is contained in F. A probabilistic argument (similar to the end of the proof of Lemma 8) establishes that F has at most $O(\log n)$ vertices not from K. As K has at most $O(\log n)$ vertices not from the maximum clique (by Proposition 6), the total number of vertices in F that are not members of the maximum clique is at most $O(\log n)$. ◀

Finally, in the last step of our algorithm we find a maximum clique in F, and this is a maximum clique in G. This last step can be performed in polynomial time by a standard algorithm (used for example to show that vertex cover is fixed parameter tractable). For every non-edge in the subgraph induced on F, at least one of its end-vertices needs to be removed. Try both possibilities in parallel, and recurse on each subgraph that remains. The recursion terminates when the graph is a clique. The shortest branch in the recursion gives the maximum clique. As only $O(\log n)$ vertices need to be removed in order to obtain a clique, the depth of the recursion is at most $O(\log n)$, and consequently the running time (which is exponential in the depth) is polynomial in n.

This completes our overview of our algorithm for finding a clique in $G \sim AG(n, \frac{1}{2}, k)$ when $k > c\sqrt{n}$ for a sufficiently large constant $c > 0$. To complete the proof of Theorem 1 we need to also address the case that $k > \varepsilon\sqrt{n}$ for arbitrarily small constant ε. This we do (as in [1]) by guessing $t \simeq 2\log\frac{c}{\epsilon}$ vertices from K (there are n^t possibilities to try, and we try all of them), and considering the subgraph of G induced on their common neighbors. This subgraph corresponds to a subgraph of $G' \simeq G(n, \frac{1}{2})$ with roughy $n' \simeq 2^{-t}n$ vertices, and a planted clique of size $\varepsilon\sqrt{n} - t \simeq c\sqrt{n'}$. Now on this new graph G" we can invoke the algorithm based on the theta function. (Technical remark. The proof that $\vartheta(\bar{G}") \leq k + O(\log n)$ uses the fact that Theorem 7 holds with extremely high probability.)

The many details that were omitted from the above overview of the proof of Theorem 1 can be found in [6].

2.2 Finding cliques by enumeration

In this section we prove Theorem 2.

Let $p = n^{\delta-1}$ for $0 < \delta < 1$, and consider first $G' \sim G(n, p)$ (hence G' has average degree roughly n^δ). For every size $t \geq 1$, let N_t denote the number of cliques of size t in G'. The expectation (over choice of $G' \sim G(n, p)$) satisfies:

$$\mathbb{E}[N_t] = \binom{n}{t} p^{\binom{t}{2}} \leq \frac{1}{t!} n^{\frac{\delta-1}{2}t^2 + \frac{3-\delta}{2}t}$$

The exponent is maximized when $t = \frac{3-\delta}{2(1-\delta)}$. For the maximizing (not necessarily integer) t, the exponent equals $\frac{(3-\delta)^2}{8(1-\delta)}$. We denote this last expression by e_δ, and note that $e_\delta = O(\frac{1}{1-\delta})$. The expected number of cliques of all sizes is then:

$$\sum_{t\geq 1} \mathbb{E}[N_t] \leq n + \sum_{t \geq 2} \frac{1}{t!} n^{\frac{\delta-1}{2}t^2 + \frac{3-\delta}{2}t} \leq n^{e_\delta}$$

(The last inequality holds for sufficiently large n.) By Markov's inequality, with probability at least $1 - \frac{1}{n}$, the actual number of cliques in G' is at most $n^{e_\delta + 1}$. (Stronger concentration results can be used here, but are not needed for the proof of Theorem 2.)

Now, for arbitrary $1 \leq k \leq n$, let the adversary plant a clique K of size k in G', thus creating the graph $G \sim G(n, p, k)$. As every subgraph of K is a clique, the total number of cliques in G is at least 2^k, which might be exponential in n (if k is large). However, the number of maximal cliques in G (a clique is maximal if it is not contained in any larger clique) is much smaller. Given a maximal clique C in G, consider C', the subgraph of C not containing any vertex from K. C' is a clique in G' (which is nonempty, except for one special case of $C = K$). C' uniquely determines C, as the remaining vertices in C are precisely the set of common neighbors of C' in K (this is because the clique C is maximal). Consequently, the number of maximal cliques in G is not larger than the number of cliques in G'.

As all maximal cliques in a graph can be enumerated in time linear in their number times some polynomial in n (see e.g. [15] and references therein), one can list all maximal cliques in G in time $n^{e_\beta + O(1)}$ (this holds with probability at least $1 - \frac{1}{n}$, over the choice of G', regardless of where the adversary plants clique K), and output the largest one.

This completes the proof of Theorem 2.

2.3 Proving NP-hardness results

In this section we provide an overview of the proof of Theorem 3. Our proof is an adaptation to our setting of a proof technique developed in [2].

Recall that we are considering a graph $G \sim A\bar{G}(n,p,k)$ (adversarial planted independent set) with $p = n^{\delta-1}$ and $0 < \delta < 1$. Let us first explain why the algorithm described in Section 2.1 fails when $k = cn^{1-\frac{\delta}{2}}$ (whereas if the independent set is planted at random, algorithms based on the theta function are known to succeed). The problem is that the bound in Theorem 7 is not true anymore, and instead one has the much weaker bound of $\vartheta(G) \le k + n^{1-\delta}\log n$. Following the steps of the algorithm of Section 2.1, in the final step, we would need to remove a minimum vertex cover from F. However, now the upper bound on the size of this vertex cover is $O(n^{1-\delta}\log n)$ rather than $O(\log n)$. Consequently, we do not know of a polynomial time algorithm that will do so. It may seem that we also do not know that no such algorithm exists. After all, F is not an arbitrary worst case instance for vertex cover, but rather an instance derived from a random graph. However, our NP-hardness result shows that indeed this obstacle is insurmountable, unless NP has randomized polynomial time algorithms. We remark that using an approximation algorithm for vertex cover in the last step of the algorithm of Section 2.1 does allow one to find in G an independent set of size $k - O(n^{1-\delta}\log n) = (1 - o(1))k$, and the NP-hardness result applies only because we insist on finding an independent set of size at least k.

Let us proceed now with an overview of our NP hardness proof. We do so for the case that $k = \frac{n}{3}$ (for which we can easily find the maximum independent set if the planted independent set is random). Assume for the sake of contradiction that ALG is a polynomial time algorithm that with high probability over choice of $G' \sim G(n,p)$, for every planted independent set of size $k = \frac{n}{3}$, it finds in the resulting graph G an independent set of size k.

We now introduce a class $\mathcal{H}$ of graphs that, in anticipation of the proofs that will follow, is required to have the following three properties. (Two of the properties are stated below in a qualitative manner, but they have precise quantitative requirements in the proofs that follow.)

1. Solving maximum independent set on graphs from this class is NP-hard.
2. Graphs in this class are very sparse.
3. The number of vertices in each graph is small.

Given the above requirements, we choose $0 < \varepsilon < \min[\frac{\delta}{2}, 1-\delta]$, and let $\mathcal{H}$ be the class of *balanced* graphs on n^{ϵ} vertices, and of average degree $2 + \delta$. (A graph H is *balanced* if no subgraph of H has average degree larger than the average degree of H.) Given a graph $H \in \mathcal{H}$ and a parameter k', it is NP-hard to determine whether H has an independent of size at least k' or not. We will reach a contradiction to the existence of ALG by showing how ALG could be used in order to find in H an independent set of size k', if one exists. For this, we use the following randomized algorithm ALGRAND.

1. Generate a random graph $G' \sim G(n,p)$.
2. Plant in G' a random copy of H (that is, pick $|H|$ random vertices in G' and replace the subgraph induced on them by H). We refer to the resulting distribution as $G_H(n,p)$, and to the graph sampled from this distribution as G_H. Observe that the number of vertices in G_H that have a neighbor in H is with high probability not larger than $|H|n^{\delta} \le \frac{n}{2}$.
3. Within the non-neighbors of H, plant at random an independent set of size $k - k'$. We refer to the resulting distribution as $G_H(n,p,k)$, and to the graph sampled from this distribution as $\tilde{G}_H$. Observe that with extremely high probability, $\alpha(\tilde{G}_H \setminus H) = k - k'$. Hence we may assume that this indeed holds. If furthermore $\alpha(H) \ge k'$, then $\alpha(\tilde{G}_H) \ge k$.
4. Run ALG on $\tilde{G}_H$. We say that ALGRAND succeeds if ALG outputs an independent set IS of size k. Observe that then at least k' vertices of H are in IS, and hence ALGRAND finds an independent set of size k' in H.

If H does not have an independent set of size k', ALGRAND surely fails to output such an independent set. But if H does have an independent set of size k', why should ALGRAND succeed? This is because ALG (which is used in ALGRAND) is fooled to think that the graph $\tilde{G}_H$ generated by ALGRAND was generated from $A\bar{G}(n,p,k)$, and on such graphs ALG does find independent sets of size k. And why is ALG fooled? This is because the distribution of graphs generated by ALGRAND is statistically close to a distribution that can be created by the adversary in the $A\bar{G}(n,p,k)$ model. Specifically, consider the following distribution that we refer to as $A_HG(n,p,k)$.

1. Generate $G' \sim G(n,p)$.
2. The computationally unbounded adversary finds within G' all subsets of vertices of size $|H|$ such that the subgraph induced on them is H. (If there is no such subset, fail.) Choose one such copy of H uniformly at random.
3. As H is assumed to have an independent set of size k', plant an independent set K of size k as follows. k' of the vertices of K are vertices of an independent set in the selected copy of H. The remaining $k-k'$ vertices of K are chosen at random among the vertices of G' that have no neighbor at all in the copy of H. (Observe that we expect there to be at least roughly $n - |H|n^\delta \geq \frac{n}{2}$ such vertices, and with extremely high probability the actual number will be at least $\frac{n}{3} > k - k'$.)

▶ **Theorem 10.** *The two distributions, $\tilde{G}_H \sim G_H(n,p,k)$ generated by ALGRAND and $G \sim A_HG(n,p,k)$ generated by the adversary, are statistically similar to each other.*

The proof of Theorem 10 appears in [6]. Here we explain the main ideas in the proof. A minimum requirement for the theorem to hold is that $G' \sim G(n,p)$ typically contains at least one copy of H (otherwise $A_HG(n,p,k)$ fails to produce any output). But this by itself does not suffice. Intuitively, the condition we need is that G' typically contains many copies of H. Then the fact that $G_H(n,p)$ of ALGRAND adds another copy of H to G' does not appear to make much of a difference to G', because G' anyway has many copies of H. Hopefully, this will imply that $G' \sim G(n,p)$ and $G_H \sim G_H(n,p)$ come from two distributions that are statistically close. This intuition is basically correct, though another ingredient (a concentration result) is also needed. Specifically, we need the following lemma (stated informally).

▶ **Lemma 11.** *For $G' \in G(n,p)$ (with p and H as above), the expected number of copies of H in G' is very high (2^{n^η} for some $\eta > 0$ that depends on δ and ϵ). Moreover, with high probability, the actual number of copies of H in G' is very close to its expectation.*

The proof of Lemma 11 is based on known techniques (first and second moment methods). It uses in an essential way the fact that the graph H is sparse (average degree barely above 2) and does not have many vertices (these properties hold by definition of the class $\mathcal{H}$). Armed with Lemma 11, we then prove the following Lemma.

▶ **Lemma 12.** *The two distributions $G(n,p)$ and $G_H(n,p)$ are statistically similar to each other.*

Lemma 12 is proved by considering graphs $G' \sim G(n,p)$ that do contain a copy of H (Lemma 11 establishes that this is a typical case), and comparing for each such graph the probability of it being generated by $G_H(n,p)$ with the probability of it being generated by $G(n,p)$. Conveniently, the ratio between these probabilities is the same as the ratio between the actual number of copies of H in the given graph G', and the expected number of copies of H in a random $G' \sim G(n,p)$. By Lemma 11, for most graphs, this ratio is close to 1.

Theorem 10 follows quite easily from Lemma 12. Consequently ALG's performance on the distributions $G_H(n,p,k)$ and $A_HG(n,p,k)$ is similar. By our assumption, ALG finds (with high probability) an independent set of size k in $G \sim A_HG(n,p,k)$, which now implies that it also does so for $\tilde{G}_H \sim G_H(n,p,k)$. But as argued above, finding an independent set of size k in $\tilde{G}_H \sim G_H(n,p,k)$ implies that ALGRAND finds an independent set of size k' in $H \in \mathcal{H}$, thus solving an NP-hard problem. Hence the assumption that there is a polynomial time algorithm ALG that can find independent sets of size k in $G \sim A\bar{G}(n,p,k)$ implies that NP has randomized polynomial time algorithms.

3 Additional results

In the main part of the paper we only described what we view as our main results. The appendix contains all missing proofs, and some additional results and extensions, not described above. For example, one may ask for which value of $p \leq \frac{1}{2}$ the transition occurs from being able to find the maximum independent set in $G \sim A\bar{G}(n,p,k)$ in polynomial time, to the problem becoming NP hard. Our results show a gradual transition. For constant p the problem remains polynomial time solvable, and then, as p continues to decrease, the running time of our algorithms becomes super polynomial, and grows gradually towards exponential complexity. Establishing this type of behavior does not require new proof ideas, but rather only the substitution of different parameters in the existing proofs. Consequently, some theorems that were stated here only in special cases (e.g., Theorem 7 that was stated only for $p = \frac{1}{2}$) are restated in the appendix in a more general way (e.g., replacing $\frac{1}{2}$ by p), and a more general proof is provided.

Though this is not shown in the appendix, our hardness results (for finding adversarially planted independent sets) also imply a gradual transition, providing NP-hardness results when $p = n^{\delta-1}$, and as p grows (e.g., into the range $p = \frac{1}{(\log n)^c}$) the NP-hardness results are replaced by hardness results under stronger assumptions, such as (a randomized version of) the exponential time hypothesis. This is because for $p = \frac{1}{(\log n)^c}$ we need to limit the size of the graphs $H \in \mathcal{H}$ to be only polylogarithmic in n, as for larger sizes the proofs in Section 2.3 fail.

An interesting range of parameters that remains open is that of $p = \frac{d}{n}$ for some large constant d. The case of a random planted independent set of size $\sqrt{\frac{c}{d}}n$ (for some sufficiently large constant $c > 0$ independent of d) was addressed in [9]. In such sparse graphs, the planted independent set is unlikely to be the maximum independent set. The main result in [9] is a polynomial time algorithm that with high probability finds the maximum independent set in that range of parameters. It would be interesting to see whether the positive results extend to the case of adversarial planted independent set. We remark that neither Theorem 1 nor Theorem 3 apply in this range of parameters.

References

1 Noga Alon, Michael Krivelevich, and Benny Sudakov. Finding a large hidden clique in a random graph. *Random Struct. Algorithms*, 13(3-4):457–466, 1998.

2 Roee David and Uriel Feige. On the effect of randomness on planted 3-coloring models. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016*, pages 77–90, 2016.

3 Yael Dekel, Ori Gurel-Gurevich, and Yuval Peres. Finding hidden cliques in linear time with high probability. *Combinatorics, Probability & Computing*, 23(1):29–49, 2014.

4 Yash Deshpande and Andrea Montanari. Finding hidden cliques of size $\sqrt{N/e}$ in nearly linear time. *Foundations of Computational Mathematics*, 15(4):1069–1128, 2015.

5 Uriel Feige. Introduction to semi-random models. In Tim Roughgarden, editor, *Beyond the Worst-Case Analysis of Algorithms.* to appear, 2020. URL: `http://www.wisdom.weizmann.ac.il/~feige/mypapers/bwca.pdf`.

6 Uriel Feige and Vadim Grinberg. How to hide a clique?, 2020. `arXiv:2004.12258`.

7 Uriel Feige and Robert Krauthgamer. Finding and certifying a large hidden clique in a semirandom graph. *Random Struct. Algorithms*, 16(2):195–208, 2000.

8 Uriel Feige and Robert Krauthgamer. The probable value of the Lovász–Schrijver relaxations for maximum independent set. *SIAM J. Comput.*, 32(2):345–370, 2003.

9 Uriel Feige and Eran Ofek. Finding a maximum independent set in a sparse random graph. *SIAM J. Discrete Math.*, 22(2):693–718, 2008.

10 Uriel Feige and Dorit Ron. Finding hidden cliques in linear time. In *21st International Meeting on Probabilistic, Combinatorial, and Asymptotic Methods in the Analysis of Algorithms (AofA'10)*, pages 189–204, 2010.

11 Mark Jerrum. Large cliques elude the metropolis process. *Random Struct. Algorithms*, 3(4):347–360, 1992.

12 Ferenc Juhász. The asymptotic behaviour of Lovász' ϑ function for random graphs. *Combinatorica*, 2:153–155, 1982.

13 Ludek Kucera. Expected complexity of graph partitioning problems. *Discrete Applied Mathematics*, 57:193–212, 1995.

14 Lászlo Lovász. On the Shannon capacity of a graph. *IEEE Transactions on Information Theory*, 25(1):1–7, 1979.

15 Kazuhisa Makino and Takeaki Uno. New algorithms for enumerating all maximal cliques. In *SWAT*, pages 260–272, July 2004. `doi:10.1007/978-3-540-27810-8_23`.

16 Raghu Meka, Aaron Potechin, and Avi Wigderson. Sum-of-squares lower bounds for planted clique. In *Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '15, pages 87–96, 2015. `doi:10.1145/2746539.2746600`.

Sampling Arbitrary Subgraphs Exactly Uniformly in Sublinear Time

Hendrik Fichtenberger
Department of Computer Science, TU Dortmund, Germany
hendrik.fichtenberger@tu-dortmund.de

Mingze Gao
Department of Computer Science, University of Sheffield, UK
noahgao0015@gmail.com

Pan Peng
Department of Computer Science, University of Sheffield, UK
p.peng@sheffield.ac.uk

Abstract

We present a simple sublinear-time algorithm for sampling an arbitrary subgraph H *exactly uniformly* from a graph G, to which the algorithm has access by performing the following types of queries: (1) uniform vertex queries, (2) degree queries, (3) neighbor queries, (4) pair queries and (5) edge sampling queries. The query complexity and running time of our algorithm are $\tilde{O}(\min\{m, \frac{m^{\rho(H)}}{\#H}\})$ and $\tilde{O}(\frac{m^{\rho(H)}}{\#H})$, respectively, where $\rho(H)$ is the fractional edge-cover of H and $\#H$ is the number of copies of H in G. For any clique on r vertices, i.e., $H = K_r$, our algorithm is almost optimal as any algorithm that samples an H from any distribution that has $\Omega(1)$ total probability mass on the set of all copies of H must perform $\Omega(\min\{m, \frac{m^{\rho(H)}}{\#H\cdot(cr)^r}\})$ queries.

Together with the query and time complexities of the $(1 \pm \varepsilon)$-approximation algorithm for the number of subgraphs H by Assadi et al. [3] and the lower bound by Eden and Rosenbaum [12] for approximately counting cliques, our results suggest that in our query model, approximately counting cliques is “equivalent to” exactly uniformly sampling cliques, in the sense that the query and time complexities of exactly uniform sampling and randomized approximate counting are within polylogarithmic factor of each other. This stands in interesting contrast to an analogous relation between approximate counting and almost uniformly sampling for self-reducible problems in the polynomial-time regime by Jerrum, Valiant and Vazirani [18].

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases Graph sampling, Graph algorithms, Sublinear-time algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.45

Category Track A: Algorithms, Complexity and Games

Acknowledgements We would like to thank the anonymous reviewers for their detailed comments. In particular, we would like to thank an anonymous reviewer for their suggestion to improve the presentation of the proof of Theorem 2 and their comment on applications, which we included as future work.

1 Introduction

“*Given a huge real graph, how can we derive a representative sample?*” is a first question asked by Leskove and Faloutsos in their seminal work on graph mining [20], which is motivated by the practical concern that most classical graph algorithms are too expensive for massive graphs (with millions or billions of vertices), and graph sampling seems essential for lifting the dilemma.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 45; pp. 45:1–45:13
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In this paper, we study the question of how to sample a subgraph H uniformly at random from the set of all subgraphs that are isomorphic to H contained in a large graph G in *sublinear time*, where the algorithm is given query access to the graph G. That is, the algorithm only probes a small portion of the graph while still returning a sample with provable performance guarantee. Such a question is relevant for statistical reasons: we might need a few representative and unbiased motifs from a large network [21], or edge-colored subgraphs in a structured database [4], in a limited time. A subroutine for extracting a uniform sample of H is also useful in streaming (e.g., [1]), parallel and distributed computing (e.g., [15]) and other randomized graph algorithms (e.g., [17]).

Currently, our understanding of the above question is still rather limited. Kaufman et al. gave the first algorithm for sampling an edge almost uniformly at random [19]. Eden and Rosenbaum gave a simpler and faster algorithm [13]. Both works considered the *general graph model*, where an algorithm is allowed to perform the following queries, where each query will be answered in constant time:

- **uniform vertex query:** the algorithm can sample a vertex uniformly at random;
- **degree query:** for any vertex v, the algorithm can query its degree d_v;
- **neighbor query:** for any vertex v and index $i \leq d_v$, the algorithm can query the i-th neighbor of v;
- **pair query:** for any two vertices u, v, the algorithm can query if there is an edge between u, v.

In [13], Eden and Rosenbaum gave an algorithm that takes as input a graph with n vertices and m edges (where m is unknown to the algorithm), uses $\tilde{O}(n/\sqrt{m})$ queries[1] in expectation and returns an edge e that is sampled with probability $(1 \pm \varepsilon)/m$ (i.e., almost uniformly at random). This is almost optimal in the sense that any algorithm that samples an edge from an almost-uniform distribution requires $\Omega(n/\sqrt{m})$ queries. In their sublinear-time algorithm for approximately counting the number cliques [10] (see below), Eden, Ron and Seshadhri use a procedure to sample cliques incident to a suitable vertex subset S almost uniformly at random. However, for an arbitrary subgraph H, it is still unclear how to obtain an almost uniform sample in sublinear time.

Approximate counting in sublinear-time. In contrast to sampling subgraphs (almost) uniformly at random, the very related line of research on approximate counting the number of subgraphs in sublinear time has made some remarkable progress in the past few years. Feige gave a $(2 + \varepsilon)$-approximation algorithm with $\tilde{O}(n/\sqrt{m})$ queries for the average degree, which is equivalent to estimating the number of edges, of a graph in the model that only uses vertex sampling and degree queries [14]. He also showed that any $(2 - o(1))$-approximation for the average degree using only vertex and degree queries requires $\Omega(n)$ queries. Goldreich and Ron then gave a $(1 + \varepsilon)$-approximation algorithm with $\tilde{O}(n/\sqrt{m})$ queries for the average degree in the model that allows vertex sampling, degree and neighbor queries [16].

Eden et al. recently gave the first sublinear-time algorithm for $(1 \pm \varepsilon)$-approximating the number of triangles [7]. Later, Eden, Ron and Seshadhri generalized it to $(1 \pm \varepsilon)$-approximating the number of r-cliques K_r [10] in the general graph model that allows vertex sampling, degree, neighbor and vertex-pair queries. The query complexity and running time of their algorithms for r-clique K_r counting are $\tilde{O}(\frac{n}{(\#K_r)^{1/3}} + \min\{m, \frac{m^{r/2}}{\#K_r}\})$ and $\tilde{O}(\frac{n}{(\#K_r)^{1/3}} + \frac{m^{r/2}}{\#K_r})$ respectively, for any $r \geq 3$, where $\#K_r$ is the number of copies of K_r in G. Furthermore, in boths works it was proved that the query complexities of the respective algorithms are optimal up to polylogarithmic dependencies on n, ϵ and r.

[1] Throughout the paper, we use $\tilde{O}(\cdot)$ to suppress any dependencies on the parameter ε, the size of the corresponding subgraph H and $\log(n)$-terms.

Later, Assadi et al. [3] gave a sublinear-time algorithm for $(1 \pm \varepsilon)$-approximating the number of copies of an arbitrary subgraph H in the *augmented general graph model* [2]. That is, besides the aforementioned vertex sampling, degree, neighbor and pair queries, the algorithm is allowed to perform the following type of queries:

- **edge sampling query** the algorithm can sample an edge uniformly at random.

The algorithm in [3] uses $\tilde{O}(\min\{m, \frac{m^{\rho(H)}}{\#H}\})$ queries and $\tilde{O}(\frac{m^{\rho(H)}}{\#H})$ time, where $\rho(H)$ is the fractional edge-cover of H and $\#H$ is the number of copies of H in G. For the special case $H = K_r$, their algorithm performs $\tilde{O}(\min\{m, \frac{m^{r/2}}{\#K_r}\})$ queries and runs in $\tilde{O}(\frac{m^{r/2}}{\#K_r})$ time, which do not have the additive term $\frac{n}{(\#K_r)^{1/3}}$ in the query complexity and running time of the algorithms in [7, 10]. Eden and Rosenbaum provided simple proofs that most of the aforementioned results are nearly optimal in terms of their query complexities by reducing from communication complexity problems [12]. Further investigation of sampling an edge and estimating subgraphs in low arboricity graphs [8, 9] and approximately counting stars [2] has also been performed.

Relation of approximate counting and almost uniform sampling. One of our original motivations is to investigate the relation of approximate counting and almost uniform sampling in the sublinear-time regime. That is, we are interested in the question whether *in the sublinear-time regime, is almost uniform sampling "computationally comparable" to approximate counting, or is it strictly harder or easier, in terms of the query and/or time complexities for solving these two problems?* Indeed, in the polynomial-time regime, Jerrum, Valiant and Vazirani showed that for self-reducible problems (e.g., counting the number of perfect matchings of a graph), approximating counting is "equivalent to" almost uniform sampling [18], in the sense that the time complexities of almost uniform sampling and randomized approximate counting are within polynomial factor of each other. Such a result has been instrumental for the development of the area of approximate counting (e.g., [23]). It is natural to ask if similar relations between approximate counting and sampling hold in the sublinear-time regime.

1.1 Our Results

In this paper, we consider the problem of (almost) uniformly sampling a subgraph in the augmented general graph model. As mentioned above, this model has been studied in [2, 3], in which the authors find that "allowing edge-sample queries results in considerably simpler and more general algorithms for subgraph counting and is hence worth studying on its own". On the other hand, allowing edge sampling queries is also natural in models where neighbor queries are allowed, e.g., in the well-studied bounded-degree model and the general model: most graph representations that allow efficient neighbor queries (e.g., GEXF, GML or GraphML) store edges in linear data structures, which often allows efficient (nearly) uniformly sampling of edges. We refer to [3] for a deeper discussion on allowing edge sampling queries from both theoretical and practical perspectives.

We prove the following upper bound on sampling subgraphs (exactly) uniformly at random and provide a corresponding algorithm in Section 3.

▶ **Theorem 1.** *Let H be an arbitrary subgraph. Let $G = (V, E)$ be a graph with n vertices and m edges. There exists an algorithm in the augmented general graph model that uses $\tilde{O}(\min\{m, \frac{m^{\rho(H)}}{\#H}\})$ queries in expectation, and with probability at least $2/3$, returns a copy of H, if $\#H > 0$. Each returned H is sampled according to the uniform distribution over all copies of H in G. The expected running time of the algorithm is $\tilde{O}(\frac{m^{\rho(H)}}{\#H})$.*

We stress that our sampler is an exactly uniform sampler, i.e., the returned H is sampled from the uniform distribution, while to the best of our knowledge, the previous sublinear-time subgraph sampling algorithms are only *almost* uniform samplers. That is, they return an edge or a clique that is sampled from a distribution that is *close* to the corresponding uniform distribution. Indeed, it has been cast as an open question if it is possible to sample an edge exactly uniformly at random in the general graph model in [11].

Our algorithm is based on one idea from [3] (see also [4]) that uses the fractional edge cover to partition a subgraph H into stars and odd cycles (i.e., Lemma 7). The authors of [3] also provided a scheme called *subgraph-sampler trees* for recursively sampling stars and odd cycles that compose H, while the resulting distribution is not (almost) uniform distribution. Instead, we show that one can sample stars and odd cycles by using rejection sampling in parallel (or, more precisely, sequentially but independently of each other) and check whether they form a copy of H.

To complement our algorithmic result, we give a lower bound on the query complexity for sampling a clique in sublinear time by using a simple reduction from [12]. We show the following theorem and present its proof in Section 4.

▶ **Theorem 2.** *Let $r \geq 3$ be an integer. Suppose $\mathcal{A}$ is an algorithm in the augmented general graph model that for any graph $G = (V, E)$ on n vertices and m edges returns an arbitrary r-clique K_r, if one exists; furthermore, each returned clique K_r is sampled according to a distribution $\mathcal{D}$, such that the total probability mass of $\mathcal{D}$ on the set of all copies of K_r is $\Omega(1)$. Then $\mathcal{A}$ requires $\Omega(\min\{m, \frac{m^{r/2}}{\#K_r \cdot (cr)^r}\})$ queries, for some absolute constant $c > 0$.*

Note that the above theorem gives a lower bound for sampling K_r from almost every non-trivial distribution $\mathcal{D}$. In particular, it holds if $\#K_r > 0$ and $\mathcal{D}$ is a distribution that is only supported on the set of all copies of K_r, e.g., the (almost) uniform distribution on these copies. Together with the query and time complexities of the $(1 \pm \varepsilon)$-approximation algorithm for the number of subgraphs H by Assadi et al. [3] and the lower bound by Eden and Rosenbaum [12] for approximately counting cliques, our Theorems 1 and 2 imply that in the augmented general graph model, *approximately* counting the number of cliques is equivalent to *exactly* sampling cliques in the sense that the query and time complexities of them are within a polylogarithmic factor of each other.

Future Work. Considering real-world applications, it would be interesting to relax the guarantees of the queries available to the algorithm. In particular, one may not be able to sample vertices or edges *exactly* uniformly at random, but only *approximately* uniformly. For example, there exist works that consider weaker query models in which even uniform vertex query is disallowed, and instead they sample vertices almost uniformly at random by performing random walks from some fixed vertex (see, e.g., [5, 6]). Implementing these changes in the model would result in a weaker guarantee for the distribution of sampled subgraphs in Theorem 1 but would be potentially more practical.

2 Preliminaries

Let $G = (V, E)$ be a simple graph with $|V| = n$ vertices and $|E| = m$ edges. For a vertex $v \in V$, we denote by d_v the degree of the vertex, by Γ_v the set of all the neighbors of v, and by E_v the set of edges incident to v. We fix a total order on vertices denoted by $\prec$ as follows:

▶ **Definition 3.** *For any two vertices u and v, we say that $u \prec v$ if $d_u < d_v$ or $d_u = d_v$ and u appears before v in the lexicographic order.*

For any two vertices, we denote by Γ_{uv} the set of the shared neighbors of u and v that are larger than u with respect to "$\prec$", i.e., $\Gamma_{uv} = \{w \mid w \in \Gamma_u \cap \Gamma_v \wedge u \prec w\}$. Sometimes, we view our graph $G = (V, E)$ as a directed graph $(V, \vec{E})$ by treating each undirected edge $e = \{u, v\} \in E$ as two directed edges $\vec{e}_1 = (u, v)$ and $\vec{e}_2 = (v, u)$. The following was proven in [7].

▶ **Lemma 4** ([7]). *For any vertex v, the number of neighbors w of v such that $v \prec w$ is at most $\sqrt{2m}$.*

Given a graph H, we say that a subgraph H' of G is a *copy* or an *instance* of H if H' is isomorphic to H. An isomorphism-preserving mapping from H to a copy of H in G is called an *embedding* of H in G.

Rejection Sampling. Given a starting distribution $\vec{p}$ and a target distribution $\vec{q}$ supported on a set R, let $M := \max_{a \in R} \frac{\vec{q}(a)}{\vec{p}(a)}$. Algorithm 1 is called *rejection sampling*.

Algorithm 1 Rejection sampling with starting distribution $\vec{p}$ and target distribution $\vec{q}$.

```
procedure RejectionSampling($\vec{p}, \vec{q}$)
    $M \leftarrow \max_{a \in R} \frac{\vec{q}(a)}{\vec{p}(a)}$
    while true do
        sample $a$ from $\vec{p}$.
        sample a number $t \in [0, 1]$ uniformly at random.
        if $t \leq \frac{\vec{q}(a)}{M \cdot \vec{p}(a)}$ then
            return $a$
```

Observe that when the algorithm terminates, the probability that a is returned is $\vec{q}(a)$ for every $a \in R$. The following lemma is known.

▶ **Lemma 5** ([22]). *The expected number of iterations of* RejectionSampling*($\vec{p}, \vec{q}$) is M.*

Edge Cover and Graph Decomposition. We use the following definition of the fractional edge cover of a graph and a decomposition result based on it by Assadi et al. [3].

▶ **Definition 6** (Fractional Edge-Cover Number). *A fractional edge-cover of $H(V_H, E_H)$ is a mapping $\psi : E_H \to [0, 1]$ such that for each vertex $v \in V_H$, $\sum_{e \in E_H, v \in e} \psi(e) \geq 1$. The fractional edge-cover number $\rho(H)$ of H is the minimum value of $\sum_{e \in E_H} \psi(e)$ among all fractional edge-covers ψ.*

Let C_k denote the cycle of length k. Let S_k denote a star with k petals, i.e., $S_k = (\{u, v_1, \ldots, v_k\}, \cup_{i \in [k]} \{u, v_k\})$. Let K_k denote a clique on k vertices. It is known that $\rho(C_{2k+1}) = k + 1/2$, $\rho(S_k) = k$ and $\rho(K_k) = k/2$.

▶ **Lemma 7** ([3]). *Any subgraph H can be decomposed into a collection of vertex-disjoint odd cycles $\overline{C_1}, \ldots, \overline{C_o}$ and star graphs $\overline{S_1}, \ldots, \overline{S_s}$ such that*

$$\rho(H) = \sum_{i=1}^{o} \rho(\overline{C_i}) + \sum_{j=1}^{s} \rho(\overline{S_j}).$$

By a result of Atserias, Grohe and Marx [4], the number of instances of H in a graph G with m edges is $O(m^{\rho(H)})$.

3 Sampling an Arbitrary Subgraph H

In this section, we present sampling algorithms for odd cycles and stars and show how to combine them to obtain a sampling algorithm for arbitrary subgraphs. Note that we do not need to know the exact number of edges m to run our algorithm; it is sufficient to have a constant approximation $\hat{m}$ of m so that $m \le \hat{m} \le cm$ for some $c > 1$. Such an approximation can be obtained by using the algorithm from [14, 16]. This increases the query complexity only by a constant factor. For the sake of simplicity, we will continue to use m in the following.

3.1 Sampling an Odd-Length Cycle

We describe our algorithm SampleOddCycle for sampling a uniformly random odd-length k-cycle. For any instance of C_{2k+1} in the input graph, our goal is to guarantee that it will be sampled with probability $\frac{1}{m^{k+1/2}}$. Let $e_1, \dots, e_{2k+1}$ be a sequence of edges that represents a cycle of length $2k+1$. While we can use edge sampling to sample every second edge of the first $2k$ edges sequentially, i.e., $e_1, e_3, \dots, e_{2k-1}$, and query the edges inbetween, i.e., $e_2, \dots, e_{2k-2}$, by vertex pair queries, we use a different strategy to sample e_{2k} and e_{2k+1}. Let $\{u, v\} = e_1$. If u has low degree, i.e., $d_u \le \sqrt{2m}$, we can afford to sample each neighbor of u with probability $1/\sqrt{2m}$ and fail if no neighbor is sampled. In particular, we need that a distinguished neighbor x_1 of u is sampled with probability at least $1/\sqrt{2m}$. However, if $d_u \ge \sqrt{2m}$, this is too costly. Instead, we invoke rejection sampling with the following starting distribution and target distribution.

▶ **Definition 8.** *Let u, v be two vertices such that $d_u > \sqrt{2m}$. Let $\vec{p}_u$ be a (starting) distribution with support Γ_u such that:*

$$\vec{p}_u(w) = \frac{1}{d_u}, \quad w \in \Gamma_u \tag{1}$$

Let $\vec{q}_u$ be a (target) distribution with support Γ_u such that:

$$\vec{q}_u(w) = \begin{cases} \frac{1}{\sqrt{2m}}, & w \in \Gamma_{uv} \\ \left(1 - \frac{|\Gamma_{uv}|}{\sqrt{2m}}\right) \cdot \frac{1}{d_u - |\Gamma_{uv}|}, & w \notin \Gamma_{uv} \end{cases} \tag{2}$$

We note that by Lemma 4, it always holds that $|\Gamma_{uv}| \le \sqrt{2m}$. Furthermore,

$$\sum_{w \in \Gamma_u} \vec{q}_u(w) = \sum_{w \in \Gamma_{uv}} \frac{1}{\sqrt{2m}} + \sum_{w \notin \Gamma_{uv}} \left(1 - \frac{|\Gamma_{uv}|}{\sqrt{2m}}\right) \cdot \frac{1}{d_u - |\Gamma_{uv}|}$$
$$= \frac{|\Gamma_{uv}|}{\sqrt{2m}} + (d_u - |\Gamma_{uv}|)\left(1 - \frac{|\Gamma_{uv}|}{\sqrt{2m}}\right) \cdot \frac{1}{d_u - |\Gamma_{uv}|} = 1.$$

Thus the distribution $\vec{q}_u$ is well-defined. Let $M_u = \max_{w \in \Gamma_u} \frac{\vec{q}_u(w)}{\vec{p}_u(w)}$ (as in Algorithm 1). Then, M_u is bounded as follows.

▶ **Lemma 9.** *Let M_u be defined as above. Recall that $d_u > \sqrt{2m}$. Then $M_u = \frac{d_u}{\sqrt{2m}}$.*

Proof. If $w \in \Gamma_{uv}$, we have that $\frac{\vec{q}_u(w)}{\vec{p}_u(w)} = \frac{d_u}{\sqrt{2m}}$. If $w \notin \Gamma_{uv}$, we have that

$$\frac{\vec{q}_u(w)}{\vec{p}_u(w)} = \frac{d_u(1 - \frac{|\Gamma_{uv}|}{\sqrt{2m}})}{d_u - |\Gamma_{uv}|} = \frac{d_u(\sqrt{2m} - |\Gamma_{uv}|)}{\sqrt{2m}(d_u - |\Gamma_{uv}|)} \le \frac{d_u}{\sqrt{2m}}, \tag{3}$$

where the last inequality uses the fact that $d_u > \sqrt{2m}$. ◀

Algorithm 2 Sampling a wedge.

```
procedure SampleWedge($(G, u, v)$)
    if $d_u \le \sqrt{2m}$ then
        sample a number $i \in \{1, \dots \sqrt{2m}\}$ uniformly at random
        if $i > d_u$ then
            return Fail
        $w \leftarrow i^{th}$ neighbor of $u$
    else
        $w \leftarrow$ RejectionSampling($(\vec{p}_u, \vec{q}_u)$)        ▷ see Definition 8
    return $w$
```

Algorithm 3 Sampling a cycle of length $2k + 1$.

```
procedure SampleOddCycle($(G, 2k + 1)$)
    sample $k$ directed edges $(u_1, v_1), \dots, (u_k, v_k)$ u.a.r. and i.i.d.
    if $u_1, v_1, \dots, u_k, v_k$ is a path of length $2k - 1$ and $u_1 \prec v_1$, $\forall i > 1 : u_1 \prec u_i, v_i$ then
        if SampleWedge($(G, u_1, v_k)$) returns $w$ and $w \prec v_1$ then
            return $\{(u_1, v_1), \dots, (u_k, v_k)\} \cup \{(v_k, w), (w, u_1)\}$
    return Fail
```

As there exists a linear number of automorphisms for every cycle, it is crucial in our algorithm to define a unique embedding based on the order of vertices for every instance of a k-cycle. Otherwise, bounding the probability that an instance is sampled *exactly* uniformly is hard as some instance might be sampled less likely because, e.g., its edges participate in many overlapping cycles. We take care of this by enforcing that only uniquely defined embeddings are sampled in SampleOddCycle. In particular, we sample k directed edges $(u_1, v_1), \dots, (u_k, v_k)$ independently and uniformly at random and require that (i) they induce a path $u_1, v_1, u_2, \dots, v_k$ and (ii) for the first edge (u_1, v_1), u_1 is the smallest vertex according to the order "$\prec$" among all $u_i, v_i, i \ge 1$. Then, we call SampleWedge on the two ends u_1, v_k of this path to close a cycle and define an orientation of this cycle by requiring that $w \prec v_1$, where for $(v_k, w) = e_{2k+1}$. If any of these requirements is not met, we have not sampled the uniquely defined embedding we are looking for, and the algorithm fails.

▶ **Lemma 10.** *For any instance of an odd cycle C_{2k+1} in G, the probability that it will be returned by SampleOddCycle$(G, 2k + 1)$ is $\frac{1}{(2m)^{k+1/2}}$.*

Proof. Let $\mathcal{C}_{2k+1}$ be any instance of a cycle of odd length $2k + 1$ in G. Let x_0 be the smallest vertex on $\mathcal{C}_{2k+1}$ according to the total order "$\prec$". Let x_1, x_{2k} be the two neighbors of x_0 on $\mathcal{C}_{2k+1}$ such that $x_1 \prec x_{2k}$. Then, we let x_i denote the vertices on $\mathcal{C}_{2k+1}$ such that $(x_i, x_{i+1}) \in E(\mathcal{C}_{2k+1})$ for $0 \le i \le 2k - 1$ and $(x_{2k}, x_0) \in E(\mathcal{C}_{2k+1})$. Note that for any $\mathcal{C}_{2k+1}$, there is a *unique* way of mapping its vertices to x_i, for $0 \le i \le 2k$. Thus, SampleOddCycle returns $\mathcal{C}_{2k+1}$ if and only if

1. $u_1 = x_0$ and $v_1 = x_{2k}$;
2. $u_i = x_{2k-2i+3}$ and $v_i = x_{2k-2i+2}$ for $2 \le i \le k$;
3. SampleWedge(G, u_1, v_k) returns x_1.

Event 1 occurs with probability $1/(2m)$, and event 2 occurs with probability $1/(2m)^{k-1}$, as each directed edge is sampled with probability $1/(2m)$.

Now we bound the probability of event 3. In the call to SampleWedge, let $u := u_1$ and $v := v_k$, which satisfies that $u \prec v$. We first note that if $d_u < \sqrt{2m}$ in SampleWedge(G, u_1, v_k), then the vertex x_1 will be sampled with probability $1/\sqrt{2m}$. Now we consider the case that $d_u \geq \sqrt{2m}$. Then, RejectionSampling$(\vec{p}_u, \vec{q}_u)$ will return x_1 with probability $\vec{q}_{u_1}(x_1) = \frac{1}{\sqrt{2m}}$, as x_1 is a common neighbor of u_1, v_k and $u_1 \prec x_1$. Thus in both cases, the probability that event 3 occurs is $\frac{1}{\sqrt{2m}}$. Therefore, the probability that SampleOddCycle returns $\mathcal{C}_{2k+1}$ is $\frac{1}{\sqrt{2m}} \cdot \frac{1}{2m} \cdot (\frac{1}{2m})^{k-1} = \frac{1}{(2m)^{k+1/2}}$. ◀

3.2 Sampling a Star

Similarly to odd cycles, we observe that every k-star admits an exponential number of automorphisms. Therefore, we enforce a unique embedding of every instance of a k-star in our sampling algorithm SampleStar. Let $e_1, \dots, e_k$ be the petals of an instance of a k-star. We sample $e_1, \dots, e_k$ sequentially. If these edges form a star, we output it only if the leaves where sampled in ascending order with respect to "$\prec$".

Algorithm 4 Sampling a star with k petals.

```
procedure SampleStar($(G, k)$)
    Sequentially sample $k$ directed edges $\{(u_1, v_1), \dots, (u_k, v_k)\}$ u.a.r. and i.i.d.
    if $u_1 = u_2 = \dots = u_k$ and $v_1 \prec v_2 \prec \dots \prec v_k$ then
        return $(u_1, v_1, \dots, v_k)$
    return Fail
```

▶ **Lemma 11.** *For any instance of a k-star S_k in G, the probability that it will be returned by the algorithm* SampleStar(G, k) *is* $\frac{1}{(2m)^k}$.

Proof. Consider any instance of S_k with root x and petals $y_1, \dots, y_k$ such that $y_1 \prec \dots y_k$. Note that it will be returned by SampleStar if and only if all the directed edges $(x, y_1), \dots, (x, y_k)$ are sequentially sampled, which occurs with probability $1/(2m)^k$. ◀

3.3 Sampling H

Let H be a subgraph. It can be decomposed into collections of o odd cycles $\overline{C_i}$ and s stars $\overline{S_j}$ as given in Lemma 7. We say that H has a (decomposition) *type* $\overline{T} = \{\overline{C_1}, \dots, \overline{C_o}, \overline{S_1}, \dots, \overline{S_s}\}$.

▶ **Definition 12.** *Given a graph G, for each potential* instance $\mathcal{H}$ *of H, we say that $\mathcal{H}$ can be decomposed into* configurations $\mathcal{T} = \{\mathcal{C}_1, \dots, \mathcal{C}_o, \mathcal{S}_1, \dots, \mathcal{S}_s\}$ *with respect to type $\overline{T} = \{\overline{C_1}, \dots, \overline{C_o}, \overline{S_1}, \dots, \overline{S_s}\}$, if*

1. *$\mathcal{C}_i \cong \overline{C_i}$ for any $1 \leq i \leq o$, and $\mathcal{S}_j \cong \overline{S}_j$, for any $1 \leq i \leq s$*
2. *all the remaining edges of H between vertices specified in $\mathcal{T}$ all are present in G.*

We let $f_{\overline{T}}(H)$ denote the number of all possible configurations $\mathcal{T}$ into which H can be decomposed with respect to $\overline{T}$.

Algorithm 5 Sampling a copy of subgraph H.

```
procedure SampleSubgraph($G, H$)
    Let $\overline{T} = \{\overline{C_1}, \dots, \overline{C_o}, \overline{S_1}, \dots, \overline{S_s}\}$ denote a (decomposition) type of $H$.
    for all $i = 1 \dots o$ do
        if SampleOddCycle($G, |E(\overline{C}_i)|$) returns a cycle $\mathcal{C}$ then
            $\mathcal{C}_i \leftarrow \mathcal{C}$
        else
            return Fail
    for all $j = 1 \dots s$ do
        if SampleStar($G, |V(\overline{S}_j)| - 1$) returns a star $\mathcal{S}$ then
            $\mathcal{S}_j \leftarrow \mathcal{S}$
        else
            return Fail
    Query all edges $(\bigcup_{i \in [o]} V(\mathcal{C}_i) \cup \bigcup_{j \in [s]} V(\mathcal{S}_j))^2$
    if $S := (\mathcal{C}_1, \dots, \mathcal{C}_o, \mathcal{S}_1, \dots, \mathcal{S}_s)$ forms a copy of $H$ then
        flip a coin and with probability $\frac{1}{f_{\overline{T}}(H)}$: return $S$
    return Fail
```

▶ **Lemma 13.** *For any instance of a subgraph H in G, the probability that it will be returned by the algorithm* SampleSubgraph*(G, H) is* $\frac{1}{(2m)^{\rho(H)}}$.

Proof. For any instance $\mathcal{H}$ of H in G, and any configuration $\mathcal{T} = \{\mathcal{C}_1, \dots, \mathcal{C}_O, \mathcal{S}_1, \dots, \mathcal{S}_s\}$ of $\mathcal{H}$ with respect to $\overline{T}$, $\mathcal{H}$ will be returned by SampleSubgraph(G, H) if and only if

1. $\mathcal{C}_i$ is returned in Algorithm 5 for each $1 \le i \le o$, and $\mathcal{S}_j$ is returned in Algorithm 5 for any $1 \le j \le s$;
2. the configuration is returned with probability $\frac{1}{f_{\overline{T}}(H)}$ in Algorithm 5.

By Lemma 10, each $\mathcal{C}_i$ will be returned with probability $\frac{1}{(2m)^{|E(\overline{C_i})|/2}} = \frac{1}{(2m)^{\rho(\overline{C_i})}}$. By Lemma 11 each $\mathcal{S}_j$ will be returned with probability $\frac{1}{(2m)^{|V(\overline{S_j})|-1}} = \frac{1}{(2m)^{\rho(\overline{S_j})}}$. Thus, $\mathcal{T}$ will be returned with probability

$$\prod_{i=1}^{o} \frac{1}{(2m)^{\rho(\overline{C_i})}} \cdot \prod_{j=1}^{s} \frac{1}{(2m)^{\rho(\overline{S_j})}} \cdot \frac{1}{f_{\overline{T}}(H)} = \frac{1}{(2m)^{\rho(H)}} \cdot \frac{1}{f_{\overline{T}}(H)}.$$

Finally, since there are $f_{\overline{T}}(H)$ configurations of $\mathcal{H}$ with respect to $\overline{T}$, the instance will be returned with probability $f_{\overline{T}}(H) \cdot \frac{1}{(2m)^{\rho(H)}} \cdot \frac{1}{f_{\overline{T}}(H)} = \frac{1}{(2m)^{\rho(H)}}$. ◀

3.4 The Final Sampler

Let X_H be an estimate of $\#H$. Such an estimate can be obtained by, e.g., the subgraph counting algorithm of Assadi et al. [3] in expected time $\tilde{O}(m^{\rho(H)}/\#H)$. We show that by sufficiently many calls to SampleSubgraph, we can obtain a uniformly random sample of an instance of H with constant probability.

Algorithm 6 Sampling a copy of subgraph H uniformly at random.

```
1: procedure SampleSubgraphUniformly($(G, H, X_H)$)
2:     for all $j = 1, \ldots, q = 10 \cdot (2m)^{\rho(H)}/X_H$ do
3:         Invoke SampleSubgraph($(G, H)$)
4:         if a subgraph $H$ is returned then return $H$
5:     return Fail
```

▶ **Lemma 14.** *If $\#H \leq X_H \leq 2\#H$, then Algorithm* SampleSubgraphUniformly(G, H, X_H) *returns a copy H with probability at least $2/3$. The distribution induced by the algorithm is (exactly) uniform over the set of all instances of H in G.*

Proof. Since $\#H \leq X_H \leq 2\#H$, the probability that no instance of H is returned in $q = 10 \cdot (2m)^{\rho(H)}/X_H$ invocations is at most

$$\left(1 - \frac{\#H}{(2m)^{\rho(H)}}\right)^q \leq e^{-\frac{\#H}{(2m)^{\rho(H)}} \cdot q} < \frac{1}{3}$$

by Lemma 13. Let $\mathcal{H}$ be an instance of H. By Lemma 13, the probability that SampleSubgraph(H) returns $\mathcal{H}$ is $\frac{1}{(2m)^{\rho(H)}}$. Thus, the probability that SampleSubgraphUniformly(G, H) successfully output an instance of H is

$$\frac{\#H}{(2m)^{\rho(H)}}.$$

Conditioned on the event that SampleSubgraphUniformly(G, H) succeeds, the probability that any specific instance $\mathcal{H}$ will be returned is

$$p_{\mathcal{H}} = \frac{\frac{1}{(2m)^{\rho(H)}}}{\frac{\#H}{(2m)^{\rho(H)}}} = \frac{1}{\#H}.$$

That is, with probability at least $\frac{2}{3}$, an instance $\mathcal{H}$ is sampled from the uniform distribution over all the instances of H in G. ◀

Finally, we prove the expected query and time complexity of SampleSubgraphUniformly.

▶ **Lemma 15.** *The expected query and time complexity of* SampleSubgraphUniformly(G, H, X_H) *is $O(m^{\rho(H)}/X_H)$.*

Proof. We analyze the query complexity of SampleOddCycle$(G, 2k + 1)$ for $d_{u_1} < \sqrt{2m}$ and $d_{u_1} \geq \sqrt{2m}$ separately. The probability that $d_{u_1} < \sqrt{2m}$ is at most 1, and the query complexity is at most $O(1)$ in this case.

To bound the probability that SampleWedge(G, u_1, v_k) is invoked such that $d_{u_1} \geq \sqrt{2m}$, recall that sampling an edge uniformly at random is equivalent to sampling a vertex proportionally to its degree and selecting a neighbor uniformly at random. The probability to sample a neighbor x of u_1 is $1/d_{u_1}$. There at most $2m/\sqrt{2m} = \sqrt{2m}$ vertices that have degree at least $\sqrt{2m}$, so the probability that a uniformly random neighbor v_1 of u_1 has degree at least $\sqrt{2m}$ is at most $\sqrt{2m}/d_{u_1}$. Therefore, the probability that v_1 has degree at least $\sqrt{2m}$, which is implied by the check $u_1 \prec v_1$ in line 3, is bounded by $\sqrt{2m}/d_{u_1}$. By Lemmas 5 and 9, the expected number of queries in SampleWedge(G, u_1, v_k) is at most $M \leq d_{u_1}/\sqrt{2m}$ if $d_{u_1} \geq \sqrt{2m}$. Thus, the expected query complexity of SampleOddCycle$(G, 2k + 1)$ is bounded by

$$\sum_{\substack{u_1\in V\\ d_{u_1}<\sqrt{2m}}} \frac{d_{u_1}}{2m}\cdot O(1) + \sum_{\substack{u_1\in V\\ d_{u_1}\geq\sqrt{2m}}} \frac{d_{u_1}}{2m}\cdot\frac{\sqrt{2m}}{d_{u_1}}\cdot\frac{d_{u_1}}{\sqrt{2m}} \leq O(1) + \sum_{\substack{u_1\in V\\ d_{u_1}\geq\sqrt{2m}}} \frac{d_{u_1}}{2m} = O(1).$$

The expected query complexity of SampleStar(G, k) is bounded by $k \in O(1)$. It follows that the expected query complexity of SampleSubgraph(G, H) is at most $(o + s + |H|^2)\cdot O(1) \subseteq |H|\cdot O(1)$. The expected query complexity of SampleSubgraphUniformly(G, H) is $O((2m)^{\rho(H)}/X_H\cdot|H|^2) = \tilde{O}((2m)^{\rho(H)}/X_H)$. To bound the expected running time, we observe that every loop in our algorithm issues at least one query, and we only perform isomorphism checks on subgraphs of constant size. Thus the running time is still $\tilde{O}((2m)^{\rho(H)}/X_H)$. ◀

The proof of Theorem 1 follows almost directly from Lemmas 14 and 15.

Proof of Theorem 1. For the case that $m \geq m^{\rho(H)}/\#H$, the claim follows from Lemmas 14 and 15. If $m < m^{\rho(H)}/\#H$, we can query the whole graph, which requires $O(m)$ degree and neighbor queries, store the graph and answer the queries of the algorithm from this internal memory. ◀

4 Proof of Theorem 2

In this section, we give the proof of Theorem 2, which follows by adapting the proofs for the lower bounds on the query complexity for approximate counting subgraphs given by Eden and Rosenbaum [12].

▶ **Theorem 16** (see Theorems 4.7 and B.1 in [12]). *For any choices of $n, m, r, c_r > 0$, there exist families of graphs with n vertices and m edges, $\mathcal{F}_0$ and $\mathcal{F}_1$, such that*

- *all graphs in $\mathcal{F}_0$ are K_r-free,*
- *all graphs in $\mathcal{F}_1$ contain at least c_r copies of K_r,*
- *and any algorithm in the augmented general graph model that distinguishes a graph $G \in \mathcal{F}_0$ from $G \in \mathcal{F}_1$ with probability $\Omega(1)$ requires $\Omega(\min\{m, m^{r/2}/c_r(cr)^r\})$ queries for some constant $c > 0$.*

Now we prove our Theorem 2.

Proof of Theorem 2. Let $\mathcal{A}$ be an algorithm that for any graph $G = (V, E)$ on n vertices and m edges returns an arbitrary r-clique K_r, if one exists; and each K_r is sampled according to $\mathcal{D}$, using $f(m, r, \#K_r) \in o(\min\{m, \frac{m^{r/2}}{\#K_r\cdot(cr)^r}\})$ neighbor, degree, pair and edge sampling queries.

Let $n, m, c_r > 0$ and let $\mathcal{F}_0, \mathcal{F}_1$ be the families from Theorem 16. Consider the following algorithm $\mathcal{A}'$: run $\mathcal{A}$ on a graph from $\mathcal{F}_0 \cup \mathcal{F}_1$ and terminate $\mathcal{A}$ if it did not produce a K_r after $f(m, r, c_r)$ queries. If it output a clique, $\mathcal{A}'$ claims that $G \in \mathcal{F}_1$, otherwise it claims that $G \in \mathcal{F}_0$. By the assumption, $\mathcal{A}$ returns a clique after at most $f(m, r, c_r)$ queries with probability $\Omega(1)$ if $G \in \mathcal{F}_1$ because then G contains at least c_r copies of K_r and the probability mass of $\mathcal{D}$ on the set of all copies of K_r is $\Omega(1)$. Otherwise, $G \in \mathcal{F}_0$, which implies that G contains no triangle. Therefore, $\mathcal{A}$ cannot output a triangle from G.

It follows that $\mathcal{A}'$ can distinguish $\mathcal{F}_0$ and $\mathcal{F}_1$, which is a contradiction to Theorem 16. ◀

References

1 Nesreen K Ahmed, Nick Duffield, Theodore L Willke, and Ryan A Rossi. On sampling from massive graph streams. *Proceedings of the VLDB Endowment*, 10(11), 2017. `doi:10.14778/3137628.3137651`.

2 Maryam Aliakbarpour, Amartya Shankha Biswas, Themis Gouleakis, John Peebles, Ronitt Rubinfeld, and Anak Yodpinyanee. Sublinear-Time Algorithms for Counting Star Subgraphs via Edge Sampling. *Algorithmica*, 2017. `doi:10.1007/s00453-017-0287-3`.

3 Sepehr Assadi, Michael Kapralov, and Sanjeev Khanna. A Simple Sublinear-Time Algorithm for Counting Arbitrary Subgraphs via Edge Sampling. In *Proceedings of the 10th Innovations in Theoretical Computer Science Conference (ITCS)*, volume 124. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018. `doi:10.4230/LIPIcs.ITCS.2019.6`.

4 Albert Atserias, Martin Grohe, and Dániel Marx. Size bounds and query plans for relational joins. In *Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2008. `doi:10.1109/FOCS.2008.43`.

5 Anna Ben-Hamou, Roberto I Oliveira, and Yuval Peres. Estimating graph parameters via random walks with restarts. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, 2018. `doi:10.1137/1.9781611975031.111`.

6 Flavio Chiericetti, Anirban Dasgupta, Ravi Kumar, Silvio Lattanzi, and Tamás Sarlós. On sampling nodes in a network. In *Proceedings of the 25th International Conference on World Wide Web*, 2016. `doi:10.1145/2872427.2883045`.

7 Talya Eden, Amit Levi, Dana Ron, and C Seshadhri. Approximately counting triangles in sublinear time. *SIAM Journal on Computing*, 46(5), 2017. `doi:10.1137/15M1054389`.

8 Talya Eden, Dana Ron, and Will Rosenbaum. The Arboricity Captures the Complexity of Sampling Edges. In *46th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 132. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019. `doi:10.4230/LIPIcs.ICALP.2019.52`.

9 Talya Eden, Dana Ron, and C. Seshadhri. Faster sublinear approximations of k-cliques for low arboricity graphs. *CoRR*, abs/1811.04425, 2018. `arXiv:1811.04425`.

10 Talya Eden, Dana Ron, and C Seshadhri. On approximating the number of k-cliques in sublinear time. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, 2018. `doi:10.1145/3188745.3188810`.

11 Talya Eden and Will Rosenbaum. On sampling edges almost uniformly. *arXiv preprint*, 2017. `arXiv:1706.09748`.

12 Talya Eden and Will Rosenbaum. Lower Bounds for Approximating Graph Parameters via Communication Complexity. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, volume 116. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018. `doi:10.4230/LIPIcs.APPROX-RANDOM.2018.11`.

13 Talya Eden and Will Rosenbaum. On Sampling Edges Almost Uniformly. In *1st Symposium on Simplicity in Algorithms (SOSA)*, volume 61. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018. `doi:10.4230/OASIcs.SOSA.2018.7`.

14 Uriel Feige. On sums of independent random variables with unbounded variance and estimating the average degree in a graph. *SIAM Journal on Computing*, 35(4), 2006. `doi:10.1137/S0097539704447304`.

15 Weiming Feng, Yuxin Sun, and Yitong Yin. What can be sampled locally? In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, 2017. `doi:10.1007/s00446-018-0332-8`.

16 Oded Goldreich and Dana Ron. Approximating average parameters of graphs. *Random Structures & Algorithms*, 32(4), 2008. `doi:10.1002/rsa.20203`.

17 Pili Hu and Wing Cheong Lau. A survey and taxonomy of graph sampling. *arXiv preprint*, 2013. `arXiv:1308.5865`.

18 Mark R Jerrum, Leslie G Valiant, and Vijay V Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theoretical Computer Science*, 43, 1986. doi:10.5555/11534.11537.

19 Tali Kaufman, Michael Krivelevich, and Dana Ron. Tight bounds for testing bipartiteness in general graphs. *SIAM Journal on Computing*, 33(6), 2004. doi:10.1137/S0097539703436424.

20 Jure Leskovec and Christos Faloutsos. Sampling from large graphs. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2006. doi:10.1145/1150402.1150479.

21 R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. Network motifs: Simple building blocks of complex networks. *Science*, 298(5594), 2002. doi:10.1126/science.298.5594.824.

22 Von Neumann. Various techniques used in connection with random digits. *National Bureau of Standards, Applied Math Series*, 12, 1950.

23 Alistair Sinclair and Mark Jerrum. Approximate counting, uniform generation and rapidly mixing markov chains. *Information and Computation*, 82(1), 1989. doi:10.1016/0890-5401(89)90067-9.

A Water-Filling Primal-Dual Algorithm for Approximating Non-Linear Covering Problems

Andrés Fielbaum
Department of Cognitive Robotics, Faculty of Mechanical, Maritime and Materials Engineering, TU Delft, The Netherlands
a.s.fielbaumschnitzler@tudelft.nl

Ignacio Morales
Departamento de Ingeniería Industrial, Escuela de Ingeniería,
Pontificia Universidad Católica, Santiago, Chile
inmorales@uc.cl

José Verschae
Instituto de Ingeniería Matemática y Computacional,
Facultad de Matemáticas y Escuela de Ingeniería,
Pontificia Universidad Católica, Santiago, Chile
jverschae@uc.cl

Abstract

Obtaining strong linear relaxations of capacitated covering problems constitute a significant technical challenge even for simple settings. For one of the most basic cases, the Knapsack-Cover (Min-Knapsack) problem, the relaxation based on *knapsack-cover inequalities* has an integrality gap of 2. These inequalities are exploited in more general problems, many of which admit primal-dual approximation algorithms.

Inspired by problems from power and transport systems, we introduce a general setting in which items can be taken fractionally to cover a given demand. The cost incurred by an item is given by an arbitrary non-decreasing function of the chosen fraction. We generalize the knapsack-cover inequalities to this setting an use them to obtain a $(2+\varepsilon)$-approximate primal-dual algorithm. Our procedure has a natural interpretation as a bucket-filling algorithm which effectively overcomes the difficulties implied by having different slopes in the cost functions. More precisely, when some superior segment of an item presents a low slope, it helps to increase the priority of inferior segments. We also present a rounding algorithm with an approximation guarantee of 2.

We generalize our algorithm to the Unsplittable Flow-Cover problem on a line, also for the setting of fractional items with non-linear costs. For this problem we obtain a $(4+\varepsilon)$-approximation algorithm in polynomial time, almost matching the 4-approximation algorithm known for the classical setting.

2012 ACM Subject Classification Theory of computation → Packing and covering problems; Mathematics of computing → Discrete optimization; Mathematics of computing → Linear programming

Keywords and phrases Knapsack-Cover Inequalities, Non-Linear Knapsack-Cover, Primal-Dual, Water-Filling Algorithm

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.46

Category Track A: Algorithms, Complexity and Games

Related Version A full version of this manuscript can be found at https://arxiv.org/abs/1912.12151.

Funding This article was partially funded by Fondecyt Project Nr. 1181527.

1 Introduction

Covering problems have been heavily studied by the combinatorial optimization community. Understanding their polyhedral descriptions, and how to approximate them, is a challenging and important task even for simple variants. One of the main tools for obtaining strong linear relaxations of covering problems are the *knapsack-cover inequalities*, introduced by Carr et al. [9], and have been used extensively [4, 18, 3, 3, 8, 10, 19, 1, 20, 2, 22]. Although the inequalities are exponentially many, they can be approximately separated up to a factor of $1+\varepsilon$ in polynomial time. In many cases they are well adjusted for primal-dual algorithms, which avoid having to solve the relaxation and yield faster combinatorial algorithms [4, 8, 10, 20].

In the classical Knapsack-Cover problem we are given a set of n items N and a demand $D \in \mathbb{N}$. Each item i has an associated covering capacity u_i and cost c_i. The objective is to choose a subset of items covering D at a minimum cost. We introduce a new natural generalization of this problem motivated by applications on the operation of power systems and the design of public transport systems. In this version we can choose items partially at a given cost, which might be non-linear. More precisely, each item $i \in N$ have an associated non-decreasing cost function $f_i : \mathbb{N} \to \mathbb{Q}_{\geq 0} \cup \{\infty\}$. We must choose a number $x_i \in \mathbb{N}$ for each i such that $\sum_{i\in N} x_i \geq D$. The cost of such solution is $\sum_{i\in N} f_i(x_i)$. We can reduce the Knapsack-Cover problem to this setting by considering $f_i(0) = 0$, $f_i(1) = \ldots = f_i(u_i) = c_i$ and $f_i(x) = \infty$ for $x > u_i$. We say that we are in the *list model* if the input contains the numbers $f_i(0), f_i(1), \ldots, f_i(D)$ explicitly as a list. In this case the reduction above is pseudo-polynomial. On the other hand, if each f_i is given by an oracle that outputs $f_i(x)$ for any x, we say that we are in the *oracle model*. In this case the reduction above can be made polynomial. We call our newly introduced problem the *Non-Linear Knapsack-Cover problem*. Our setting also generalizes the *Single-Demand Facility Location* problem studied by Carnes and Shmoys [8]. In this setting each item (facility) has an activation cost b_i and then the cost grows linearly at a rate of a_i, that is, $f_i(0) = 0$, $f_i(x) = b_i + c_i x$ for $x \in \{1, \ldots, u_i\}$, and $f_i(x) = \infty$ otherwise.

More generally, we study the Non-Linear variant of the Unsplittable Flow-Cover problem on a path (UFP-cover), that extends the Non-Linear Knapsack-Cover problem. In the original UFP-cover problem, first considered by Bar-Noy et al. [4], we have a discrete interval $I = \{1, \ldots, k\}$ and a set N of n items, each one characterized by a capacity or height u_i, a cost c_i, and a sub-interval $I_i \subseteq \{1, \ldots, k\}$. We also have a demand D_t for each $t \in I$. The problem consists on selecting the cheapest set of items such that the total height at any point in I is at least the demand, that is, we must pick a set S minimizing $\sum_{i\in S} c_i$ such that $\sum_{i\in S: I_i \ni t} u_i \geq D_t$ for all $t \in I$. In this paper we generalize this problem to the case where items can be taken partially. As before we are giving a non-decreasing function $f_i = \mathbb{N} \to \mathbb{Q}_{\geq 0} \cup \{\infty\}$ for each item. We can choose to set the height of any item to a value $x_i \in \mathbb{N}$ by paying a cost $f_i(x_i)$. We mush choose heights in order to cover the demand at each point $t \in I$ at a minimum total cost $\sum_i f_i(x_i)$. Notice that this setting generalizes Non-Linear Knapsack-Cover.

In this article we provide a generalization of the knapsack-cover inequalities to the Non-Linear Knapsack-Cover problem, which we also apply to Non-Linear UFP-Cover. The obtained relaxations yield primal-dual algorithms matching the classical settings. Namely, for Non-Linear Knapsack-Cover we show a 2-approximation algorithm, and for Non-Linear UFP-Cover a 4-approximation algorithm, both running in polynomial time in the list model. For the oracle model, they can be adapted to yield a $(2+\varepsilon)$- and $(4+\varepsilon)$-approximation, respectively, in polynomial time. Additionally, we show a rounding technique for the Non-Linear Knapsack-Cover case also achieving a 2-approximation for the list model, together with a polynomial time separation algorithm.

Applications

One of our main motivations for considering non-linear cost functions comes from the Unit Commitment Problem (UCP), a prominent problem in the operation of power systems. In its most basic version, a central planner, called Independent System Operator (ISO), must schedule the production of energy generated from a given set of power plants, in order to satisfy a given demand. For the case of one time period, the problem corresponds to Non-Linear Knapsack-Cover. More precisely, items correspond to power plants, and $f_i(x)$ is the cost of producing x units of power by power plant i. A common issue in this setting is that plants incur fixed costs for starting production, and after the resource is available, a minimum amount of energy must be produced. It is worth noticing that after paying the fixed cost the behavior of the cost functions might be non-linear and are often modelled by convex quadratic functions [24].

On the other hand, Non-Linear UFP-Cover appears in the optimization of transport systems. Consider an avenue with several bus stops $\{1,\dots,k\}$. We interpret the avenue as a path network where bus stops correspond to edges. Passengers need to move (in a single direction) within bus stops, and hence, we associate to each passenger a set of consecutive bus stops (i.e., a path) which they need to traverse. The set of paths defines a flow that needs to traverse the network. Hence, we obtain a demand D_t at each bus stop t, representing the total number of passengers (amount of flow) that must traverse it. On the supply side, there are potential bus transit lines, each covering some sub-path of the avenue, and hence covering the demand on some subset of consecutive bus stops $I_i = \{f_i, f_i+1, \dots, l_i\} \subseteq \{1,\dots,k\}$. As the planner of this system, we must choose which lines to operate. Additionally, for each chosen line, we must choose its operation frequency and the type of vehicle to use. Such combinations of frequency and vehicle define the amount of passengers (demand) the line can transport. Assume that for line i and each demand x, we can optimally choose (that is, we have an oracle) the frequency and vehicle combination to minimize the operating cost of the line, which we call $f_i(x)$. In other words, for each line i, we must choose a demand $x_i \geq 0$ to cover, such that the total operation cost $\sum_i f_i(x_i)$ is minimized. It is not hard to see that covering the demand at each bus stop guarantees that all passengers can be transported. Hence, it suffices that $\sum_{i:t\in I_i} x_i \geq D_t$ holds for each bus stop $t \in \{1,\dots,k\}$. Hence, we obtain an instance of Non-Linear UFP-Cover.

We might wonder what type of cost functions f_i one can get in this setting. There is a vast literature concerning economies of scale in public transport lines: for instance, Mohring [21] states that there are economies of scale in public transport, Fielbaum et al. [14] show that they get exhausted. Coulombel and Monchambert [12] propose that the system could face diseconomies of scale when the demand exceeds certain thresholds. Hence, techniques to manage non-linear functions (that can have convex and concave regions) are needed.

Related Work

The use of the primal-dual method to derive approximation algorithms is introduced by Bar-Yehuda and Even [5] and Chvátal [11], becoming one of the major tools for designing approximation algorithms [23]. Bar-Noy et al. [4] are the firsts to consider the primal-dual framework based on knapsack-cover inequalities. However, they pose their algorithm in the equivalent local-ratio framework [6], even before the knapsack-cover inequalities were introduced and without stating the underlying LP-relaxation. Their techniques yield a 4-approximation algorithm and their analysis is tight [10]. Additionally, this problem admits a quasi-polynomial time approximation scheme (QPTAS) [16]. On the other hand, Carnes and Shmoys [8] give an explicit description of the primal-dual method, obtaining a 2-approximation algorithm for Knapsack-Cover, the Single-Demand Facility Location problem, and the more general Single-item Lot-Sizing problem with Linear Holding Costs. Cheung et al. [10] consider the Generalized Min-Sum Scheduling problem on a single machine without release dates; they obtain a $(4+\varepsilon)$-approximation algorithm based on the primal-dual framework on an LP with knapsack-cover inequalities. Finally, McCormick et al. [20] consider covering problems with precedence constraints, where they give a primal-dual algorithm with approximation ratio equal to the width of the precedence relations. We remark that Non-Linear Knapsack-Cover can be modeled within this framework, but applying this result to this case yields an unbounded approximation guarantee.

Outside the primal-dual framework, the literature is rich on the use of the knapsack-cover inequalities and its generalizations together with rounding techniques. The problems considered include the Min-Sum General Scheduling problem on a single [3] and multiple [22, 2] machines, the Uniform [18] and Non-Uniform [19] Capacitated Multi-item Lot-sizing problem, and Capacitated Facility Location [1]. On the other hand, it is not known if there exists a compact set of constraints matching the strength of the knapsack-cover inequalities. Recently, Bazzi et al. [7] gave a formulation with an integrality gap of $2+\varepsilon$ for the Knapsack-Cover problem with a quasipolinomial number of inequalities.

It is also worth mentioning that the most common technique for dealing with non-linear cost functions in capacitated covering problems is a doubling technique: split the cost function in segments where the function doubles. Then, each segment can be considered independently as a single item. This technique removes the precedence dependence between different segments, at factor 4 loss in the approximation ratio; see for example [3]. Similarly, with a randomized shifting strategy an e-approximation is achievable [17, 15]. Our approach strengthen the knapsack-cover inequalities and allows to avoid the extra loss.

Our Contribution

Let z_{ij} be a binary variable that represents whether $x_i \geq j$ in the solution, i.e., if the j-th unitary *segment* of item i is *taken*. Defining $g_{ij} = f_i(j) - f_i(j-1)$, then the cost of any solution is $\sum_{ij} g_{ij} z_{ij}$, and for any solution to be feasible it must hold for all i, j that if $z_{ij} = 1$ then $z_{i,j-1} = 1$. In a greedy algorithm, one might be tempted to take segments with low g_{ij}. This fails as such segments might be preceded by another segment with $g_{ik} \gg g_{ij}$ for $k < j$. This poses two fundamental questions when assessing the value of a segment: (i) how to take into consideration (mandatory) preceding segments of high cost? (ii) how to take into account low costs segments to the right, specially considering segments that finally might not be part of the final solution (since the demand can be completely covered by previous segments)?

We introduce a natural variant of the knapsack-cover inequalities for non-linear cost functions. These generalize the basic version of the inequalities, as well the generalization of Carnes and Shmoys [8] for the Single-Demand Facility Location Problem. Our inequalities are then used to derive a primal-dual algorithm that helps to handle the fundamental questions stated above. Our algorithm can be interpreted as a water-filling algorithm. Each segment j of an item i has a corresponding *bucket* B_{ij} of capacity g_{ij}, representing an inequality in the dual linear program. All buckets for a given item i are placed on a stairway, where bucket B_{ij} is on the j-th step of the stairs. A segment is taken, i.e. we set $z_{ij} = 1$, if its corresponding bucket and all previous ones (which are in lower steps of the stairs) are full. Water reaches buckets through two mechanisms. Water from an external source is poured directly into each bucket at a rate of either 1 or 0 (units of water per time unit). The first time a bucket B_{ij} becomes full, then the water arriving to this bucket spills to bucket $B_{i,j-1}$, which now fills at a rate of 2 (as long as B_{ij} is still receiving water from the external source). If $B_{i,j-1}$ also becomes full and $j > 2$, then the water pouring into B_{ij} and $B_{i,j-1}$ spills to $B_{i,j-2}$ which now fills at a rate of 3, etc. For a bucket to receive water from the external source it must satisfy two properties: (i) its corresponding segment is not yet in the primal solution, and (ii) all previous segments of the item are not enough to cover the remaining demand. Our primal-dual algorithm helps to take care of the tensions implied by the questions above by making buckets filling faster due to water spilled from higher buckets, and prevents spilling water from a bucket if they are so high that they are useless to help covering the remaining demand.

For the case of Non-Linear UFP-Cover, our algorithm works similarly. However, the primal solution constructed with the algorithm might contain redundant segments due to sub-intervals of I that might be covered in subsequent steps of the algorithm. For this reason we need to perform a *reverse-delete* (or pruning) strategy to remove unnecessary segments, in the reverse order in which they were introduced in the primal solution.

We notice that, for both algorithms, our analysis is tight as they achieve the same performance guarantee as their classic variants [8, 10]. Additionally, the integrality gap of our formulation for the Non-Linear Knapsack-Cover problem is also 2, as the same lower bound of the the classical setting holds [9].

Finally, we show a rounding technique for the LP relaxation of the Non-Linear Knapsack-Cover problem and a polynomial time separation algorithm for the generalized knapsack-cover inequalities in the list model. These results, together with some of the proofs and extra details, can be found in the full version of this manuscript [13].

2 A Generalization of the Knapsack-Cover Inequalities for Non-Linear Knapsack-Cover

We first study the Non-Linear Knapsack-Cover problem. Recall that in this setting we consider a demand $D \in \mathbb{N}$ and a set N of n items, each with a non-decreasing function f_i. We assume that all f_i are defined over a common domain $\{0, 1, \dots, m\}$, for some $m \leq D$, and that $f_i(0) = 0$. Hence, each item $i \in N$ has m *segments* of unit length, indexed by a common set $M = \{1, \dots, m\}$, each having a unit cost $g_{ij} = f_i(j) - f_i(j-1) \geq 0$. In what follows we assume that our instance admits a feasible solution. We start by considering the list model.

It is worth mentioning that the problem described can be solved in polynomial time (respectively pseudo-polynomial) in the list (respectively oracle) model by a straightforward adaptation of the classical dynamic program for Knapsack. In the oracle model the problem is (weakly) NP-hard as it contains Knapsack-Cover as a special case. For this model the dynamic program can be turned into an FPTAS also by adapting well known rounding techniques [23]. However, these techniques alone cannot handle Non-Linear UFP-Cover.

2.1 Knapsack-Cover Inequalities for Non-Linear Costs

To write a linear relaxation of this problem, consider $a \in \{0, \dots, m\}^N$, where a_i represents that all segments $j \in \{1, \dots, a_i\}$ have been taken already for item $i \in N$ (and $a_i = 0$ represents that no segment of i is taken yet). We face the residual problem, where we must decide about segments not taken yet, and we must cover the residual demand $D(a) := \max\{D - \sum_{i \in N} a_i, 0\}$. Recall that z_{ij} is a variable that indicates whether the j-th segment of item i is taken. Since we must only cover the residual demand $D(a)$, there is an optimal solution where $z_{ij} = 0$ for $j > a_i + D(a)$; therefore, we conclude that for item i we can only take up to segment $m_i(a) := \min\{m, a_i + D(a)\}$.

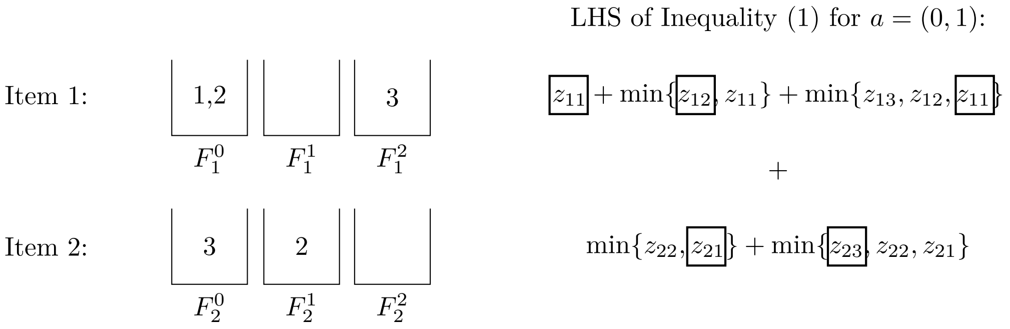

LHS of Inequality (1) for $a = (0,1)$:
Item 1:
1,2
3
F_1^0
F_1^1
F_1^2
$\boxed{z_{11}} + \min\{\boxed{z_{12}}, z_{11}\} + \min\{z_{13}, z_{12}, \boxed{z_{11}}\}$
+
Item 2:
3
2
F_2^0
F_2^1
F_2^2
$\min\{z_{22}, \boxed{z_{21}}\} + \min\{\boxed{z_{23}}, z_{22}, z_{21}\}$

Figure 1 Example of an inequality in (1) for a given pair (a, F), with $n = 2$ items, $m = 3$ segments and demand $D = 4$. We consider a vector a with $a_1 = 0, a_2 = 1$, and hence $m_1(a) = m_2(a) = 3$. The left side of the figure shows a specific index F. On the right is shown the left-hand-side (LHS) of the convex inequality in (1), where we cover a term with a square if the corresponding term is chosen for the inequality in (2) indexed by (a, F). This yields the inequality $z_{11} + z_{12} + z_{11} + z_{21} + z_{23} \geq D(a)$, where $D(a) = 3$. That is, $2z_{11} + z_{12} + z_{21} + z_{23} \geq D(a)$, implying that $\tau(1, 1, a, F) = 2, \tau(1, 2, a, F) = \tau(2, 1, a, F) = \tau(2, 3, a, F) = 1$, and all the other $\tau(i, j, a, F)$ parameters equal 0.

To obtain a linear program, we relax the condition that $z_{ij} = 1$ implies $z_{ik} = 1$ for $k < j$. To do so, note that in a feasible solution variable z_{ij} should never be larger than $\min\{z_{ij}, z_{i,j-1}, \ldots, z_{i1}\}$, and hence we can replace in our formulation the appearance of z_{ij} by this minimum. We conclude that the following is a relaxation of the Non-Linear Knapsack-Cover problem, which we call [GKC]:

$$\min \sum_{i \in N, j \in M} g_{ij} z_{ij}$$

$$\sum_{i \in N} \sum_{j = a_i + 1}^{m_i(a)} \min\{z_{ij}, z_{i,j-1}, \ldots, z_{i1}\} \geq D(a) \qquad \text{for all } a \in \{0, \ldots, m\}^N, \tag{1}$$

$$z_{ij} \geq 0 \qquad \text{for all } i \in N, j \in M.$$

We call the set of inequalities (1) the *knapsack-cover inequalities for non-linear costs*. This relaxation can be easily linearized. Indeed, if a program has a constraint of the form $\min\{x_1, x_2\} \geq b$, then we can replace it with $x_1 \geq b$ and $x_2 \geq b$. More generally, if the constraint is $\min\{x_1, x_2\} + \min\{x_3, x_4\} \geq b$, then we must consider all constraints $x_i + x_j \geq b$ for all $i \in \{1, 2\}$ and $j \in \{3, 4\}$. More generally, convex inequality in (1) can be replaced with exponentially many linear ones. The linear inequalities are constructed by replacing each summand $\min\{z_{ij}, z_{i,j-1}, \ldots, z_{i1}\}$ in (1) by one of its terms $z_{ij}, z_{i,j-1}, \ldots, z_{i1}$.

Each of the new linear inequalities will be indexed by a pair (a, F). The chosen notation will prove useful when describing and analyzing the water-filling algorithm below, as they will indicate which buckets are spilling water and which are receiving it. Consider a fixed vector a as above. The index F indicates, for each item $i \in N$ and segment $j \in M$, which term z_{ip} is selected from the set $\{z_{ij}, \ldots, z_{i1}\}$ for each i, j. A given F can be thought as an array of containers $(F_i^k)_{i \in N, k \in \{0, \ldots, m-1\}}$. Each item $i \in N$ corresponds to a row of this array, with containers (sets) $F_i^0, \ldots, F_i^{m-1}$. We distribute the set $\{a_i + 1, \ldots, m\}$ within these containers, and thus $\bigcup_{k=0}^{m-1} F_i^k = \{a_i + 1, \ldots, m\}$. Assigning an index $j \in \{a_i + 1, \ldots, m\}$ to F_i^k represents that in that inequality we select $z_{i,j-k}$ from $\{z_{ij}, \ldots, z_{i1}\}$. See Figure 1 for a concrete example of this construction. The obtained set of inequalities is given by

$$\sum_{i\in N}\sum_{k=0}^{m-1}\sum_{j\in F_i^k: j\leq m_i(a)} z_{i,j-k} \geq D(a) \qquad \text{for all } (a,F)\in\mathcal{F}, \tag{2}$$

where $\mathcal{F}$ is the set of all ordered pairs (a, F) with $a \in \{0,\ldots,m\}^n$, and F satisfies (i) $\dot\bigcup_{k=0}^{m-1} F_i^k = \{a_i + 1,\ldots,m\}$, and (ii) if $j \in F_i^k$ then $j - k \geq 1$ for all k, j. Let us consider now a given pair (a, F), an item i, and $j > a_i$. The term z_{ij} might appear several times in the respective constraint (2), depending on how many "minimums" are replaced by z_{ij}. If $k \in F_i^{k-j}$, then $\min\{z_{ik}, z_{i,k-1},\ldots,z_{i1}\}$ is replaced by z_{ij}. Moreover, recall that the residual demand $D(a)$ will never be covered by a segment z_{ij} for $j > m_i(a)$, and hence the number of times that z_{ij} appears in the left-hand-side of the inequality is

$$\tau(i,j,a,F) = |\{k\geq j : k\in F_i^{k-j}, k\leq m_i(a)\}|. \tag{3}$$

For a concrete example consider Figure 1.

With this definition, we obtain a linear relaxation that is equivalent to [GKC].

▶ **Lemma 1.** *The convex program [GKC] is equivalent to*

$$\begin{aligned}
\textit{[P-GKC]:}\min &\sum_{i\in N}\sum_{j\in M} z_{ij}g_{ij} \\
\textit{s.t.} &\sum_{i\in N}\sum_{j=1}^{m}\tau(i,j,a,F)\cdot z_{ij}\geq D(a) \qquad \textit{for all } (a,F)\in\mathcal{F}, \qquad (4)\\
& z\geq 0.
\end{aligned}$$

A routinary computation yields that the dual of this linear program, which we call [D-GKC], is the problem of maximizing $\sum_{(a,F)\in\mathcal{F}} D(a)v_{aF}$ subject to $v \geq 0$ and

$$\sum_{(a,F)\in\mathcal{F}} \tau(i,j,a,F)\cdot v_{aF}\leq g_{ij} \quad \text{for all } i\in N, j\in M. \tag{5}$$

2.2 A 2-approximate Primal-Dual Algorithm

We provide a primal-dual 2-approximation algorithm based on the LP-relaxation [P-GKC] and its dual [D-GKC]. It is worth having in mind the bucket representation of the algorithm given above in the introduction.

Algorithm description

The water-filling algorithm described in the introduction is an intuitive representation of a greedy algorithm for the dual [D-GKC]. Each bucket corresponds to a dual inequality: Each of the inequalities in the dual [D-GKC] represents a bucket, the left hand size corresponds to the amount of water in the bucket, while the right hand side is its capacity. The greedy dual algorithm raises dual variables one by one starting from a dual solution $v \equiv 0$. In each iteration of the main loop, we raise a variable v_{aF}. The index a is chosen such that a_i represents the largest value ℓ for which all buckets $B_{i1},\ldots,B_{i\ell}$ are full (or equivalently, $z_{i1} = \ldots = z_{i\ell} = 1$), for each $i \in N$. To choose F, a segment j will belong to F_i^k if and only if the water from the external source falling into bucket B_{ij} (if any) spills down to bucket $B_{i,j-k}$. Number k is chosen such that it is the smallest number for which $B_{i,j-k}$ is not full, representing the idea that the water of full buckets falls down to the previous buckets on the stairs. Also, buckets receiving water from the external source are the buckets B_{ij} with

$a_i + 1 \leq j \leq m_i(a)$. This way, $\tau(i, j, a, F)$ corresponds to the filling rates of bucket B_{ij} in the current iteration, which considers the water directly from the external source and the water spilled from higher buckets. We stop raising variable v_{aF} as soon as one dual inequality becomes tight, i.e., some bucket B_{ij} becomes full. After, we update the value of z by setting $z_{ik} = 1$ if $B_{i\ell}$ is full for all $\ell \leq k$. Notice as we do not require that $k \leq m_i(a)$, and hence the returned primal solution might not satisfy the total demand exactly. Finally, we update a and F as described above and repeat the main loop until the residual demand $D(a)$ equals 0. The pseudo-code is given in Algorithm 1. The algorithm calls a sub-routine (Line 14) that explains how to update F when a bucket gets full but cannot be taken, which is given in Algorithm 2.

Algorithm 1 Primal-Dual Water-Filling Algorithm for Non-Linear Knapsack-Cover.

```
1:  $z, v \leftarrow 0$;  % primal and dual solutions.
2:  $a \leftarrow 0$;  % $a_i$ represents the largest value for which buckets $B_{i1}, \ldots, B_{i,a_i}$ are full.
3:  $F_i^k \leftarrow \emptyset \quad \forall i \in N, k \in M$;
4:  $F_i^0 \leftarrow M$;  % $j \in F_i^k$ iff water from bucket $B_{ij}$ falls to bucket $B_{i,j-k}$.
5:  while $D(a) > 0$ do
6:      Increase $v_{aF}$ until a dual constraint indexed by $(i, j)$, for some item $i \in N$ and segment
    $j \in M$, becomes tight.   % Bucket $B_{ij}$ becomes full.
          % Update $z_{ij}$:
7:      if $j = a_i + 1$ then
8:          Let $q > a_i$ be the maximum number such that $B_{i,a_i+1}, \ldots, B_{iq}$ are full.
9:          for $\ell = j, \ldots, q$ do
10:             $z_{i\ell} \leftarrow 1$;  % Take available segments.
11:         end for
12:         $a_i \leftarrow q$;
13:     else % If we cannot raise variable $z_{ij}$:
14:         $F \leftarrow \text{Update}(F, i, j, a)$  % Call Algorithm 2 to update the sets $F$
15:     end if
16: end while
17: return $v, z$.
```

Algorithm 2 Updating Buckets Subroutine

```
input  $a, i, j, F$   % $a, F$ represent the current state of the buckets, $i, j$ represent which
is the bucket that just got full.  We require $j > a_i + 1$.
Let $p < j$ be the maximum number so that $B_{ip}$ is not full.
Let $q > j$ be the minimum number so that $B_{iq}$ is not full (and $q = m + 1$ if $B_{ij}, \ldots, B_{im}$
are all full).
for $\ell = j, j + 1, \ldots, q - 1$ do
    $F_i^{\ell-j} \leftarrow F_i^{\ell-j} \setminus \{\ell\}$ and $F_i^{\ell-p} \leftarrow F_i^{\ell-p} \cup \{\ell\}$  % The water from the external source
falling to $B_{i\ell}$, which was previously spilling to bucket $B_{ij}$, now spills to bucket $B_{ip}$.
end for
return $F$
```

Analysis

The algorithm terminates, as each iteration of the main loop (Line 5) corresponds to some bucket that becomes full, so we enter the while loop at most nm times. The main challenge is to show that the algorithm is 2-approximate.

It follows directly that the dual solution constructed is feasible through the execution of the algorithm. The primal solution is feasible as we kept iterating the main loop until the residual demand $D(a)$ is zero. As in most approximate primal-dual algorithms, the crux of the analysis is to show that an approximate form of the complementary slackness conditions are satisfied. This is summarized in the next key lemma.

▶ **Lemma 2.** *Let $\bar{v}, \bar{z}$ be the primal and dual solutions computed by the algorithm. Then,for all $(a, F) \in \mathcal{F}$ such that $\bar{v}_{aF} > 0$, it holds that*

$$\sum_{i\in N}\sum_{j=1}^{m} \tau(i,j,a,F)\bar{z}_{ij} \leq 2D(a).$$

Proof. Let $(\bar{\imath}, \bar{\jmath})$ be the indices of the dual inequality that became tight in the last iteration of the algorithm, and let $\bar{z}, \bar{v}$ be the output primal and dual solutions.

Let us fix a variable $v_{aF} > 0$ and consider the iteration of the main loop of the algorithm where we were raising that variable. For a given item $i \in N$, the expression $\sum_{j\geq 1} \tau(i,j,a,F)\bar{z}_{ij}$ represents the total number of buckets that are receiving water from the external source and whose water is spilling to some bucket that ends up in the final solution. We analyze the last item separately from the other items. Regarding item $\bar{\imath}$, notice that $\sum_{j\geq 1} \tau(\bar{\imath},j,a,F)\bar{z}_{\bar{\imath}j} \leq D(a)$, just because the buckets obtaining water from the external source are in the interval $a_{\bar{\imath}} + 1, \ldots, m_{\bar{\imath}}(a)$, which are at most $D(a)$ many.

Consider now $i \neq \bar{\imath}$ and a bucket B_{ij} that is "part"of $\tau(i, j', a, F)$ for some segment j' included in the final solution, that is, either $j' = j$ or B_{ij} is pouring into $B_{ij'}$, case in which all the buckets between j and j' are full in this iteration of the algorithm. Then, as $\bar{z}_{ij'} = 1$, by construction B_{ij} will be taken as well; that is, $\bar{z}_{ij} = 1$. Additionally, no water (either directly or indirectly) reaches a bucket B_{ik} with $k \leq a_i$, and hence $\tau(i,k,a,F) = 0$. So the quantity $\sum_{i\in N\setminus\{\bar{\imath}\}}\sum_{j\geq 1} \tau(i,j,a,F)\bar{z}_{ij}$ is upper bounded by the total number of buckets in the final solution, of items other than $\bar{\imath}$, that are above a. This number cannot be higher than $D(a)$, otherwise the algorithm would have finished before filling the last bucket $B_{\bar{\imath},\bar{\jmath}}$. ◀

The proof of the next theorem follows from the previous lemma and standard techniques from primal-dual analysis. The details are deferred to the full version.

▶ **Theorem 3.** *Algorithm 1 is a polynomial time 2-approximation algorithm for Non-Linear Knapsack-Cover in the list model.*

The extension of this theorem to the oracle model is given in Section 4.

3 Unsplittable Flow-Cover on the Line

We now show that extending the ideas of Section 2 we can also achieve a 4-approximation for the Non-Linear UFP-Cover problem. Recall that an instance of this problem is given by an interval $I = \{1, \ldots, k\}$, a set N of n items, where each item is characterized by a capacity or height u_i, a cost c_i, and a sub-interval $I_i \subseteq \{1, \ldots, k\}$. We also have a demand D_t for each $t \in I$.

In the non-linear case, we can choose the height of each item from within a set $M = \{1, \ldots, m\}$. In other words, each item consists of a list of (vertical) segments, and one must choose a prefix of them. Again, we first focus on the list model where the costs of segments $g_{i1}, g_{i2}, \ldots, g_{im}$ are given in a list. As before, we use variables $z_{ij} \in \{0,1\}$ to represent if segment j of item i is in the solution. With this the cost is $\sum_{i \in N} \sum_{j \in M} z_{ij} g_{ij}$ and the constraint that each demand must be covered is given by

$$\sum_{i \in N : t \in I_i} \sum_{j \in M} z_{ij} \geq D_t \text{ for all } t \in I. \tag{6}$$

Finally, we require that $z_{i,j+1} = 1 \Rightarrow z_{ij} = 1$ for all $i \in N, j \in \{1, \ldots, m-1\}$.

We now present the problem using the generalized knapsack-cover inequalities, and the relaxation explained in Section 2, applied to each inequality in (6) separately. For this, define $D_t(a) = \max\left(D_t - \sum_{i:t \in I_i} a_i, 0\right)$ and $\tau(i,j,a,F,t) = \{k \geq j : k \in F_i^{k-j}, k \leq m_i(a)\}$, where $m_i(a) = \min\{m, a_i + D_t(a)\}$. The relaxed primal problem, which we call [P-UFP], asks to minimize $\sum_{i,j} z_{ij} g_{ij}$ for $z \geq 0$ subject to

$$\sum_{i:t \in I_i} \sum_{j \geq 1} \tau(i,j,a,F,t) \cdot z_{ij} \geq D_t(a) \quad \text{for all } (t,a,F) \in \mathcal{H}$$

where $\mathcal{H}$ is the set of triplets (t,a,F) where $t \in I$ and $(a,F) \in \mathcal{F}$, as defined in Section 2.2. This yields a dual relaxation [D-UFP], whose objective is $\max \sum_{(a,t,F) \in \mathcal{H}} v_{atF} \cdot D_t(a)$, and we must optimize over all $v \geq 0$ satisfying

$$\sum_{(a,t,F) \in \mathcal{H} : t \in I_i} \tau(i,j,a,F,t) \cdot v_{atF} \leq g_{ij} \qquad \text{for all } i \in N, j \in M. \tag{7}$$

Algorithm Description

Our primal-dual algorithm is given in Algorithm 3. In this case our approach has two phases. During the *growing* phase (Lines 5–15), we construct a dual solution, which then directly implies a feasible primal solution. In the *pruning* phase (Lines 16–20) we remove unnecessary segments from the primal solution. As before, for each item $i \in N$ we have a stair of buckets, where each bucket B_{ij} corresponds to a given inequality in the dual, indexed by $j \in M$ and $i \in N$. In each iteration of the growing phase buckets receive water (that might fall to inferior buckets) from an external source at a rate of 1 or 0. Once we define the rates, the water dynamics work in exactly the same way as in Section 2.2: water reaching a given bucket that is full is spilled to the next bucket to the left until it reaches a bucket that is not full. The only difference is that only some of the items receive water. More precisely, in a given iteration of the growing phase, we select $t \in I$ with largest unsatisfied demand $D_t(a)$ (break ties arbitrarily). This is a greedy criterion to increase the dual objective function as fast as possible. Only buckets for items $i \in I$ such that $t \in I_i$ receive water from the external source. For such an item i, the subset of buckets receiving water from the external source are again buckets B_{ij} with $j \in \{a_i + 1, \ldots, m_i(a)\}$. The water dynamics can be emulated by raising a single dual variable at a time. Notice that the only difference to the dual in Section 2 is that when raising a given variable v_{atF}, only inequalities for items $i \in N$ where $t \in I_i$ are affected, corresponding to the fact that only buckets corresponding to such items receive water from the external source.

When one or more buckets become full, we pick one of these buckets. As before, a full bucket means that the corresponding segment (i,j) is available. We take a given segment, that is, we define a primal variable z_{ij} to 1, as soon as all preceding buckets of item i are available.

In other words, if B_{ij} becomes full for $j = a_i + 1$, then we take set $z_{ij} = \ldots = z_{iq} = 1$ where $B_{ij}, \ldots, B_{iq}$ are full but $B_{i,q+1}$ is not. This is the case even if $q > m_i(a)$. All taken buckets (or segments) are considered to be a "block", denoted by b_{ij} (where j denotes the first segment of the block). After this we update t and continue with a new iteration of the main loop of the growing phase.

Although this first phase gives a feasible primal solution, some blocks might have become redundant, that is, the solution would remain feasible without them. In the second phase we remove redundant blocks when we can. To do this, we check for each block b_{ij}, in reverse order in which they were added, whether removing the block from the primal solution makes the given primal unfeasible. If b_{ij} is redundant and it is the superior block (i.e., the block containing the highest segment that is still in the solution) of its item, we remove it; if b_{ij} is redundant but there are blocks over it in the solution when it is checked, we cannot remove it (the solution would become unfeasible). Note that doing so, all the superior blocks that are in the final solution are not redundant.

Algorithm 3 Primal-Dual Algorithm for Non-Linear UFP-Cover.

```
$z$, $v \leftarrow 0$   % primal and dual solutions.
$a \leftarrow 0$
$F_i^k \leftarrow \emptyset$ for all $i, k \geq 1$; $F_i^0 \leftarrow M$
$\mathcal{B}_i \leftarrow \emptyset$ for all $i$   % Set containing the blocks of item $i$.
while $D_t(a) > 0$ for some $t$ do
    Select $t$ that maximizes $D_t(a)$ (break ties arbitrarily).
    Increase $v_{atF}$ until a dual constraint indexed by $(i, j)$, for some item $i$ and segment $j$,
    becomes tight. Break ties in favor of a bucket with smallest index $j$.
    if $j > a_i + 1$ then
        $F \leftarrow \text{Update}(F, i, j, a)$   % Water pouring into $B_{ij}$ pours into a lower bucket.
    else
        Let $q > a_i$ be the maximum number such that $j' \notin F_i^0$ for all $j' = j+1, \ldots, q$   % we
        take all full buckets that poured into $B_{i,a_i+1}$, even the ones that are now truncated.
        Set $a_i \leftarrow q$
        Set $\mathcal{B}_i \leftarrow \mathcal{B}_i \cup b_{ij}$, with $b_{ij} = \{j, \ldots, q\}$   % $b_{ij}$ is a block that enters the primal
        solution.
    end if
end while
for all $b_{ij}$ in reversed order in which they are defined in the growing phase do
    if $b_{ij}$ can be removed from the primal solution without leaving any demand unsatisfied
    and $j \geq j'$ for all $j'$ such that $b_{ij'} \in \mathcal{B}_i$ then
            % We eliminate redundant blocks, unless they have a superior block over it
        $\mathcal{B}_i \leftarrow \mathcal{B}_i \setminus b_{i,j}$
    end if
end for
return $z$, where $z_{ij} = 1$ for $j \leq \sum_{b_{ik} \in \mathcal{B}_i} |b_{ik}|$.
```

The proof of correctness is analogous as in Section 2.2. To show the approximation factor we need the following key lemma.

▶ **Lemma 4.** *Consider the output (z, v) of the algorithm. Let (a, t, F) such that $v_{atF} > 0$. Then*

$$\sum_{i: t \in I_i} \sum_{j \geq 1} \tau(i, j, a, F, t) \cdot z_{ij} \leq 4 D_t(a).$$

Proof. Let us fix a variable $v_{atF} > 0$ raised in the main loop of the growing phase. Denote by $\mathcal{B}_i$ the set of blocks for each i at the end of the pruning phase. Out of those, consider the ones that are above a, and that contribute to fulfill the demand in t, that is, $S_{ta} = \{b_{ij} \in \cup_{i \in N} \mathcal{B}_i : t \in I_i, j \geq a_i + 1\}$. Let us denote by $\bar{b}_i$ the superior block of each item (i.e. $\bar{b}_i = b_{ij}$ with $b_{ij} \in S_{ta}$ and $j \geq j'$ for all j' such that $b_{ij'} \in \mathcal{B}_i$). For each of these superior blocks, as they were not removed, it must exist some $t_i \in I$ such that its demand would become unsatisfied when removing $\bar{b}_i$, which is of course also true if we look only at the blocks in S_{ta}, i.e.,

$$D_{t_i}(a) > \left(\sum_{b_{kj} \in S_{ta}} |b_{kj}| \right) - |\bar{b}_i|. \tag{8}$$

Inequality (8) is true because we removed the blocks in a reversed order, and the blocks that conform a were introduced before $\bar{b}_i$ in the growing phase. Let us classify the blocks in S_{ta} into two subsets, $S^L_{ta} = \{b_{i\ell} \in S_{ta} : t_i \leq t\}$ and $S^R_{ta} = \{b_{i\ell} \in S_{ta} : t_i > t\}$.

We divide the proof of Lemma 4 into two analogous inequalities. Let us show that

$$\sum_{i: t \in I_i} \sum_{j: b_{ij} \in S^R_{ta}} z_{ij} \tau(i, a, t, F) \leq 2 D_t(a). \tag{9}$$

To do this, define $t^R = \min\{t_i : \bar{b}_i \in S^R_{ta}\}$. Note that t^R is covered by every interval I_i with $\bar{b}_i$ in S^R_{ta}, as they cover t (which is at most t^R) and their t_i (which is larger or equal than t^R). Define (i_1, j_1) such that $t_{i_1} = t^R$ and $\bar{b}_{i_1} = b_{i_1, j_1}$. On the one hand, by definition of τ, we have that $z_{i_1, j_1} \tau(i_1, j_1, a, t, F) \leq \tau(i_1, j_1, a, t, F) \leq D_t(a)$. On the other hand we study

$$\sum_{i: t \in I_i} \sum_{\substack{j: B_{ij} \in S^R_{ta}, \\ (i,j) \neq (i_1, j_1)}} z_{ij} \tau(i, j, a, F, t).$$

Consider an item $i \neq i_1$, and the iteration while increasing variable v_{atF}. The summands are the number of buckets that were spilling over each of the segments above a_i that are in the final solution (because we only sum when $z_{ij} = 1$, and buckets B_{ik} for $k \leq a_i$ do not receive water). This quantity cannot be higher than the sum of the cardinality of all the blocks above a_i in the final solution (recall that when blocks are taken in Line 11, they include truncated buckets that have poured onto the taken segments). For i_1, the same argument holds, but the superior block $\bar{b}_{i_1}$ does not need to be considered because it never poured onto the inferior blocks (otherwise they would have been the same block). Thus it holds that

$$\sum_{i: t \in I_i} \sum_{\substack{j: b_{ij} \in S^R_{ta}, \\ (i,j) \neq (i_1, j_1)}} z_{ij} \tau(i, j, a, F, t) \leq \sum_{\substack{b_{ij} \in S^R_{ta}, \\ (i,j) \neq (i_1, j_1)}} |b_{ij}| \leq D_{t^R}(a) \leq D_t(a). \quad ◀$$

With this lemma we can show the following main results. The first proof follows from standard LP techniques which are completely analogous to the proof of Theorem 3. The extension to the oracle model is given in Section 4.

▶ **Theorem 5.** *Algorithm 3 is a polynomial 4-approximation algorithm for Non-Linear UFP-Cover in the list model.*

4 Arbitrary non-decreasing functions

We now show how to adapt our algorithms in Sections 2 and 3 for the more general oracle model. Here we assume that each function $f_i : \{0, \dots, m\} \to \mathbb{Q}_{\geq 0} \cup \{\infty\}$, where $m \leq D$ is not necessarily polynomially bounded. We assume that each function f_i is given by an oracle, such that a polynomial number of bits is enough to describe all values $f_i(x)$. Using standard techniques, we first approximate each function f_i by a piece-wise constant function with a polynomial number of steps. After, we discuss how to emulate Algorithm 1 and 3 in polynomial time for such functions.

First of all, by scaling we can assume, without loss of generality, that $f_i(x) \in \mathbb{N} \cup \{\infty\}$.

▶ **Lemma 6.** *Consider $\varepsilon > 0$ and let $f : \{1, \dots, m\} \to \mathbb{N} \cup \{\infty\}$ be a non-decreasing function. Let $f_{\max}$ be the maximum finite value of $f(x)$. There exists a non-decreasing piece-wise constant function $\tilde{f}$ such that*

$$f(x) \leq \tilde{f}(x) \leq (1+\varepsilon) f(x) \qquad \textit{for all } x \in \{1, \dots, m\},$$

where $(\tilde{f}(x))_{x \in \mathbb{N}}$ takes at most $\lceil \log_{1+\varepsilon} f_{\max} \rceil + 1$ many different values. The function $\tilde{f}$ can be computed in polynomial time.

Proof. To prove the lemma we can assume that $f(x) > 0$, as the values $f(x) = 0$ just corresponds to separate piece in $\tilde{f}$. For all other $x \in \{1, \dots, m\}$, we can simply set $\tilde{f}(x) = (1+\varepsilon)^{\lceil \log_{1+\varepsilon} f(x) \rceil}$. The constant-wise pieces (intervals) of $\tilde{f}$ can be easily computed in polynomial time with a binary search approach. ◀

We now sketch how to adapt the algorithms of Sections 2 and 3 for this scenario. Let $\tilde{f}_i$ be the obtained function after applying the last lemma to f_i. We partition the set $\{0, 1, \dots, m\}$ in intervals $J_{i1}, J_{i2}, \dots, J_{im_i}$ correspondent to the piece-wise constant pieces of $\tilde{f}_i$, that is $\tilde{f}_i(x) = \tilde{f}_i(x')$ if $x, x' \in J_{ik}$. We denote by $u_{ik} \in \mathbb{N}$ the cardinality of interval $J_{ik} \subseteq \mathbb{N}$. To adapt the algorithms, note that as they originally deal with unitary segments, a piecewise constant function can be replaced (preserving the same costs for any solution) by a piecewise constant function with a pseudopolynomial number of unitary segments. More precisely, if $J_{ik} = \{\ell, \ell+1, \dots, u\}$, then $g_{i\ell} = \tilde{f}_i(\ell) - \tilde{f}_i(\ell - 1)$, and $g_{ir} = 0$ for all $r \in \{\ell+1, \dots, u\}$. Applying our algorithms to this instance would imply a pseudopolynomial running time. However, as all but m_i many buckets for item i has zero capacity g_{ij}, we can handle all of them simultaneously to make our algorithms run in polynomial time.

To do this, we can process all segments in J_{ik} in a single step: when the algorithm begins, all their respective buckets but the first one get instantaneously full, so the first one will receive water at a rate equal to the length of the constant interval (equivalently, the interval J_{ij} is represented by a bucket of height g_{ij} that gets filled at a rate u_{ij}). Any time a bucket pours onto some inferior bucket, its rate also increases by the length of the interval corresponding to the pouring bucket. Truncations, given by the fact that in a given iteration only buckets B_{ij} for $j \in \{a_i + 1, \dots, m_i(a)\}$ get water from the external source for each item i, make these rates diminish accordingly. With these rules, the algorithms can be easily adapted to run in polynomial time implying the following theorems.

▶ **Theorem 7.** *There exists a polynomial time $(2+\varepsilon)$-approximation for the Knapsack-Cover Problem with Non-Linear Costs and arbitrary non-decreasing functions.*

▶ **Theorem 8.** *There exists a $(4+\varepsilon)$-approximation for the Unsplittable Flow-Cover on the Line Problem with Non-Linear Costs and arbitrary non-decreasing functions.*

References

1 H.-C. An, M. Singh, and O. Svensson. LP-based algorithms for capacitated facility location. *SIAM Journal on Computing*, 46(1):272–306, 2017.

2 N. Bansal and J. Batra. Geometry of scheduling on multiple machines. *arXiv:1907.05473 [cs]*, 2019. arXiv:1907.05473.

3 N. Bansal and K. Pruhs. The geometry of scheduling. *SIAM Journal on Computing*, 43(5):1684–1698, 2014.

4 A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. Naor, and B. Schieber. A unified approach to approximating resource allocation and scheduling. *Journal of the ACM*, 48(5):1069–1090, 2001.

5 R. Bar-Yehuda and S. Even. A linear-time approximation algorithm for the weighted vertex cover problem. *Journal of Algorithms*, 2(2):198–203, 1981.

6 R. Bar-Yehuda and D. Rawitz. On the equivalence between the primal-dual schema and the local ratio technique. *SIAM Journal on Discrete Mathematics*, 19(3):762–797, 2005.

7 A. Bazzi, S. Fiorini, S. Huang, and O. Svensson. Small extended formulation for knapsack cover inequalities from monotone circuits. *Theory of Computing*, 14(1):1–29, 2018.

8 Tim Carnes and David B. Shmoys. Primal-dual schema for capacitated covering problems. *Mathematical Programming*, 153(2):289–308, 2015.

9 R. D. Carr, L. K. Fleischer, V. J. Leung, and C. A. Phillips. Strengthening integrality gaps for capacitated network design and covering problems. In *Proceedings of the 11th annual ACM-SIAM symposium on Discrete algorithms (SODA 2000)*, pages 106–115, 2000.

10 M. Cheung, J. Mestre, D. B. Shmoys, and J. Verschae. A primal-dual approximation algorithm for min-sum single-machine scheduling problems. *SIAM Journal on Computing*, 31(2):825–838, 2017.

11 V. Chvátal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.

12 N. Coulombel and N. Monchambert. Diseconomies of scale and subsidies in urban public transportation. *HAL:02373768*, 2019.

13 A. Fielbaum, I. Morales, and J. Verschae. A water-filling primal-dual algorithm for approximating non-linear covering problems. *arXiv:1912.12151 [cs.DS]*, 2019. arXiv:1912.12151.

14 Andrés Fielbaum, Sergio Jara-Diaz, and Antonio Gschwender. Beyond the Mohring effect: Scale economies induced by transit lines structures design. *Economics of Transportation*, 22:100–163, 2020.

15 M. Goemans and J. Kleinberg. An improved approximation ratio for the minimum latency problem. *Mathematical Programming*, 82(1):111–124, 1998.

16 W. Höhn, J. Mestre, and A. Wiese. How unsplittable-flow-covering helps scheduling with job-dependent cost functions. In *Automata, Languages, and Programming (ICALP 2014)*. Springer, 2014.

17 M.-Y. Kao, J. H. Reif, and S. R. Tate. Searching in an unknown environment: An optimal randomized algorithm for the cow-path problem. *Information and Computation*, 131(1):63–79, 1996.

18 Retsef Levi, Andrea Lodi, and Maxim Sviridenko. Approximation algorithms for the capacitated multi-item lot-sizing problem via flow-cover inequalities. *Mathematics of Operations Research*, 33(2):461–474, 2008.

19 S. Li. Constant approximation algorithm for non-uniform capacitated multi-item lot-sizing via strong covering inequalities. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2017)*, pages 2311–2325, 2017.

20 S. Thomas McCormick, Britta Peis, J. Verschae, and A. Wierz. Primal-dual algorithms for precedence constrained covering problems. *Algorithmica*, 78(3):771–787, 2017.

21 Herbert Mohring. Optimisation and scale economies in urban bus transport. *American Economic Review*, 62(4):591–604, 1972.

22 B. Moseley. Scheduling to approximate minimization objectives on identical machines. In *Automata, Languages, and Programming (ICALP 2019)*, pages 86:1–86:14, 2019.

23 D. P. Williamson and D. B. Shmoys. *The design of approximation algorithms*. Cambridge University Press, 1 edition, 2011.

24 J. Zhu. *Optimization of power system operation*, volume 47. John Wiley & Sons, 2015.

Scattering and Sparse Partitions, and Their Applications

Arnold Filtser
Columbia University, United States, New York, NY, USA
http://www.cs.columbia.edu/~arnoldf/
arnold273@gmail.com

Abstract

A partition $\mathcal{P}$ of a weighted graph G is (σ,τ,Δ)-sparse if every cluster has diameter at most Δ, and every ball of radius Δ/σ intersects at most τ clusters. Similarly, $\mathcal{P}$ is (σ,τ,Δ)-scattering if instead for balls we require that every shortest path of length at most Δ/σ intersects at most τ clusters. Given a graph G that admits a (σ,τ,Δ)-sparse partition for all $\Delta>0$, Jia et al. [STOC05] constructed a solution for the Universal Steiner Tree problem (and also Universal TSP) with stretch $O(\tau\sigma^2\log_\tau n)$. Given a graph G that admits a (σ,τ,Δ)-scattering partition for all $\Delta>0$, we construct a solution for the Steiner Point Removal problem with stretch $O(\tau^3\sigma^3)$. We then construct sparse and scattering partitions for various different graph families, receiving many new results for the Universal Steiner Tree and Steiner Point Removal problems.

2012 ACM Subject Classification Theory of computation

Keywords and phrases Scattering partitions, sparse partitions, sparse covers, Steiner point removal, Universal Steiner tree, Universal TSP

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.47

Category Track A: Algorithms, Complexity and Games

Related Version A full version of the paper is available at https://arxiv.org/abs/2001.04447. The reader is encouraged to read the full version of the paper [31].

Funding The research was supported by the Simons Foundation.

Acknowledgements The author would like to thank Alexandr Andoni, Robert Krauthgamer, Jason Li, Ofer Neiman, Anastasios Sidiropoulos, and Ohad Trabelsi for helpful discussions.

1 Introduction

Graph and metric clustering are widely used for various algorithmic applications (e.g., divide and conquer). Such partitions come in a variety of forms, satisfying different requirements. This paper is dedicated to the study of bounded diameter partitions, where small neighborhoods are guaranteed to intersects only a bounded number of clusters.

The first problem we study is the *Steiner Point Removal* (SPR) problem. Here we are given an undirected weighted graph $G=(V,E,w)$ and a subset of terminals $K\subseteq V$ of size k (the non-terminal vertices are called Steiner vertices). The goal is to construct a new weighted graph $M=(K,E',w')$, with the terminals as its vertex set, such that: (1) M is a graph minor of G, and (2) the distance between every pair of terminals t,t' in M is distorted by at most a multiplicative factor of α, formally, $\forall t,t'\in K,\quad d_G(t,t')\le d_M(t,t')\le\alpha\cdot d_G(t,t')$. Property (1) expresses preservation of the topological structure of the original graph. For example if G was planar, so will M be. Whereas property (2) expresses preservation of the geometric structure of the original graph, that is, distances between terminals. The question is thus: given a graph family $\mathcal{F}$, what is the minimal α such that every graph in $\mathcal{F}$ with a terminal set of size k will admit a solution to the SPR problem with distortion α.

Consider a weighted graph $G = (V, E, w)$ with a shortest path metric d_G. The *weak* diameter of a cluster $C \subseteq V$ is the maximal distance between a pair of vertices in the cluster w.r.t. d_G (i.e., $\max_{u,v\in C} d_G(u, v)$). The *strong* diameter is the maximal distance w.r.t. the shortest path metric in the induced graph $G[C]$ (i.e., $\max_{u,v\in C} d_{G[C]}(u, v)$). A partition $\mathcal{P}$ of G has weak (resp. strong) diameter Δ if every cluster $C \in \mathcal{P}$ has weak (resp. strong) diameter at most Δ. Partition $\mathcal{P}$ is *connected*, if the graph induced by every cluster $C \in \mathcal{P}$ is connected. Given a shortest path $\mathcal{I} = \{v_0, v_1, \dots, v_s\}$, denote by $Z_{\mathcal{I}}(\mathcal{P}) = \sum_{C\in\mathcal{P}} \mathbb{1}_{C\cap\mathcal{I}\neq\emptyset}$ the number of clusters in $\mathcal{P}$ intersecting $\mathcal{I}$. If $Z_{\mathcal{I}}(\mathcal{P}) \le \tau$, we say that $\mathcal{I}$ is τ-*scattered* by $\mathcal{P}$.

▶ **Definition 1** (Scattering Partition). *Given a weighted graph $G = (V, E, w)$, we say that a partition $\mathcal{P}$ is (σ, τ, Δ)-scattering if the following conditions hold:*

- *$\mathcal{P}$ is connected and has weak diameter Δ.*
- *Every shortest path $\mathcal{I}$ of length at most Δ/σ is τ-scattered by $\mathcal{P}$, i.e., $Z_{\mathcal{I}}(\mathcal{P}) \le \tau$.*

*We say that a graph G is (σ, τ)-*scatterable *if for every parameter Δ, G admits an (σ, τ, Δ)-scattering partition that can be computed efficiently.*

The main contribution of this paper is the finding that scattering partitions imply solutions for the SPR problem. The proof appears in Section 3.[1]

▶ **Theorem 2** (Scattering Partitions imply SPR). *Let $G = (V, E, w)$ be a weighted graph such that for every subset $A \subseteq V$, $G[A]$ is $(1, \tau)$-scatterable. Let $K \subseteq V$ be some subset of terminals. Then there is a solution to the SPR problem with distortion $O(\tau^3)$ that can be computed efficiently.*

Jia, Lin, Noubir, Rajaraman, and Sundaram [45] [2] defined the notion of *sparse partitions*, which is closely related to scattering partitions. Let $\mathcal{P}$ be a partition. Given a ball $B = B_G(x, r)$, denote by $Z_B(\mathcal{P}) = \sum_{C\in\mathcal{P}} \mathbb{1}_{C\cap B\neq\emptyset}$ the number of clusters in $\mathcal{P}$ intersecting B.

▶ **Definition 3** (Strong/Weak Sparse Partition). *Given a weighted graph $G = (V, E, w)$, we say that a partition $\mathcal{P}$ is (σ, τ, Δ)-weak (resp. strong) sparse partition if the following conditions hold:*

- *$\mathcal{P}$ has weak (resp. strong) diameterΔ.*
- *Every ball $B = B_G(v, r)$ of radius $r \le \Delta/\sigma$ intersects at most τ clusters, i.e., $Z_B(\mathcal{P}) \le \tau$.*

*We say that a graph G admits an (σ, τ)-*weak *(resp. strong) sparse partition scheme if for every parameter Δ, G admits an efficiently computable (σ, τ, Δ)-weak (resp. strong) sparse partition.*

Jia et al. [45] found a connection between sparse partitions to the *Universal Steiner Tree Problem* (UST).[3] Consider a complete weighted graph $G = (V, E, w)$ (or a metric space (X, d)) where there is a special server vertex $\mathrm{rt} \in V$, which is frequently required to multicast messages to different subsets of clients $S \subseteq V$. The cost of a multicast is the total weight of all edges used for the communication. Given a subset S, the optimal solution is to use the minimal Steiner tree spanning $S \cup \{\mathrm{rt}\}$. In order to implement an infrastructure for multicasting, or in order to make routing decisions much faster (and not compute it from scratch once S is given), a better solution will be to compute a *Universal Steiner Tree* (UST). A UST is a tree T over V, such that for every subset S, the message will be sent using the

[1] In Observation 2 we argue that (σ, τ, Δ)-scattering partition is also $(1, \tau\sigma, \Delta)$-scattering.

[2] Awerbuch and Peleg [8] were the first to study sparse covers (see Definition 5). Their notion of sparse partition is somewhat different from the one used here (introduced by [45]).

[3] A closely related problem is the *Universal Traveling Salesman Problem* (UTSP), see Section 1.4.

sub-tree $T(S)$ spanning $S \cup \{\mathrm{rt}\}$. The stretch of T is the maximum ratio among all subsets $S \subseteq X$ between the weight of $T(S)$ and the weight of the minimal Steiner tree spanning $S \cup \{\mathrm{rt}\}$, $\max_{S\subseteq X} \frac{w(T(S))}{\mathrm{Opt}(S\cup\{\mathrm{rt}\})}$.

Jia et al. [45] proved that given a sparse partition scheme, one can efficiently construct a UST with low stretch (the same statement holds w.r.t. UTSP as well).

▶ **Theorem 4** (Sparse Partitions imply UST, [45]). *Suppose that an n-vertex graph G admits an (σ, τ)-weak sparse partition scheme, then there is a polynomial time algorithm that given a root $\mathrm{rt} \in V$ computes a UST with stretch $O(\tau\sigma^2 \log_\tau n)$.*

Jia et al. [45] constructed $(O(\log n), O(\log n))$-weak sparse partition scheme for general graphs, receiving a solution with stretch polylog(n) for the UST problem. In some instances the communication is allowed to flow only in certain routes. It is therefore natural to consider the case where $G = (V, E, w)$ is not a complete graph, and the UST is required to be a subgraph of G. Busch, Jaikumar, Radhakrishnan, Rajaraman, and Srivathsan [12] proved a theorem in the spirit of Theorem 4, stating that given a (σ, τ, γ)-*hierarchical strong sparse partition*, one can efficiently construct a subgraph UST with stretch $O(\sigma^2\tau^2\gamma \log n)$. A (σ, τ, γ)-hierarchical strong sparse partition is a laminar collection of partitions $\{\mathcal{P}_i\}_{i\ge 0}$ such that $\mathcal{P}_i$ is (σ, τ, γ^i)-strong sparse partition which is a refinement of $\mathcal{P}_{i+1}$.[4] Busch et al. constructed a $\left(2^{O(\sqrt{\log n})}, 2^{O(\sqrt{\log n})}, 2^{O(\sqrt{\log n})}\right)$-hierarchical strong sparse partition, obtaining a $2^{O(\sqrt{\log n})}$ stretch algorithm for the subgraph UST problem. We tend to believe that poly-logarithmic stretch should be possible. It is therefore interesting to construct strong sparse partitions, as it eventually may lead to hierarchical ones.

A notion which is closely related to sparse partitions is *sparse covers*.

▶ **Definition 5** (Strong/Weak Sparse cover). *Given a weighted graph $G = (V, E, w)$, a (σ, τ, Δ)-weak (resp. strong) sparse cover is a set of clusters $\mathcal{C} \subset 2^V$, where all the clusters have weak (resp. strong) diameter at most Δ, and the following conditions hold:*

- *Cover: $\forall u \in V$, exists $C \in \mathcal{C}$ such that $B_G(u, \frac{\Delta}{\sigma}) \subseteq C$.*
- *Sparsity: every vertex $u \in V$ belongs to at most $|\{C \in \mathcal{C} \mid u \in C\}| \le \tau$ clusters.*

*We say that a graph G admits an (σ, τ)-*weak *(resp. strong) sparse cover scheme if for every parameter Δ, G admits an (σ, τ, Δ)-weak (resp. strong) sparse cover that can be computed efficiently.*

It was (implicitly) proven in [45] that given (σ, τ, Δ)-weak sparse cover $\mathcal{C}$, one can construct an (σ, τ, Δ)-weak sparse partition. In fact, most previous constructions of weak sparse partitions were based on sparse covers.

1.1 Previous results

SPR. Given an n-point tree, Gupta [38] provided an upper bound of 8 for the SPR problem (on trees). This result were recently reproved by the author, Krauthgamer, and Trabelsi [33] using the `Relaxed-Voronoi` framework. Chan, Xia, Konjevod, and Richa [14] provided a lower bound of 8 for trees. This is the best known lower bound for the general SPR problem. Basu and Gupta [11] provided an $O(1)$ upper bound for the family of outerplanar graphs.[5] For general n-vertex graphs with k terminals the author [27, 29] recently proved an $O(\log k)$ upper bound for the SPR problem using the `Relaxed-Voronoi` framework, improving upon

[4] We assume here w.l.o.g. that the minimal distance in G is 1.

[5] Actually the manuscript [11] was never published, and thus did not go through a peer review process.

previous works by Kamma, Krauthgamer, and Nguyen [46] ($O(\log^5 k)$), and Cheung [16] ($O(\log^2 k)$) (which were based on the `Ball-Growing` algorithm). Interestingly, there are no results on any other restricted graph family, although several attempts have been made (see [25, 50, 17]).

UST. Given an n-point metric space and root rt, Gupta, Hajiaghayi and Räcke [39] constructed a UST with stretch $O(\log^2 n)$, improving upon a previous $O(\log^4 n/\log\log n)$ result by [45]. [45] is based on sparse partitions, while [39] is based on *tree covers*. Jia et al. [45] proved a lower bound of $\Omega(\log n)$ to the UST problem, based on a lower bound to the online Steiner tree problem by Alon and Azar [6]. Using the same argument, they [45] proved an $\Omega(\frac{\log n}{\log\log n})$ lower bound for the case where the space is the $n \times n$ grid (using [43]). Given a space with doubling dimension ddim, [6] Jia et al. [45] provided a solution with stretch $2^{O(\mathrm{ddim})} \cdot \log n$, using sparse partitions. Given an n vertex planar graph, Busch, LaFortune, and Tirthapura [13] proved an $O(\log n)$ upper bound (improving over Hajiaghayi, Kleinberg, and Leighton [42]). More generally, for graphs G excluding a fixed minor, both Hajiaghayi et al. [42] (implicitly) and Busch et al. [13] (explicitly) provided a solution with stretch $O(\log^2 n)$. Both constructions used sparse covers. Finally, Busch et al. [12] constructed a subgraph UST with stretch polylog(n) for graphs excluding a fixed minor (using hierarchical strong sparse partitions).

Scattering Partitions. As we are the first to define scattering partitions there is not much previous work. Nonetheless, Kamma et al. [46] implicitly proved that general n-vertex graphs are $(O(\log n), O(\log n))$-scatterable.[7]

Sparse Covers and Partitions. Awerbuch and Peleg [8] introduced the notion of sparse covers and constructed $(O(\log n), O(\log n))$-strong sparse cover scheme for n-vertex weighted graphs.[8] Jia et al. [45] induced an $(O(\log n), O(\log n))$-weak sparse partition scheme. Hajiaghayi et al. [42] constructed an $(O(1), O(\log n))$-weak sparse cover scheme for n-vertex planar graph, concluding an $(O(1), O(\log n))$-weak sparse partition scheme. Their construction is based on the [48] clustering algorithm. Abraham, Gavoille, Malkhi, and Wieder [5] constructed $(O(r^2), 2^{O(r)} \cdot r!)$-strong sparse cover scheme for K_r-free graphs. Busch et al. [13] constructed a $(48, 18)$-strong sparse cover scheme for planar graphs [9] and $(8, O(\log n))$-strong sparse cover scheme for graphs excluding a fixed minor, concluding a $(48, 18)$ and $(8, O(\log n))$-weak sparse partition schemes for these families (respectively). For graphs with doubling dimension ddim, Jia et al. [45] constructed an $(1, 8^{\mathrm{ddim}})$-weak sparse scheme. Abraham et al. [3] constructed a $(2, 4^{\mathrm{ddim}})$-strong sparse cover scheme. In a companion paper, the author [28] constructed an $(O(\mathrm{ddim}), O(\mathrm{ddim} \cdot \log \mathrm{ddim}))$-strong sparse cover scheme.[10] Busch et al. [12] constructed $\big(O(\log^4 n), O(\log^3 n), O(\log^4 n)\big)$-hierarchical strong sparse partition for graphs excluding a fixed minor.

[6] A metric space (X, d) has doubling dimension ddim if every ball of radius $2r$ can be covered by 2^{ddim} balls of radius r. The doubling dimension of a graph is the doubling dimension of its induced shortest path metric.

[7] This follows from Theorem 1.6 in [46] by choosing parameters $t = \beta = O(\log n)$ and using union bounds over all n^2 shortest paths. Note that they assume that for every pair of vertices there is a unique shortest path.

[8] More generally, for $k \in \mathbb{N}$, [8] constructed a $(2k-1, 2k \cdot n^{\frac{1}{k}})$-strong sparse cover scheme.

[9] Busch et al. argued that they constructed $(24, 18)$-strong sparse covering scheme. However they measured radius rather than diameter.

[10] More generally, for a parameter $t = \Omega(1)$, [28] constructed $\big(O(t), O(2^{\mathrm{ddim}/t} \cdot \mathrm{ddim} \cdot \log t)\big)$-sparse cover scheme.

1.2 Our Contribution

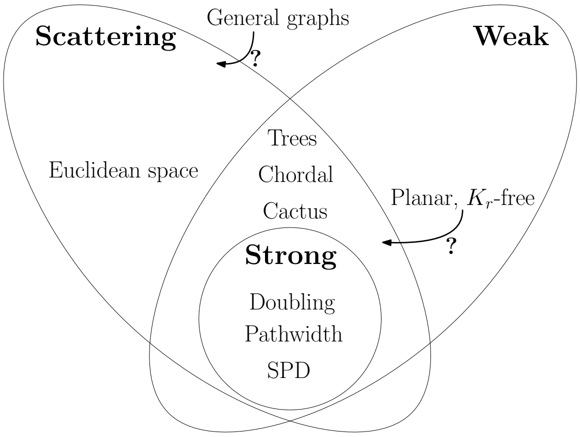


Figure 1 Classification of various graph families according to the possibility of construction different partitions. Graphs with bounded doubling dimension or SPD [14] (pathwidth) admit strong sparse partitions with parameters depending only on the dimension/SPDdepth. Trees, Chordal and Cactus graphs admit both $(O(1), O(1))$-weak sparse and scattering partitions, while similar strong partitions are impossible. $\mathbb{R}^d$ with norm 2 admit $(1, 2d)$ scattering partition while weak sparse partition with constant padding will have an exponential number of intersections. Planar graphs admit $(O(1), O(1))$-weak sparse partitions, while it is an open question whether similar scattering partitions exist. Finally, while sparse partitions for general graphs are well understood, we lack a lower bound for scattering partitions.

Formal statements, and proofs of all our partitions are differed to the full version [31]. The main contribution of this paper is the definition of scattering partition and the finding that good scattering partitions imply low distortion solutions for the SPR problem (Theorem 2). We construct various scattering and sparse partition schemes for many different graph families, and systematically classify them according to the partition types they admit. In addition, we provide several lower bounds. The specific partitions and lower bounds are described below. Our findings are summarized in Table 1, while the resulting classification is illustrated in Figure 1.

Recall that [45] (implicitly) showed that sparse covers imply weak sparse partitions. We show that the opposite direction is also true. That is, given a (σ, τ, Δ)-weak sparse partition, one can construct an $(\sigma + 2, \tau, (1 + \frac{2}{\sigma})\Delta)$-weak sparse cover. Interestingly, in addition we show that strong sparse partitions imply strong sparse covers, while the opposite is not true. Specifically there are graph families that admit $(O(1), O(1))$-strong sparse cover schemes, while there are no constants σ, τ, such that they admit (σ, τ)-strong sparse partitions. Description of our findings on the connection between sparse partitions and sparse covers, and a classification of various graph families are differed to the full version [31].

The scattering partitions we construct imply new solutions for the SPR problem previously unknown. Specifically, for every graph with pathwidth ρ we provide a solution to the SPR problem with distortion poly(ρ), independent of the number of terminals. After trees [38] and outerplanar graphs [11] [5], this is the first graph family to have solution for the SPR problem independent from the number of terminals (although attempts were made). Furthermore, we obtain solution with constant distortion for Chordal and Cactus graphs.[11]

[11] Note that the family of cactus graph is contained in the family of outerplanar graph. Basu and Gupta [11] solved the SPR problem directly on outerplanar graphs with constant distortion. However, this manuscript was never published. See also 5.

Table 1 Summery of the various new/old, weak/strong scattering/sparse partitions.
Table footnotes: ♣ More generally, there is a partition $\mathcal{P}$ s.t. every ball of radius $\frac{\Delta}{8\alpha}$ intersects at most $\tilde{O}(n^{1/\alpha})$ clusters, for all $\alpha > 1$ simultaneously. ■ More generally, it must hold that $\tau \geq n^{\Omega(1/\sigma)}$. ♠ More generally, there is a partition $\mathcal{P}$ s.t. every ball of radius $\Omega(\frac{\Delta}{\alpha})$ intersects at most $O(2^{\mathrm{ddim}/\alpha})$ clusters, for all $\alpha > 1$ simultaneously. ♦ More generally, it must hold that $\tau > (1 + \frac{1}{2\sigma})^d$. ★ Note that this lower bound holds chordal/cactus/planar/K_r-free graphs. More generally, it must hold that $\tau \geq \Omega(n^{2/\sigma+1})$.

Family	Partition type	Padding (σ)	#inter. (τ)	Ref.
General n-vertex Graphs	Weak	$O(\log n)$	$O(\log n)$	[45]
	Scattering	$O(\log n)$	$O(\log n)$	[46]
	Strong	$O(\log n)$	$O(\log n)$	This paper ♣
	Weak L.B.	$\Omega(\log n/\log\log n)$	$O(\log n)$	This paper ■
ddim **doubling dimension**	Weak	1	8^{ddim}	[45]
	Strong	$O(\mathrm{ddim})$	$\tilde{O}(\mathrm{ddim})$	This paper ♠
Euclidean space $(\mathbb{R}^d, \|\cdot\|_2)$	Scattering	1	$2d$	This paper
	Weak L.B.	$O(1)$ $\Omega(d/\log d)$	$2^{\Omega(d)}$ $\mathrm{poly}(d)$	This paper ♦
Trees	Scattering	2	3	This paper
	Weak	4	3	This paper
	Strong L.B.	$\log n/\log\log n$ $\sqrt{\log n}$	$\log n$ $2^{\sqrt{\log n}}$	This paper ★
Pathwidth ρ **(SPDdepth** [14]**)**	Strong	$O(\rho)$	$O(\rho^2)$	This paper
	Weak	8	5ρ	This paper
Chordal	Scattering	2	3	This paper
	Weak	24	3	This paper
K_r **free**	Weak	$O(r^2)$	2^r	This paper
Cactus	Scattering	4	5	This paper

The weak sparse partitions we construct imply improved solutions for the UST (and UTSP) problem. Specifically, we conclude that for graphs with doubling dimension ddim a UST (and UTSP) with stretch $\mathrm{poly}(\mathrm{ddim}) \cdot \log n$ can be efficiently computed, providing an exponential improvement in the dependence on ddim compared with the previous state of the art [45] of $2^{O(\mathrm{ddim})} \cdot \log n$. For K_r-minor free graphs we conclude that an UST (or UTSP) with stretch $2^{O(r)} \cdot \log n$ can be efficiently computed, providing a quadratic improvement in the dependence on n compared with the previous state of the art [39] of $O(\log^2 n)$. [12] Finally, for pathwidth ρ graphs (or more generally, graph with SPDdepth ρ) we can compute a UST (or UTSP) with stretch $O(\rho \cdot \log n)$, improving over previous solutions that were exponential in ρ (based on the fact that pathwidth ρ graphs are $K_{\rho+2}$-minor free).

Before we proceed to describe our partitions we make two observations.

[12] This result is a mere corollary obtained by assembling previously existing parts together. Mysteriously, although UTSP on minor free graphs was studied before [42, 13], this corollary was never drawn.

▶ **Observation 1.** *Every (σ, τ, Δ)-strong sparse partition is also scattering partition and weak sparse partition with the same parameters.*

▶ **Observation 2.** *Every (σ, τ, Δ)-scattering partition is also $(1, \sigma\tau, \Delta)$-scattering partition.*

Observation 1 follows as every path of weight $\sigma\Delta$ is contained in a ball of radius $\sigma\Delta$. Observation 2 follows as every shortest path of length $\le \Delta$ can be assembled as a concatenation of at most σ shortest paths of length $\le \frac{\Delta}{\sigma}$.

Next we survey the partitions for various graph families. Formal statements and proofs are differed to the full version [31].

General Graphs. Given an n-vertex general graph and parameter $\Delta > 0$ we construct a single partition $\mathcal{P}$ which is simultaneously $\big(8k, O(n^{1/k} \cdot \log n), \Delta\big)$-strong sparse partition for all parameters $k \ge 1$. Thus we generalize the result of [45] and obtained a strong diameter guarantee. This partition implies that general graphs are $(O(\log n), O(\log n))$-scatterable (reproving [46] via an easier proof), inducing a solution for the SPR problem with stretch $\text{polylog}(|K|)$. While quantitatively better solutions are known, this one is arguably the simplest, and induced by a general framework. Further, we provide a lower bound, showing that if all n-vertex graphs admit (σ, τ)-weak sparse partition scheme, then $\tau \ge n^{\Omega(\frac{1}{\sigma})}$. In particular there is no sparse partition scheme with parameters smaller than $(\Omega(\log n/\log\log n), \Omega(\log n))$. This implies that both our results and [45] are tight up to second order terms. Although we do not provide any lower bound for scattering partitions, we present some evidence that general graphs are not $(O(1), O(1))$-scatterable. Specifically, we define a stronger notion of partitions called *super-scattering* and show that general graphs are not $(1, \Omega(\log n))$-super scatterable.

Trees. Trees are the most basic of the restricted graph families. Weak sparse partitions for trees follows from the existence of sparse covers. Nevertheless, in order to improve parameters and understanding we construct $(4, 3)$-weak sparse partition scheme for trees. Further, we prove that trees are $(2, 3)$-scatterable. Finally, we show that there are no good strong sparse partition for trees. Specifically, we prove that if all n-vertex trees admit (σ, τ)-strong sparse partition scheme, then $\tau \ge \frac{1}{3} \cdot n^{\frac{2}{\sigma+1}}$. This implies that for strong sparse partitions, trees are essentially as bad as general graphs.

Doubling Dimension. We prove that for every graph with doubling dimension ddim and parameter $\Delta > 0$, there is a partition $\mathcal{P}$ which is simultaneously $\big(58\alpha, 2^{\text{ddim}/\alpha} \cdot \tilde{O}(\text{ddim}), \Delta\big)$-strong sparse partition for all parameters $\alpha \ge 1$. Note that this implies an $\big(O(\text{ddim}), \tilde{O}(\text{ddim})\big)$-strong sparse partition scheme.

Euclidean Space. We prove that the d-dimensional Euclidean space $(\mathbb{R}^d, \|\cdot\|_2)$ is $(1, 2d)$-scatterable [13], while for every (σ, τ)-weak sparse partition scheme it holds that $\tau > (1 + \frac{1}{2\sigma})^d$. In particular, if σ is at most a constant, then τ must be exponential. This provides an interesting example of a family where scattering partitions have considerably better parameters than sparse partitions.

[13] In Euclidean space, we say that a partition is (σ, τ, Δ)-scattering if every interval of length Δ/σ intersects at most τ clusters.

SPDdepth.[14] We prove that every graph with SPDdepth ρ (in particular graph with pathwidth ρ) admit $\big(O(\rho), O(\rho^2)\big)$-strong sparse partition scheme. Further, we prove that such graphs admit $(8, 5\rho)$-weak sparse partition scheme.

Chordal Graphs. We prove that every Chordal graph is $(2, 3)$-scatterable.

Cactus Graphs. We prove that every Cactus graph is $(4, 5)$-scatterable.

1.3 Technical Ideas

Scattering Partition Imply SPR. Similarly to previous works on the SPR problem, we construct a minor via a terminal partition. That is, a partition of V into k connected clusters, where each cluster contains a single terminal. The minor is then induced by contracting all the internal edges. Intuitively, to obtain small distortion, one needs to ensure that every Steiner vertex is clustered into a terminal not much further than its closest terminal, and that every shortest path between a pair of terminals intersects only a small number of clusters. However, the local partitioning of each area in the graph requires a different scale, according to the distance to the closest terminal. Our approach is similar in spirit to the algorithm of Englert et al. [25], who constructed a minor with small expected distortion[15] using stochastic decomposition for all possible distance scales. We however, work in the more restrictive regime of worst case distortion guarantee. Glossing over many details, we create different scattering partitions to different areas, where vertices at distance $\approx \Delta$ to the terminal set are partitioned using a $(1, \tau, \Delta)$-scattering partition. Afterwards, we assemble the different clusters from the partitions in all possible scales into a single terminal partition. We use the scattering property twice. First to argue that each vertex v is clustered to a terminal at distance at most $O(\tau) \cdot D(v)$ (here $D(v)$ is the distance to the closest terminal). Second, to argue that every shortest path where all the vertices are at similar distance to the terminal set, intersect the clusters of at most $O(\tau^2)$ terminals.

1.4 Related Work

In the functional analysis community, the notion of *Nagata dimension* was studied. The Nagata dimension of a metric space (X, d), $\dim_N X$, is the infimum over all integers n such that there exists a constant c s.t. X admits a $(c, n+1)$-weak sparse partition scheme. In contrast, in this paper our goal is to minimize this constant c. See [53] and the references therein.

A closely related problem to UST is the *Universal Traveling Salesman Problem* (UTSP). Consider a postman providing post service for a set X of clients with n different locations (with distance measure d_X). Each morning the postman receives a subset $S \subset X$ of the required deliveries for the day. In order to minimize the total tour length, one solution may be to compute each morning an (approximation of an) Optimal TSP tour for the set S. An alternative solution will be to compute a *Universal TSP* (UTSP) tour. This is a universal tour R containing all the points X. Given a subset S, $R(S)$ is the tour visiting all the points in S w.r.t. the order induced by R. Given a tour T denote its length by $|T|$. The *stretch* of R is the maximum ratio among all subsets $S \subseteq X$ between the length of $R(S)$ and the length of the optimal TSP tour on S, $\max_{S\subseteq X} \frac{|R(S)|}{|\mathrm{Opt}(S)|}$.

[14] Every (weighted) path graph has an SPDdepth 1. A graph G has an SPDdepth ρ if there exist a *shortest path* P, such that every connected component in $G \setminus P$ has an SPDdepth $\rho - 1$. This family includes graphs with pathwidth at most ρ, and more. See [2].

[15] A distribution $\mathcal{D}$ over solutions to the SPR problem has expected distortion α if $\forall t, t' \in K$, $\mathbb{E}_{M\sim\mathcal{D}}[d_M(t, t')] \le \alpha \cdot d_G(t, t')$.

All the sparse partition based upper bounds for the UST problem translated directly to the UTSP problem with the same parameters. The first to study the problem were Platzman and Bartholdi [57], who given n points in the Euclidean plane constructed a solution with stretch $O(\log n)$, using *space filling curves*. Recently, Christodoulou, and Sgouritsa [18] proved a lower and upper bound of $\Theta(\log n/\log\log n)$ for the $n \times n$ grid, improving a previous $\Omega(\sqrt[6]{\log n/\log\log n})$ lower bound of Hajiaghayi, Kleinberg, and Leighton [42] (and the $O(\log n)$ upper bound of [57]). For general n vertex graphs Gupta et al. [39] proved an $O(\log^2 n)$ upper bound, while Gorodezky, Kleinberg, Shmoys, and Spencer [36] proved an $\Omega(\log n)$ lower bound. From the computational point of view, Schalekamp and Shmoys [59] showed that if the input graph is a tree, an UTSP with optimal stretch can be computed efficiently.

The *A Priori TSP* problem is similar to the UTSP problem. In addition there is a distribution $\mathcal{D}$ over subsets $S \subseteq V$ and the stretch of tour a R is the expected ratio between the induced solution to optimal $\mathbb{E}_{S\sim\mathcal{D}}\frac{|R(S)|}{|\text{Opt}(S)|}$ (instead of a worst case like in UTSP). Similarly, *A Priori Steiner Tree* was studied (usually omitting rt from the problem). See [44, 59, 36] for further details. Another similar problem is the *Online (or dynamic) Steiner Tree* problem. Here the set S of vertices that should be connected is evolving over time, see [43, 6, 37] and references therein.

Unlike the definition used in this paper (taken from [45]), sparse partitions were also defined in the literature as partitions where only a small fraction of the edges are inter-cluster (see for example [5]). A closely related notion to sparse partitions are padded and separating decompositions. A graph G is β-decomposable if for every $\Delta > 0$, there is a distribution $\mathcal{D}$ over Δ bounded partitions such that for every $u, v \in V$, the probability that u and v belong to different clusters is at most $\beta \cdot \frac{d_G(u,v)}{\Delta}$. Note that by linearity of expectation, a path $\mathcal{I}$ of length Δ/σ intersects at most $1 + \beta/\sigma$ clusters in expectation. For comparison, in scattering partition we replace the distribution by a single partition and receive a bound on the number of intersections in the worst case. See [48, 9, 26, 40, 1, 5, 4, 34, 28] for further details.

Englert et al. [25] showed that every graph which is β-decomposable, admits a distribution $\mathcal{D}$ over solution to the SPR problem with expected distortion $O(\beta\log\beta)$. [15] In particular this implies constant expected distortion for graphs excluding a fixed minor, or bounded doubling dimension.

For a set K of terminals of size k, Krauthgamer, Nguyen and Zondiner [50] showed that if we allow the minor M to contain at most $\binom{k}{2}^2$ Steiner vertices (in addition to the terminals), then distortion 1 can be achieved. They further showed that for graphs with constant treewidth, $O(k^2)$ Steiner points will suffice for distortion 1. Cheung, Gramoz and Henzinger [17] showed that allowing $O(k^{2+\frac{2}{t}})$ Steiner vertices, one can achieve distortion $2t-1$. For planar graphs, Cheung et al. al. achieved $1+\epsilon$ distortion with $\tilde{O}((\frac{k}{\epsilon})^2)$ Steiner points.

There is a long line of work focusing on preserving the cut/flow structure among the terminals by a graph minor. See [56, 54, 15, 55, 25, 19, 51, 7, 35, 52].

There were works studying metric embeddings and metric data structures concerned with preserving distances among terminals, or from terminals to other vertices, out of the context of minors. See [20, 58, 41, 47, 21, 22, 10, 23, 49, 32, 24].

2 Preliminaries

All the logarithms in the paper are in base 2. We use $\tilde{O}$ notation to suppress constants and logarithmic factors, that is $\tilde{O}(f(j)) = f(j)\cdot\text{polylog}(f(j))$.

Graphs. We consider connected undirected graphs $G = (V, E)$ with edge weights $w : E \to \mathbb{R}_{\geq 0}$. Let d_G denote the shortest path metric in G. $B_G(v, r) = \{u \in V \mid d_G(v, u) \leq r\}$ is the ball of radius r around v. For a vertex $v \in V$ and a subset $A \subseteq V$, let $d_G(x, A) := \min_{a \in A} d_G(x, a)$, where $d_G(x, \emptyset) = \infty$. For a subset of vertices $A \subseteq V$, let $G[A]$ denote the induced graph on A, and let $G \setminus A := G[V \setminus A]$.

Special graph families. A graph H is a *minor* of a graph G if we can obtain H from G by edge deletions/contractions, and vertex deletions. A graph family $\mathcal{G}$ is H*-minor-free* if no graph $G \in \mathcal{G}$ has H as a minor. Some examples of minor free graph families are planar graphs (K_5 and $K_{3,3}$ free), outerplanar graphs (K_4 and $K_{3,2}$ free), series-parallel graphs (K_4 free), Cactus graphs (also known as tree of cycles) (⧅ free), and trees (K_3 free).

Given a graph $G = (V, E)$, a *tree decomposition* of G is a tree T with nodes $B_1, \dots, B_s$ (called *bags*) where each B_i is a subset of V such that the following properties hold:

- For every edge $\{u, v\} \in E$, there is a bag B_i containing both u and v.
- For every vertex $v \in V$, the set of bags containing v form a connected subtree of T.

The *width* of a tree decomposition is $\max_i\{|B_i| - 1\}$. The *treewidth* of G is the minimal width of a tree decomposition of G. A *path decomposition* of G is a special kind of tree decomposition where the underlying tree is a path. The *pathwidth* of G is the minimal width of a path decomposition of G.

Chordal graphs are unweighted graphs where each cycle of length greater then 4 contains a chordal. In other words, if the induced graph on a set of vertices V' is the cycle graph, than necessarily $|V'| \leq 3$. Chordal graphs contain interval graphs, subtree intersection graphs and other interesting sub families. A characterization of Chordal graphs is that they have a tree decomposition such that each bag is a clique. That is, there is a tree decomposition T of G where there is no upper bound on the size of a bag, but for every bag $B \in T$ the induced graph $G[B]$ is a clique.

A *Cactus graph* (a.k.a. tree of cycles) is a graph where each edge belongs to at most one simple cycle. Alternatively it can be defined as the graph family that excludes K_4 minus an edge (⧅) as a minor.

Abraham et al. [2] defined *shortest path decompositions* (SPDs) of "low depth". Every (weighted) path graph has an SPDdepth 1. A graph G has an SPDdepth k if there exist a *shortest path* P, such that every connected component in $G \setminus P$ has an SPDdepth $k - 1$. In other words, given a graph, in SPD we hierarchically delete shortest paths from each connected component, until no vertices remain. See [2] for formal definition (or full version [31]). Every graph with pathwidth ρ has SPDdepth at most $\rho + 1$, treewidth ρ implies SPDdepth at most $O(\rho \log n)$, and every graph excluding a fixed minor has SPDdepth $O(\log n)$. See [2, 30] for further details and applications.

3 From Scattering Partitions to SPR: Proof of Theorem 2

We will assume w.l.o.g. that the minimal pairwise distance in the graph is exactly 1, otherwise we can scale all the weights accordingly. The set of terminals denoted $K = \{t_1, \dots, t_k\}$. For every vertex $v \in V$, denote by $D(v) = d_G(v, K)$ the distance to its closest terminal. Note that $\min_{v \in V \setminus K} D(v) \geq 1$.

Similarly to previous papers on the SPR problem, we will create a minor using *terminal partitions*. Specifically, we partition the vertices into k connected clusters, with a single terminal in each cluster. Such a partition induces a minor by contracting all the internal edges in each cluster. More formally, a partition $\{V_1, \dots, V_k\}$ of V is called a terminal partition (w.r.t to K) if for every $1 \leq i \leq k$, $t_i \in V_i$, and the induced graph $G[V_i]$ is connected. For a

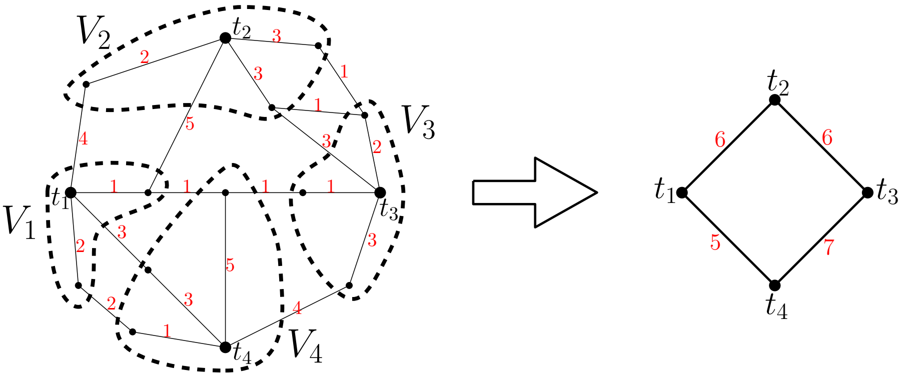


Figure 2 The left side of the figure contains a weighted graph $G = (V, E)$, with weights specified in red, and four terminals $\{t_1, t_2, t_3, t_4\}$. The dashed black curves represent a terminal partition of the vertex set V into the subsets V_1, V_2, V_3, V_4. The right side of the figure represents the minor M induced by the terminal partition. The distortion is realized between t_1 and t_3, and is $\frac{d_M(t_1,t_3)}{d_G(t_1,t_3)} = \frac{12}{4} = 3$.

vertex $v \in V_i$, we say that v is assigned to t_i. See Figure 2 for an illustration. The *induced minor* by the terminal partition $\{V_1, \dots, V_k\}$, is a minor M, where each set V_i is contracted into a single vertex called (abusing notation) t_i. Note that there is an edge in M from t_i to t_j if and only if there are vertices $v_i \in V_i$ and $v_j \in V_j$ such that $\{v_i, v_j\} \in E$. We determine the weight of the edge $\{t_i, t_j\} \in E(M)$ to be $d_G(t_i, t_j)$. Note that by the triangle inequality, for every pair of (not necessarily neighboring) terminals t_i, t_j, it holds that $d_M(t_i, t_j) \geq d_G(t_i, t_j)$. The *distortion* of the induced minor is $\max_{i,j} \frac{d_M(t_i,t_j)}{d_G(t_i,t_j)}$.

3.1 Algorithm

For $i \geq 1$, set $\mathcal{R}_i = \{v \in V \mid 2^{i-1} \leq D(v) < 2^i\}$ to be the set of vertices at distance between 2^{i-1} and 2^i from K. Set $\mathcal{R}_0 = K$. We create the terminal partition in an iterative manner, where initially each set $V_i = \{t_i\}$ is a singleton, and gradually more vertices are joining. We will denote the stage of the terminal partition after i steps, using a function $f_i : V \to K \cup \{\perp\}$. For a yet unassigned vertex v we write $f_i(v) = \perp$, otherwise the vertex v will be assigned to $f_i(v)$. Initially for every terminal t_j, $f_0(t_j) = t_j$ while for every Steiner vertex $v \in V \setminus K$, $f_0(v) = \perp$. In iteration i we will define f_i by "extending" f_{i-1}. That is, unassigned vertices may be assigned (i.e., for v such that $f_{i-1}(v) = \perp$ it might be $f_i(v) = t_j$), while the function will remain the same on the set of assigned vertices ($f_{i-1}(v) \neq \perp \Rightarrow f_i(v) = f_{i-1}(v)$). We will guarantee that all the vertices in $\mathcal{R}_i$ will be assigned in f_i. In particular, after $\log(\max_v D(v))$ steps, all the vertices will be assigned. Denote by V_i the set of vertices assigned by f_i. Initially $V_0 = K = \mathcal{R}_0$. By induction we will assume that $\cup_{j \leq i-1} \mathcal{R}_j \subseteq V_{i-1}$. Let $G_i = G[V \setminus V_{i-1}]$ be the graph induced by the set of yet unassigned vertices. Fix $\Delta_i = 2^{i-1}$. Let $\mathcal{P}_i$ be an $(1, \tau, \Delta_i)$-scattering partition of G_i. Let $\mathcal{C}_i \subseteq \mathcal{P}_i$ be the set of clusters C which contain at least one vertex $v \in \mathcal{R}_i$. All the vertices in $\cup \mathcal{C}_i$ will be assigned by f_i.

We say that a cluster $C \in \mathcal{C}_i$ is at level 1, noting $\delta_i(C) = 1$, if there is an edge $\{v, u_C\}$ (in G) from a vertex $v \in C$ to a vertex $u_C \in V_{i-1}$ of weight at most 2^i. In general, $\delta_i(C) = l$, if l is the minimal index such that there is an edge $\{v, u_C\}$ from a vertex $v \in C$ to a vertex $u_C \in C'$ of weight at most 2^i, such that $\delta_i(C') = l - 1$. In both cases u_C is called the *linking* vertex of C. Next, we define f_i based on f_{i-1}. For every vertex $v \in V_{i-1}$ set $f_i(v) = f_{i-1}(v)$.

For every vertex not in $\cup\mathcal{C}_i$ (or V_{i-1}) set $f_i(v) =\perp$. For a cluster $C \in \mathcal{C}_i$ s.t. $\delta_i(C) = 1$, let $u_C \in V_{i-1}$ be its linking vertex. For every $v \in C$ set $f_i(v) = f_i(u_C)$. Generally, for level l suppose that f_i is already defined on all the clusters of level $l-1$. Let $C \in \mathcal{C}_i$ s.t. $\delta_i(C) = l$. Let u_C be the linking vertex of C. For every $v \in C$, set $f_i(v) = f_i(u_C)$. Note that for every cluster, all the vertices are mapped to the same terminal. This finishes the definition of f_i.

The algorithm continues until there is f_i where all the Steiner vertices are assigned. Set $f = f_i$. The algorithm returns the terminal-centered minor M of G induced by $\{f^{-1}(t_1), \dots, f^{-1}(t_k)\}$.

3.2 Basic Properties

It is straightforward from the construction that $f^{-1}(t_1), \dots, f^{-1}(t_k)$ define a terminal partition. We will prove that every vertex v will be assigned during either iteration $\lceil \log D(v) \rceil$ or $\lceil \log D(v) \rceil - 1$ (Claim 9), to a terminal at distance at most $O(\tau) \cdot D(v)$ from v (Corollary 8). We begin by arguing that in each iteration, the maximum possible level of a cluster is τ.

▷ Claim 6. For every cluster $C \in \mathcal{C}_i$, $\delta_i(C) \leq \tau$.

Proof. Consider a cluster $C \in \mathcal{C}_i$, and let $v \in C$ be a vertex s.t. $D(v) \leq 2^i$. Let $P = \{v = v_0, \dots, v_s\}$ be a prefix of the $D(v)$ length path from v to its closest terminal such that v_s has a neighbor in V_{i-1}. Note that P has (weighted) length at most $2^{i-1} = \Delta_i$ (as all vertices v' for which $D(v') \leq 2^{i-1}$ are necessarily clustered). $\mathcal{P}_i$ is a $(1, \tau, \Delta_i)$-scattering partition. Hence the vertices of P are partitioned to $\tau' \leq \tau$ clusters $C_1, \dots, C_{\tau'}$ where $v_s \in C_1$, $v_0 \in C_{\tau'}$ and there is an edge from C_j to C_{j+1} of weight at most $2^{i-1} < 2^i$, while the edge from v_s towards V_{i-1} is of weight at most 2^i. It holds that $\delta_i(C_1) = 1$, and by induction $\delta(C_j) \leq j$. In particular $\delta(C) \leq \tau' \leq \tau$. ◁

▷ Claim 7. For every vertex v which is assigned during the i'th iteration (i.e., $v \in C \in \mathcal{C}_i$) it holds that $d_G(v, f(v)) \leq 3\tau \cdot 2^i$.

Proof. The proof is by induction on i. For $i = 0$ the assertion holds trivially as every terminal is assigned to itself. We will assume the assertion for $i-1$ and prove it for i. Let $C \in \mathcal{C}_i$ be some cluster, and let $v \in C$. Suppose first that $\delta_i(C) = 1$. Let $u_C \in V_{i-1}$ be the linking vertex of C. By the induction hypothesis $d_G(u_C, f(u_C)) \leq 3\tau \cdot 2^{i-1}$. As the diameter of C is bounded by 2^{i-1}, and the weight of the edge towards u_C is at most 2^i we conclude $d_G(v, f(v)) \leq d_G(v, u_C) + d_G(u_C, f(u_C)) \leq (2^{i-1} + 2^i) + 3\tau \cdot 2^{i-1} = 3 \cdot 2^{i-1} + 3\tau \cdot 2^{i-1}$. Generally, for $\delta_i(C) = l$, we argue by induction that for every $v \in C$ it holds that $d_G(v, f(v)) \leq l \cdot 3 \cdot 2^{i-1} + 3\tau \cdot 2^{i-1}$. Indeed, let u_C by the linking vertex of C. By the induction hypothesis it holds that $d_G(u_C, f(u_C)) \leq (l-1) \cdot 3 \cdot 2^{i-1} + 3\tau \cdot 2^{i-1}$. Using similar arguments, it holds that $d_G(v, f(v)) \leq d_G(v, u_C) + d_G(u_C, f(u_C)) \leq (2^{i-1} + 2^i) + (l-1) \cdot 3 \cdot 2^{i-1} + 3\tau \cdot 2^{i-1} = l \cdot 3 \cdot 2^{i-1} + 3\tau \cdot 2^{i-1}$. Using Claim 6, $d_G(v, f(v)) \leq 3\tau \cdot 2^{i-1} + 3\tau \cdot 2^{i-1} = 3\tau \cdot 2^i$ as required. ◁

▶ **Corollary 8.** *For every vertex v it holds that $d_G(v, f(v)) < 6\tau \cdot D(v)$.*

Proof. Let $i \geq 0$ such that $2^{i-1} < D(v) \leq 2^i$. The vertex v is assigned at iteration i or earlier. By Claim 7 we conclude $d_G(v, f(v)) \leq 3\tau \cdot 2^i < 6\tau \cdot D(v)$. ◀

▷ Claim 9. Consider a vertex v such that $2^{i-1} < D(v) \leq 2^i$. Then v is assigned either at iteration $i-1$ or i.

Proof. Clearly if v remains un-assigned until iteration i, it will be assigned during the i'th iteration. Suppose that v was assigned during iteration j. Then v belongs to a cluster $C \in \mathcal{C}_j$. In particular there is a vertex $u \in C$ such that $D(u) \le 2^j$. As C has diameter at most 2^{j-1}, it holds that

$$2^{i-1} < D(v) \le D(u) + d_G(v,u) \le 2^j + 2^{j-1} = 3 \cdot 2^{j-1} \ .$$

i, j are integers, hence $j \ge i-1$. ◁

3.3 Distortion Analysis

In this section we analyze the distortion of the minor induced by the terminal partition created by our algorithm. We have several variables that are defined with respect to the algorithm. Note that all these definitions are for analysis purposes only, and have no impact on the execution of the algorithm. Consider a pair of terminals t and t'. Let $P_{t,t'} = \{t = v_0, \dots, v_\gamma = t'\}$ be the shortest path from t to t' in G. We can assume that there are no terminals in $P_{t,t'}$ other than t, t'. This is because if we will prove the distortion guarantee for every pair of terminals t, t' such that $P_{t,t'} \cap K = \{t, t'\}$, then by the triangle inequality the distortion guarantee will hold for all terminal pairs.

Detours. The terminals t, t' are fixed. During the execution of the algorithm, for every terminal t_j we will maintain a *detour* $\mathcal{D}_{t_j}$ (or shortly $\mathcal{D}_j$). A detour is a consecutive subinterval $\{a_j, \dots, b_j\}$ of $P_{t,t'}$, where $a_j \in \mathcal{D}_j$ is the leftmost (i.e., with minimal index) vertex in the detour and b_j is the rightmost. Initially $\mathcal{D}_t = \{t\}$ and $\mathcal{D}_{t'} = \{t'\}$, while for every $t_j \notin \{t, t'\}$, $\mathcal{D}_j = \emptyset$. Every pair of detours $\mathcal{D}_j, \mathcal{D}_{j'}$ will be disjoint throughout the execution of the algorithm.

A vertex $v \in P_{t,t'}$ is *active* if and only if it does not belong to any detour. It will hold that every active vertex is necessarily unassigned (while there might be unassigned vertices which are inactive). Initially, t, t' are *inactive*, while all the other vertices of $P_{t,t'}$ are active. Consider the i'th iteration of the algorithm. We go over the terminals according to an arbitrary order $\{t_1, \dots, t_k\}$. Consider the terminal t_j with detour $\mathcal{D}_j = \{a_j, \dots, b_j\}$ (which might be empty). If no active vertices are assigned to t_j we do nothing. Otherwise, let $a'_j \in P_{t,t'}$ (resp. b'_j) be the leftmost (resp. rightmost) active vertex that was assigned to t_j during the i'th iteration. Set a_j to be vertex with minimal index between the former a_j and a'_j (a'_j if there was no a_j). Similarly b_j is the vertex with maximal index between the former b_j and b'_j. $\mathcal{D}_j$ is updated to be $\{a_j, \dots, b_j\}$. All the vertices in $\{a_j, \dots, b_j\} = \mathcal{D}_j$ become inactive. Note that a vertex might become inactive while remaining yet unassigned.

Consider an additional detour $\mathcal{D}_{j'}$. Before the updating of $\mathcal{D}_j$ at iteration i, $\mathcal{D}_j, \mathcal{D}_{j'}$ are disjoint. If a'_j, b'_j were active they cannot belong to $\mathcal{D}_{j'}$. Thus after the update, a_j, b_j did not belong to $\mathcal{D}_{j'}$ as well. However, it is possible that after the update $\mathcal{D}_j$ and $\mathcal{D}_{j'}$ are no longer disjoint. The only such possibility is when $\mathcal{D}_{j'} \subset \mathcal{D}_j$. In such a case, we set $\mathcal{D}_{j'} \leftarrow \emptyset$, maintaining the disjointness property (while not changing the (in)active status of any vertex).

After we nullify all the detours that were contained in $\mathcal{D}_j$, we will proceed to treat the next terminals in turn. Once we finish going over all the terminals, we proceed to the $i+1$ iteration. Eventually, all the vertices cease to be active, and in particular belong to some detour. In other words, all the vertices of $P_{t,t'}$ are partitioned to consecutive disjoint detours $\mathcal{D}_{\ell_1}, \dots, \mathcal{D}_{\ell_s}$.

Intervals. For an *interval* $Q=\{v_a,\dots,v_b\}\subseteq P_{t,t'}$, the *internal length* is $L(Q)=d_G(v_a,v_b)$, while the *external length* is $L^+(Q)=d_G(v_{a-1},v_{b+1})$.[16] We denote by $D(Q)=D(v_a)$ the distance from the leftmost vertex $v_a\in Q$ to its closest terminal. Set $c_{\text{int}}=\frac{1}{7}$ ("int" for interval). We partition the vertices in $P_{t,t'}$ into consecutive intervals $\mathcal{Q}$, such that for every $Q\in\mathcal{Q}$,

$$L(Q)\le c_{\text{int}}\cdot D(Q)\le L^+(Q)\ . \tag{3.1}$$

Such a partition could be obtained as follows: Sweep along the path $P_{t,t'}$ in a greedy manner, after partitioning the prefix $v_0,\dots,v_{h-1}$, to construct the next interval Q, simply pick the minimal index s such that $L^+(\{v_h,\dots,v_{h+s}\})\ge c_{\text{int}}\cdot D(v_h)$. By the minimality of s, $L(\{v_h,\dots,v_{h+s}\})\le L^+(\{v_h,\dots,v_{h+s-1}\})\le c_{\text{int}}\cdot D(v_h)$ (in the case $s=0$, trivially $L(\{v_h\})=0\le c_{\text{int}}\cdot D(v_h)$). Note that such s could always be found, as $L^+(\{v_h,\dots,v_\gamma=t'\})=d_G(v_{h-1},t')\ge d_G(v_h,t')\ge D(v_h)=D(Q)$.

Consider some interval $Q=\{v_a,\dots,v_b\}\in\mathcal{Q}$. For every vertex $v\in Q$, by triangle inequality it holds that $D(Q)-L(Q)\le D(v)\le D(Q)+L(Q)$. Therefore,

$$(1-c_{\text{int}})D(Q)\le D(v)\le(1+c_{\text{int}})D(Q)\ . \tag{3.2}$$

Note that the set $\mathcal{Q}$ of intervals is determined before the execution of the algorithm, and is never changed. In particular, it is independent from the set of detours (which evolves during the execution of the algorithm).
For an interval Q, we denote by i_Q the first iteration when some vertex v belonging to the interval Q is assigned.

▷ Claim 10. All Q vertices are assigned in either iteration i_Q or i_Q+1.

Proof. Let $u\in Q$ be some vertex which is assigned during iteration i_Q. Then u belongs to a cluster $C\in\mathcal{C}_{i_Q}$, containing a vertex $u'\in C$ such that $D(u')\le 2^{i_Q}$. As C has diameter at most 2^{i_Q-1}, it holds that $2^{i_Q}\ge D(u')\ge D(u)-d_G(u,u')\ge D(u)-2^{i_Q-1}$. Hence $D(u)\le\frac{3}{2}\cdot 2^{i_Q}$. It follows that

$$D(Q)\overset{(3.2)}{\le}\frac{1}{1-c_{\text{int}}}\cdot D(u)\le\frac{3}{2}\cdot\frac{1}{1-c_{\text{int}}}\cdot 2^{i_Q}\ . \tag{3.3}$$

For every vertex $v\in Q$ it holds that,

$$D(v)\le D(Q)+L(Q)\overset{(3.2)}{\le}(1+c_{\text{int}})\cdot D(Q\overset{(3.3)}{\le}\frac{3}{2}\cdot\frac{(1+c_{\text{int}})}{(1-c_{\text{int}})}\cdot 2^{i_Q}=2^{i_Q+1}\ .$$

Therefore, in the i_Q+1 iteration, all the (yet unassigned) vertices of Q will necessarily be assigned. ◁

▶ **Lemma 11.** *Consider an interval $Q\in\mathcal{Q}$. Then the vertices of Q are partitioned into at most $O(\tau^2)$ different detours.*

Proof. By definition, by the end of the i_Q-1'th iteration all the vertices of Q are unassigned. We first consider the case where by the end i_Q-1'th iteration some vertex $v\in Q$ is inactive. It holds that v belongs to some detour $\mathcal{D}_j$. As all the vertices of Q are unassigned, necessarily $Q\subset\mathcal{D}_j$. In particular, all the vertices of Q belong to a single detour. This property will not change till the end of the algorithm, thus the lemma follows.

[16] For ease of notation we will denote $v_{-1}=t$ and $v_{\gamma+1}=t'$.

Next, we consider the case where by the end of the $i_Q - 1$'th iteration all the vertices of Q are active. The algorithm at iteration i_Q creates an $\left(1, \tau, \Delta_{i_Q}\right)$-scattering partition $\mathcal{P}_{i_Q}$. The length of Q is bounded by

$$L(Q) \overset{(3.1)}{\leq} c_{\text{int}} \cdot D(Q) \overset{(3.3)}{\leq} c_{\text{int}} \cdot \frac{3}{2} \cdot \frac{1}{1 - c_{\text{int}}} \cdot 2^{i_Q} = \frac{1}{4} \cdot 2^{i_Q} < \Delta_{i_Q} \tag{3.4}$$

Hence Q is partitioned by $\mathcal{P}_{i_Q}$ to $\tau' \leq \tau$ clusters $C_1, \dots, C_{\tau'} \in \mathcal{P}_{i_Q}$. It follows that by the end of the i_Q'th iteration, the inactive vertices in Q are partitioned to at most τ detours. If all the vertices in Q become inactive, then we are done, as the number of detours covering Q can only decrease further in the algorithm (as a result of detour nullification). Hence we will assume that some of Q vertices remain active.

A *slice* is a maximal sub-interval $S \subseteq Q$ of active vertices. The active vertices in Q are partitioned to at most $\tau + 1$ slices $S_1, S_2, \dots, S_{\tau''}$.[17] By the end of the $i_Q + 1$ iteration, according to Claim 10 all Q vertices will be assigned, and in particular belong to some detour. The algorithm creates a $\left(\Delta_{i_Q+1}, \tau, 1\right)$-scattering partition $\mathcal{P}_{i_Q+1}$ of the unassigned vertices. By equation (3.4) the length of every slice S is bounded by $L(S) \leq L(Q) \leq \frac{1}{4} \cdot 2^{i_Q} \leq \Delta_{i_Q+1}$. Therefore the vertices S intersect at most τ clusters of $\mathcal{P}_{i_Q+1}$, and thus will be partitioned to at most τ detours. Some detours might get nullified, however in the worst case, by the end of the $i_Q + 1$ iteration, the vertices in $\cup_i S_i$ are partitioned to at most $\tau \cdot (\tau + 1)$ detours. In particular all the vertices in Q are partitioned to at most $O(\tau^2)$ detours. As the number of detours covering Q can only decrease further in the algorithm, the lemma follows. ◀

By the end of algorithm, we will *charge* the intervals for the detours. Consider the detour $\mathcal{D}_j = \{a_j, \dots, b_j\}$ of t_j. Let $Q_j \in \mathcal{Q}$ be the interval containing a_j. We will charge Q_j for the detour $\mathcal{D}_j$. Denote by $X(Q)$ the number of detours for which the interval Q is charged for. By Lemma 11, $X(Q) = O(\tau^2)$ for every interval $Q \in \mathcal{Q}$.

Recall that by the end of the algorithm, all the vertices of $P_{t,t'}$ are partitioned to consecutive disjoint detours $\mathcal{D}_{\ell_1}, \dots, \mathcal{D}_{\ell_s}$, where $\mathcal{D}_{\ell_j} = \{a_{\ell_j}, \dots, b_{\ell_j}\}$ and a_{ℓ_j}, b_{ℓ_j} belong to the cluster of t_{ℓ_j}. In particular $t_{\ell_1} = t$ and $t_{\ell_s} = t'$, as each terminal belongs to the cluster of itself. Moreover, for every $j < s$, there is an edge $\{b_{\ell_j}, a_{\ell_{j+1}}\}$ in G between the cluster of t_{ℓ_j} to that of $t_{\ell_{j+1}}$. Therefore, in the minor induced by the partition there is an edge between t_{ℓ_j} to $t_{\ell_{j+1}}$. We conclude

$$\begin{aligned} d_M(t, t') &\leq \sum_{j=1}^{s-1} d_G(t_{\ell_j}, t_{\ell_{j+1}}) \leq \sum_{j=1}^{s-1} \left[d_G(t_{\ell_j}, a_{\ell_j}) + d_G(a_{\ell_j}, a_{\ell_{j+1}}) + d_G(a_{\ell_{j+1}}, t_{\ell_{j+1}}) \right] \\ &\leq \sum_{j=1}^{s-1} d_G(a_{\ell_j}, a_{\ell_{j+1}}) + 2 \sum_{j=1}^{s} d_G(t_{\ell_j}, a_{\ell_j}) \ . \end{aligned}$$

Note that $\sum_{j=1}^{s-1} d_G(a_{\ell_j}, a_{\ell_{j+1}}) \leq d_G(t, t')$ as $P_{t,t'}$ is a shortest path. Denote by Q_{ℓ_j} the interval containing a_{ℓ_j}. By Corollary 8,

$$d_G(t_{\ell_j}, a_{\ell_j}) = d_G(a_{\ell_j}, f(a_{\ell_j})) \leq O(\tau) \cdot D(a_{\ell_j}) \overset{(3.2)}{=} O(\tau) \cdot D(Q_{\ell_j}) \overset{(3.1)}{=} O(\tau) \cdot L^+(Q_{\ell_j}) \ .$$

By changing the order of summation we get

$$\sum_{j=1}^{s} d_G(t_{\ell_j}, a_{\ell_j}) = O(\tau) \cdot \sum_{Q \in \mathcal{Q}} X(Q) \cdot L^+(Q) = O(\tau^3) \cdot \sum_{Q \in \mathcal{Q}} L^+(Q) \ .$$

Finally, note that $\sum_{Q \in \mathcal{Q}} L^+(Q) \leq 2 \cdot d_G(t, t')$ as every edge in $P_{t,t'}$ is counted at most twice. We conclude $d_M(t, t') \leq O(\tau^3) \cdot d_G(t, t')$. Theorem 2 now follows.

[17] Actually, as at least one Q vertex remained active, at the beginning of the $i_Q + 1$ iteration the inactive vertices of Q partitioned to at most $\tau - 1$ detours. Therefore the maximal number of slices is τ.

4 Discussion and Open Problems

In this paper we defined scattering partitions, and showed how to apply them in order to construct solutions to the SPR problem. We proved an equivalence between sparse partitions and sparse covers. Finally, we constructed many sparse and scattering partitions for different graph families (and lower bounds), implying new results for the SPR, UST, and UTSP problems. An additional contribution of this paper is a considerable list of (all but question (5)) new intriguing open questions and conjectures.

1. **Planar graphs:** The SPR problem is most fascinating and relevant for graph families which are closed under taking a minor. Note that already for planar graphs (or even treewidth 2 graphs), the best upper bound for the SPR problem is $O(\log k)$ (same as general graphs), while the only lower bound is 8. The most important open question coming out of this paper is the following conjecture:

   ▶ **Conjecture 1.** *Every graph family excluding a fixed minor is $(O(1), O(1))$-scatterable.*

   Note that proving this conjecture for a family $\mathcal{F}$, will imply a solution to the SPR problem with constant distortion. Proving the conjecture for planar graphs will be fascinating. However, it is already open for outerplanar graphs, and graphs with treewidth 2.

2. **Scattering Partitions for General Graphs:** While we provide almost tight upper and lower bounds for sparse partitions, for scattering partitions, the story is different.

   ▶ **Conjecture 2.** *Consider an n vertex weighted graph G such that between every pair of vertices there is a unique shortet path. Then G is $(1, O(\log n))$-scatterable. Furthermore, this is tight.*

   In the full version [31], we provide some evidence that Conjecture 2 cannot be pushed further. However, any nontrivial lower bound will be interesting. Furthermore, every lower bound larger than 8 for the general SPR problem will be intriguing.

3. **Doubling graphs:** While we constructed strong sparse partition for doubling graphs (which imply scattering), it has no implication for the SPR problem. This is due to the fact that Theorem 2 required scattering partition for every induced subgraph. As induced subgraphs of a doubling graph might have unbounded doubling dimension, the proof fails to follow through. We leave the required readjustments to future work.

4. **Sparse Covers:** We classify various graph families according to the type of partitions/-covers they admit. We currently lack any example of a graph family that admits weak sparse covers but does not admit strong sparse covers. It will be interesting to find such an example, or even more so to prove that every graph that admits weak sparse cover, also has strong sparse cover with (somewhat) similar parameters.

References

1 Ittai Abraham, Yair Bartal, and Ofer Neiman. Advances in metric embedding theory. *Advances in Mathematics*, 228(6):3026–3126, 2011. doi:10.1016/j.aim.2011.08.003.

2 Ittai Abraham, Arnold Filtser, Anupam Gupta, and Ofer Neiman. Metric embedding via shortest path decompositions. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 952–963, 2018. Full version: arxiv:1708.04073. doi:10.1145/3188745.3188808.

3 Ittai Abraham, Cyril Gavoille, Andrew V. Goldberg, and Dahlia Malkhi. Routing in networks with low doubling dimension. In *26th IEEE International Conference on Distributed Computing Systems (ICDCS 2006), 4-7 July 2006, Lisboa, Portugal*, page 75, 2006. doi:10.1109/ICDCS.2006.72.

4 Ittai Abraham, Cyril Gavoille, Anupam Gupta, Ofer Neiman, and Kunal Talwar. Cops, robbers, and threatening skeletons: Padded decomposition for minor-free graphs. *SIAM J. Comput.*, 48(3):1120–1145, 2019. doi:10.1137/17M1112406.

5 Ittai Abraham, Cyril Gavoille, Dahlia Malkhi, and Udi Wieder. Strong-diameter decompositions of minor free graphs. *Theory Comput. Syst.*, 47(4):837–855, 2010. doi:10.1007/s00224-010-9283-6.

6 Noga Alon and Yossi Azar. On-line steiner trees in the euclidean plane. In *Proceedings of the Eighth Annual Symposium on Computational Geometry, Berlin, Germany, June 10-12, 1992*, pages 337–343, 1992. doi:10.1145/142675.142744.

7 Alexandr Andoni, Anupam Gupta, and Robert Krauthgamer. Towards $(1+\epsilon)$-approximate flow sparsifiers. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 279–293, 2014. doi:10.1137/1.9781611973402.20.

8 Baruch Awerbuch and David Peleg. Sparse partitions (extended abstract). In *31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri, USA, October 22-24, 1990, Volume II*, pages 503–513, 1990. doi:10.1109/FSCS.1990.89571.

9 Yair Bartal. Probabilistic approximations of metric spaces and its algorithmic applications. In *37th Annual Symposium on Foundations of Computer Science, FOCS '96, Burlington, Vermont, USA, 14-16 October, 1996*, pages 184–193, 1996. doi:10.1109/SFCS.1996.548477.

10 Yair Bartal, Arnold Filtser, and Ofer Neiman. On notions of distortion and an almost minimum spanning tree with constant average distortion. *J. Comput. Syst. Sci.*, 105:116–129, 2019. doi:10.1016/j.jcss.2019.04.006.

11 A. Basu and A. Gupta. Steiner point removal in graph metrics. Unpublished Manuscript, available from http://www.ams.jhu.edu/~abasu9/papers/SPR.pdf, 2008.

12 Costas Busch, Chinmoy Dutta, Jaikumar Radhakrishnan, Rajmohan Rajaraman, and Srinivasagopalan Srivathsan. Split and join: Strong partitions and universal steiner trees for graphs. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 81–90, 2012. doi:10.1109/FOCS.2012.45.

13 Costas Busch, Ryan LaFortune, and Srikanta Tirthapura. Sparse covers for planar graphs and graphs that exclude a fixed minor. *Algorithmica*, 69(3):658–684, 2014. doi:10.1007/s00453-013-9757-4.

14 T.-H. Chan, Donglin Xia, Goran Konjevod, and Andrea Richa. A tight lower bound for the steiner point removal problem on trees. In *Proceedings of the 9th International Conference on Approximation Algorithms for Combinatorial Optimization Problems, and 10th International Conference on Randomization and Computation*, APPROX'06/RANDOM'06, pages 70–81, Berlin, Heidelberg, 2006. Springer-Verlag. doi:10.1007/11830924_9.

15 Moses Charikar, Tom Leighton, Shi Li, and Ankur Moitra. Vertex sparsifiers and abstract rounding algorithms. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 265–274, 2010. doi:10.1109/FOCS.2010.32.

16 Yun Kuen Cheung. Steiner point removal - distant terminals don't (really) bother. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1353–1360, 2018. doi:10.1137/1.9781611975031.89.

17 Yun Kuen Cheung, Gramoz Goranci, and Monika Henzinger. Graph minors for preserving terminal distances approximately - lower and upper bounds. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, pages 131:1–131:14, 2016. doi:10.4230/LIPIcs.ICALP.2016.131.

18 George Christodoulou and Alkmini Sgouritsa. An improved upper bound for the universal TSP on the grid. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 1006–1021, 2017. doi:10.1137/1.9781611974782.64.

19 Julia Chuzhoy. On vertex sparsifiers with steiner nodes. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 673–688, 2012. doi:10.1145/2213977.2214039.

20 Don Coppersmith and Michael Elkin. Sparse sourcewise and pairwise distance preservers. *SIAM J. Discrete Math.*, 20(2):463–501, 2006. doi:10.1137/050630696.

21 Michael Elkin, Arnold Filtser, and Ofer Neiman. Terminal embeddings. *Theor. Comput. Sci.*, 697:1–36, 2017. doi:10.1016/j.tcs.2017.06.021.

22 Michael Elkin, Arnold Filtser, and Ofer Neiman. Prioritized metric structures and embedding. *SIAM J. Comput.*, 47(3):829–858, 2018. doi:10.1137/17M1118749.

23 Michael Elkin and Ofer Neiman. Near isometric terminal embeddings for doubling metrics. In *34th International Symposium on Computational Geometry, SoCG 2018, June 11-14, 2018, Budapest, Hungary*, pages 36:1–36:15, 2018. doi:10.4230/LIPIcs.SoCG.2018.36.

24 Michael Elkin and Ofer Neiman. Lossless prioritized embeddings. *CoRR*, abs/1907.06983, 2019. To appear in SODA2020. arXiv:1907.06983.

25 Matthias Englert, Anupam Gupta, Robert Krauthgamer, Harald Räcke, Inbal Talgam-Cohen, and Kunal Talwar. Vertex sparsifiers: New results from old techniques. *SIAM J. Comput.*, 43(4):1239–1262, 2014. doi:10.1137/130908440.

26 Jittat Fakcharoenphol and Kunal Talwar. An improved decomposition theorem for graphs excluding a fixed minor. In *Approximation, Randomization, and Combinatorial Optimization: Algorithms and Techniques, 6th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2003 and 7th International Workshop on Randomization and Approximation Techniques in Computer Science, RANDOM 2003, Princeton, NJ, USA, August 24-26, 2003, Proceedings*, pages 36–46, 2003. doi:10.1007/978-3-540-45198-3_4.

27 Arnold Filtser. Steiner point removal with distortion $O(\log k)$. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1361–1373, 2018. doi:10.1137/1.9781611975031.90.

28 Arnold Filtser. On strong diameter padded decompositions. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2019, September 20-22, 2019, Massachusetts Institute of Technology, Cambridge, MA, USA.*, pages 6:1–6:21, 2019. doi:10.4230/LIPIcs.APPROX-RANDOM.2019.6.

29 Arnold Filtser. Steiner point removal with distortion $O(\log k)$ using the relaxed-voronoi algorithm. *SIAM J. Comput.*, 48(2):249–278, 2019. doi:10.1137/18M1184400.

30 Arnold Filtser. A face cover perspective to ℓ_1 embeddings of planar graphs. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1945–1954, 2020. doi:10.1137/1.9781611975994.120.

31 Arnold Filtser. Scattering and sparse partitions, and their applications. *CoRR*, abs/2001.04447, 2020. arXiv:2001.04447.

32 Arnold Filtser, Lee-Ad Gottlieb, and Robert Krauthgamer. Labelings vs. embeddings: On distributed representations of distances. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1063–1075, 2020. doi:10.1137/1.9781611975994.65.

33 Arnold Filtser, Robert Krauthgamer, and Ohad Trabelsi. Relaxed voronoi: A simple framework for terminal-clustering problems. In *2nd Symposium on Simplicity in Algorithms, SOSA@SODA 2019, January 8-9, 2019 - San Diego, CA, USA*, pages 10:1–10:14, 2019. doi:10.4230/OASIcs.SOSA.2019.10.

34 Arnold Filtser and Ofer Neiman. Light spanners for high dimensional norms via stochastic decompositions. In *26th Annual European Symposium on Algorithms, ESA 2018, August 20-22, 2018, Helsinki, Finland*, pages 29:1–29:15, 2018. doi:10.4230/LIPIcs.ESA.2018.29.

35 Gramoz Goranci, Monika Henzinger, and Pan Peng. Improved guarantees for vertex sparsification in planar graphs. In *25th Annual European Symposium on Algorithms, ESA 2017, September 4-6, 2017, Vienna, Austria*, pages 44:1–44:14, 2017. doi:10.4230/LIPIcs.ESA.2017.44.

36 Igor Gorodezky, Robert D. Kleinberg, David B. Shmoys, and Gwen Spencer. Improved lower bounds for the universal and *a priori* TSP. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 13th International Workshop, APPROX 2010, and 14th International Workshop, RANDOM 2010, Barcelona, Spain, September 1-3, 2010. Proceedings*, pages 178–191, 2010. doi:10.1007/978-3-642-15369-3_14.

37 Albert Gu, Anupam Gupta, and Amit Kumar. The power of deferral: Maintaining a constant-competitive steiner tree online. *SIAM J. Comput.*, 45(1):1–28, 2016. doi:10.1137/140955276.

38 Anupam Gupta. Steiner points in tree metrics don't (really) help. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '01, pages 220–227, Philadelphia, PA, USA, 2001. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=365411.365448.

39 Anupam Gupta, Mohammad Taghi Hajiaghayi, and Harald Räcke. Oblivious network design. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, Miami, Florida, USA, January 22-26, 2006*, pages 970–979, 2006. URL: http://dl.acm.org/citation.cfm?id=1109557.1109665.

40 Anupam Gupta, Robert Krauthgamer, and James R. Lee. Bounded geometries, fractals, and low-distortion embeddings. In *44th Symposium on Foundations of Computer Science (FOCS 2003), 11-14 October 2003, Cambridge, MA, USA, Proceedings*, pages 534–543, 2003. doi:10.1109/SFCS.2003.1238226.

41 Anupam Gupta, Viswanath Nagarajan, and R. Ravi. An improved approximation algorithm for requirement cut. *Oper. Res. Lett.*, 38(4):322–325, 2010. doi:10.1016/j.orl.2010.02.009.

42 Mohammad Taghi Hajiaghayi, Robert D. Kleinberg, and Frank Thomson Leighton. Improved lower and upper bounds for universal TSP in planar metrics. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, Miami, Florida, USA, January 22-26, 2006*, pages 649–658, 2006. URL: http://dl.acm.org/citation.cfm?id=1109557.1109628.

43 Makoto Imase and Bernard M. Waxman. Dynamic steiner tree problem. *SIAM J. Discrete Math.*, 4(3):369–384, 1991. doi:10.1137/0404033.

44 Patrick Jaillet. A priori solution of a traveling salesman problem in which a random subset of the customers are visited. *Operations Research*, 36(6):929–936, 1988. doi:10.1287/opre.36.6.929.

45 Lujun Jia, Guolong Lin, Guevara Noubir, Rajmohan Rajaraman, and Ravi Sundaram. Universal approximations for tsp, steiner tree, and set cover. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, pages 386–395, 2005. doi:10.1145/1060590.1060649.

46 Lior Kamma, Robert Krauthgamer, and Huy L. Nguyen. Cutting corners cheaply, or how to remove steiner points. *SIAM J. Comput.*, 44(4):975–995, 2015. doi:10.1137/140951382.

47 Telikepalli Kavitha and Nithin M. Varma. Small stretch pairwise spanners. In *Proceedings of the 40th International Conference on Automata, Languages, and Programming - Volume Part I*, ICALP'13, pages 601–612, Berlin, Heidelberg, 2013. Springer-Verlag. doi:10.1007/978-3-642-39206-1_51.

48 Philip N. Klein, Serge A. Plotkin, and Satish Rao. Excluded minors, network decomposition, and multicommodity flow. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, pages 682–690, 1993. doi:10.1145/167088.167261.

49 Robert Krauthgamer, James R. Lee, and Havana Rika. Flow-cut gaps and face covers in planar graphs. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 525–534, 2019. doi:10.1137/1.9781611975482.33.

50 Robert Krauthgamer, Huy L. Nguyen, and Tamar Zondiner. Preserving terminal distances using minors. *SIAM J. Discrete Math.*, 28(1):127–141, 2014. doi:10.1137/120888843.

51 Robert Krauthgamer and Inbal Rika. Mimicking networks and succinct representations of terminal cuts. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 1789–1799, 2013. doi:10.1137/1.9781611973105.128.

52 Robert Krauthgamer and Inbal Rika. Refined vertex sparsifiers of planar graphs. *CoRR*, abs/1702.05951, 2017. arXiv:1702.05951.

53 Urs Lang and Thilo Schlichenmaier. Nagata dimension, quasisymmetric embeddings, and lipschitz extensions. *International Mathematics Research Notices*, 2005(58):3625–3655, 2005. doi:10.1155/IMRN.2005.3625.

54 Frank Thomson Leighton and Ankur Moitra. Extensions and limits to vertex sparsification. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 47–56, 2010. doi:10.1145/1806689.1806698.

55 Konstantin Makarychev and Yury Makarychev. Metric extension operators, vertex sparsifiers and lipschitz extendability. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 255–264, 2010. doi:10.1109/FOCS.2010.31.

56 Ankur Moitra. Approximation algorithms for multicommodity-type problems with guarantees independent of the graph size. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA*, pages 3–12, 2009. doi:10.1109/FOCS.2009.28.

57 Loren K. Platzman and John J. Bartholdi, III. Spacefilling curves and the planar travelling salesman problem. *J. ACM*, 36(4):719–737, October 1989. doi:10.1145/76359.76361.

58 Liam Roditty, Mikkel Thorup, and Uri Zwick. Deterministic constructions of approximate distance oracles and spanners. In *Automata, Languages and Programming, 32nd International Colloquium, ICALP 2005, Lisbon, Portugal, July 11-15, 2005, Proceedings*, pages 261–272, 2005. doi:10.1007/11523468_22.

59 Frans Schalekamp and David B. Shmoys. Algorithms for the universal and a priori TSP. *Oper. Res. Lett.*, 36(1):1–3, 2008. doi:10.1016/j.orl.2007.04.009.

Approximate Nearest Neighbor for Curves – Simple, Efficient, and Deterministic

Arnold Filtser
Department of Computer Science, Columbia University, New York, NY, USA
http://www.cs.columbia.edu/~arnoldf/
arnold273@gmail.com

Omrit Filtser
Department of Applied Mathematics and Statistics, Stony Brook University, NY, USA
https://omrit.filtser.com/
omrit.filtser@gmail.com

Matthew J. Katz
Department of Computer Science, Ben-Gurion University of the Negev, Beer Sheva, Israel
matya@cs.bgu.ac.il

Abstract

In the $(1+\varepsilon, r)$-*approximate near-neighbor* problem for curves (ANNC) under some similarity measure δ, the goal is to construct a data structure for a given set $\mathcal{C}$ of curves that supports approximate near-neighbor queries: Given a query curve Q, if there exists a curve $C \in \mathcal{C}$ such that $\delta(Q,C) \le r$, then return a curve $C' \in \mathcal{C}$ with $\delta(Q,C') \le (1+\varepsilon)r$. There exists an efficient reduction from the $(1+\varepsilon)$-*approximate nearest-neighbor* problem to ANNC, where in the former problem the answer to a query is a curve $C \in \mathcal{C}$ with $\delta(Q,C) \le (1+\varepsilon)\cdot\delta(Q,C^*)$, where C^* is the curve of $\mathcal{C}$ most similar to Q.

Given a set $\mathcal{C}$ of n curves, each consisting of m points in d dimensions, we construct a data structure for ANNC that uses $n \cdot O(\frac{1}{\varepsilon})^{md}$ storage space and has $O(md)$ query time (for a query curve of length m), where the similarity measure between two curves is their discrete Fréchet or dynamic time warping distance. Our method is simple to implement, deterministic, and results in an exponential improvement in both query time and storage space compared to all previous bounds.

Further, we also consider the *asymmetric* version of ANNC, where the length of the query curves is $k \ll m$, and obtain essentially the same storage and query bounds as above, except that m is replaced by k. Finally, we apply our method to a version of approximate range counting for curves and achieve similar bounds.

2012 ACM Subject Classification Theory of computation → Computational geometry; Theory of computation → Design and analysis of algorithms

Keywords and phrases polygonal curves, Fréchet distance, dynamic time warping, approximation algorithms, (asymmetric) approximate nearest neighbor, range counting

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.48

Category Track A: Algorithms, Complexity and Games

Related Version https://arxiv.org/abs/1902.07562

Funding *Arnold Filtser*: Supported by the Simons Foundation.
Omrit Filtser: Supported by the Eric and Wendy Schmidt Fund for Strategic Innovation, by the Council for Higher Education of Israel, and by Ben-Gurion University of the Negev.
Matthew J. Katz: Supported by grant 1884/16 from the Israel Science Foundation.

Acknowledgements We wish to thank Boris Aronov for helpful discussions on the problems studied in this paper.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 48; pp. 48:1–48:19
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Nearest neighbor search is a fundamental and well-studied problem that has various applications in machine learning, data analysis, and classification. This important task also arises in applications where the recorded instances are trajectories or polygonal curves modeling, for example, epigenetic and surgical processes, market value fluctuations, population growth, the number of the requests per hour received at some web-page, and even the response of a football player in a given situation.

Let $\mathcal{C}$ be a set of n curves, each consisting of at most m points in d dimensions, and let δ be some distance measure for curves. In the *nearest-neighbor* problem for curves, the goal is to construct a data structure for $\mathcal{C}$ that supports nearest-neighbor queries, that is, given a query curve Q of length at most m, return the curve $C^* \in \mathcal{C}$ closest to Q (according to δ). The approximation version of this problem is the $(1+\varepsilon)$*-approximate nearest-neighbor* problem, where the answer to a query Q is a curve $C \in \mathcal{C}$ with $\delta(Q,C) \leq (1+\varepsilon)\delta(Q,C^*)$. We study a decision version of this approximation problem, which is called the $(1+\varepsilon,r)$*-approximate near-neighbor* problem for curves (ANNC). Here, if there exists a curve in $\mathcal{C}$ that lies within distance r of the query curve Q, one has to return a curve in $\mathcal{C}$ that lies within distance $(1+\varepsilon)r$ of Q. Note that there exists a reduction from the $(1+\varepsilon)$-approximate nearest-neighbor problem to the $(1+\varepsilon,r)$-approximate near-neighbor problem [14, 22, 13], at the cost of an additional logarithmic factor in the query time and an $O(\log^2 n)$ factor in the storage space.

In practice, it is often the case that the query curves are significantly shorter than the input curves (e.g., Google-search queries). Thus, we also study the *asymmetric setting* of $(1+\varepsilon,r)$-ANNC, where each of the input curves has complexity at most m, while each query curve has complexity at most $k \ll m$.

There are many methods that are used in real-world applications for comparing curves, and one of the most prevalent is the (discrete) *Fréchet* distance (DFD for short), which is often described by the following analogy. Two frogs are hopping from vertex to vertex along two polygonal curves. At each step, one of the frogs or both frogs may advance to the next vertex on its curve. The discrete Fréchet distance is defined as the smallest maximum distance between the frogs that can be achieved in such a joint sequence of hops. Another useful distance measure for curves or time series is the *dynamic time warping* distance (DTW for short), in which instead of taking the smallest maximum distance we take the smallest sum of distances.

In the last several years, a series of papers have been written investigating the approximate near-neighbor problem for curves (ANNC) and its variants under the Fréchet distance [15, 6, 10, 7, 1, 12, 2] (see Table 1), and several different approaches and sophisticated methods were utilized in order to provide efficient data structures. Up to now, all data structures for ANNC under DFD have either an exponential in m query time, or an infeasible storage space bound. In this paper, for the first time, we manage to remove the exponential factor from the query time, while also significantly reducing the space consumption. Our approach consists of a discretization of space based on the input curves, which allows us to prepare a small set of curves that captures all possible queries approximately.

Indyk [15] was the first to give a deterministic near-neighbor data structure for curves under DFD. The data structure achieves an approximation factor of $O((\log m + \log\log n)^{t-1})$ given some trade-off parameter $t > 1$. Its space consumption is very high, $O(m^2|X|)^{tm^{1/t}} \cdot n^{2t}$, where $|X|$ is the size of the domain on which the curves are defined, and the query time is $(m \log n)^{O(t)}$. In Table 1 we set $t = 1 + o(1)$ to obtain a constant approximation factor.

Later, Driemel and Silvestri [10] presented a locality-sensitive-hashing scheme for curves under DFD, improving the result of Indyk for short curves. Their data structure uses $O(2^{4md} n \log n)$ space and answers queries in $O(2^{4md} \log n)$ time with an approximation factor of $O(d^{3/2})$. They also provide a trade-off between approximation quality and computational performance: for $d = O(1)$, and given a parameter $k \in [m]$, a data structure of size $O(2^{2k} m^{k-1} n \log n + mn)$ is constructed that answers queries in $O(2^{2k} m^k \log n)$ time with an approximation factor of $O(m/k)$. They also show that this result can be applied to DTW, but only for the extreme of the trade-off which gives an $O(m)$ approximation.

Recently, Emiris and Psarros [12] presented near-neighbor data structures for curves under both DFD and DTW. Their algorithm provides an approximation factor of $(1+\varepsilon)$, at the expense of increased space usage and preprocessing time. They use the idea that for a fixed alignment between two curves (i.e., a given sequence of hops of the two frogs), the problem can be reduced to the near-neighbor problem for points in ℓ_∞ product of ℓ_2 spaces. Their basic idea is to construct a data structure for every possible alignment. Once a query is given, they query all these data structures and return the closest curve found. This approach is responsible for the 2^m factor in their query time. Furthermore, they generalize this approach using randomized projections of ℓ_p-products of Euclidean metrics (for any $p \geq 1$), and define the $\ell_{p,2}$-distance for curves (for $p \geq 1$), which is exactly DFD when $p = \infty$, and DTW when $p = 1$ (see Section 2). The space used by their data structure is $\tilde{O}(n) \cdot (2 + \frac{d}{\log m})^{O(m^{1+1/\varepsilon} \cdot d \log(1/\varepsilon))}$ with query $\tilde{O}(d m^{1+1/\varepsilon} \cdot 2^{4m} \log n)$ for DFD and $\tilde{O}(n) \cdot \frac{1}{\varepsilon}^{O(md)}$ space and $\tilde{O}(d \cdot 2^{4m} \log n)$ query for DTW.

Subsequent work. In a recent manuscript, Driemel, Psarros, and Schmidt [9], study the asymmetric setting of $(1+\varepsilon, r)$-ANNC under DFD. They follow our approach of preparing in advance the answers to all relevant queries on a discretization of the space, to construct a randomized data structure with space in $n \cdot O\left(\frac{kd^{3/2}}{\varepsilon}\right)^{dk}$ and query time in $O(dk)$. They also show how to derandomize their data structure, at the cost of increasing the space to $d^{3/2} n k \varepsilon^{-1} \cdot O\left(\frac{kd^{3/2}}{\varepsilon}\right)^{dk}$, and the query time to $O(d^{5/2} k^2 \varepsilon^{-1} (\log n + kd \log \frac{kd}{\varepsilon}))$. This provides additional evidence that our approach to ANNC, although quite simple and easy to implement, seems to produce more efficient data structures than those obtained using tools such as LSH and randomized projections. Moreover, in this version of our manuscript we show how to improve upon the results in [9] for the asymmetric setting.

Our results. We present a data structure for the $(1+\varepsilon, r)$-approximate near-neighbor problem using a bucketing method. We construct a relatively small set of curves $\mathcal{I}$ such that given a query curve Q, if there exists some curve in $\mathcal{C}$ within distance r of Q, then one of the curves in $\mathcal{I}$ must be very close to Q. The points of the curves in $\mathcal{I}$ are chosen from a simple discretization of space, thus, while it is not surprising that we get the best query time, it is surprising that we achieve a better space bound. Moreover, while the analysis of the space bounds is rather involved, the implementation of our data structures remain simple in practice.

See Table 1 for a summary of our results. In the table, we do not state our result for the general $\ell_{p,2}$-distance. Instead, we state our results for the two most important cases, i.e. DFD and DTW, and compare them with previous work. Note that our results substantially improve the current state of the art for any $p \geq 1$. In particular, we remove the exponential dependence on m in the query bounds and significantly improve the space bounds.

Our results for the asymmetric setting, where the query curve Q has complexity $k \ll m$, are summarized in Table 2. We show that in the asymmetric setting for DFD, our data structure can be slightly modified in order to achieve query time and storage space independent of m. Moreover, the storage space and query time matches those of the symmetric setting, by replacing m with k.

We also apply our methods to an approximation version of range counting for curves (for the general $\ell_{p,2}$ distance) and achieve bounds similar to those of our ANNC data structure. Moreover, at the cost of an additional $O(n)$-factor in the space bound, we can also answer the corresponding approximation version of range searching, thus answering a question of Afshani and Driemel [1], with respect to DFD.

We note that our approach with obvious modifications works also in a dynamic setting, that is, we can construct an efficient dynamic data structure for ANNC as well as for other related problems such as range counting and range reporting for curves.

Another significant advantage of our approach is that, unlike some of the previous solutions, our data structure always returns an answer, and never returns a curve at distance greater than $(1+\varepsilon)r$ from the query curve, i.e., there are no false positives. This is an important property of our solution, due to the fact that verifying the validity of the answer (i.e., computing the distance between two curves) cannot be done in strongly subquadratic time (assuming SETH, see [4]), which is already more than our query time (for $d < m$).

Table 1 Our approximate near-neighbor data structure under DFD and DTW compared to the previous results.

	Space	Query	Approx.	Comments
DFD	$O(m^2\|X\|)^{m^{1-o(1)}} \cdot n^{2+o(1)}$	$(m \log n)^{O(1)}$	$O(1)$	deterministic, [15]
	$O(2^{4md} n \log n)$	$O(2^{4md} \log n)$	$O(d^{3/2})$	randomized, using LSH [10]
	$\tilde{O}(n) \cdot (2 + \frac{d}{\log m})^{O(m^{1+1/\varepsilon} \cdot d \log(\frac{1}{\varepsilon}))}$	$\tilde{O}(dm^{1+1/\varepsilon} \cdot 2^{4m} \log n)$	$1+\varepsilon$	randomized, [12]
	$n \cdot O(\frac{1}{\varepsilon})^{md}$	$O(md)$	$1+\varepsilon$	deter. (rand. construction), Theorem 9
DTW	$O(n \log n + mn)$	$O(m \log n)$	$O(m)$	randomized, using LSH, $d = O(1)$, [10]
	$\tilde{O}(n) \cdot \frac{1}{\varepsilon}^{O(md)}$	$\tilde{O}(d \cdot 2^{4m} \log n)$	$1+\varepsilon$	randomized, [12]
	$n \cdot O(\frac{1}{\varepsilon})^{m(d+1)}$	$O(md)$	$1+\varepsilon$	deter. (rand. construction), Theorem 15

More related work. De Berg, Gudmundsson, and Mehrabi [7] described a dynamic data structure for approximate nearest neighbor for curves (which can also be used for other types of queries such as range reporting), under the (continuous) Fréchet distance. Their data structure uses $n \cdot O\left(\frac{1}{\varepsilon}\right)^{2m}$ space and has $O(m)$ query time, but with an additive error of $\varepsilon \cdot \mathit{reach}(Q)$, where $\mathit{reach}(Q)$ is the maximum distance between the start vertex of the query curve Q and any other vertex of Q. Furthermore, their query procedure might fail when the distance to the nearest neighbor is relatively large.

Afshani and Driemel [1] studied (exact) range searching under both the discrete and continuous Fréchet distance. In this problem, the goal is to preprocess $\mathcal{C}$ such that given a query curve Q of length m_q and a radius r, all the curves in $\mathcal{C}$ that are within distance r from

Table 2 Summary of previous and current results for the asymmetric approximate near-neighbor data structure for curves. All the results in the table are w.r.t. DFD. The approximation ratio is $1+\varepsilon$ for $\varepsilon \in (0,1)$, and our data structures always succeed. Historic note: [9] is a subsequent work to the first version of this paper arXiv:1902.07562. In this second version we also apply our counting techniques to the asymmetric cases.

Space	Query	Deterministic construction?	Reference
$n \cdot \left(O(\frac{kd^{3/2}}{\varepsilon})^{kd}\right)$	$O(kd)$	no	[9]
$n \cdot \left(O(\frac{kd^{3/2}}{\varepsilon})^{kd+1}\right)$	$O(\frac{k^2d^{5/2}}{\varepsilon}(\log n + kd\log(\frac{kd}{\varepsilon})))$	yes	[9]
$n \cdot O(\frac{1}{\varepsilon})^{kd}$	$O(kd)$	no	Theorem 11
$n \cdot O(\frac{1}{\varepsilon})^{kd}$	$O(kd\log(\frac{nkd}{\varepsilon}))$	yes	Theorem 18

Q can be found efficiently. For DFD, their data structure uses $O(n(\log\log n)^{m-1})$ space and has $O(n^{1-\frac{1}{d}} \cdot \log^{O(m)} n \cdot m_q^{O(d)})$ query time, where m_q is limited to $\log^{O(1)} n$. Additionally, they provide a lower bound in the pointer model, stating that every data structure with $Q(n)+O(k)$ query time, where k is the output size, has to use roughly $\Omega\left((n/Q(n))^2\right)$ space in the worst case (even for $m_q = 1$). Afshani and Driemel conclude their paper by asking whether more efficient data structures might be constructed if one allows approximation.

De Berg, Cook IV, and Gudmundsson [6], considered the following range counting problem under the continuous Fréchet distance. Given a polygonal curve C with m vertices, they show how to preprocess it into a data structure of size $O(k \cdot \text{polylog(m)})$, so that, given a query segment s, one can return a constant approximation of the number of subcurves of C that lie within distance r of s in $O(\frac{m}{\sqrt{k}} \cdot \text{polylog(m)})$ time, where k is a parameter between m and m^2.

Aronov et al. [2] managed to obtain practical bounds for two cases of the asymmetric $(1+\varepsilon, r)$-ANNC under DFD: (i) when Q is a line segment (i.e., $k=2$), or (ii) when $\mathcal{C}$ consists of line segments (i.e., $m=2$). The bounds on the size of the data structure and query time are nearly linear in the size of the input and query curve, respectively. Specifically, for the case where $k=2$, they achieve query time $O(\log^4(\frac{n}{\varepsilon}))$ and storage space $O(n\frac{1}{\varepsilon^4}\log^4(\frac{n}{\varepsilon}))$. They also provide efficient data structures for several other variants of the problem: the (exact) NNC where ℓ_∞ is used for interpoint distances, and the case where the location of the input curves is only fixed up to translation.

1.1 Technical ideas

We use a discretization of the space, by laying a d-dimensional uniform grid with edge length $\frac{\varepsilon r}{\sqrt{d}}$. The main ingredient in our data structure is then a relatively small set $\mathcal{I}$ of curves defined by grid points, which represents all possible queries. For each curve in $\mathcal{I}$ we store an index of a close enough curve from the input set $\mathcal{C}$. Given a query Q sufficiently close to some curve in $\mathcal{C}$, we find a representative Q' in $\mathcal{I}$ by simply rounding Q's vertices and return the index of the curve stored for Q'.

Given a point $x \in \mathbb{R}^d$, the number of grid points that are within distance $(1+\varepsilon)r$ from x is bounded by $O(\frac{1}{\varepsilon})^d$ (Corollary 7). Thus, given a curve C of length m, the total number of grid points that are within distance $(1+\varepsilon)r$ from one of its vertices is $m \cdot O(\frac{1}{\varepsilon})^d$. Naively, the number of curves needed to represent all possible queries of length m within distance

r of C is bounded by the number of ways to choose m points with repetitions from a set of grid points of size $m \cdot O(\frac{1}{\varepsilon})^d$, which is bounded by $m^m \cdot O(\frac{1}{\varepsilon})^{md}$. This infeasible bound on the storage space might be the reason why more sophisticated solutions for ANNC have been suggested throughout the years.

One of the main technical contributions of this paper is an analysis leading to a significantly better bound, if we store only candidate curves that are within distance $(1+\varepsilon)r$ from C. Actually, in Section 3 we show that for the case of DFD, it is sufficient to store a set of representative curves of size only $O(\frac{1}{\varepsilon})^{md}$ for each input curve. The basic idea is to bound the number of representatives that can be obtained by some fixed alignment between C and the candidate curve (see Claim 8).

For the general case of $\ell_{p,2}$-distance (including DTW), we are minimizing the sum of distances instead of the maximum distance (as in DFD). Thus, we have to use a more dense grid (with edge length $\frac{\varepsilon r}{(2m)^{1/p}\sqrt{d}}$), and the situation becomes more complicated. First, unlike DFD, the triangle inequality does not hold for $\ell_{p,2}$-distance in general (including DTW). Second, since DFD is a min-max measure, the choice of different vertices for a representative curve is "independent" in a sense, whereas for $\ell_{p,2}$-distance in general, the choice of different vertices depends on their sum of distances from the input curve. Using more careful counting arguments and analysis of the alignment between two curves, we are able to show that in this case the number of representative curves that our data structure has to store per input curve is bounded by $O(\frac{1}{\varepsilon})^{m(d+1)}$ (see Claim 13).

To store the set $\mathcal{I}$ we simply use a dictionary, which can be implemented using a hash table and guarantees a query time linear in the size of the query. To obtain a fully deterministic solution, one can use a search tree instead. However, a naive implementation using a binary search tree results in an additional factor of $O(\log|\mathcal{I}|) = O(md\log(\frac{n}{\varepsilon}))$ to the query time, i.e., in a query time of $O(m^2d^2\log(\frac{n}{\varepsilon}))$. We show how to implement the dictionary using a prefix tree, exploiting the fact that the vertices of the curves in $\mathcal{I}$ are from a relatively small set of grid points, which improves the query time to $O(md\log(\frac{nmd}{\varepsilon}))$.

For the asymmetric setting (where the length of a query is $k \ll m$), we use simplifications of the input curves in order to obtain bounds that are independent of m. Given a curve C of length m, a simplification Π of C is a curve of length $k \ll m$ that is relatively close to C. Simplifications were used in order to provide approximate solutions in several asymmetric versions of problems on curves, such as clustering [5], and distance oracles [8, 9].

By the triangle inequality for DFD, every query curve Q within distance r from an input curve C is at distance at most $2r$ from the simplification Π (where Π is within distance r from C). Thus, it is enough to prepare for query curves at distance at most $2r$ from Π, which follows from previous arguments. Note that the query time and storage space are independent of m.

2 Preliminaries

To simplify the presentation, we assume throughout the paper that all the input curves have exactly the same size, m, and all the query curves have exactly the same size, either m or k, depending on whether we are considering the standard or the asymmetric version. This assumption can be easily removed (see Remark 16 at the end of Section 5).

Let $\mathcal{C}$ be a set of n curves, each consisting of m points in d dimensions, and let δ be some distance measure for curves.

▶ **Problem 1** ($(1+\varepsilon)$-approximate nearest-neighbor for curves). *Given a parameter $0 < \varepsilon \leq 1$, preprocess $\mathcal{C}$ into a data structure that given a query curve Q, returns a curve $C' \in \mathcal{C}$, such that $\delta(Q, C') \leq (1+\varepsilon) \cdot \delta(Q, C)$, where C is the curve in $\mathcal{C}$ closest to Q.*

▶ **Problem 2** ($(1+\varepsilon, r)$-approximate near-neighbor for curves). *Given a parameter r and $0 < \varepsilon \leq 1$, preprocess $\mathcal{C}$ into a data structure that given a query curve Q, if there exists a curve $C_i \in \mathcal{C}$ such that $\delta(Q, C_i) \leq r$, returns a curve $C_j \in \mathcal{C}$ such that $\delta(Q, C_j) \leq (1+\varepsilon)r$.*

▶ **Problem 3** (Asymmetric $(1+\varepsilon, r)$-approximate near-neighbor for curves). *Given parameters r,k, and $0 < \varepsilon \leq 1$, preprocess $\mathcal{C}$ into a data structure that given a query curve Q of length k, if there exists a curve $C_i \in \mathcal{C}$ such that $\delta(Q, C_i) \leq r$, returns a curve $C_j \in \mathcal{C}$ such that $\delta(Q, C_j) \leq (1+\varepsilon)r$.*

Curve alignment. Given two integers m_1, m_2, let $\tau := \langle (i_1, j_1), \ldots, (i_t, j_t) \rangle$ be a sequence of pairs where $i_1 = j_1 = 1$, $i_t = m_1$,$j_t = m_2$, and for each $1 < k \leq t$, one of the following conditions holds:

(i) $i_k = i_{k-1} + 1$ and $j_k = j_{k-1}$,
(ii) $i_k = i_{k-1}$ and $j_k = j_{k-1} + 1$, or
(iii) $i_k = i_{k-1} + 1$ and $j_k = j_{k-1} + 1$.

We call such a sequence τ an *alignment* of two curves.

Let $P = (p_1, \ldots, p_{m_1})$ and $Q = (q_1, \ldots, q_{m_2})$ be two curves of lengths m_1 and m_2, respectively, in d dimensions. We say that an alignment τ w.r.t. P and Q matches p_i and p_j if $(i, j) \in \tau$.

Discrete Fréchet distance (DFD). The *Fréchet cost* of an alignment τ w.r.t. P and Q is $\sigma_{dF}(\tau(P, Q)) := \max_{(i,j)\in\tau} \|p_i - q_j\|_2$. The discrete Fréchet distance is defined over the set $\mathcal{T}$ of all alignments as

$$d_{dF}(P, Q) = \min_{\tau\in\mathcal{T}} \sigma_{dF}(\tau(P, Q)).$$

Dynamic time wrapping (DTW). The *time warping cost* of an alignment τ w.r.t. P and Q is $\sigma_{DTW}(\tau(P, Q)) := \sum_{(i,j)\in\tau} \|p_i - q_j\|_2$. The DTW distance is defined over the set $\mathcal{T}$ of all alignments as

$$d_{DTW}(P, Q) = \min_{\tau\in\mathcal{T}} \sigma_{DTW}(\tau(P, Q)).$$

$\ell_{p,2}$-distance for curves. The *$\ell_{p,2}$-cost* of an alignment τ w.r.t. P and Q is $\sigma_{p,2}(\tau(P, Q)) := \left(\sum_{(i,j)\in\tau} \|p_i - q_j\|_2^p\right)^{1/p}$. The $\ell_{p,2}$-distance between P and Q is defined over the set $\mathcal{T}$ of all alignments as

$$d_{p,2}(P, Q) = \min_{\tau\in\mathcal{T}} \sigma_{p,2}(\tau(P, Q)).$$

Notice that $\ell_{p,2}$-distance is a generalization of DFD and DTW, in the sense that $\sigma_{dF} = \sigma_{\infty,2}$ and $d_{dF} = d_{\infty,2}$, $\sigma_{DTW} = \sigma_{1,2}$ and $d_{DTW} = d_{1,2}$. Also note that DFD satisfies the triangle inequality, but DTW and $\ell_{p,2}$-distance (for $p \neq \infty$) do not (see Section 5 for details).

Emiris and Psarros [12] showed that the number of all possible alignments of two curves is in $O(m \cdot 2^{2m})$. We reduce this bound by counting only alignments that can determine the $\ell_{p,2}$-distance between two curves.[1] More formally, let τ be an alignment. If there

[1] Since our storage space is already in $O(\frac{1}{\varepsilon})^{md}$, and $m \cdot 2^{2m} \leq 3^{2m}$ is in $O(1)^{md}$, we could have used this larger upper bound. However, in Lemma 4 we show a tight upper bound on the number of relevant alignments, which may be useful for other applications.

exists an alignment τ' such that $\tau' \subset \tau$, then clearly $\sigma_{p,2}(\tau'(P,Q)) \leq \sigma_{p,2}(\tau(P,Q))$, for any $1 \leq p \leq \infty$ and for any two curves P and Q. In this case, we say that τ cannot determine the $\ell_{p,2}$-distance between two curves.

▶ **Lemma 4.** *The number of different alignments that can determine the $\ell_{p,2}$-distance between two m-curves (for any $1 \leq p \leq \infty$) is at most $O(\frac{2^{2m}}{\sqrt{m}})$.*

Proof. Let $\tau = \langle (i_1, j_1), \ldots, (i_t, j_t) \rangle$ be an alignment. Notice that $m \leq t \leq 2m-1$. By definition, τ has 3 types of (consecutive) subsequences of length two:

(i) $\langle (i_k, j_k), (i_k+1, j_k) \rangle$,
(ii) $\langle (i_k, j_k), (i_k, j_k+1) \rangle$, and
(iii) $\langle (i_k, j_k), (i_k+1, j_k+1) \rangle$.

Denote by $\mathcal{T}_1$ the set of all alignments that do not contain any subsequence of type (iii). Then, any $\tau_1 \in \mathcal{T}_1$ is of length exactly $2m-1$. Moreover, τ_1 contains exactly $2m-2$ subsequences of length two, of which $m-1$ are of type (i) and $m-1$ are of type (ii). Therefore, $|\mathcal{T}_1| = \binom{2m-2}{m-1} = O(\frac{2^{2m}}{\sqrt{m}})$.

Assume that an alignment τ contains a subsequence of the form $(i_k, j_k-1), (i_k, j_k), (i_k+1, j_k)$, for some $1 < k \leq t-1$. Notice that removing the pair (i_k, j_k) from τ results in a legal alignment τ', such that $\sigma_{p,2}(\tau'(P,Q)) \leq \sigma_{p,2}(\tau(P,Q))$, for any $1 \leq p \leq \infty$ and two curves P, Q. We call the pair (i_k, j_k) a *redundant pair.* Similarly, if τ contains a subsequence of the form $(i_k-1, j_k), (i_k, j_k), (i_k, j_k+1)$, for some $1 < k \leq t-1$, then the pair (i_k, j_k) is also a redundant pair. Therefore we only care about alignments that do not contain any redundant pairs. Denote by $\mathcal{T}_2$ the set of all alignments that do not contain any redundant pairs, then any $\tau_2 \in \mathcal{T}_2$ contains at least one subsequence of type (iii).

We claim that for any alignment $\tau_2 \in \mathcal{T}_2$, there exists a unique alignment $\tau_1 \in \mathcal{T}_1$. Indeed, if we add the redundant pair (i_l, j_l+1) between (i_l, j_l) and (i_l+1, j_l+1) for each subsequence of type (iii) in τ_2, we obtain an alignment $\tau_1 \in \mathcal{T}_1$. Moreover, since τ_2 does not contain any redundant pairs, the reverse operation on τ_1 results in τ_2. Thus we obtain $|\mathcal{T}_2| \leq |\mathcal{T}_1| = O(\frac{2^{2m}}{\sqrt{m}})$. ◀

Points and balls. Given a point $x \in \mathbb{R}^d$ and a real number $R > 0$, we denote by $B_p^d(x, R)$ the d-dimensional ball under the ℓ_p norm with center x and radius R, i.e., a point $y \in \mathbb{R}^d$ is in $B_p^d(x, R)$ if and only if $\|x - y\|_p \leq R$, where $\|x - y\|_p = \left(\sum_{i=1}^d |x_i - y_i|^p\right)^{1/p}$. Let $B_p^d(R) = B_p^d(\mathbf{0}, R)$, and let $V_p^d(R)$ be the volume (w.r.t. Lebesgue measure) of $B_p^d(R)$, then

$$V_p^d(R) = \frac{2^d \Gamma(1+1/p)^d}{\Gamma(1+d/p)} R^d,$$

where $\Gamma(\cdot)$ is Euler's Gamma function (an extension of the factorial function). For $p = 2$ and $p = 1$, we get

$$V_2^d(R) = \frac{\pi^{d/2}}{\Gamma(1+d/2)} R^d \quad \text{and} \quad V_1^d(R) = \frac{2^d}{d!} R^d.$$

Our approach consists of a discretization of the space using lattice points, i.e., points from $\mathbb{Z}^d$.

▶ **Lemma 5.** *The number of lattice points in the d-dimensional ball of radius R under the ℓ_p norm (i.e., in $B_p^d(R)$) is bounded by $V_p^d(R + d^{1/p})$.*

Proof. With each lattice point $z = (z_1, z_2, \ldots, z_d)$, $z_i \in \mathbb{Z}$, we match the d-dimensional lattice cube $C(z) = [z_1, z_1 + 1] \times [z_2, z_2 + 1] \times \cdots \times [z_d, z_d + 1]$. Notice that $z \in C(z)$, and the ℓ_p-diameter of a lattice cube is $d^{1/p}$. Therefore, the number of lattice points in the ℓ_p^d-ball of radius R is bounded by the number of lattice cubes that are contained in a ℓ_p^d-ball with radius $R + d^{1/p}$. This number is bounded by $V_p^d(R + d^{1/p})$ divided by the volume of a lattice cube, which is $1^d = 1$. ◀

▶ Remark 6. In general, in all our data structures we do not assume any bound on the dimension d. However, using dimension reduction techniques, we may assume that $d \le O(\frac{\log(nm)}{\varepsilon^2})$. See Section 9 for details.

3 Discrete Fréchet distance (DFD)

Consider the infinite d-dimensional grid with edge length $\frac{\varepsilon r}{\sqrt{d}}$. Given a point x in $\mathbb{R}^d$, by rounding one can find in $O(d)$ time the grid point x' closest to x, and $\|x - x'\|_2 \le \frac{\varepsilon r}{2}$. Let $G(x, R)$ denote the set of grid points that are contained in $B_2^d(x, R)$.

▶ **Corollary 7.** $|G(x, (1 + \varepsilon)r)| = O(\frac{1}{\varepsilon})^d$.

Proof. We scale our grid so that the edge length is 1, hence we are looking for the number of lattice points in $B_2^d(x, \frac{1+\varepsilon}{\varepsilon}\sqrt{d})$. By Lemma 5 we get that this number is bounded by the volume of the d-dimensional ball of radius $\frac{1+\varepsilon}{\varepsilon}\sqrt{d} + \sqrt{d} \le \frac{3\sqrt{d}}{\varepsilon}$. Using Stirling's formula we conclude that

$$V_2^d\left(\frac{3\sqrt{d}}{\varepsilon}\right) = \frac{\pi^{\frac{d}{2}}}{\Gamma(\frac{d}{2}+1)} \cdot \left(\frac{3\sqrt{d}}{\varepsilon}\right)^d \le \left(\frac{\alpha}{\varepsilon}\right)^d ,$$

where α is a constant. For example, if d is even, then

$$V_2^d\left(\frac{3\sqrt{d}}{\varepsilon}\right) = \frac{\pi^{\frac{d}{2}}}{(\frac{d}{2})!} \cdot \left(\frac{3\sqrt{d}}{\varepsilon}\right)^d \le \frac{\pi^{\frac{d}{2}}}{\sqrt{2\pi}(d/2)^{d/2+1/2}e^{-d/2}} \cdot \left(\frac{3\sqrt{d}}{\varepsilon}\right)^d \le \left(\frac{12.4}{\varepsilon}\right)^d = O\left(\frac{1}{\varepsilon}\right)^d .$$

◀

Denote by p_j^i the j'th point of C_i, and let $G_i = \bigcup_{1 \le j \le m} G(p_j^i, (1+\varepsilon)r)$ and $\mathcal{G} = \bigcup_{1 \le i \le n} G_i$, then by the above corollary we have $|G_i| = m \cdot O(\frac{1}{\varepsilon})^d$ and $|\mathcal{G}| = mn \cdot O(\frac{1}{\varepsilon})^d$. Let $\mathcal{I}_i$ be the set of all curves $\overline{Q} = (x_1, x_2, \ldots, x_m)$ with points from G_i, such that $d_{dF}(C_i, \overline{Q}) \le (1 + \frac{\varepsilon}{2})r$.

▷ Claim 8. $|\mathcal{I}_i| = O(\frac{1}{\varepsilon})^{md}$ and it can be computed in $O(\frac{1}{\varepsilon})^{md}$ time.

Proof. Let $\overline{Q} \in \mathcal{I}_i$ and let τ be an alignment with $\sigma_{dF}(\tau(C_i, \overline{Q})) \le (1 + \frac{\varepsilon}{2})r$. For each $1 \le k \le m$ let j_k be the smallest index such that $(j_k, k) \in \tau$. In other words, j_k is the smallest index that is matched to k by the alignment τ. Since $d_{dF}(C_i, \overline{Q}) \le (1 + \frac{\varepsilon}{2})r$, we have $x_k \in B_2^d(p_{j_k}^i, (1 + \frac{\varepsilon}{2})r)$, for $k = 1, \ldots, m$. This means that for any curve $\overline{Q} \in \mathcal{I}_i$ such that $\sigma_{dF}(\tau(C_i, \overline{Q})) \le (1 + \frac{\varepsilon}{2})r$, we have $x_k \in G(p_{j_k}^i, (1 + \frac{\varepsilon}{2})r)$, for $k = 1, \ldots, m$. By Corollary 7, the number of ways to choose a grid point x_k from $G(p_{j_k}^i, (1 + \frac{\varepsilon}{2})r)$ is bounded by $O(\frac{1}{\varepsilon})^d$.

We conclude that given an alignment τ, the number of curves $\overline{Q}$ with m points from G_i such that $\sigma_{dF}(\tau(C_i, \overline{Q})) \le (1 + \frac{\varepsilon}{2})r$ is bounded by $O(\frac{1}{\varepsilon})^{md}$. Finally, by Lemma 4, the total number of curves in $\mathcal{I}_i$ is bounded by $2^{2m} \cdot O(\frac{1}{\varepsilon})^{md} = O(\frac{1}{\varepsilon})^{md}$.

To construct $\mathcal{I}_i$ we compute, for each of the $O(\frac{1}{\varepsilon})^{md}$ candidates, its discrete Fréchet distance to C_i. Thus, we construct $\mathcal{I}_i$ in total time $O(\frac{1}{\varepsilon})^{md} \cdot O(m^2) = O(\frac{1}{\varepsilon})^{md}$. (The latter equality is true, since clearly $(\frac{\alpha}{\varepsilon})^{md} \cdot O(m^2) \le (\frac{c\alpha}{\varepsilon})^{md}$, i.e., $O(m^2) \le c^{md}$, where α is the constant from Corollary 7 and $c > 1$ is a sufficiently large constant.) ◁

The data structure. Denote $\mathcal{I} = \bigcup_{1\leq i\leq n} \mathcal{I}_i$, so $|\mathcal{I}| \leq n \cdot O(\frac{1}{\varepsilon})^{md}$ and we construct $\mathcal{I}$ in total time $n \cdot O(\frac{1}{\varepsilon})^{md}$. Next, we would like to store the set $\mathcal{I}$ in a dictionary (a hash table or a lookup table) $\mathcal{D}$, such that given a query curve Q, one can find Q in $\mathcal{D}$ (if it exists) in $O(md)$ time. We use Cuckoo Hashing [21] to construct a (dynamic) dictionary of linear space, constant worst-case query and deletion time, and constant expected amortized insertion time. We insert the curves of $\mathcal{I}$ into the dictionary $\mathcal{D}$ as follows. For each $1 \leq i \leq n$ and curve $\overline{Q} \in \mathcal{I}_i$, if $\overline{Q} \notin \mathcal{D}$, insert $\overline{Q}$ into $\mathcal{D}$, and set $C(\overline{Q}) \leftarrow C_i$. The storage space required for $\mathcal{D}$ is $O(|\mathcal{I}|)$, and to construct it we perform $|\mathcal{I}|$ insertions and look-up operations which take in total $O(|\mathcal{I}| \cdot md) = O(|\mathcal{I}|)$ expected time.

The query algorithm. Let $Q = (q_1, \ldots, q_m)$ be the query curve. The query algorithm is as follows: For each $1 \leq k \leq m$ find the grid point q'_k (not necessarily from $\mathcal{G}$) closest to q_k. This can be done in $O(md)$ time by rounding. Then, search for the curve $Q' = (q'_1, \ldots, q'_m)$ in the dictionary $\mathcal{D}$. If Q' is in $\mathcal{D}$, return $C(Q')$, otherwise, return NO. The total query time is then $O(md)$.

Correctness. Consider a query curve $Q = (q_1, \ldots, q_m)$. Assume that there exists a curve $C_i \in \mathcal{C}$ such that $d_{dF}(C_i, Q) \leq r$. We show that the query algorithm returns a curve C^* with $d_{dF}(C^*, Q) \leq (1+\varepsilon)r$.

Consider a point $q_k \in Q$. Denote by $q'_k \in \mathcal{G}$ the grid point closest to q_k, and let $Q' = (q'_1, \ldots, q'_m)$. We have $\|q_k - q'_k\|_2 \leq \frac{\varepsilon r}{2}$, so $d_{dF}(Q, Q') \leq \frac{\varepsilon r}{2}$. By the triangle inequality,

$$d_{dF}(C_i, Q') \leq d_{dF}(C_i, Q) + d_{dF}(Q, Q') \leq r + \frac{\varepsilon r}{2} = (1 + \frac{\varepsilon}{2})r,$$

so Q' is in $\mathcal{I}_i \subseteq \mathcal{I}$. This means that $\mathcal{D}$ contains Q' with a curve $C(Q') \in \mathcal{C}$ such that $d_{dF}(C(Q'), Q') \leq (1 + \frac{\varepsilon}{2})r$, and the query algorithm returns $C(Q')$. Now, again by the triangle inequality,

$$d_{dF}(C(Q'), Q) \leq d_{dF}(C(Q'), Q') + d_{dF}(Q', Q) \leq (1 + \frac{\varepsilon}{2})r + \frac{\varepsilon r}{2} = (1+\varepsilon)r.$$

We obtain the following theorem.

▶ **Theorem 9.** *There exists a data structure for the $(1+\varepsilon, r)$-ANNC under DFD, with $n \cdot O(\frac{1}{\varepsilon})^{md}$ space, $n \cdot O(\frac{1}{\varepsilon})^{md}$ expected preprocessing time, and $O(md)$ query time.*

Table 3 Comparing our ANN data structure to previous structures, for a fixed ε (say $\varepsilon = 1/2$).

m	Reference	Space	Query	Approx.
$\log n$	[10]	$O(n^{4d+1} \log n)$	$\tilde{O}(n^{4d})$	$d\sqrt{d}$
	[12]	$n^{\Omega(d \log n)}$	$\tilde{O}(dn^4)$	$1+\varepsilon$
	Theorem 9	$n^{O(d)}$	$O(d \log n)$	$1+\varepsilon$
$O(1)$	[10]	$2^{O(d)} n \log n$	$2^{O(d)} \cdot \log n$	$d\sqrt{d}$
	[12]	$d^{O(d)} \tilde{O}(n)$	$O(d \log n)$	$1+\varepsilon$
	Theorem 9	$2^{O(d)} n$	$O(d)$	$1+\varepsilon$

4 The asymmetric setting under DFD

In this section, we show how to easily adapt our data structure to the asymmetric setting, by using simplifications of length at most k instead of the original input curves.

Bereg et al. [3] showed that given a curve C consisting of m points in 3D, and a parameter $r > 0$, there is an algorithm that runs in $O(m \log m)$ time and returns a simplification Π with minimum number of vertices such that $d_{dF}(C, \Pi) \leq r$. Their algorithm generalizes to higher dimensions, using an approximation algorithm for the minimum enclosing ball problem (see Kumar et al. [17]). In this section, we use the following generalization of their original approach ([3], Theorem 1). More details are given in Section 8.

▶ **Lemma 10.** *Let C be a curve consisting of m points in $\mathbb{R}^d$. Given parameters $k \leq m$, $r > 0$, and $\varepsilon \in (0, 1]$, there is an algorithm that runs in $O\left(\frac{d \cdot m \log m}{\varepsilon} + m \cdot \text{poly}\frac{1}{\varepsilon}\right)$ time that either returns a simplification Π consisting of k points such that $d_{dF}(C, \Pi) \leq (1+\varepsilon)r$, or declares that for every simplification Π with k points, it holds that $d_{dF}(C, \Pi) > r$.*

For each $C_i \in \mathcal{C}$, using Lemma 10 with parameter $\varepsilon = 1$, we find a curve Π_i of length k such that $d_{dF}(C_i, \Pi_i) \leq 2r$. If we fail to find such a curve, then we can ignore C_i, because it means that $d_{dF}(Q, C_i) > r$ for any curve Q of length k.

To reduce the space consumption of our data structure, we only store candidate curves of length k that are close enough to the simplifications Π_i. However, since the distance between the simplification Π_i and the input curve C_i could be up to $2r$, storing the answers for the set of candidate curves that are within distance $(1 + \frac{\varepsilon}{2})r$ from Π_i is not enough, because a query Q that is within distance $(1+\varepsilon)r$ from C_i might be as far as $(3+\varepsilon)r$ from Π_i. Thus, instead, we insert into our data structure all the curves that are within distance $4r$ from Π_i. This allows us to capture all query curves that are within distance r from C_i.

The data structure. We construct our data structure for the original (symmetric) version, with the following modifications. The set of input curves is $\mathcal{P} = \{\Pi_1, \ldots, \Pi_n\}$ (instead of $\mathcal{C}$), and the radius parameter is $4r$ (instead of r), but the grid edge length remains $\frac{\varepsilon r}{\sqrt{d}}$. In addition, we let $\mathcal{I}'_i$ be the set of all curves $\overline{Q}$ with k points from G_i, such that $d_{dF}(\overline{Q}, \Pi_i) \leq 4r$, and $\mathcal{I}_i$ will be the set of all curves $\overline{Q} \in \mathcal{I}'_i$ such that $d_{dF}(\overline{Q}, C_i) \leq (1 + \frac{\varepsilon}{2})r$. We insert the curves in $\mathcal{I}_i$ into the database $\mathcal{D}$ as before: For each $\overline{Q} \in \mathcal{I}_i$, if $\overline{Q} \notin \mathcal{D}$, insert $\overline{Q}$ into $\mathcal{D}$ and set $C(\overline{Q}) \leftarrow C_i$.

Notice that using $4r$ instead of r, increases the ratio between the radius and the grid edge length by only a factor of 4, and therefore the bound on $|\mathcal{I}'_i|$ does not change, except that m is replaced by k. Therefore, the bounds on the storage space and query time are similar to those of the original data structure, where m is replaced by k. Thus, the storage space is in $n \cdot O(\frac{1}{\varepsilon})^{kd}$ and the query time is in $O(kd)$. As for the preprocessing time, we get an additional term of $O(nmd \log m)$ for computing the simplifications $\Pi_1, \ldots, \Pi_n$. We also need to compute the distances $d_{dF}(C_i, \overline{Q})$ in the construction of $\mathcal{I}_i$, for $1 \leq i \leq n$, which takes $n \cdot O(\frac{1}{\varepsilon})^{kd} \cdot O(mkd) = nm \cdot O(\frac{1}{\varepsilon})^{kd}$ time in total (as $kd \leq 2^{kd}$). Thus the total expected preprocessing time is $O(nmd \log m) + nm \cdot O(\frac{1}{\varepsilon})^{kd} = nm \cdot \big(O(d \log m) + O(\frac{1}{\varepsilon})^{kd}\big)$.

Correctness. Consider a query curve Q, and assume that there exists a curve $C_i \in \mathcal{C}$ such that $d_{dF}(C_i, Q) \leq r$. Then, Π_i is a curve of length k and $d_{dF}(C_i, \Pi_i) \leq 2r$. As in the previous section, let Q' be the curve computed by the query algorithm, then $d_{dF}(Q', Q) \leq \frac{\varepsilon r}{2}$. By the triangle inequality, we have $d_{dF}(Q', C_i) \leq d_{dF}(Q', Q) + d_{dF}(Q, C_i) \leq (1 + \frac{\varepsilon}{2})r$, and

$$d_{dF}(Q', \Pi_i) \leq d_{dF}(Q', C_i) + d_{dF}(C_i, \Pi_i) \leq (1 + \frac{\varepsilon}{2})r + 2r \leq 4r.$$

Therefore our data structure contains Q', and the query algorithm returns $C(Q')$, where $d_{dF}(C(Q'),Q') \leq (1+\frac{\varepsilon}{2})r$. Finally, again by the triangle inequality, we have

$$d_{dF}(C(Q'),Q) \leq d_{dF}(C(Q'),Q') + d_{dF}(Q',Q) \leq (1+\frac{\varepsilon}{2})r + \frac{\varepsilon r}{2} = (1+\varepsilon)r.$$

We obtain the following theorem.

▶ **Theorem 11.** *There exists a data structure for the asymmetric $(1+\varepsilon, r)$-ANNC under DFD, with $n \cdot O(\frac{1}{\varepsilon})^{dk}$ space, $nm \cdot \big(O(d \log m) + O(\frac{1}{\varepsilon})^{kd}\big)$ expected preprocessing time, and $O(kd)$ query time.*

5 $\ell_{p,2}$-distance of polygonal curves

For the near-neighbor problem under the $\ell_{p,2}$-distance, we use the same basic approach as in Section 3, but with two small modifications. The first is that we set the grid's edge length to $\frac{\varepsilon r}{(2m)^{1/p}\sqrt{d}}$, and redefine $G(x,R)$, G_i, and $\mathcal{G}$, as in Section 3 but with respect to the new edge length of our grid. The second modification is that we redefine $\mathcal{I}_i$ to be the set of all curves $\overline{Q} = (x_1, x_2, \dots, x_m)$ with points from $\mathcal{G}$, such that $d_{p,2}(C_i, \overline{Q}) \leq (1+\frac{\varepsilon}{2})r$.

We assume without loss of generality from now and to the end of this section that $r = 1$ (we can simply scale the entire space by $1/r$), so the grid's edge length is $\frac{\varepsilon}{(2m)^{1/p}\sqrt{d}}$. The following corollary is respective to Corollary 7.

▶ **Corollary 12.** $|G(x,R)| = O\left(1 + \frac{m^{1/p}}{\varepsilon}R\right)^d$.

Proof. We scale our grid so that the edge length is 1, hence we are looking for the number of lattice points in $B_2^d(x, \frac{(2m)^{1/p}\sqrt{d}}{\varepsilon}R)$. By Lemma 5 we get that this number is bounded by the volume of the d-dimensional ball of radius $(1 + \frac{(2m)^{1/p}}{\varepsilon}R)\sqrt{d}$. Using Stirling's formula we conclude,

$$V_2^d\left(\left(1 + \frac{(2m)^{1/p}}{\varepsilon}R\right)\sqrt{d}\right) = \frac{\pi^{\frac{d}{2}}}{\Gamma(\frac{d}{2}+1)} \cdot \left(\left(1 + \frac{(2m)^{1/p}}{\varepsilon}R\right)\sqrt{d}\right)^d = \alpha^d \cdot \left(1 + \frac{m^{1/p}}{\varepsilon}R\right)^d$$

where α is a constant (approximately $4.13 \cdot 2^{1/p}$). ◀

In the following claim we bound the size of $\mathcal{I}_i$, which, surprisingly, is independent of p.

▷ Claim 13. $|\mathcal{I}_i| = O(\frac{1}{\varepsilon})^{m(d+1)}$ and it can be computed in $O(\frac{1}{\varepsilon})^{m(d+1)}$ time.

Proof. Let $\overline{Q} = (x_1, x_2, \dots, x_m) \in \mathcal{I}_i$, and let τ be an alignment with $\sigma_{p,2}(\tau(C_i, \overline{Q})) \leq (1+\frac{\varepsilon}{2})$. For each $1 \leq k \leq m$ let j_k be the smallest index such that $(j_k, k) \in \tau$. In other words, j_k is the smallest index that is matched to k by the alignment τ.

Set $R_k = \|x_k - p^i_{j_k}\|_2$, then we have $\|(R_1, \dots, R_m)\|_p \leq \sigma_{p,2}(\tau(C_i, \overline{Q})) \leq (1+\frac{\varepsilon}{2})$.

Let $\alpha_k = \left\lceil \frac{m^{1/p}}{\varepsilon} R_k \right\rceil$. By triangle inequality,

$$\begin{aligned} \|(\alpha_1, \alpha_2, \dots, \alpha_m)\|_p &\leq \frac{m^{1/p}}{\varepsilon}\|(R_1, R_2, \dots, R_m)\|_p + m^{1/p} \\ &\leq \frac{m^{1/p}}{\varepsilon}\left(1 + \frac{\varepsilon}{2}\right) + m^{1/p} < \left(2 + \frac{1}{\varepsilon}\right) m^{1/p}. \end{aligned}$$

Clearly, $x_k \in B_2^d(p^i_{j_k}, \alpha_k \frac{\varepsilon}{m^{1/p}})$.

We conclude that for each curve $\overline{Q} = (x_1, x_2, \ldots, x_m) \in \mathcal{I}_i$ there exists an alignment τ such that $\sigma_{p,2}(\tau(C_i, \overline{Q})) \leq 1 + \frac{\varepsilon}{2}$, and a sequence of integers $(\alpha_1, \ldots, \alpha_m)$ such that $\|(\alpha_1, \alpha_2, \ldots, \alpha_m)\|_p \leq (2 + \frac{1}{\varepsilon})m^{1/p}$ and $x_k \in B_2^d(p^i_{j_k}, \alpha_k \frac{\varepsilon}{m^{1/p}})$, for $k = 1, \ldots, m$. Therefore, the number of curves in $\mathcal{I}_i$ is bounded by the multiplication of three numbers:

1. The number of alignments that can determine the distance, which is at most 2^{2m} by Lemma 4.
2. The number of ways to choose a sequence of m positive integers $\alpha_1, \ldots, \alpha_m$ such that $\|(\alpha_1, \alpha_2, \ldots, \alpha_m)\|_p \leq (2 + \frac{1}{\varepsilon})m^{1/p}$, which is bounded by the number of lattice points in $B_p^m((2 + \frac{1}{\varepsilon})m^{1/p})$ (the m-dimensional ℓ_p-ball of radius $(2 + \frac{1}{\varepsilon})m^{1/p}$). By Lemma 5, this number is bounded by

$$V_p^m((2 + \frac{1}{\varepsilon})m^{1/p} + m^{1/p}) \leq V_p^m(\frac{4m^{1/p}}{\varepsilon}) = \frac{2^m \Gamma(1 + 1/p)^m}{\Gamma(1 + m/p)} \left(\frac{4m^{1/p}}{\varepsilon}\right)^m = O(\frac{1}{\varepsilon})^m \ ,$$

   where the last equality follows as $\frac{m^{m/p}}{\Gamma(1+m/p)} = O(1)^m$.
3. The number of ways to choose a curve $(x_1, x_2, \ldots, x_m)$, such that $x_k \in G(p^i_{j_k}, \alpha_k \frac{\varepsilon}{m^{1/p}})$, for $k = 1, \ldots, m$. By Corollary 12, the number of grid points in $G(p^i_{j_k}, \alpha_k \frac{\varepsilon}{m^{1/p}})$ is $O(1+\alpha_k)^d$, so the number of ways to choose $(x_1, x_2, \ldots, x_m)$ is at most $\Pi_{k=1}^m O(1+\alpha_k)^d = O(1)^{md} \left(\Pi_{k=1}^m (1 + \alpha_k)\right)^d$. By the inequality of arithmetic and geometric means we have

$$\begin{aligned}
(\Pi_{k=1}^m (1 + \alpha_k)^p)^{1/p} &\leq \left(\frac{\sum_{k=1}^m (1 + \alpha_k)^p}{m}\right)^{m/p} = \left(\frac{\|(1 + \alpha_1, \ldots, 1 + \alpha_m)\|_p}{m^{1/p}}\right)^m \\
&\leq \left(\frac{\|1\|_p + \|(\alpha_1, \ldots, \alpha_m)\|_p}{m^{1/p}}\right)^m \\
&\leq \left(\frac{m^{1/p} + (2 + \frac{1}{\varepsilon})m^{1/p}}{m^{1/p}}\right)^m = O(\frac{1}{\varepsilon})^m,
\end{aligned}$$

   so $\Pi_{k=1}^m O(1 + \alpha_k)^d = O(1)^{md} O(\frac{1}{\varepsilon})^{md} = O(\frac{1}{\varepsilon})^{md}$.

Finally, $|\mathcal{I}_i| \leq 2^{2m} \cdot O(\frac{1}{\varepsilon})^m \cdot O(\frac{1}{\varepsilon})^{md} \leq O(\frac{1}{\varepsilon})^{m(d+1)}$. ◁

The data structure and query algorithm are similar to those we described for DFD, and the size of $\mathcal{I}_i$ and $\mathcal{I}$ is roughly the same (here there is an additional $O(\frac{1}{\varepsilon})^m$ factor in the space bound). Therefore, the query time, storage space, and preprocessing time are roughly similar, but we still need to show that the algorithm is correct.

Correctness. Consider a query curve $Q = (q_1, \ldots, q_m)$. Assume that there exists a curve $C_i \in \mathcal{C}$ such that $d_{p,2}(C_i, Q) \leq 1$. We will show that the query algorithm returns a curve C^* with $d_{p,2}(C^*, Q) \leq 1 + \varepsilon$.

Consider a point $q_k \in Q$. Denote by $q'_k \in \mathcal{G}$ the grid point closest to q_k, and let $Q' = (q'_1, \ldots, q'_m)$. We have $\|q_k - q'_k\|_2 \leq \frac{\varepsilon}{2(2m)^{1/p}}$. Let τ be an alignment such that the $\ell_{p,2}$-cost of τ w.r.t. C_i and Q is at most 1. Unlike the Fréchet distance, $\ell_{p,2}$-distance for curves does not satisfy the triangle inequality. However, by the triangle inequality under ℓ_2 and ℓ_p, we get that the $\ell_{p,2}$-cost of τ w.r.t. C_i and Q' is

$$\sigma_{p,2}(\tau(C_i,Q')) = \left(\sum_{(j,t)\in\tau} \|p^i_j - q'_t\|_2^p\right)^{1/p} \leq \left(\sum_{(j,t)\in\tau} \left(\|p^i_j - q_t\|_2 + \|q_t - q'_t\|_2\right)^p\right)^{1/p}$$
$$\leq \left(\sum_{(j,t)\in\tau} \|p^i_j - q_t\|_2^p\right)^{1/p} + \left(\sum_{(j,t)\in\tau} \|q_t - q'_t\|_2^p\right)^{1/p}$$
$$\leq 1 + \left(2m\left(\frac{\varepsilon}{2(2m)^{1/p}}\right)^p\right)^{1/p} \leq 1 + \frac{\varepsilon}{2}.$$

So $d_{p,2}(C_i, Q') \leq 1 + \frac{\varepsilon}{2}$, and thus Q' is in $I_i \subseteq \mathcal{I}$. This means that $\mathcal{T}$ contains Q' with a curve $C(Q') \in \mathcal{C}$ such that $d_{p,2}(C(Q'), Q') \leq 1 + \frac{\varepsilon}{2}$, and the query algorithm returns $C(Q')$. Now, again by the same argument (using an alignment with $\ell_{p,2}$-cost at most $1 + \frac{\varepsilon}{2}$ w.r.t. $C(Q')$ and Q'), we get that $d_{p,2}(C(Q'), Q) \leq 1 + \frac{\varepsilon}{2} + \left(2m\left(\frac{\varepsilon}{2(2m)^{1/p}}\right)^p\right)^{1/p} = 1 + \varepsilon$.

We obtain the following theorem.

▶ **Theorem 14.** *There exists a data structure for the $(1+\varepsilon, r)$-ANNC under $\ell_{p,2}$-distance, with $n \cdot O(\frac{1}{\varepsilon})^{m(d+1)}$ space, $n \cdot O(\frac{1}{\varepsilon})^{m(d+1)}$ expected preprocessing time, and $O(md)$ query time.*

As mentioned in the preliminaries section, the DTW distance between two curves equals to their $\ell_{1,2}$-distance, and therefore we obtain the following theorem.

▶ **Theorem 15.** *There exists a data structure for the $(1+\varepsilon, r)$-ANNC under DTW, with $n \cdot O(\frac{1}{\varepsilon})^{m(d+1)}$ space, $n \cdot O(\frac{1}{\varepsilon})^{m(d+1)}$ expected preprocessing time, and $O(md)$ query time.*

▶ Remark 16 (Dealing with query curves and input curves of varying size). For the case of DFD, our assumption that all query curves are of length exactly k can be easily removed, by constructing k data structures $\mathcal{D}_1, \ldots, \mathcal{D}_k$, where $\mathcal{D}_i$ is our data structure constructed for query curves of length i (instead of k), for $1 \leq i \leq k$. Clearly, the query time does not change. The storage space is multiplied by k, so in the case of DFD we have storage space $nk \cdot O(\frac{1}{\varepsilon})^{kd}$, but $k < 2^{kd}$, so the storage space remains $n \cdot O(\frac{1}{\varepsilon})^{kd}$.

For the case of $d_{p,2}$ we can deal with queries of all sizes up to m. Our construction in Section 5 can be modified in a straightforward manner to deal with queries of size k, the space guarantee however will depend on m, upper bounded by $n \cdot O(\frac{1}{\varepsilon})^{m(d+1)}$, as in Theorem 14. From here, we can use the same approach as above.

6 A deterministic construction using a prefix tree

When implementing the dictionary $\mathcal{D}$ as a hash table, the construction of the data structure is randomized and thus in the worst case we might get higher prepeocessing time. To avoid this, we can implement $\mathcal{D}$ as a prefix tree.

6.1 Discrete Fréchet distance

In this section we describe the implementation of $\mathcal{D}$ as a prefix tree in the case of ANNC under DFD.

We can construct a prefix tree $\mathcal{T}$ for the curves in $\mathcal{I}$, where any path in $\mathcal{T}$ from the root to a leaf corresponds to a curve that is stored in it. For each $1 \leq i \leq n$ and curve $\overline{Q} \in \mathcal{I}_i$, if $\overline{Q} \notin \mathcal{T}$, insert $\overline{Q}$ into $\mathcal{T}$, and set $C(\overline{Q}) \leftarrow C_i$.

Each node $v \in \mathcal{T}$ corresponds to a grid point from $\mathcal{G}$. Denote the set of v's children by $N(v)$. We store with v a multilevel search tree on $N(v)$, with a level for each coordinate. The points in $\mathcal{G}$ are the grid points contained in nm balls of radius $(1+\varepsilon)r$. Thus when projecting these points to a single dimension, the number of 1-dimensional points is at most $nm \cdot \frac{\sqrt{d}(1+\varepsilon)2r}{\varepsilon r} = O(\frac{nm\sqrt{d}}{\varepsilon})$. So in each level of the search tree on $N(v)$ we have $O(\frac{nm\sqrt{d}}{\varepsilon})$ 1-dimensional points, so the query time is $O(d\log(\frac{nmd}{\varepsilon}))$.

Inserting a curve of length m to the tree $\mathcal{T}$ takes $O(md\log(\frac{nmd}{\varepsilon}))$ time. Since $\mathcal{T}$ is a compact representation of $|\mathcal{I}| = n \cdot O(\frac{1}{\varepsilon})^{dm}$ curves of length m, the number of nodes in $\mathcal{T}$ is $m \cdot |\mathcal{I}| = nm \cdot O(\frac{1}{\varepsilon})^{dm}$. Each node $v \in \mathcal{T}$ contains a search tree for its children of size $O(d \cdot |N(v)|)$, and $\sum_{v\in\mathcal{T}} |N(v)| = nm \cdot O(\frac{1}{\varepsilon})^{dm}$ so the total space complexity is $O(nmd) \cdot O(\frac{1}{\varepsilon})^{md} = n \cdot O(\frac{1}{\varepsilon})^{md}$. Constructing $\mathcal{T}$ takes $O(|\mathcal{I}| \cdot md\log(\frac{nmd}{\varepsilon})) = n\log(\frac{nmd}{\varepsilon}) \cdot O(\frac{1}{\varepsilon})^{md}$ time.

▶ **Theorem 17.** *There exists a data structure for the* $(1+\varepsilon, r)$*-ANNC under DFD, with* $n \cdot O(\frac{1}{\varepsilon})^{dm}$ *space,* $n \cdot \log(\frac{n}{\varepsilon}) \cdot O(\frac{1}{\varepsilon})^{md}$ *preprocessing time, and* $O(md\log(\frac{nmd}{\varepsilon}))$ *query time.*

Similarly, for the asymmetric case we obtain the following theorem.

▶ **Theorem 18.** *There exists a data structure for the asymmetric* $(1+\varepsilon, r)$*-ANNC under DFD, with* $n \cdot O(\frac{1}{\varepsilon})^{dk}$ *space,* $nm\log(\frac{n}{\varepsilon}) \cdot \big(O(d\log m) + O(\frac{1}{\varepsilon})^{kd}\big)$ *preprocessing time, and* $O(kd\log(\frac{nkd}{\varepsilon}))$ *query time.*

6.2 $\ell_{p,2}$-distance

For the case of ANNC under $\ell_{p,2}$-distance, the total number of curves stored in the tree $\mathcal{T}$ is roughly the same as in the case of DFD. We only need to show that for a given node v of the tree $\mathcal{T}$, the upper bound on the size and query time of the search tree associated with it are similar.

The grid points corresponding to the nodes in $N(v)$ are from n sets of m balls with radius $(1+\varepsilon)$. When projecting the grid points in one of the balls to a single dimension, the number of 1-dimensional points is at most $\frac{m^{1/p}\sqrt{d}}{\varepsilon} \cdot (1+\varepsilon)$, so the total number of projected points is at most $\frac{nm^{1+\frac{1}{p}}\sqrt{d}}{\varepsilon} \cdot (1+\varepsilon)$.

Thus in each level of the search tree of v we have $O(\frac{nm^2\sqrt{d}}{\varepsilon})$ 1-dimensional points, so the query time is $O(d\log(\frac{nmd}{\varepsilon}))$, and inserting a curve of length m into the tree $\mathcal{T}$ takes $O(md\log(\frac{nmd}{\varepsilon}))$ time. Note that the size of the search tree of v remains $O(d \cdot |N(v)|)$.

We conclude that the total space complexity is $O(\frac{nm^2\sqrt{d}}{\varepsilon}) \cdot O(\frac{1}{\varepsilon})^{m(d+1)} = n \cdot O(\frac{1}{\varepsilon})^{m(d+1)}$, constructing $\mathcal{T}$ takes $O(|\mathcal{I}| \cdot md\log(nmd/\varepsilon)) = n\log(\frac{n}{\varepsilon}) \cdot O(\frac{1}{\varepsilon})^{m(d+1)}$ time, and the total query time is $O(md\log(\frac{nmd}{\varepsilon}))$.

▶ **Theorem 19.** *There exists a data structure for the* $(1+\varepsilon, r)$*-ANNC under* $\ell_{p,2}$*-distance, with* $n \cdot O(\frac{1}{\varepsilon})^{m(d+1)}$ *space,* $n \cdot \log(\frac{n}{\varepsilon}) \cdot O(\frac{1}{\varepsilon})^{m(d+1)}$ *preprocessing time, and* $O(md\log(\frac{nmd}{\varepsilon}))$ *query time.*

7 Approximate range counting

In the range counting problem for curves, we are given a set $\mathcal{C}$ of n curves, each consisting of m points in d dimensions, and a distance measure for curves δ. The goal is to preprocess $\mathcal{C}$ into a data structure that given a query curve Q and a threshold value r, returns the number of curves that are within distance r from Q.

In this section we consider the following approximation version of range counting for curves, in which r is part of the input (see Remark 22). Note that by storing pointers to curves instead of just counters, we can obtain a data structure for the approximate range searching problem (at the cost of an additional $O(n)$-factor to the storage space).

▶ **Problem 20** ($(1+\varepsilon, r)$-approximate range-counting for curves). *Given a parameter r and $0 < \varepsilon \le 1$, preprocess $\mathcal{C}$ into a data structure that given a query curve Q, returns the number of all the input curves whose distance to Q is at most r plus possibly additional input curves whose distance to Q is greater than r but at most $(1+\varepsilon)r$.*

We construct the dictionary $\mathcal{D}$ (implemented as a dynamic hash table, or a prefix tree) for the curves in $\mathcal{I}$ as in Section 5, as follows. For each $1 \le i \le n$ and curve $\overline{Q} \in \mathcal{I}_i$, if $\overline{Q}$ is not in $\mathcal{D}$, insert it into $\mathcal{D}$ and initialize $C(\overline{Q}) \leftarrow 1$. Otherwise, if $\overline{Q}$ is in $\mathcal{D}$, update $C(\overline{Q}) \leftarrow C(\overline{Q}) + 1$. Notice that $C(\overline{Q})$ holds the number of curves from $\mathcal{C}$ that are within distance $(1+\frac{\varepsilon}{2})r$ to $\overline{Q}$. Given a query curve Q, we compute Q' as in Section 5. If Q' is in $\mathcal{D}$, we return $C(Q')$, otherwise, we return 0.

Clearly, the storage space, preprocessing time, and query time are similar to those in Section 5. We claim that the query algorithm returns the number of curves from $\mathcal{C}$ that are within distance r to Q plus possibly additional input curves whose distance to Q is greater than r but at most $(1+\varepsilon)r$. Indeed, let C_i be a curve such that $d_{dF}(C_i, Q) \le r$. As shown in Section 5 we get $d_{p,2}(C_i, Q') \le (1+\frac{\varepsilon}{2})r$, so Q' is in $\mathcal{I}_i$ and C_i is counted in $C(Q')$. Now let C_i be a curve such that $d_{p,2}(C_i, Q) > (1+\varepsilon)r$. If $d_{p,2}(C_i, Q') \le (1+\frac{\varepsilon}{2})r$, then by a similar argument (switching the rolls of Q and Q') we get that $d_{p,2}(C_i, Q') \le (1+\varepsilon)r$, a contradiction. So $d_{p,2}(C_i, Q') > (1+\frac{\varepsilon}{2})r$, and thus C_i is not counted in $C(Q')$.

We obtain the following theorem.

▶ **Theorem 21.** *There exists a data structure for the $(1+\varepsilon, r)$-approximate range-counting for curves under $\ell_{p,2}$-distance, with $n \cdot O(\frac{1}{\varepsilon})^{m(d+1)}$ space, $n \log(\frac{n}{\varepsilon}) \cdot O(\frac{1}{\varepsilon})^{m(d+1)}$ preprocessing time, and $O(md \log(\frac{nmd}{\varepsilon}))$ query time. (Under DFD, the exponent in the bounds for the space and preprocessing time is md rather than $m(d+1)$.)*

▶ Remark 22. When the threshold parameter r is part of the query, we call the problem the $(1+\varepsilon)$-*approximate range-counting problem.* Note that the reduction from $(1+\varepsilon)$-approximate nearest-neighbor to $(1+\varepsilon, r)$-approximate near-neighbor can be easily adapted to a reduction from $(1+\varepsilon)$-approximate range-counting to $(1+\varepsilon, r)$-approximate range-counting, more details will be given in a full version of this paper.

8 Simplification in d-dimensions

The algorithm of Bereg et al. [3] receives as an input a curve C consisting of m points in $\mathbb{R}^3$, and a parameter $r > 0$. In $O(m \log m)$ time, it returns a curve Π such that $d_{dF}(C, \Pi) \le r$, and Π has the minimum number of vertices among all curves within distance r from C. The algorithm is operating in a greedy manner, by repeatedly executing Megiddo's [19] minimum enclosing ball (MEB) algorithm for points in $\mathbb{R}^3$, which takes linear time.

We generalize the algorithm of Bereg et al. for curves in $\mathbb{R}^d$, by using an algorithm presented by Kumar et al. [17] for approximated minimum enclosing ball (AMEB) in $\mathbb{R}^d$. Formally, given a set A of n points in $\mathbb{R}^d$ and a parameter $\varepsilon \in (0, 1]$, the goal is to find an enclosing ball of A with radius $r > 0$, where the minimum enclosing ball of A has radius at least $\frac{r}{1+\varepsilon}$. The algorithm of [17] can find an AMEB in $O(\frac{nd}{\varepsilon} + \varepsilon^{-4.5} \log \frac{1}{\varepsilon})$ time. In particular, given an additional parameter $r > 0$, this algorithm either returns an enclosing ball of A with radius $(1+\varepsilon)r$, or declares that the minimum enclosing ball of A has radius larger than r.

Next, we describe our modified algorithm. Consider a curve $C = (x_1, \ldots, x_m)$, and denote $C[i,j] = (x_i, \ldots, x_j)$. The following sub-procedure takes as an input a curve A and returns a point y and an index s, such that the ball with radius $(1+\varepsilon)r$ centered at y covers the prefix $A[1,s]$, and (if $s < |A|$) the minimum enclosing ball of $A[1,s+1]$ has radius larger than r.

1. By iterative probing, using an algorithm for AMEB, find some t such that $A[1,2^t]$ can be covered by a ball of radius $(1+\varepsilon)r$, while $A[1,2^{t+1}]$ cannot be covered by a ball of radius r. If all the points in A can be enclosed by a single ball of radius $(1+\varepsilon)r$ centered at y, simply return y and $|A|$.
2. By binary search, again using an algorithm for AMEB, find some $s \in [2^t, 2^{t+1})$ such that $A[1,s]$ can be covered by a ball of radius $(1+\varepsilon)r$, and $A[1,s+1]$ cannot be covered by a ball of radius r. Let $y \in \mathbb{R}^d$ be the center of this ball. Return y and s.

Starting from the input $A = C[1,m]$, repeat the above sub-procedure such that in each step the input is the suffix of C that was not yet covered by the previous steps (i.e. $A[s+1,m]$). Let $(y_1, \ldots, y_q)$ be the sequence of output points.

Lemma 10 is an easy corollary of the following lemma.

▶ **Lemma 23.** *Let C be a curve consisting of m points in $\mathbb{R}^d$. Given parameters $r > 0$, and $\varepsilon \in (0,1]$, the algorithm above runs in $O\left(\frac{d \cdot m \log m}{\varepsilon} + m \cdot \varepsilon^{-4.5} \log \frac{1}{\varepsilon}\right)$ time and returns a curve $\Pi = (y_1, \ldots, y_q)$ such that $d_{dF}(C, \Pi) \le (1+\varepsilon)r$. Furthermore, for every curve Π' with less than q points, it holds that $d_{dF}(C, \Pi') > r$.*

Proof sketch. We start by analyzing the running time for a single iteration of the sub-procedure, when using the algorithm of [17] to find an AMEB. The total time for the first step of the sub-procedure (finding t) is

$$\sum_{i=1}^{t+1} O(\frac{2^i \cdot d}{\varepsilon} + \varepsilon^{-4.5} \log \frac{1}{\varepsilon}) = O(\frac{2^t \cdot d}{\varepsilon} + t \cdot \varepsilon^{-4.5} \log \frac{1}{\varepsilon}).$$

In the second step, there are $O(t)$ executions of [17] on a set of size at most 2^{t+1}, so the total time for this step is $t \cdot O(\frac{2^t \cdot d}{\varepsilon} + \varepsilon^{-4.5} \log \frac{1}{\varepsilon})$.

Let m_i be the length of the subcurve covered by the point y_i that was found in the i'th iteration of the sub-procedure. The total time spent for finding y_i is therefore $\log m_i \cdot O(\frac{m_i \cdot d}{\varepsilon} + \varepsilon^{-4.5} \log \frac{1}{\varepsilon})$, and the total running time of the algorithm is

$$\sum_{i=1}^{q} \log m_i \cdot O\left(\frac{m_i \cdot d}{\varepsilon} + \varepsilon^{-4.5} \log \frac{1}{\varepsilon}\right) = O\left(\frac{d \cdot m \log m}{\varepsilon} + m \cdot \varepsilon^{-4.5} \log \frac{1}{\varepsilon}\right) ,$$

where we used the the concavity of the log function, and the fact $\sum_{i=1}^{q} m_i = m$.

Next we argue the correctness. Clearly, $d_{dF}(C, \Pi) \le (1+\varepsilon)r$. Let $s_0 = 0, s_1, \ldots, s_q = m$ be the sequence of indices (of vertices in C) found during the execution of the algorithm, such that the ball of radius $(1+\varepsilon)r$ around y_i covers $C[s_{i-1}+1, s_i]$. It follows by a straightforward induction that every curve Π' with less that i points will be at distance greater than r from $C[1, s_{i-1}+1]$. The lemma now follows. ◀

9 Remark on dimension reduction

In general, when the dimension d is large, i.e. $d \gg \log(nm)$, one can use dimension reduction (using the celebrated Johnson-Lindenstrauss lemma [16]) in order to achieve a better running time, at the cost of inserting randomness in the prepossessing and query procedure. However, such an approach can work only against an oblivious adversary, as it will necessarily fail for some curves. Recently Narayanan and Nelson [20] (improving [11, 18]) proved a terminal version of the JL-lemma. Given a set K of k points in $\mathbb{R}^d$ and $\varepsilon \in (0,1)$, there is a dimension reduction function $f : \mathbb{R}^d \to \mathbb{R}^{O(\frac{\log k}{\varepsilon^2})}$ such that for every $x \in K$ and $y \in \mathbb{R}^d$ it holds that $\|x-y\|_2 \leq \|f(x)-f(y)\|_2 \leq (1+\varepsilon)\cdot\|x-y\|_2$.

This version of dimension reduction can be used such that the query remains deterministic and always succeeds. The idea is to take all the nm points from all the input curves to be the terminals, and let f be the terminal dimension reduction. We transform each input curve $P = (p_1,\dots,p_m)$ into $f(P) = (f(p_1),\dots,f(p_m))$, a curve in $\mathbb{R}^{O(\frac{\log nm}{\varepsilon^2})}$. Given a query $Q = (q_1,\dots,q_m)$ we transform it to $f(Q) = (f(q_1),\dots,f(q_m))$. Since the pairwise distances between every query point to all input points are preserved, so is the distance between the curves. Specifically, the $d_{p,2}$ distance w.r.t. any alignment τ is preserved up to a $1+\varepsilon$ factor, and therefore we can reliably use the answer received using the transformed curves.

References

1 Peyman Afshani and Anne Driemel. On the complexity of range searching among curves. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 898–917, 2018. doi:10.1137/1.9781611975031.58.

2 Boris Aronov, Omrit Filtser, Michael Horton, Matthew J. Katz, and Khadijeh Sheikhan. Efficient nearest-neighbor query and clustering of planar curves. In *Algorithms and Data Structures - 16th International Symposium, WADS 2019, Edmonton, AB, Canada, August 5-7, 2019, Proceedings*, pages 28–42, 2019. doi:10.1007/978-3-030-24766-9_3.

3 Sergey Bereg, Minghui Jiang, Wencheng Wang, Boting Yang, and Binhai Zhu. Simplifying 3d polygonal chains under the discrete Fréchet distance. In *LATIN 2008: Theoretical Informatics, 8th Latin American Symposium, Búzios, Brazil, April 7-11, 2008, Proceedings*, pages 630–641, 2008. doi:10.1007/978-3-540-78773-0_54.

4 Karl Bringmann. Why walking the dog takes time: Fréchet distance has no strongly subquadratic algorithms unless SETH fails. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 661–670, 2014. doi:10.1109/FOCS.2014.76.

5 Kevin Buchin, Anne Driemel, Joachim Gudmundsson, Michael Horton, Irina Kostitsyna, Maarten Löffler, and Martijn Struijs. Approximating (k, l)-center clustering for curves. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 2922–2938, 2019. doi:10.1137/1.9781611975482.181.

6 Mark de Berg, Atlas F. Cook IV, and Joachim Gudmundsson. Fast Fréchet queries. *Comput. Geom.*, 46(6):747–755, 2013. doi:10.1016/j.comgeo.2012.11.006.

7 Mark de Berg, Joachim Gudmundsson, and Ali D. Mehrabi. A dynamic data structure for approximate proximity queries in trajectory data. In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, GIS 2017, Redondo Beach, CA, USA, November 7-10, 2017*, pages 48:1–48:4, 2017. doi:10.1145/3139958.3140023.

8 Anne Driemel and Sariel Har-Peled. Jaywalking your dog: Computing the Fréchet distance with shortcuts. *SIAM J. Comput.*, 42(5):1830–1866, 2013. doi:10.1137/120865112.

9 Anne Driemel, Ioannis Psarros, and Melanie Schmidt. Sublinear data structures for short Fréchet queries. *CoRR*, abs/1907.04420, 2019. arXiv:1907.04420.

10 Anne Driemel and Francesco Silvestri. Locality-Sensitive Hashing of Curves. In *Proceedings of the 33rd International Symposium on Computational Geometry*, volume 77, pages 37:1–

37:16, Brisbane, Australia, July 2017. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SoCG.2017.37.

11 Michael Elkin, Arnold Filtser, and Ofer Neiman. Terminal embeddings. *Theor. Comput. Sci.*, 697:1–36, 2017. doi:10.1016/j.tcs.2017.06.021.

12 Ioannis Z. Emiris and Ioannis Psarros. Products of Euclidean metrics and applications to proximity questions among curves. In *34th International Symposium on Computational Geometry, SoCG 2018, June 11-14, 2018, Budapest, Hungary*, pages 37:1–37:13, 2018. doi:10.4230/LIPIcs.SoCG.2018.37.

13 Sariel Har-Peled, Piotr Indyk, and Rajeev Motwani. Approximate nearest neighbor: Towards removing the curse of dimensionality. *Theory of Computing*, 8(1):321–350, 2012. doi:10.4086/toc.2012.v008a014.

14 Piotr Indyk. *High-dimensional computational geometry*. PhD thesis, Stanford University, 2000. see here.

15 Piotr Indyk. Approximate nearest neighbor algorithms for Fréchet distance via product metrics. In *Proceedings of the 8th Symposium on Computational Geometry*, pages 102–106, Barcelona, Spain, June 2002. ACM Press. doi:10.1145/513400.513414.

16 William Johnson and Joram Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. *Contemporary Mathematics*, 26:189–206, 1984. doi:10.1090/conm/026/737400.

17 Piyush Kumar, Joseph S. B. Mitchell, and E. Alper Yildirim. Comuting core-sets and approximate smallest enclosing hyperspheres in high dimensions. In *Proceedings of the Fifth Workshop on Algorithm Engineering and Experiments, Baltimore, MD, USA, January 11, 2003*, pages 45–55, 2003.

18 Sepideh Mahabadi, Konstantin Makarychev, Yury Makarychev, and Ilya P. Razenshteyn. Nonlinear dimension reduction via outer bi-Lipschitz extensions. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 1088–1101, 2018. doi:10.1145/3188745.3188828.

19 Nimrod Megiddo. Linear programming in linear time when the dimension is fixed. *J. ACM*, 31(1):114–127, 1984. doi:10.1145/2422.322418.

20 Shyam Narayanan and Jelani Nelson. Optimal terminal dimensionality reduction in Euclidean space. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 1064–1069, 2019. doi:10.1145/3313276.3316307.

21 Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. *Journal of Algorithms*, 51(2):122–144, 2004. doi:10.1016/j.jalgor.2003.12.002.

22 Gregory Shakhnarovich, Trevor Darrell, and Piotr Indyk. *Nearest-neighbor methods in learning and vision: theory and practice (neural information processing)*. The MIT press, 2006. see here.

Computation of Hadwiger Number and Related Contraction Problems: Tight Lower Bounds

Fedor V. Fomin
University of Bergen, Norway
fomin@ii.uib.no

Daniel Lokshtanov
University of California, Santa Barbara, CA, USA
daniello@ucsb.edu

Ivan Mihajlin
University of California, San Diego, CA, USA
imikhail@cs.ucsd.edu

Saket Saurabh
Department of Informatics, University of Bergen, Norway
The Institute of Mathematical Sciences, Chennai, India
saket@imsc.res.in

Meirav Zehavi
Ben-Gurion University of the Negev, Beer-Sheva, Israel
meiravze@bgu.ac.il

Abstract

We prove that the Hadwiger number of an n-vertex graph G (the maximum size of a clique minor in G) cannot be computed in time $n^{o(n)}$, unless the Exponential Time Hypothesis (ETH) fails. This resolves a well-known open question in the area of exact exponential algorithms. The technique developed for resolving the Hadwiger number problem has a wider applicability. We use it to rule out the existence of $n^{o(n)}$-time algorithms (up to ETH) for a large class of computational problems concerning edge contractions in graphs.

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

Keywords and phrases Hadwiger Number, Exponential-Time Hypothesis, Exact Algorithms, Edge Contraction Problems

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.49

Category Track A: Algorithms, Complexity and Games

Related Version The full version of this paper is available at `http://arxiv.org/abs/2004.11621`.

Funding *Fedor V. Fomin*: Research Council of Norway via the project MULTIVAL.
Daniel Lokshtanov: European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant no. 715744), and United States - Israel Binational Science Foundation grant no. 2018302.
Saket Saurabh: European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant no. 819416), and Swarnajayanti Fellowship grant DST/SJF/MSA-01/2017-18.
Meirav Zehavi: Israel Science Foundation grant no. 1176/18, and United States – Israel Binational Science Foundation grant no. 2018302.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 49; pp. 49:1–49:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

The *Hadwiger number* $h(G)$ of a graph G is the largest number h for which the complete graph K_h is a minor of G. Equivalently, $h(G)$ is the maximum size of the largest complete graph that can be obtained from G by contracting edges. It is named after Hugo Hadwiger, who conjectured in 1943 that the Hadwiger number of G is always at least as large as its chromatic number. According to Bollobás, Catlin, and Erdős, this conjecture remains "one of the deepest unsolved problems in graph theory" [4].

The Hadwiger number of an n-vertex graph G can be easily computed in time $n^{\mathcal{O}(n)}$ by brute-forcing through all possible partitions of the vertex set of G into connected sets, contracting each set into one vertex and checking whether the resulting graph is a complete graph. The question whether the Hadwiger number of a graph can be computed in single-exponential $2^{\mathcal{O}(n)}$ time was previously asked in [1, 6, 13]. Our main result provides a negative answer to this open question.

▶ **Theorem 1.** *Unless the Exponential Time Hypothesis (ETH) is false, there does not exist an algorithm computing the Hadwiger number of an n-vertex graph in time $n^{o(n)}$.*

The interest in the complexity of the Hadwiger number is naturally explained by the recent developments in the area of exact exponential algorithms, that is, algorithms solving intractable problems significantly faster than the trivial exhaustive search, though still in exponential time [8]. Within the last decade, significant progress on upper and lower bounds of exponential algorithms has been achieved. Drastic improvements over brute-force algorithms were obtained for a number of fundamental problems like Graph Coloring [3] and Hamiltonicity [2]. On the other hand, by making use of the ETH, lower bounds could be obtained for 2-CSP [15] or for Subgraph Isomorphism and Graph Homomorphism [6].

Graph Minor (deciding whether a graph G contains a graph H as a minor) is a fundamental problem in graph theory and graph algorithms. Graph Minor could be seen as special case of a general graph embedding problem where one wants to embed a graph H into graph G. In what follows we will use n to denote the number of vertices in G and h to denote the number of vertices in H. By the theorem of Robertson and Seymour [14], there exists a computable function f and an algorithm that, for given graphs G and H, checks in time $f(h) \cdot n^3$ whether H is a minor of G. Thus the problem is fixed-parameter tractable (FPT) being parameterized by H. On the other hand, Cygan et al. [6] proved that unless the ETH fails, this problem cannot be solved in time $n^{o(n)}$ even in the case when $|V(G)| = |V(H)|$. Other interesting embedding problems that are strongly related to Graph Minor include the following problems.

- Subgraph Isomorphism: Given two graphs G and H, decide whether G contains a subgraph isomorphic to H. This problem cannot be solved in time $n^{o(n)}$ when $|V(G)| = |V(H)|$, unless the ETH fails [6]. In the special case called Clique, when H is a clique, a brute-force algorithm checking for every vertex subset of G whether it is a clique of size h solves the problem in time $n^{\mathcal{O}(h)}$. The same algorithm also runs in single-exponential time $\mathcal{O}(2^n n^2)$. It is also known that Clique is W[1]-hard parameterized by h and cannot be solved in time $f(h) \cdot n^{o(h)}$ for any function f unless the ETH fails [7, 5].
- Graph Homomorphism: Given two graphs G and H, decide whether there exists a homomorphism from G to H. (A *homomorphism* $G \to H$ from an undirected graph G to an undirected graph H is a mapping from the vertex set of G to that of H such that the image of every edge of G is an edge of H.) This problem is trivially solvable in time $h^{\mathcal{O}(n)}$, and an algorithm of running time $h^{o(n)}$ for this problem would yield the failure of

the ETH [6]. However, for the special case of H being a clique, Graph Homomorphism is equivalent to h-Coloring (deciding whether the chromatic number of G is at most h), and thus is solvable in single-exponential time $2^n \cdot n^{\mathcal{O}(1)}$ [3, 12]. When the graph G is a complete graph, the problem is equivalent to finding a clique of size n in H, and then is solvable in time $2^h \cdot h^{\mathcal{O}(1)}$.

- Topological Graph Minor: Given two graphs G and H, decide whether G contains H as a topological minor. (We say that a graph H is a subdivision of a graph G if G can be obtained from H by contracting only edges incident with at least one vertex of degree two. A graph H is called a *topological minor* of a graph G if a subdivision of H is isomorphic to a subgraph of G.) This problem is, perhaps, the closest "relative" of Graph Minor. Grohe et al. [11] gave an algorithm of running time $f(h) \cdot n^3$ for this problem for some computable function f. Similar to Graph Minor and Subgraph Isomorphism, this problem cannot be solved in time $n^{o(n)}$ when $|V(G)| = |V(H)|$, unless the ETH fails [6]. However for the special case of the problem with H being a complete graph, Lingas and Wahlen [13] gave a single-exponential algorithm solving the problem in time $2^{\mathcal{O}(n)}$.

Thus all the above graph embedding "relatives" of Graph Minor are solvable in single-exponential time when graph H is a clique. However, from the perspective of exact exponential algorithms, Theorem 1 implies that finding the largest clique minor is the most difficult problem out of them all. This is why we find the lower bound provided by Theorem 1 surprising. Moreover, from the perspective of parameterized complexity, finding a clique minor of size h, which is FPT, is actually easier than finding a clique (as a subgraph) of size h, which is W[1]-hard, as well as from finding an h-coloring of a graph, which is para-NP-hard.

Theorem 1 also answers another question of Cygan et al. [6], who asked whether deciding if a graph H can be obtained from a graph G only by edge contractions, could be resolved in single-exponential time. By Theorem 1, the existence of such an algorithm is highly unlikely even when the graph H is a complete graph. Moreover, the technique developed to prove Theorem 1, appears to be extremely useful to rule out the existence of $n^{o(n)}$-time algorithms for various contraction problems. We formalize our results with the following $\mathcal{F}$-Contraction problem. Let $\mathcal{F}$ be a graph class. Given a graph G and $t \in \mathbb{N}$, the task is to decide whether there exists a subset $F \subseteq E(G)$ of size at most t such that $G/F \in \mathcal{F}$ (where G/F is the graph obtained from G by contracting the edges in F). We prove that in each of the cases of $\mathcal{F}$-Contraction where $\mathcal{F}$ is the family of chordal graphs, interval graphs, proper interval graphs, threshold graphs, trivially perfect graphs, split graphs, complete split graphs and perfect graphs, unless the ETH fails, $\mathcal{F}$-Contraction is not solvable in time $n^{o(n)}$. For lack of space, some of these results are relegated to the full version of this paper (see [9]).

Technical Details. A summary of the reductions presented in this paper is given in Fig. 1. To prove our lower bounds, we first revisit the proof of Cygan et al. [6] for the ETH-hardness of a problem called List Subgraph Isomorphism. Informally, in this problem we are given two graphs G and H on the same number of vertices, as well as a list of vertices in H for each vertex in G, and we need to find a copy of G in H so that each vertex u in G is mapped to a vertex v in H that belongs to its list (i.e. v belongs to the list of u). We prove that the instances produced by the reduction (after some modification) of [6] have a very useful property that we crucially exploit later. Specifically, we construct a proper coloring of G as well as a proper coloring of H, and show that every vertex v in H that belongs to the list of some vertex u is, in fact, of the same color as u.

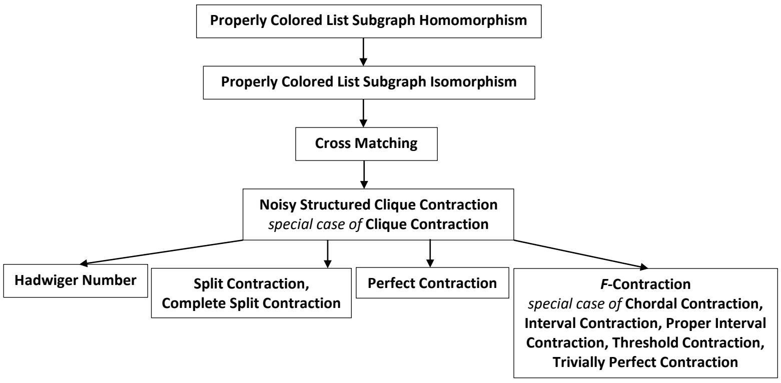

Properly Colored List Subgraph Homomorphism
Properly Colored List Subgraph Isomorphism
Cross Matching
Noisy Structured Clique Contraction
special case of Clique Contraction
Hadwiger Number
Split Contraction, Complete Split Contraction
Perfect Contraction
F-Contraction
special case of Chordal Contraction, Interval Contraction, Proper Interval Contraction, Threshold Contraction, Trivially Perfect Contraction

Figure 1 A summary of the problems considered in this paper, and the reductions between them.

Having proved the above, we turn to prove the ETH-hardness of a special case of Clique Contraction where the input graph is highly structured. To this end, we introduce an intermediate problem called Cross Matching. Informally, in this problem we are given a graph L with a partition (A, B) of its vertex set, and need to find a perfect matching between A and B whose contraction gives a clique. To see the connection between this problem and List Subgraph Isomorphism, think of the subgraph of L induced by one side of the partition – say, A – as a representation of the *complement* of G, and the subgraph of L induced by the other side of the partition as a representation of H. Then, the edges that go across A and B in a perfect matching can be thought of as a mapping of the vertices of G to the vertices of H. The crossing edges of L are easily defined such that necessarily a vertex of G can only be matched to a vertex in its list. In particular, we would like to enforce that every "non-edge" of the complement of G (which corresponds to an edge of G) would have to be mapped to an edge of H in order to obtain a clique. However, the troublesome part is that non-edges of the complement of G may also be "filled" (to eventually get a clique) using crossing edges rather than only edges of H. To argue that this critical issue does not arise, we crucially rely on the proper colorings of G and H.

Now, for the connection between Cross Matching and Clique Contraction, note that a solution to an instance of Cross Matching is clearly a solution to the instance of Clique Contraction defined by the same graph, but the other direction is not true. By adding certain vertices and edges to the graph of an instance of Cross Matching, we enforce all solutions to be perfect matchings between A and B. In particular, we construct the instances of Clique Contraction in a highly structured manner that allows us to derive not only the ETH-hardness of Clique Contraction itself, but to build upon them and further derive ETH-hardness for a wide variety of other contraction problems. In particular, we show that the addition of "noise" (that is, extra vertices and edges) to any structured instance of Clique Contraction has very limited effect. Roughly speaking, we show that the edges in the "noise" and the edges going across the "noise" and core of the graph (that is, the original vertices corresponding to the structured instance of Clique Contraction) are not "helpful" when trying to create a clique on the core (i.e. it is not helpful to try to use these edges in order to fill non-edges between vertices in the core). Depending on the contraction problem at hand, the noise is slightly different, but the proof technique stays the same – first showing that the core must yield a clique, and then using the argument above (in fact, in all cases but that of perfect graphs, we are able to invoke the argument as a black box) to show that the noise is, in a sense, irrelevant.

Preliminaries. As we only use standard notations, we present them only in the full version of this paper.

2 Lower Bound: Prop-Colored List Subgraph Isomorphism

In this section we build upon the work of Cygan et al. [6] and show a lower bound for a problem called Properly Colored List Subgraph Isomorphism (Prop-Col LSI). Intuitively, Prop-Col LSI is a variant of Spanning Subgraph Isomorphism where given two graphs G and H, we ask whether G is isomorphic to some spanning subgraph of H. The input to the variant consists also of proper colorings of G and H and an additional labeling of vertices in G by subsets of vertices in H of the same color, so that each vertex in G can be mapped only to vertices in H contained in its list. Formally, it is defined as follows.

Properly Colored List Subgraph Isomorphism (Prop-Col LSI)
Input: Graphs G and H with proper colorings $c_G : V(G) \to \{1, \dots, k\}$ and $c_H : V(H) \to \{1, \dots, k\}$ for some $k \in \mathbb{N}$, respectively, and a function $\ell : V(G) \to 2^{V(H)}$ such that for every $u \in V(G)$ and $v \in \ell(u)$, $c_G(u) = c_H(v)$.
Question: Does there exist a bijective function $\varphi : V(G) \to V(H)$ such that *(i)* for every $\{u, v\} \in E(G)$, $\{\varphi(u), \varphi(v)\} \in E(H)$, and *(ii)* for every $u \in V(G)$, $\varphi(u) \in \ell(u)$?

Notice that as the function φ above is bijective rather than only injective, we seek a spanning subgraph. Our objective is to prove the following statement.

▶ **Lemma 2.** *Unless the ETH is false, there does not exist an algorithm that solves Prop-Col LSI in time $n^{o(n)}$ where $n = |V(G)|$.*

In [6], the authors considered the two problems defined below. Intuitively, the second is defined as Prop-Col LSI when no proper colorings of H and G are given (and hence the labeling of vertices in G is not restricted accordingly); the first is defined as the second when we seek a homomorphism rather than an isomorphism (i.e., the sought function φ may not be injective) and also $|V(G)|$ may not be equal to $|V(H)|$ (thus φ may neither be onto).

List Subgraph Homomorphism (LSH)
Input: Graphs G and H, and a function $\ell : V(G) \to 2^{V(H)}$.
Question: Does there exist a function $\varphi : V(G) \to V(H)$ such that *(i)* for every $\{u, v\} \in E(G)$, $\{\varphi(u), \varphi(v)\} \in E(H)$, and *(ii)* for every $u \in V(G)$, $\varphi(u) \in \ell(u)$?

List Subgraph Isomorphism (LSI)
Input: Graphs G and H where $|V(G)| = |V(H)|$, and a function $\ell : V(G) \to 2^{V(H)}$.
Question: Does there exist a bijective function $\varphi : V(G) \to V(H)$ such that *(i)* for every $\{u, v\} \in E(G)$, $\{\varphi(u), \varphi(v)\} \in E(H)$, and *(ii)* for every $u \in V(G)$, $\varphi(u) \in \ell(u)$?

The proof of hardness of LSI consists of two parts:

- Showing ETH-hardness of LSH.
- Giving a fine-grained reduction from LSH to LSI.

We cannot use the hardness of LSI as a black box because Prop-Col LSI is a special case of LSI. Nevertheless, we will prove that the instances generated by the reduction (with a minor crucial modification) of Cygan et al. [6] have the additional properties required to make them instances of our special case.

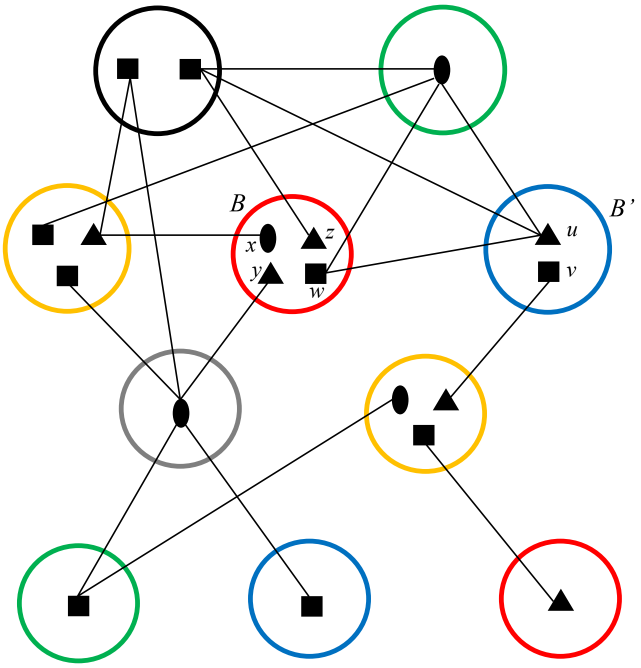


Figure 2 The reduction in Definition 4. The vertices of G are depicted by black shapes, where each distinct shape represents a different color (say, square is 1, rectangle is 2 and oval is 3), and the vertices of $\widetilde{G}$ are depicted by circles enclosing the vertex sets identifies with them, where the color of a vertex is the color of its circle (say, black is 1, green is 2, yellow is 3, red is 4, blue is 5 and grey is 6). Edges (of both graphs) are depicted by black lines. (The graph $\widetilde{H}$ is not shown). Then, the function ϕ_B is defined as follows: $\phi_B(1) = z, \phi_B(2) = \phi_B(5) = w, \phi_B(3) = x, \phi_B(4) = 0$, and $\phi_B(6) = y$. Moreover, the function $\phi_{B'}$ is defined as follows: $\phi_{B'}(1) = \phi_{B'}(2) = \phi_{B'}(4) = u, \phi_{B'}(3) = v$, and $\phi_{B'}(5) = \phi_{B'}(6) = 0$. With respect to B and B', the labeling ℓ is defined as follows: $\ell(B) = \{(R,4) : R[1] \neq 0, R[2] = R[5] \neq 0, R[3] \neq 0, R[4] = 0, R[6] \neq 0\}$, and $\ell(B') = \{(R,5) : R[1] = R[2] = R[4] \neq 0, R[3] \neq 0, R[5] = R[6] = 0\}$.

Lower Bound: Properly Colored Subgraph Homomorphism. Adapting the scheme of Cygan et al. [6] to our purpose, we will first show that finding a homomorphism remains hard if it has to preserve a given proper coloring:

> Properly Colored List Subgraph Homomorphism (Prop-Col LSH)
> **Input:** Graphs G and H with proper colorings $c_G : V(G) \to \{1,\dots,k\}$ and $c_H : V(H) \to \{1,\dots,k\}$ for some $k \in \mathbb{N}$, respectively, and a function $\ell : V(G) \to 2^{V(H)}$ such that for every $u \in V(G)$ and $v \in \ell(u)$, $c_G(u) = c_H(v)$.
> **Question:** Does there exist a function $\varphi : V(G) \to V(H)$ such that *(i)* for every $\{u,v\} \in E(G)$, $\{\varphi(u),\varphi(v)\} \in E(H)$, and *(ii)* for every $u \in V(G)$, $\varphi(u) \in \ell(u)$?

In [6], the authors gave a reduction from the 3-Coloring problem on n-vertex graphs of degree 4 (which is known not to be solvable in time $2^{o(n)}$ unless the ETH fails), which generates equivalent instances (G', H', ℓ) of LSH where both $|V(G')|$ and $|V(H')|$ are bounded by $\mathcal{O}(\frac{n}{\log n})$. This proves that LSH is not solvable in time $n^{o(n)}$ where $n = \max\{|V(G)|, |V(H)|\}$ unless the ETH fails. For their reduction, Cygan et al. [6] considered the notion of a *grouping* (also known as *quotient graph*) $\widetilde{G}$ of a graph G is a graph with vertex set $V(\widetilde{G}) = \{B_1, B_2, \dots, B_t\}$ where $(B_1, B_2, \dots, B_t)$ is a partition of $V(G)$ for some $t \in \mathbb{N}$ and for any distinct $i, j \in \{1,\dots,t\}$, the vertices B_i and B_j are adjacent in $\widetilde{G}$ if and only if there exist $u \in B_i$ and $v \in B_j$ that are adjacent in G. Specifically, they computed a grouping with a coloring having specific properties as stated in the following lemma (see also Fig. 2.).

▶ **Lemma 3** (Lemma 3.2 in [6]). *For any constant $d \geq 1$, there exist positive integers $\lambda = \lambda(d)$, $n_0 = n_0(d)$ and a polynomial time algorithm that for a given graph G on $n \geq n_0$ vertices of maximum degree d and a positive integer $r \leq \sqrt{\frac{n}{2\lambda}}$, finds a grouping $\widetilde{G}$ of G and a coloring $\widetilde{c} : V(\widetilde{G}) \to [\lambda r]$ with the following properties:*

1. $|V(\widetilde{G})| \leq |V(G)|/r$;
2. *The coloring $\widetilde{c}$ is a proper coloring of $\widetilde{G}^2$;*[1]
3. *Each vertex of $\widetilde{G}$ is an independent set in G;*
4. *For any edge $\{B_i, B_j\} \in E(\widetilde{G})$, there exists exactly one pair $(u, v) \in B_i \times B_j$ such that $\{u, v\} \in E(G)$.*

Now, we describe the reduction of [6]. Here, without loss of generality, it is assumed that G has no isolated vertices, else they can be removed. An explanation of the intuition behind this somewhat technical definition is given below it.

▶ **Definition 4.** *For any instance G of 3-Coloring where G has degree d and a positive integer $r = o(\sqrt{|V(G)|})$, the instance* `reduce`$(G) = (\widetilde{G}, \widetilde{H}, \ell)$ *of LSH is defined as follows.*

- ***The graph $\widetilde{G}$.*** *Let $\widetilde{G}$ and $\widetilde{c} : V(\widetilde{G}) \to \{1, 2, \ldots, L\}$ be the grouping and coloring given by Lemma 3 where $L = \lambda(d)r$. Additionally, for each $B \in V(\widetilde{G})$, define $\phi_B : \{1, 2, \ldots, L\} \to B \cup \{0\}$ as follows: for any $i \in \{1, 2, \ldots, L\}$, if there exists (u, v, B') such that $u \in B$ and $v \in B'$, $\{u, v\} \in E(G)$ and $\widetilde{c}(B') = i$, then $\phi_B(i) = u$, and otherwise $\phi_B(i) = 0$.*[2]
- ***The graph $\widetilde{H}$.*** *Let $V(\widetilde{H}) = \{(R, l) : R \in \{0, 1, 2, 3\}^L, l \in L\}$,*[3] *and $E(\widetilde{H}) = \{\{(R, l), (R', l')\} : R[l'] \neq R'[l]\}$.*
- ***The labeling ℓ.*** *For any $B \in V(\widetilde{G})$, let $\ell(B)$ contain all vertices $(R, l) \in V(\widetilde{H})$ such that $\widetilde{c}(B) = l$, and there exists $f : B \to \{1, 2, 3\}$ such that for all $i \in \{1, 2, \ldots, L\}$, either $\phi_B(i) = R[i] = 0$ or both $\phi_B(i) \neq 0$ and $f(\phi_B(i)) = R[i]$.*

Intuitively, for every vertex $B \in V(\widetilde{G})$, the function ϕ_B can be interpreted as follows. It is the assignment, for every possible color $i \in \{1, \ldots, L\}$, of the unique vertex u within the vertex set identified with B itself that is adjacent to some vertex in the vertex subset identified with some vertex $B' \in V(\widetilde{G})$ colored i, if such a vertex u exists (else the assignment is of 0). In a sense, B thus stores the information on the identity of each vertex within it that is adjacent (in G) to some vertex outside of it, where each such internal vertex is uniquely accessed by specifying the color of the vertex in $\widetilde{G}$ whose identified vertex set contains the *neighbor*. With respect to the graph $\widetilde{H}$ and labeling ℓ, we interpret each vertex $(R, l) \in V(\widetilde{H})$ as a "placeholder" (i.e. potential assignment of the sought function φ) for any vertex $B \in V(\widetilde{G})$ that "complies with the pattern encoded by the pair (R, l)" as follows. First and straightforwardly, B must be colored l. Here, we remind that the colors of vertices in $\widetilde{G}$ belong to $\{1, \ldots, L\}$, while vertices in G are colored 1, 2 or 3 only. Then, the second requirement is that we can recolor (by f) the vertices in B so that the color of each vertex in B that is adjacent (in G) to some vertex outside B is as encoded by the vector R – that is, for each color $i \in \{1, \ldots, L\}$, if the vertex $\phi_B(i)$ is defined (i.e., $\phi_B(i) \neq 0$), then its color (which is $1, 2$ or 3) must be equal to the i-th entry of R. (More intuition is given in Fig. 2.)

Now, we state the correctness of the reduction.

[1] The square G^2 of a graph G is the graph on vertex set $V(G)$ and edge set $\{\{u, v\} : \{u, v\} \in E(G)$ or there exists $w \in V(G)$ with $\{u, w\}, \{v, w\} \in E(G)\}$.

[2] The uniqueness of u (if it exists), and thus the validity of ϕ_B, follows from Properties 2 and 4 in Lemma 3.

[3] That is, R is a vector with L entries where each entry is 0, 1, 2 or 3.

▶ **Lemma 5** (Lemma 3.3 in [6]). *For any instance G of 3-Coloring where G is an n-vertex graph of degree d, and a positive integer $r = o(\sqrt{|V(G)|})$, the instance* $\texttt{reduce}(G) = (\widetilde{G}, \widetilde{H}, \ell)$ *is computable in time polynomial in the sizes of $G, \widetilde{G}$ and $\widetilde{H}$, and has the following properties.*

- *G is a Yes-instance of 3-Coloring if and only if $(\widetilde{G}, \widetilde{H}, \ell)$ is a Yes-instance of LSH.*
- *$|V(\widetilde{G})| \le n/r$, and $|V(\widetilde{H})| \le \gamma(d)^r$ where γ is some computable function of d.*

We next prove that we can add colorings to the instance $\texttt{reduce}(G) = (\widetilde{G}, \widetilde{H}, \ell)$ of LSH in order to cast it as an instance of Prop-Col LSH while making a minor mandatory modification to the graph $\widetilde{H}$.

▶ **Lemma 6.** *Given an instance* $\texttt{reduce}(G) = (\widetilde{G}, \widetilde{H}, \ell)$ *of LSH, an equivalent instance $(\widetilde{G}, \widetilde{H}', c_{\widetilde{G}}, c_{\widetilde{H}'}, \ell)$ of Prop-Col LSH, where $\widetilde{H}'$ is a subgraph of $\widetilde{H}$, is computable in polynomial time.*

Proof. Define $c_{\widetilde{G}} = \widetilde{c}$ where $\widetilde{c}$ is the coloring of $\widetilde{G}$ in Definition 4. Additionally, let $\widetilde{H}'$ be the subgraph of $\widetilde{H}$ induced by the vertex set $\{(R, l) \in V(\widetilde{H})$: there exists $B \in V(\widetilde{G})$ such that $(R, l) \in \ell(B)\}$. Then, define $c_{\widetilde{H}'} : V(\widetilde{H}') \to \{1, 2, \dots, L\}$ as follows: for any $(R, l) \in V(\widetilde{H}')$, define $c_{\widetilde{H}'}((R, l)) = l$. Notice that, by the definition of $V(\widetilde{H}')$, every set assigned by ℓ is subset of $V(\widetilde{H}')$.

First, we assert that $(\widetilde{G}, \widetilde{H}', c_{\widetilde{G}}, c_{\widetilde{H}'}, \ell)$ is an instance of Prop-Col LSH. To this end, we need to verify that the three following properties hold.

1. $c_{\widetilde{G}}$ is a proper coloring of $\widetilde{G}$.
2. $c_{\widetilde{H}'}$ is a proper coloring of $\widetilde{H}'$.
3. For every $B \in V(\widetilde{G})$ and $(R, l) \in \ell(B)$, it holds that $c_{\widetilde{G}}(B) = c_{\widetilde{H}'}((R, l))$.

By the definition of $c_{\widetilde{G}}$, it is a proper coloring of $\widetilde{G}^2$, which is a supergraph of $\widetilde{G}$. Thus, $c_{\widetilde{G}}$ is a proper coloring of $\widetilde{G}$.

Now, we argue that $c_{\widetilde{H}'}$ is a proper coloring of $\widetilde{H}'$. To this end, consider some edge $\{(R, l), (R', l')\} \in E(\widetilde{H}')$. We need to show that $c_{\widetilde{H}'}((R, l)) \neq c_{\widetilde{H}'}((R', l'))$. By the definition of $c_{\widetilde{H}'}$, we have that $c_{\widetilde{H}'}((R, l)) = l$ and $c_{\widetilde{H}'}((R', l')) = l'$, and therefore it suffices to show that $l \neq l'$. By the definition of $E(\widetilde{H})$ (which is a superset of $E(\widetilde{H}')$), we have that $R[l'] \neq R'[l]$. Thus, necessarily at least one among $R[l']$ and $R'[l]$ is not 0, and so we suppose w.l.o.g. that $R[l']$ is not 0. Furthermore, since $(R, l) \in V(\widetilde{H}')$, we have that there exists $B \in E(\widetilde{G})$ such that $(R, l) \in \ell(B)$. Thus,

- $\widetilde{c}(B) = l$.
- There exists $f : B \to \{1, 2, 3\}$ such that for all $i \in \{1, 2, \dots, L\}$, either $\phi_B(i) = R[i] = 0$ or both $\phi_B(i) \neq 0$ and $f(\phi_B(i)) = R[i]$.

From the second property, and because $R[l'] \neq 0$, we necessarily have that both $\phi_B(l') \neq 0$ and $f(\phi_B(l')) = R[l']$. In particular, by the definition of ϕ_B, having $\phi_B(l') \neq 0$ means that there exists (u, v, B') such that $u \in B$, $v \in B'$, $\{u, v\} \in E(G)$ and $\widetilde{c}(B') = l'$. By the definition of $\widetilde{G}$ as a grouping of G, having $u \in B$, $v \in B'$ and $\{u, v\} \in E(G)$ implies that $\{B, B'\} \in E(\widetilde{G})$. Because $\widetilde{c}$ is a proper coloring of $\widetilde{G}$, this means that $\widetilde{c}(B) \neq \widetilde{c}(B')$. Since $\widetilde{c}(B) = l$ and $\widetilde{c}(B') = l'$, we derive that $l \neq l'$. Hence, $c_{\widetilde{H}'}$ is indeed a proper coloring of $\widetilde{H}'$.

To conclude that $(\widetilde{G}, \widetilde{H}', c_{\widetilde{G}}, c_{\widetilde{H}'}, \ell)$ is indeed an instance of Prop-Col LSH, it remains to assert that for every $B \in V(\widetilde{G})$ and $(R, l) \in \ell(B)$, it holds that $c_{\widetilde{G}}(B) = c_{\widetilde{H}'}((R, l))$. To this end, consider some $B \in V(\widetilde{G})$ and $(R, l) \in \ell(B)$. By the definition of ℓ (recall Definition 4), $(R, l) \in \ell(B)$ implies that $\widetilde{c}(B) = l$. As $c_{\widetilde{G}} = \widetilde{c}$, we have that $c_{\widetilde{G}}(B) = l$. Moreover, the definition of $c_{\widetilde{H}'}$ directly implies that $c_{\widetilde{H}'}((R, l)) = l$. Thus, $c_{\widetilde{G}}(B) = c_{\widetilde{H}'}((R, l))$.

Finally, we argue that $(\widetilde{G}, \widetilde{H}, \ell)$ is a Yes-instance of LSH if and only if $(\widetilde{G}, \widetilde{H}', c_{\widetilde{G}}, c_{\widetilde{H}'}, \ell)$ is a Yes-instance of Prop-Col LSH. In one direction, because $\widetilde{H}'$ is a subgraph of $\widetilde{H}$, it is immediate that if $(\widetilde{G}, \widetilde{H}', c_{\widetilde{G}}, c_{\widetilde{H}'}, \ell)$ is a Yes-instance of Prop-Col LSH, then so is $(\widetilde{G}, \widetilde{H}, \ell)$. For the other direction, suppose that $(\widetilde{G}, \widetilde{H}, \ell)$ is a Yes-instance of LSH. Thus, there exists a function $\varphi : V(\widetilde{G}) \to V(\widetilde{H})$ such that *(i)* for every $\{B, B'\} \in E(\widetilde{G})$, $\{\varphi(B), \varphi(B')\} \in E(\widetilde{H})$, and *(ii)* for every $B \in V(\widetilde{G})$, $\varphi(B) \in \ell(B)$. In particular, directly by the definition of $V(\widetilde{H}')$, the second condition implies that for every $B \in V(\widetilde{G})$, it holds that $\varphi(B) \in V(\widetilde{H}')$. Thus, because $\widetilde{H}'$ is an induced subgraph of $\widetilde{H}$, it holds that for every $\{B, B'\} \in E(\widetilde{G})$, $\{\varphi(B), \varphi(B')\} \in E(\widetilde{H}')$. Therefore, φ witnesses that $(\widetilde{G}, \widetilde{H}', c_{\widetilde{G}}, c_{\widetilde{H}'}, \ell)$ is a Yes-instance of Prop-Col LSH. ◀

We are now ready to assert the hardness of Prop-Col LSH. The proof, based on Lemmas 3, 5 and 6, can be found in the full version of this paper.

▶ **Lemma 7.** *Unless the ETH is false, there does not exist an algorithm that solves* Prop-Col *LSH in time $n^{o(n)}$ where $n = \max(|V(G)|, |V(H)|)$.*

From Graph Homomorphism to Subgraph Isomorphism. In this part, we observe that the reduction of [6] from LSH to LSI can be essentially used as is to serve as a reduction from Prop-Col LSH to Prop-Col LSI. For the sake of completeness, we give the full details (and the conclusion of the proof of Lemma 2) in the full version of this paper.

3 Lower Bound for the Cross Matching Problem

In this section, towards the proof of a lower bound for Clique Contraction, we prove a lower bound for an intermediate problem called Cross Matching that somewhat resembles Clique Contraction, and which is defined as follows.

Cross Matching
Input: A graph G with a partition (A, B) of $V(G)$ where $|A| = |B|$.
Question: Does there exist a perfect matching M in G such that every edge in M has one endpoint in A and the other in B, and G/M is a clique?

Our objective is to prove the following statement.

▶ **Lemma 8.** *Unless the ETH is false, there does not exist an algorithm that solves* Cross Matching *in time $n^{o(n)}$ where $n = |A|$.*

Proof. Towards a contradiction, suppose that there exists an algorithm, denoted by MatchingAlg, that solves Cross Matching in time $n^{o(n)}$ where n is the number of vertices in the set A in the input. We will show that this implies the existence of an algorithm, denoted by LSIAlg, that solves Prop-Col LSI in time $n^{o(n)}$ where n is the number of vertices in the input graph G, thereby contradicting Lemma 2 and hence completing the proof.

We define the execution of LSIAlg as follows. Given an instance (G, H, c_G, c_H, ℓ) of Prop-Col LSI, LSIAlg constructs an instance (L, A, B) of Cross Matching as follows (see Fig. 3):

- $V(L) = V(\overline{G}) \cup V(H)$.
- $E(L) = E(\overline{G}) \cup E(H) \cup \{\{u, v\} : u \in V(G), v \in L(u)\}$.
- $A = V(\overline{G})$ and $B = V(H)$.

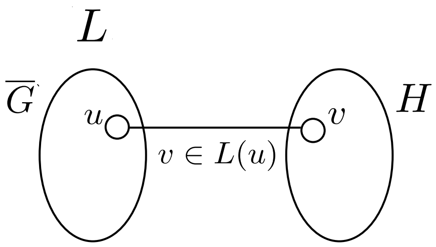


Figure 3 The construction of an instance of Cross Matching in the proof of Lemma 8.

Then, LSIAlg calls MatchingAlg with (L, A, B) as input, and returns the answer of this call.

Denote $n = |V(G)|$, and notice that $|A| = |B| = n$. Thus, because MatchingAlg runs in time $|A|^{o(|A|)} = n^{o(n)}$, so does LSIAlg.

For the correctness of the algorithm, first suppose that (G, H, c_G, c_H, ℓ) is a Yes-instance of Prop-Col LSI. This means that there exists a bijective function $\varphi : V(G) \to V(H)$ such that *(i)* for every $\{u, v\} \in E(G)$, $\{\varphi(u), \varphi(v)\} \in E(H)$, and *(ii)* for every $u \in V(G)$, $\varphi(u) \in L(u)$. Having φ at hand, we will show that (L, A, B) is a Yes-instance, which will imply that the call to MatchingAlg with (L, A, B) as input returns Yes, and hence LSIAlg returns Yes.

Based on φ, we define a subset $M \subseteq E(L)$ as follows: $M = \{\{u, \varphi(u)\} : u \in A\}$. Notice that the containment of M in $E(L)$ follows from the definition of $E(L)$ and Condition *(ii)* above. Moreover, by the definition of A, B and because φ is bijective, it further follows that M is a perfect matching in L such that every edge in M has one endpoint in A and the other in B. Thus, to conclude that (L, A, B) is a Yes-instance, it remains to argue that L/M is a clique. To this end, we consider two arbitrary vertices x and y of L/M, and prove that they are adjacent in L/M. Necessarily x is a vertex that replaced two vertices $u \in A$ and $u' \in B$ such that $\{u, u'\} \in M$, and y is a vertex that replaced two vertices $v \in A \setminus \{u\}$ and $v' \in B \setminus \{u'\}$ such that $\{v, v'\} \in M$. By the definition of contraction, to show that x and y are adjacent in L/M, it suffices to show that u and v are adjacent in L or u' and v' are adjacent in L (or both). To this end, suppose that u and v are not adjacent in L, else we are done. By the definition of $E(L)$, this means that $\{u, v\} \notin E(\overline{G})$ and hence $\{u, v\} \in E(G)$. By Condition *(i)* above, we derive that $\{\varphi(u), \varphi(v)\} \in E(H)$. By the definition of M, we know that $u' = \varphi(u)$ and $v' = \varphi(v)$, therefore $\{u', v'\} \in E(H)$. In turn, by the definition of $E(L)$, we get that $\{u', v'\} \in E(L)$. Thus, the proof of the forward direction is complete.

Now, suppose that LSIAlg returns Yes, which means that the call to MatchingAlg with (L, A, B) returns Yes. Thus, (L, A, B) is a Yes-instance, which means that there exists a perfect matching M in G such that every edge in M has one endpoint in A and the other in B, and G/M is a clique. We define a function $\varphi : A \to B$ as follows. For every $u \in V(G)$, let $\varphi(u) = v$ where v is the unique vertex in B such that $\{u, v\} \in M$; the existence and uniqueness of v follows from the supposition that M is a perfect matching such that every edge in M has one endpoint in A and the other in B. Furthermore, by the definition of A, B and the edges in $E(L)$ with one endpoint in A and the other in B, it directly follows that φ is a bijective mapping between $V(G)$ and $V(H)$ such that for every $u \in V(G)$, it holds that $\varphi(u) \in L(u)$. Thus, it remains to argue that for every edge $\{u, v\} \in E(G)$, it holds that $\{\varphi(u), \varphi(v)\} \in E(H)$. To this end, consider some arbitrary edge $\{u, v\} \in E(G)$, and denote $u' = \varphi(u)$ and $v' = \varphi(v)$. Because L/M is a clique and M is a matching that, by the definition of φ, necessarily contains both $\{u, u'\}$ and $\{v, v'\}$, we derive that at least one of the following four conditions must be satisfied: *(i)* $\{u, v\} \in E(L)$; *(ii)* $\{u', v'\} \in E(L)$; *(iii)*

$\{u,v'\} \in E(L)$; *(iv)* $\{v,u'\} \in E(L)$. Because $\{u,v\} \in E(G)$, we have that $\{u,v\} \notin E(\overline{G})$ and therefore $\{u,v\} \notin E(L)$. Thus, we are left with Conditions *(ii)*, *(iii)* and *(iv)*. Now, we will crucially rely on the proper colorings of G and H to rule out the satisfaction of Conditions *(iii)* and *(iv)*.

▷ Claim 9. For any two edges $\{x,x'\},\{y,y'\} \in E(L)$ such that $\{x,y\} \in E(G)$ and $x',y' \in V(H)$, it holds that neither $\{x,y'\}$ nor $\{y,x'\}$ belongs to $E(L)$.

Proof of Claim 9. Because c_G is a proper coloring of G and $\{x,y\} \in E(G)$, it holds that $c_G(x) \neq x_G(y)$. Because $\{x,x'\},\{y,y'\} \in E(L)$, $x,y \in V(G)$ and $x',y' \in V(H)$, and by the definition of $E(L)$, it holds that $x' \in L(x)$ and $y' \in L(y)$, and therefore $c_G(x) = c_H(x')$ and $c_G(y) = c_H(y')$. Thus, $c_G(x) \neq c_H(y')$ and $c_G(y) \neq c_H(x')$, implying that $y' \notin L(x)$ and $x' \notin L(y)$. In turn, by the definition of $E(L)$, this means that neither $\{x,y'\}$ nor $\{y,x'\}$ belongs to $E(L)$. This completes the proof of the claim. ◁

We now return to the proof of the lemma. By Claim 9, we are only left with Condition *(ii)*, that is, $\{u',v'\} \in E(L)$. However, by the definition of $E(L)$, this means that $\{u',v'\} \in E(H)$. As argued earlier, this completes the proof of the reverse direction. ◀

4 Lower Bounds: Clique Contraction and Hadwiger Number

In this section, we prove a lower bound for Clique Contraction and consequently for Hadwiger Number, defined as follows.

Clique Contraction
Input: A graph G and $t \in \mathbb{N}$.
Question: Is there a subset $F \subseteq E(G)$ of size at most t such that G/F is a clique?

Hadwiger Number
Input: A graph G and $h \in \mathbb{N}$.
Question: Is the Hadwiger number of G at least as large as h?

Our objective is to prove the following statement, where the analogous statement for Hadwiger Number (called Theorem 1 in the introduction) will follow as a corollary.

▶ **Theorem 10.** *Unless the ETH is false, there does not exist an algorithm that solves Clique Contraction in time $n^{o(n)}$ where $n = |V(G)|$.*

To make our approach adaptable to extract analogous statements for other contraction problems, we will first define a new problem called Noisy Structured Clique Contraction (which will arise in Section 5) along with a special case of it that is also a special case of Clique Contraction. Then, we will prove a crucial property of instances of Noisy Structured Clique Contraction, and afterwards we will use this property to prove Theorem 10 and its corollary. The definition of the new problem is as follows (see Fig. 4).

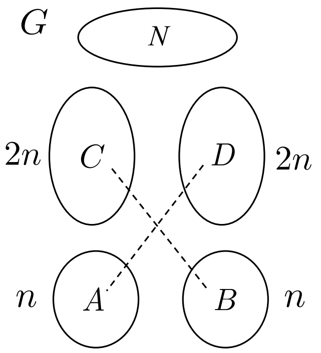


Figure 4 An instance of Noisy Structured Clique Contraction where dashed lines represent non-edges.

Noisy Structured Clique Contraction

Input: A graph G on at least $6n$ vertices for some $n \in \mathbb{N}$, and a partition (A, B, C, D, N) of $V(G)$ such that $|A| = |B| = n$, $|C| = |D| = 2n$, no vertex in A is adjacent to any vertex in D, and no vertex in B is adjacent to any vertex in C.

Question: Does there exist a subset $F \subseteq E(G)$ of size at most n such that $G[A \cup B \cup C \cup D \cup X]/F$ is a clique,[a] where $X = \{u \in N :$ there exists a vertex $v \in A \cup B \cup C \cup D$ such that u and v belong to the same connected component of $G[F]\}$?

[a] Note that F might contain edges outside $G[A \cup B \cup C \cup D \cup X]$. Then, we slightly abuse notation so that $G[A \cup B \cup C \cup D \cup X]/F$ refers to $G[A \cup B \cup C \cup D \cup X]/(F \cap E(G[A \cup B \cup C \cup D \cup X]))$.

Intuitively, the vertex set X consists of the noise (represented by N) that "interacts" with non-noise (represented by $V(G) \setminus N$) through contracted edges (in F), i.e. the vertices in N that lie together with at least one vertex in $V(G) \setminus N$ in a component that will be contracted and thereby replaced by a single vertex. We refer to the special case of Noisy Structured Clique Contraction where $N = \emptyset$ as Structured Clique Contraction. Note that Structured Clique Contraction is also a special case of Clique Contraction.

Solutions to instances of Noisy Structured Clique Contraction exhibit the following property, which will be crucial in the proof of Theorem 10 as well as results in Section 5.

▶ **Lemma 11.** *Let F be a solution to an instance (G, A, B, C, D, N, n) of Noisy Structured Clique Contraction. Then, F is a matching of size n in G such that each edge in F has one endpoint in A and the other in B.*

Proof. We first argue that every vertex in $A \cup B$ is incident to at least one edge in F. Targeting a contradiction, suppose that there exists a vertex $u \in A \cup B$ that is not incident to any edge in F. Because $|A \cup B \cup C \cup D| = 6n$, $|F| \leq n$ and $G[A \cup B \cup C \cup D \cup X]/F$ is a clique (where the last two properties follow from the supposition that F is a solution), it holds that $G[A \cup B \cup C \cup D \cup X]/F$ is a clique on at least $5n + |X|$ vertices. Hence, the degree of every vertex in $G[A \cup B \cup C \cup D \cup X]/F$, and in particular of u, should be $5n - 1 + |X|$ in $G[A \cup B \cup C \cup D \cup X]/F$. However, because no vertex in A is adjacent to any vertex in D and no vertex in B is adjacent to any vertex in C, the degree of any vertex in $A \cup B$, and in particular of u, is at most $|A \cup B| - 1 + |C \cup D|/2 + |X| = 4n - 1 + |X|$ in $G[A \cup B \cup C \cup D \cup X]$. Because u is not incident to any edge in F, its degree in $G[A \cup B \cup C \cup D \cup X]/F$ is at most its degree in $G[A \cup B \cup C \cup D \cup X]$. This is a contradiction, thus we get that indeed every vertex in $A \cup B$ is incident to at least one edge in F. From this, because $|F| \leq n$ and $|A \cup B| = 2n$, we derive that F is a perfect matching in $G[A \cup B]$.

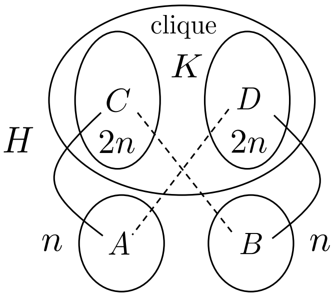


Figure 5 The construction of an instance of Structured Clique Contraction in the proof of Lemma 12 where dashed lines represent non-edges.

It remains to argue that every edge in F has one endpoint in A and the other in B. Targeting a contradiction, suppose that this is false. Because F is a perfect matching in $G[A \cup B]$, this means that there exist two vertices $a, a' \in A$ such that $\{a, a'\} \in F$. By the definition of Noisy Structured Clique Contraction, neither a nor a' is adjacent to any vertex in D. Moreover, note that $D \subseteq V(G[A \cup B \cup C \cup D \cup X]/F)$. In particular, the vertex of $G[A \cup B \cup C \cup D \cup X]/F$ yielded by the contraction of $\{a, a'\}$ is not adjacent to any vertex of D in $G[A \cup B \cup C \cup D \cup X]/F$. However, this is a contradiction because $G[A \cup B \cup C \cup D \cup X]/F$ is a clique. ◀

We now prove a lower bound for Structured Clique Contraction. Because it is a special case of Clique Contraction, this will directly yield the correctness of Theorem 10.

▶ **Lemma 12.** *Unless the ETH is false, there does not exist an algorithm that solves Structured Clique Contraction in time $n^{o(n)}$ where $n = |V(G)|$.*

Proof. Targeting a contradiction, suppose that there exists an algorithm, denoted by CliConAlg, that solves Structured Clique Contraction in time $n^{o(n)}$ where n is the number of vertices in the input graph. We will show that this implies the existence of an algorithm, denoted by MatchingAlg, that solves Cross Matching in time $n^{o(n)}$ where n is the size of the set A in the input, thereby contradicting Lemma 8 and hence completing the proof.

We define the execution of MatchingAlg as follows. Given an instance (G, A, B) of Cross Matching, MatchingAlg constructs an instance (H, A, B, C, D, n) of Structured Clique Contraction as follows (see Fig. 5):

- Let $n = |A|$, and K be a clique on $4n$ new vertices. Let (C, D) be a partition of $V(K)$ such that $|C| = |D|$.
- $V(H) = V(G) \cup V(K)$.
- $E(H) = E(G) \cup E(K) \cup \{\{a, c\} : a \in A, c \in C\} \cup \{\{b, d\} : b \in B, d \in D\}$.

Then, MatchingAlg calls CliConAlg with (H, A, B, C, D, n) as input, and returns the answer.

First, note that by construction, $|V(H)| = 6n$. Thus, because CliConAlg runs in time $|V(H)|^{o(|V(H)|)} \leq n^{o(n)}$, it follows that MatchingAlg runs in time $n^{o(n)}$.

For the correctness of the algorithm, first suppose that (G, A, B) is a Yes-instance of Cross Matching. This means that there exists a perfect matching M in G such that every edge in M has one endpoint in A and the other in B, and G/M is a clique. By the definition of $E(H)$, $M \subseteq E(H)$. We will show that H/M is a clique. As $|M| = n$, this will mean that (H, A, B, C, D, n) is a Yes-instance of Structured Clique Contraction, which will mean, in turn, that the call to CliConAlg with (H, A, B, C, D, n) as input returns Yes, and hence MatchingAlg returns Yes.

Note that $V(H/M) = V(K) \cup V(G/M)$. To show that H/M is a clique, we consider two arbitrary vertices $u, v \in V(H/M)$, and show that they are adjacent in H/M. If $u, v \in V(K)$, then because K is a clique, it is clear that $\{u, v\} \in E(H/M)$. Moreover, if $u, v \in G/M$, then because G/M is a clique, it is clear that $\{u, v\} \in E(H/M)$. Thus, one of the vertices u and v belongs to $V(G/M)$ and the other belongs to $V(K)$. We suppose w.l.o.g. that $u \notin V(K)$. Because M is a perfect matching in G such that every edge in M has one endpoint in A and the other in B, it follows that u resulted from the contraction of the edge between some $a \in A$ and some $b \in B$. If $v \in C$, then $\{a, v\} \in E(H)$, and otherwise $v \in D$ and so $\{b, v\} \in E(H)$. Thus, by the definition of contraction, we conclude that $\{u, v\} \in E(H/M)$. This completes the proof of the forward direction.

Now, suppose that MatchingAlg returns Yes, which means that the call to CliConAlg with (H, A, B, C, D, n) returns Yes. Thus, (H, A, B, C, D, n) is a Yes-instance, which means that there exists a subset $F \subseteq E(H)$ of size at most n such that H/F is a clique. We will show that F is a perfect matching in G such that every edge in F has one endpoint in A and the other in B. Because H/F is a clique, this will imply that G/F is a clique and thus that (G, A, B) is a Yes-instance of Cross Matching. To achieve this, notice that by Lemma 11, F is a matching of size n in H such that each edge in F has one endpoint in A and the other in B. Because $G = H[A \cup B]$, we have that F is a perfect matching in G. Thus, the proof of the reverse direction is complete. ◀

▶ **Corollary 13.** *Unless the ETH is false, there does not exist an algorithm that solves Hadwiger Number in time $n^{o(n)}$ where $n = |V(G)|$.*

Proof. Targeting a contradiction, suppose that there exists an algorithm, denoted by HadwigerAlg, that solves Hadwiger Number in time $n^{o(n)}$ where n is the number of vertices in the input graph. We will show that this implies the existence of an algorithm, denoted by CliConAlg, that solves Clique Contraction in time $n^{o(n)}$ where n is the number of vertices in the input graph, thereby contradicting Theorem 10 and hence completing the proof.

We define the execution of CliConAlg as follows. Given an instance (G, t) of Clique Contraction, if G is not connected, then CliConAlg returns No, and otherwise it returns Yes if and only if HadwigerAlg returns Yes when called with $(G, |V(G)| - t)$ as input. Because the call to HadwigerAlg with input $(G, |V(G)| - t)$ runs in time $n^{o(n)}$ where $n = |V(G)|$, we have that CliConAlg runs in time $n^{o(n)}$ as well.

For the correctness of the algorithm, first observe that if G is not connected, then no sequence of edge contractions can yield a clique, and hence it is correct to return No. Thus, now assume that G is connected. First, suppose that (G, t) is a Yes-instance of Clique Contraction. This means that there exists a sequence of at most t edge contractions that transforms G into a clique. In particular, this clique must have at least $|V(G)| - t$ vertices, and therefore the Hadwiger number of G is at least as large as $|V(G)| - t$. By the correctness of HadwigerAlg, its call with $(G, |V(G)| - t)$ returns Yes, and therefore CliConAlg returns Yes.

Now, suppose that CliConAlg returns Yes, which means that the call to HadwigerAlg with $(G, |V(G)| - t)$ returns Yes. By the correctness of HadwigerAlg, the clique K_h for $h = |V(G)| - t$ is a minor of G. This means that there is a sequence of vertex deletions, edge deletions and edge contractions that transforms G into K_h. In particular, this sequence can contain at most t vertex deletions and edge contractions in total. Furthermore, by replacing each vertex deletion for a vertex v by an edge contraction for some edge e incident to v (which exists because G is connected) and dropping all edge deletions, we obtain another sequence that transforms G into K_h. Because this sequence contains only edge contractions, and at most t of them, we conclude that (G, t) is a Yes-instance of Clique Contraction. ◀

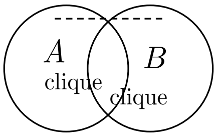


Figure 6 A two-cliques graph (see Definition 15).

5 Lower Bounds for Contraction to Graph Classes Problems

In this section, we prove lower bounds for several cases of the $\mathcal{F}$-CONTRACTION problem, defined as follows. Here, $\mathcal{F}$ is a (possibly infinite) family of graphs.

$\mathcal{F}$-CONTRACTION
Input: A graph G and $t \in \mathbb{N}$.
Question: Does there exist a subset $F \subseteq E(G)$ of size at most t such that $G/F \in \mathcal{F}$?

Notice that CLIQUE CONTRACTION is the case of $\mathcal{F}$-CONTRACTION where $\mathcal{F}$ is the family of cliques. In this section, we consider the cases of $\mathcal{F}$-CONTRACTION where $\mathcal{F}$ is the family of chordal graphs, interval graphs, proper interval graphs, threshold graphs, trivially perfect graphs, split graphs, complete split graphs and perfect graphs, also called CHORDAL CONTRACTION, INTERVAL CONTRACTION, PROPER INTERVAL CONTRACTION, THRESHOLD CONTRACTION, TRIVIALLY PERFECT CONTRACTION, SPLIT CONTRACTION, COMPLETE SPLIT CONTRACTION and PERFECT CONTRACTION, respectively. Before we define these classes formally, it will be more enlightening to first define only the class of chordal graphs as well as somewhat artificial classes of graphs that will help us prove lower bounds for many of the classes above in a unified manner.

▶ **Definition 14** (Chordal Graphs). *A graph is* chordal *if it does not contain C_ℓ for all $\ell \geq 4$ as an induced subgraph.*

Our first class of graphs is defined as follows (see Fig. 6).

▶ **Definition 15** (Two-Cliques Graphs). *A* two-cliques graph *is a graph G such that there exist $A, B \subseteq V(G)$ such that $A \cup B = V(G)$, $G[A]$ and $G[B]$ are cliques, and there do not exist vertices $a \in A \setminus B$ and $b \in B \setminus A$ such that $\{a, b\} \in E(G)$. The* two-cliques class *is the class of all two-cliques graphs.*

It should be clear that the two-cliques class is a subclass of the class of chordal graphs. Now, we further define a family of classes of graphs as follows.

▶ **Definition 16** (Non-Trivial Chordal Class). *We say that a class of graphs $\mathcal{F}$ is* non-trivial chordal *if it is a subclass of the class of chordal graphs, and a superclass of the two-cliques class.*

Clearly, the class of cliques is not a non-trivial chordal class, and the class of chordal graphs is a non-trivial chordal class. The rest of this section is divided as follows. First, in Section 5, we prove a lower bound for any non-trivial chordal class. Then, in Section 5, we prove a lower bound for some graph classes that are not non-trivial chordal.

Non-Trivial Chordal Graph Classes. Here, our objective is to prove the following theorem. Afterwards, we will derive lower bounds for several known graph classes as corollaries.

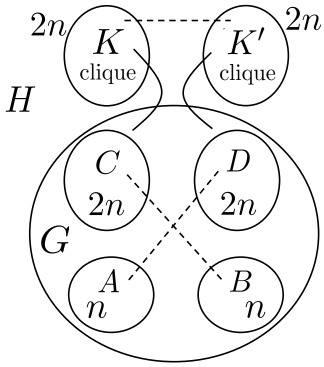


Figure 7 The construction of an instance of $\mathcal{F}$-Contraction in the proof of Theorem 17 where dashed lines represent non-edges.

▶ **Theorem 17.** *Let $\mathcal{F}$ be any non-trivial chordal graph class. Unless the ETH is false, there does not exist an algorithm that solves $\mathcal{F}$-Contraction in time $n^{o(n)}$ where $n = |V(G)|$.*

For the proof of this theorem, the following well-known property of chordal graphs will come in handy. This property is a direct consequence of the alternative characterization of the class of chordal graphs as the class of graphs that admit clique-tree decompositions, see [10].

▶ **Proposition 18.** *Let G be a chordal graph, and let u and v be two non-adjacent vertices in G. Then, $G[N(u) \cap N(v)]$ is a clique.*

We are now ready to prove Theorem 17.

Proof of Theorem 17. Targeting a contradiction, suppose that there exists an algorithm, denoted by NonTrivChordAlg, that solves $\mathcal{F}$-Contraction in time $n^{o(n)}$ where n is the number of vertices in the input graph. We will show that this implies the existence of an algorithm, denoted by CliConAlg, that solves Structured Clique Contraction in time $n^{o(n)}$ where n is the number of vertices in the input graph, thereby contradicting Lemma 12 and hence completing the proof.

We define the execution of CliConAlg as follows. Given an instance (G, A, B, C, D, n) of Structured Clique Contraction, CliConAlg constructs an instance (H, n) of $\mathcal{F}$-Contraction as follows (see Fig. 7):

- Let $n = |A|$. Moreover, let K and K' be two cliques, each on $2n$ new vertices.
- $V(H) = V(G) \cup V(K) \cup V(K')$.
- $E(H) = E(G) \cup E(K) \cup E(K') \cup \{\{u, v\} : u \in V(G), v \in V(K) \cup V(K')\}$.

Then, CliConAlg calls NonTrivChordAlg with (H, n) as input, and returns the answer of this call.

First, note that by construction, $|V(H)| = 10n$. Thus, because NonTrivChordAlg runs in time $|V(H)|^{o(|V(H)|)} \leq n^{o(n)}$, it follows that CliConAlg runs in time $n^{o(n)}$.

For the correctness of the algorithm, first suppose that (G, A, B, C, D, n) is a Yes-instance of Structured Clique Contraction. This means that there exists a subset $F \subseteq E(G)$ of size at most n such that G/F is a clique. By the definition of H, we directly derive that H/F is a two-cliques graphs, and therefore it belongs to $\mathcal{F}$. Thus, (H, n) is a Yes-instance of $\mathcal{F}$-Contraction, which means that the call to NonTrivChordAlg with (H, n) as input returns Yes, and hence CliConAlg returns Yes.

Now, suppose that CliConAlg returns Yes, which means that the call to NonTrivChordAlg with (H, n) returns Yes. Thus, (H, n) is a Yes-instance of $\mathcal{F}$-Contraction, which means that there exists a subset $F \subseteq E(H)$ of size at most n such that $H/F \in \mathcal{F}$. In particular, H/F

is a chordal graph. Based on Proposition 18, we will first show that $H[A \cup B \cup C \cup D \cup X]/F$ is a clique, where $X = \{u \in V(K) \cup V(K') :$ there exists a vertex $v \in A \cup B \cup C \cup D$ such that u and v belong to the same connected component of $H[F]\}$.

Targeting a contradiction, suppose that $H[A \cup B \cup C \cup D \cup X]/F$ is not a clique, and therefore there exist two non-adjacent vertices u and v in this graph. By the definition of X, $H[A \cup B \cup C \cup D \cup X]/F$ is equal to the subgraph of H/F induced by the set of vertices derived from connected components that contain at least one vertex from $A \cup B \cup C \cup D$. In particular, u and v are also non-adjacent vertices in H/F. By Proposition 18, this implies that $(H/F)[N_{H/F}(u) \cap N_{H/F}(v)]$ is a clique. Let $\mathcal{C}_1$ (resp. $\mathcal{C}_2$) be the set of connected components of $H[F]$ that contain at least one vertex from $V(K_1)$ (resp. $V(K_2)$). Because $|F| \leq n$ and $|V(K_1)| = |V(K_2)| = 2n$, there exists at least one component $C_1 \in \mathcal{C}_1$ (resp. $C_2 \in \mathcal{C}_2$) that does not contain any vertex from $A \cup B \cup C \cup D$. Let c_1 and c_2 be the vertices of H/F yielded by the replacement of C_1 and C_2, respectively. As all vertices in $V(K_1) \cup V(K_2)$ are adjacent to all vertices in $A \cup B \cup C \cup D$, we have that $c_1, c_2 \in N_{H/F}(u) \cap N_{H/F}(v)$. However, there do not exist a vertex in $V(K_1)$ and a vertex in $V(K_2)$ that are adjacent in H, and for every vertex in $V(K_1) \cup V(K_2)$, its neighborhood outside this set is contained in $A \cup B \cup C \cup D$. Thus, c_1 and c_2 must be non-adjacent in H/F. However, this is a contradiction to the argument that $(H/F)[N_{H/F}(u) \cap N_{H/F}(v)]$ is a clique. From this, we derive that $H[A \cup B \cup C \cup D \cup X]/F$ is indeed a clique.

Now, notice that (H, A, B, C, D, N, n) where $N = V(K_1) \cup V(K_2)$ is an instance of Noisy Structured Clique Contraction. Furthermore, since $|F| \leq n$ and we have already shown that $H[A \cup B \cup C \cup D \cup X]/F$ is a clique, we have that F is a solution to this instance. Therefore, by Lemma 11, F is a matching of size n in H such that each edge in F has one endpoint in A and the other in B. In particular, $F \subseteq E(G)$ and hence $X = \emptyset$. Because $G = H[A \cup B \cup C \cup D]$, we thus derive that G/F is a clique. Thus, we conclude that (G, A, B, C, D, n) is a Yes-instance of Structured Clique Contraction. This completes the proof of the reverse direction. ◀

Now, we give definitions for several classes of graphs for which lower bounds will follow from Theorem 18. First, a graph is an *interval graph* if there exists a set of intervals on the real line such that the vertices of the graph are in bijection with these intervals, and there exists edge between two vertices if and only if their intervals intersect. A graph is a *proper interval graph* if, in the former definition, we also add the constraint that all intervals must have the same length. A graph is a *threshold graph* if it can be constructed from a one-vertex graph by repeated applications of the following two operations: addition of a single isolated vertex to the graph; addition of a single vertex that is connected to all other vertices. A graph is *trivially perfect* if in each of its induced subgraphs, the maximum size of an independent set equals the number of maximal cliques.

It is well-known that every graph that is a (proper) interval graph, or a threshold graph, or a trivially perfect graph, is also a chordal graph (see [10]). Moreover, it is immediate to verify that the two-cliques class is a subclass of the classes of (proper) interval graphs, threshold graphs and trivially perfect graphs. Thus, these classes are non-trivial chordal graphs classes, and therefore Theorem 17 directly implies lower bounds for them:

▶ **Corollary 19.** *Unless the ETH is false, none of the following problems admits an algorithm that solves it in time $n^{o(n)}$ where $n = |V(G)|$: Chordal Contraction, Interval Contraction, Proper Interval Contraction, Threshold Contraction and Trivially Perfect Contraction.*

Other Graph Classes. In Section 4, we proved a lower bound for a class of graphs that is not non-trivial chordal, namely, the class of cliques. In the full version of this paper, we show that our approach can yield lower bounds also for other classes of graphs that are not non-trivially chordal, including the classes of Split Graphs, Complete Split Graphs and Perfect Graphs.

References

1 Akanksha Agrawal, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. Split contraction: The untold story. In *34th Symposium on Theoretical Aspects of Computer Science (STACS 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

2 Andreas Björklund. Determinant sums for undirected hamiltonicity. *SIAM J. Comput.*, 43(1):280–299, 2014. doi:10.1137/110839229.

3 Andreas Björklund, Thore Husfeldt, and Mikko Koivisto. Set partitioning via inclusion–exclusion. *SIAM J. Computing*, 39(2):546–563, 2009.

4 B. Bollobás, P. A. Catlin, and P. Erdős. Hadwiger's conjecture is true for almost every graph. *European J. Combin.*, 1(3):195–199, 1980. doi:10.1016/S0195-6698(80)80001-1.

5 Jianer Chen, Benny Chor, Michael R. Fellows, Xiuzhen Huang, David Juedes, Iyad A. Kanj, and Ge Xia. Tight lower bounds for certain parameterized NP-hard problems. *Information and Computation*, 201(2):216–231, 2005. doi:10.1016/j.ic.2005.05.001.

6 Marek Cygan, Fedor V. Fomin, Alexander Golovnev, Alexander S. Kulikov, Ivan Mihajlin, Jakub Pachocki, and Arkadiusz Socala. Tight lower bounds on graph embedding problems. *J. ACM*, 64(3):18:1–18:22, 2017. doi:10.1145/3051094.

7 Rodney G. Downey and Michael R. Fellows. *Parameterized complexity.* Springer-Verlag, New York, 1999.

8 Fedor V. Fomin and Dieter Kratsch. *Exact Exponential Algorithms.* Springer, 2010. An EATCS Series: Texts in Theoretical Computer Science.

9 Fedor V. Fomin, Daniel Lokshtanov, Ivan Mihajlin, Saket Saurabh, and Meirav Zehavi. Computation of hadwiger number and related contraction problems: Tight lower bounds. *CoRR*, abs/2004.11621, 2020. arXiv:2004.11621.

10 Martin Charles Golumbic. *Algorithmic Graph Theory and Perfect Graphs.* North-Holland Publishing Co., Amsterdam, The Netherlands, The Netherlands, 2004.

11 Martin Grohe, Ken-ichi Kawarabayashi, Dániel Marx, and Paul Wollan. Finding topological subgraphs is fixed-parameter tractable. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 479–488, 2011.

12 Eugene L. Lawler. A note on the complexity of the chromatic number problem. *Information Processing Letters*, 5(3):66–67, 1976.

13 Andrzej Lingas and Martin Wahlen. An exact algorithm for subgraph homeomorphism. *J. Discrete Algorithms*, 7(4):464–468, 2009. doi:10.1016/j.jda.2008.10.003.

14 Neil Robertson and Paul D. Seymour. Graph minors. XIII. The disjoint paths problem. *J. Combinatorial Theory Ser. B*, 63(1):65–110, 1995.

15 Patrick Traxler. The time complexity of constraint satisfaction. In *Parameterized and Exact Computation*, pages 190–201. Springer, 2008.

Node-Max-Cut and the Complexity of Equilibrium in Linear Weighted Congestion Games

Dimitris Fotakis
National Technical University of Athens, Greece
fotakis@cs.ntua.gr

Vardis Kandiros
Massachusetts Institute of Technology, Cambridge, MA, USA
kandiros@mit.edu

Thanasis Lianeas
National Technical University of Athens, Greece
lianeas@corelab.ntua.gr

Nikos Mouzakis
National Technical University of Athens, Greece
nmouzakis@corelab.ntua.gr

Panagiotis Patsilinakos
National Technical University of Athens, Greece
patsilinak@corelab.ntua.gr

Stratis Skoulakis
Singapore University of Technology and Design, Singapore
efstratios@sutd.edu.sg

Abstract

In this work, we seek a more refined understanding of the complexity of local optimum computation for Max-Cut and pure Nash equilibrium (PNE) computation for congestion games with weighted players and linear latency functions. We show that computing a PNE of linear weighted congestion games is PLS-complete either for very restricted strategy spaces, namely when player strategies are paths on a series-parallel network with a single origin and destination, or for very restricted latency functions, namely when the latency on each resource is equal to the congestion. Our results reveal a remarkable gap regarding the complexity of PNE in congestion games with weighted and unweighted players, since in case of unweighted players, a PNE can be easily computed by either a simple greedy algorithm (for series-parallel networks) or any better response dynamics (when the latency is equal to the congestion). For the latter of the results above, we need to show first that computing a local optimum of a natural restriction of Max-Cut, which we call *Node-Max-Cut*, is PLS-complete. In Node-Max-Cut, the input graph is vertex-weighted and the weight of each edge is equal to the product of the weights of its endpoints. Due to the very restricted nature of Node-Max-Cut, the reduction requires a careful combination of new gadgets with ideas and techniques from previous work. We also show how to compute efficiently a $(1+\varepsilon)$-approximate equilibrium for Node-Max-Cut, if the number of different vertex weights is constant.

2012 ACM Subject Classification Theory of computation → Exact and approximate computation of equilibria

Keywords and phrases PLS-completeness, Local-Max-Cut, Weighted Congestion Games, Equilibrium Computation

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.50

Category Track A: Algorithms, Complexity and Games

Related Version The full version of the paper is available at `https://arxiv.org/abs/1911.08704`.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 50; pp. 50:1–50:19
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Funding This work was supported by the Hellenic Foundation for Research and Innovation (H.F.R.I.) under the “First Call for H.F.R.I. Research Projects to support Faculty members and Researchers and the procurement of high-cost research equipment grant”, project: BALSAM (project id: 1424). *Stratis Skoulakis*: Supported by NRF 2018, Fellowship NRF-NRFF2018-07. Part of this work was carried out while the author was a PhD student at the National Technical University of Athens.

1 Introduction

Motivated by the remarkable success of local search in combinatorial optimization, Johnson et al. introduced [25] the complexity class Polynomial Local Search (PLS), consisting of local search problems with polynomially verifiable local optimality. PLS includes many natural complete problems (see e.g., [28, App. C]), with Circuit-Flip [25] and Max-Cut [33] among the best known ones, and lays the foundation for a principled study of the complexity of local optima computation. In the last 15 years, a significant volume of research on PLS-completeness was motivated by the problem of computing a pure Nash equilibrium of potential games (see e.g., [1, 34, 20] and the references therein), where any improving deviation by a single player decreases a potential function and its local optima correspond to pure Nash equilibria [29].

Computing a local optimum of Max-Cut under the flip neighborhood (a.k.a. Local-Max-Cut) has been one of the most widely studied problems in PLS. Given an edge-weighed graph, a cut is locally optimal if we cannot increase its weight by moving a vertex from one side of the cut to the other. Since its PLS-completeness proof by Schäffer and Yannakakis [33], researchers have shown that Local-Max-Cut remains PLS-complete for graphs with maximum degree five [9], is polynomially solvable for cubic graphs [31], and its smoothed complexity is either polynomial in complete [2] and sparse [9] graphs, or almost polynomial in general graphs [7, 10]. Moreover, due to its simplicity and versatility, Max-Cut has been widely used in PLS reductions (see e.g., [1, 20, 34]). Local-Max-Cut can also be cast as a game, where each vertex aims to maximize the total weight of its incident edges that cross the cut. Cut games are potential games (the value of the cut is the potential function), which has motivated research on efficient computation of approximate equilibria for Local-Max-Cut [3, 6]. To the best of our knowledge, apart from the work on the smoothed complexity of Local-Max-Cut (and may be that Local-Max-Cut is P-complete for unweighted graphs [33, Theorem 4.5]), there has not been any research on whether (and to which extent) additional structure on edge weights affects hardness of Local-Max-Cut.

A closely related research direction deals with the complexity of computing a pure Nash equilibrium (equilibrium or PNE, for brevity) of *congestion games* [32], a typical example of potential games [29] and among the most widely studied classes of games in Algorithmic Game Theory (see e.g., [15] for a brief account of previous work). In congestion games (or CGs, for brevity), a finite set of players compete over a finite set of resources. Strategies are resource subsets and players aim to minimize the total cost of the resources in their strategies. Each resource e is associated with a (non-negative and non-decreasing) latency function, which determines the cost of using e as a function of e’s *congestion* (i.e., the number of players including e in their strategy). Researchers have extensively studied the properties of special cases and variants of CGs. Most relevant to this work are *symmetric* (resp. *asymmetric*) CGs, where players share the same strategy set (resp. have different strategy sets), *network* CGs, where strategies correspond to paths in an underlying network, and *weighted* CGs, where player contribute to the congestion with a different weight.

Fabrikant et al. [12] proved that computing a PNE of asymmetric network CGs or symmetric CGs is PLS-complete, and that it reduces to min-cost-flow for symmetric network CGs. About the same time [17, 30] proved that weighted congestion games admit a (weighted) potential function, and thus a PNE, if the latency functions are either affine or exponential (and [23, 24] proved that in a certain sense, this restriction is necessary). Subsequently, Ackermann et al. [1] characterized the strategy sets of CGs that guarantee efficient equilibrium computation. They also used a variant of Local-Max-Cut, called *threshold games*, to simplify the PLS-completeness proof of [12] and to show that computing a PNE of asymmetric network CGs with (exponentially steep) linear latencies is PLS-complete.

On the other hand, the complexity of equilibrium computation for weighted CGs is not well understood. All the hardness results above carry over to weighted CGs, since they generalize standard CGs (where the players have unit weight). But on the positive side, we only know how to efficiently compute a PNE for weighted CGs on parallel links with general latencies [16] and for weighted CGs on parallel links with identity latency functions and asymmetric strategies [19]. Despite the significant interest in (exact or approximate) equilibrium computation for CGs (see e.g., [4, 5, 6, 27] and the references therein), we do not understand how (and to which extent) the complexity of equilibrium computation is affected by player weights. This is especially true for weighted CGs with linear latencies, which admit a potential function and their equilibrium computation is in PLS.

Contributions. We contribute to both research directions outlined above. In a nutshell, we show that equilibrium computation in linear weighted CGs is significantly harder than for standard CGs, in the sense that it is PLS-complete either for very restricted strategy spaces, namely when player strategies are paths on a series-parallel network with a single origin and destination, or for very restricted latency functions, namely when resource costs are equal to the congestion. Our main step towards proving the latter result is to show that computing a local optimum of Node-Max-Cut, a natural and interesting restriction of Max-Cut where the weight of each edge is the product of the weights of its endpoints, is PLS-complete.

More specifically, using a tight reduction from Local-Max-Cut, we first show, in Section 3.1, that equilibrium computation for linear weighted CGs on series-parallel networks with a single origin and destination is PLS-complete (Theorem 1). The reduction results in games where both the player weights and the latency slopes are exponential. Our result reveals a remarkable gap between weighted and standard CGs regarding the complexity of equilibrium computation, since for standard CGs on series-parallel networks with a single origin and destination, a PNE can be computed by a simple greedy algorithm [18].

Aiming at a deeper understanding of how different player weights affect the complexity of equilibrium computation in CGs, we show, in Section 3.2, that computing a PNE of weighted network CGs with asymmetric player strategies and identity latency functions is PLS-complete (Theorem 2). Again the gap to standard CGs is remarkable, since for standard CGs with identity latency functions, any better response dynamics converges to a PNE in polynomial time. In the reduction of Theorem 2, Node-Max-Cut plays a role similar to that of threshold games in [1, Sec. 4]. The choice of Node-Max-Cut seems necessary, in the sense that known PLS reductions, starting from Not-All-Equal Satisfiability [12] or Local-Max-Cut [1], show that equilibrium computation is hard due to the interaction of players on different resources (where latencies simulate the edge / clause weights), while in our setting, equilibrium computation is hard due to the player weights, which are the same for all resources in a player's strategy.

Node-Max-Cut is a natural restriction of Max-Cut and settling the complexity of its local optima computation may be of independent interest, both conceptually and technically. Node-Max-Cut coincides with the restriction of Max-Cut shown (weakly) NP-complete on complete graphs in the seminal paper of Karp [26], while a significant generalization of Node-Max-Cut with polynomial weights was shown P-complete in [33].

A major part of our technical effort concerns reducing Circuit-Flip to Node-Max-Cut, thus showing that computing a local optimum of Node-Max-Cut is PLS-complete (Section 5). Since Node-Max-Cut is a very restricted special case of Max-Cut, we have to start from a PLS-complete problem lying before Local-Max-Cut on the "reduction paths" of PLS. The reduction is technically involved, due to the very restricted nature of the problem. In Node-Max-Cut, every vertex contributes to the cut value of its neighbors with the same weight, and differentiation comes only as a result of the different total weight in the neighborhood of each vertex. To deal with this restriction, we combine some new carefully constructed gadgets with the gadgets used by Schäffer and Yannakakis [33], Elsässer and Tscheuschner [9]. and Gairing and Savani [20]. In general, as a very restricted special case of Max-Cut, Node-Max-Cut is a natural and convenient starting point for future PLS reductions, especially when one wants to show hardness of equilibrium computation for restricted classes of games that admit weighted potential functions (e.g., as that in [17]). So, our results may serve as a first step towards a better understanding of the complexity of (exact or approximate) equilibrium computation for weighted potential games.

We also show that a $(1+\varepsilon)$-approximate equilibrium for Node-Max-Cut, where no vertex can switch sides and increase the weight of its neighbors across the cut by a factor larger than $1+\varepsilon$, can be computed in time exponential in the number of different weights (Theorem 3). Thus, we can efficiently compute a $(1+\varepsilon)$-approximate equilibrium for Node-Max-Cut, for any $\varepsilon > 0$, if the number of different vertex weights is constant. Since similar results are not known for Max-Cut, Theorem 3 may indicate that approximate equilibrium computation for Node-Max-Cut may not be as hard as for Max-Cut. An interesting direction for further research is to investigate (i) the quality of efficiently computable approximate equilibria for Node-Max-Cut; and (ii) the smoothed complexity of its local optima.

Related Work. Existence and efficient computation of (exact or approximate) equilibria for weighted congestion games have received significant research attention. We briefly discuss here some of the most relevant previous work. There has been significant research interest in the convergence rate of best response dynamics for weighted congestion games (see e.g., [8, 4, 11, 13, 22]). Gairing et al. [19] presented a polynomial algorithm for computing a PNE for load balancing games on restricted parallel links. Caragiannis et al. [6] established existence and presented efficient algorithms for computing approximate PNE in weighted CGs with polynomial latencies (see also [14, 21]).

Bhalgat et al. [3] presented an efficient algorithm for computing a $(3+\varepsilon)$-approximate equilibrium in Max-Cut games, for any $\varepsilon > 0$. The approximation guarantee was improved to $2+\varepsilon$ in [6]. We highlight that the notion of approximate equilibrium in cut games is much stronger than the notion of approximate local optimum of Max-Cut, since the former requires that no vertex can significantly improve the total weight of its incidence edges that cross the cut (as e.g., in [3, 6]), while the latter simply requires that the total weight of the cut cannot be significantly improved (as e.g., in [6]).

Johnson et al. [25] introduced the complexity class PLS and proved that Circuit-Flip is PLS-complete. Subsequently, Schäffer and Yannakakis [33] proved that Max-Cut is PLS-complete. From a technical viewpoint, our work is close to previous work by Elsässer and

Tscheuschner [9] and Gairing and Savani [20], where they show that Local-Max-Cut in graphs of maximum degree five [9] and computing a PNE for hedonic games [20] are PLS-complete, and by Ackermann et al. [1], where they reduce Local-Max-Cut to computing a PNE in network congestion games.

2 Basic Definitions and Notation

Polynomial-Time Local Search (PLS). A *polynomial-time local search* (PLS) problem L [25, Sec. 2] is specified by a (polynomially recognizable) set of instances I_L, a set $S_L(x)$ of feasible solutions for each instance $x \in I_L$, with $|s| = O(\text{poly}(|x|)$ for every solution $s \in S_L(x)$, an objective function $f_L(s, x)$ that maps each solution $s \in S_L(x)$ to its value in instance x, and a *neighborhood* $N_L(s, x) \subseteq S_L(x)$ of feasible solutions for each $s \in S_L(x)$. Moreover, there are three polynomial-time algorithms that for any given instance $x \in I_L$: (i) the first generates an initial solution $s_0 \in S_L(x)$; (ii) the second determines whether a given s is a feasible solution and (if $s \in S_L(x)$) computes its objective value $f_L(s, x)$; and (iii) the third returns either that s is *locally optimal* or a feasible solution $s' \in N_L(s, x)$ with better objective value than s. If L is a maximization (resp. minimization) problem, a solution s is locally optimal if for all $s' \in N_L(s, x)$, $f_L(s, x) \geq f_L(s', x)$ (resp. $f_L(s, x) \leq f_L(s', x)$). If s is not locally optimal, the third algorithm returns a solution $s' \in N_L(s, x)$ with $f(s, x) < f(s', x)$ (resp. $f(s, x) > f(s', x)$). The complexity class PLS consists of all polynomial-time local search problems. By abusing the terminology, we always refer to polynomial-time local search problem simply as local search problems.

PLS Reductions and Completeness. A local search problem L is PLS-*reducible* to a local search problem L', if there are polynomial-time algorithms ϕ_1 and ϕ_2 such that (i) ϕ_1 maps any instance $x \in I_L$ of L to an instance $\phi_1(x) \in I_{L'}$ of L'; (ii) ϕ_2 maps any (solution s' of instance $\phi_1(x)$, instance x) pair, with $s' \in S_{L'}(\phi_1(x))$, to a solution $s \in S_L(x)$; and (iii) for every instance $x \in I_L$, if s' is locally optimal for $\phi_1(x)$, then $\phi_2(s', x)$ is locally optimal for x.

By definition, if a local search problem L is PLS-reducible to a local search problem L', a polynomial-time algorithm that computes a local optimum of L' implies a polynomial time algorithm that computes a local optimum of L. Moreover, a PLS-reduction is transitive. As usual, a local search problem Q is PLS-*complete*, if $Q \in$ PLS and any local search problem $L \in$ PLS is PLS-reducible to Q.

Max-Cut and Node-Max-Cut. An instance of Max-Cut consists of an undirected edge-weighted graph $G(V, E)$, where V is the set of vertices and E is the set of edges. Each edge e is associated with a positive weight w_e. A cut of G is a vertex partition $(S, V \setminus S)$, with $\emptyset \neq S \neq V$. We usually identify a cut with one of its sides (e.g., S). We denote $\delta(S) = \{\{u, v\} \in E : u \in S \wedge v \notin S\}$ the set of edges that cross the cut S. The weight (or the value) of a cut S, denoted $w(S)$, is $w(S) = \sum_{e \in \delta(S)} w_e$. In Max-Cut, the goal is to compute an optimal cut S^* of maximum value $w(S^*)$.

In Node-Max-Cut, each vertex v is associated with a positive weight w_v and the weight of each edge $e = \{u, v\}$ is $w_e = w_u w_v$, i.e. equal to the product of the weights of e's endpoints. Again the goal is to compute a cut S^* of maximum value $w(S^*)$. As optimization problems, both Max-Cut and Node-Max-Cut are NP-complete [26].

In this work, we study Max-Cut and Node-Max-Cut as local search problems under the flip neighborhood. Then, they are referred to as Local-Max-Cut and Local-Node-Max-Cut. The neighborhood $N(S)$ of a cut $(S, V \setminus S)$ consists of all cuts $(S', V \setminus S')$ where S and

S' differ by a single vertex. Namely, the cut S' is obtained from S by moving a vertex from one side of the cut to the other. A cut S is locally optimal if for all $S' \in N(S)$, $w(S) \geq w(S')$. In Local-Max-Cut (resp. Local-Node-Max-Cut), given an edge-weighted (resp. vertex-weighted) graph, the goal is to compute a locally optimal cut. Clearly, both Max-Cut and Node-Max-Cut belong to PLS. In the following, we abuse the terminology and refer to Local-Max-Cut and Local-Node-Max-Cut as Max-Cut and Node-Max-Cut, for brevity, unless we need to distinguish between the optimization and the local search problem.

Weighted Congestion Games. A *weighted congestion game* $\mathcal{G}$ consists of n players, where each player i is associated with a positive weight w_i, a set of resources E, where each resource e is associated with a non-decreasing latency function $\ell_e : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$, and a non-empty strategy set $\Sigma_i \subseteq 2^E$ for each player i. A game is linear if $\ell_e(x) = a_e x + b_e$, for some $a_e, b_e \geq 0$, for all $e \in E$. The identity latency function is $\ell(x) = x$. The player strategies are *symmetric*, if all players share the same strategy set Σ, and *asymmetric*, otherwise.

We focus on *network* weighted congestion games, where the resources E correspond to the edges of an underlying network $G(V, E)$ and the player strategies are paths on G. A network game is *single-commodity*, if G has an origin o and a destination d and the player strategies are all (simple) $o - d$ paths. A network game is *multi-commodity*, if G has an origin o_i and a destination d_i for each player i, and i's strategy set Σ_i consists of all (simple) $o_i - d_i$ paths. A single-commodity network $G(V, E)$ is *series-parallel*, if it either consists of a single edge (o, d) or can be obtained from two series-parallel networks composed either in series or in parallel (see e.g., [35] for details on composition and recognition of series-parallel networks).

A configuration $\vec{s} = (s_1, \dots, s_n)$ consists of a strategy $s_i \in \Sigma_i$ for each player i. The congestion s_e of resource e in configuration $\vec{s}$ is $s_e = \sum_{i:e \in s_i} w_i$. The cost of resource e in $\vec{s}$ is $\ell_e(s_e)$. The *individual cost* (or *cost*) $c_i(\vec{s})$ of player i in configuration $\vec{s}$ is the total cost for the resources in her strategy s_i, i.e., $c_i(\vec{s}) = \sum_{e \in s_i} \ell_e(s_e)$. A configuration $\vec{s}$ is a *pure Nash equilibrium* (equilibrium or PNE, for brevity), if for every player i and every strategy $s' \in \Sigma_i$, $c_i(\vec{s}) \leq c_i(\vec{s}_{-i}, s')$ (where $(\vec{s}_{-i}, s')$ denotes the configuration obtained from $\vec{s}$ by replacing s_i with s'). Namely, no player can improve her cost by unilaterally switching her strategy.

Equilibrium Computation and Local Search. [17] shows that for linear weighted congestion games, with latencies $\ell_e(x) = a_e x + b_e$, $\Phi(\vec{s}) = \sum_{e \in E}(a_e s_e^2 + b_e s_e) + \sum_i w_i \sum_{e \in s_i}(a_e w_i + b_e)$ changes by $2w_i(c_i(\vec{s}) - c_i(\vec{s}_{-i}, s'))$, when a player i switches from strategy s_i to strategy s' in $\vec{s}$. Hence, Φ is a weighted potential function, whose local optimal (wrt. single player deviations) correspond to PNE of the underlying game. Hence, equilibrium computation for linear weighted congestion games is in PLS. Specifically, configurations corresponds to solutions, the neighborhood $N(\vec{s})$ of a configuration $\vec{s}$ consists of all configurations $(\vec{s}_{-i}, s')$ with $s' \in \Sigma_i$, for some player i, and local optimality is defined wrt. the potential function Φ.

Max-Cut and Node-Max-Cut as Games. Local-Max-Cut and Local-Node-Max-Cut can be cast as cut games, where players correspond to vertices of $G(V, E)$, strategies $\Sigma = \{0, 1\}$ are symmetric, and configurations $\vec{s} \in \{0, 1\}^{|V|}$ correspond to cuts, e.g., $S(\vec{s}) = \{v \in V : s_v = 0\}$. Each player v aims to maximize $w_v(\vec{s}) = \sum_{e=\{u,v\} \in E: s_u \neq s_v} w_e$, that is the total weight of her incident edges that cross the cut. For Node-Max-Cut, this becomes $w_v(\vec{s}) = \sum_{u:\{u,v\} \in E \wedge s_u \neq s_v} w_u$, i.e., v aims to maximize the total weight of her neighbors across the cut. A cut $\vec{s}$ is a PNE if for all players v, $w_v(\vec{s}) \geq w_v(\vec{s}_{-i}, 1 - s_v)$. Equilibrium computation for cut games is equivalent to local optimum computation, and thus, is in PLS.

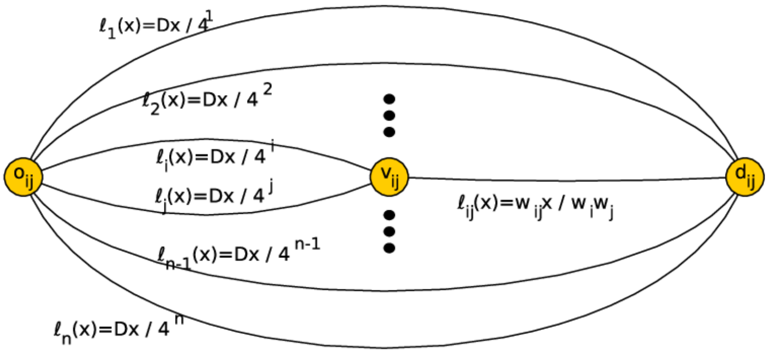


Figure 1 The series-parallel network F_{ij} that corresponds to edge $\{i, j\} \in A$.

A cut $\vec{s}$ is a $(1+\varepsilon)$-approximate equilibrium, for some $\varepsilon > 0$, if for all players v, $(1+\varepsilon)w_v(\vec{s}) \geq w_v(\vec{s}_{-i}, 1 - s_v)$. Note that the notion of $(1+\varepsilon)$-approximate equilibrium is stronger than the notion of $(1+\varepsilon)$-approximate local optimum, i.e., a cut S such that for all $S' \in N(S)$, $(1+\varepsilon)w(S) \geq w(S')$ (see also the discussion in [6]).

3 Hardness of Computing Equilibria in Weighted Congestion Games

We next show that computing a PNE in weighted congestion games with linear latencies is PLS-complete either for single-commodity series-parallel networks or for multi-commodity networks with identity latency functions.

3.1 Weighted Congestion Games on Series-Parallel Networks

▶ **Theorem 1.** *Computing a pure Nash equilibrium in weighted congestion games on single-commodity series-parallel networks with linear latency functions is PLS-complete.*

Proof sketch. Membership in PLS follows from the potential function argument of [17]. To show hardness, we present a reduction from Local-Max-Cut.

Let $H(V, A)$ be an instance of Local-Max-Cut with n vertices and m edges. Based on H, we construct a weighted congestion game on a single-commodity series-parallel network G with $3n$ players, where for every $i \in [n]$, there are three players with weight $w_i = 16^i$. Network G is a parallel composition of two identical copies of a simpler series-parallel network. We refer to these copies as G_1 and G_2. Each of G_1 and G_2 is a series composition of m simple series-parallel networks F_{ij}, each corresponding to an edge $\{i, j\} \in A$. Network F_{ij} is depicted in Figure 1, where D is assumed to be a constant chosen (polynomially) large enough. An example of the entire network G is shown in Figure 2.

In each of G_1 and G_2, there is a unique path that contains all edges with latency functions $\ell_i(x) = Dx/4^i$, for each $i \in [n]$. We refer to these paths as p_i^u for G_1 and p_i^l for G_2. In addition to the edges with latency $\ell_i(x)$, p_i^u and p_i^l include all edges with latencies $\ell_{ij}(x) = \frac{w_{ij}x}{w_i w_j} = \frac{w_{ij}x}{16^{i+j}}$, which correspond to the edges incident to vertex i in H.

Due to the choice of the player weights and the latency slopes, a player with weight w_i must choose either p_i^u or p_i^l in any PNE. We can prove this claim by induction on the player weights. The players with weight $w_n = 16^n$ have a dominant strategy to choose either p_n^u or p_n^l, since the slope of $\ell_n(x)$ is significantly smaller than the slope of any other latency $\ell_i(x)$. In fact, the slope of ℓ_n is so small that even if all other $3n - 1$ players choose one of p_n^u or p_n^l, a player with weight w_n would prefer either p_n^u or p_n^l over all other paths. Therefore, we

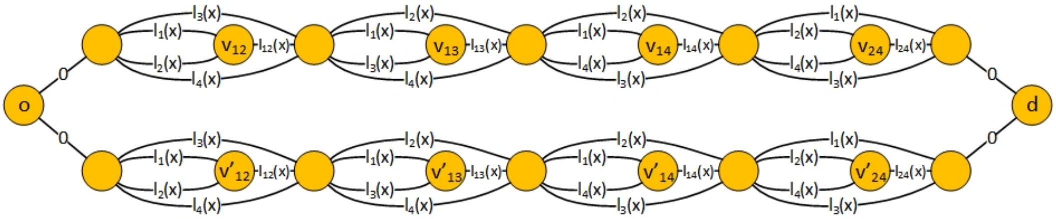


Figure 2 An example of the network G constructed in the proof of Theorem 1 for graph $H(V, A)$, with $V = \{1, 2, 3, 4\}$ and $A = \{\{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 4\}\}$. G is a parallel composition of two parts, each consisting of the smaller networks F_{12}, F_{13}, F_{14} and F_{24} (see also Figure 1) connected in series.

can assume that each of p^u_n and p^l_n are used by at least one player with weight w_n in any PNE, which would increase their latency so much that no player with smaller weight would prefer them any more. The inductive argument applies the same reasoning for players with weights w_{n-1}, who should choose either p^u_{n-1} or p^l_{n-1} in any PNE, and subsequently, for players with weights $w_{n-2}, \dots, w_1$. Hence, we conclude that for all $i \in [n]$, each of p^u_i and p^l_i is used by at least one player with weight w_i.

Moreover, we note that two players with different weights, say w_i and w_j, go through the same edge with latency $\ell_{ij}(x) = \frac{w_{ij}x}{w_i w_j}$ in G only if the corresponding edge $\{i, j\}$ is present in H. The correctness of the reduction follows the fact that a player with weight w_i aims to minimize her cost through edges with latencies ℓ_{ij} in G in the same way that in the Max-Cut instance, we want to minimize the weight of the edges incident to a vertex i and do not cross the cut. Formally, we next show that a cut S is locally optimal for the Max-Cut instance if and only if the configuration where for every $k \in S$, two players with weight w_k use p^u_k and for every $k \notin S$, two players with weight w_k use p^l_k is a PNE of the weighted congestion game on G.

Assume an equilibrium configuration and consider a player a of weight w_k that uses p^u_k together with another player of weight w_k (if this is not the case, vertex k is not included in S and we apply the symmetric argument for p^l_k). By the equilibrium condition, the cost of player a on p^u_k is at most her cost on p^l_k, which implies that

$$\sum_{k=1}^{m} \frac{2D16^k}{4^k} + \sum_{j:\{k,j\}\in A} \frac{w_{kj}(2 \cdot 16^k + x^u_j 16^j)}{16^{k+j}} \le \sum_{k=1}^{m} \frac{2D16^k}{4^k} + \sum_{j:\{k,j\}\in A} \frac{w_{kj}(2 \cdot 16^k + x^l_j 16^j)}{16^{k+j}} \,,$$

where x^u_j (resp. x^l_j) is either 1 or 2 (resp. 2 or 1) depending on whether, for each vertex j connected to vertex k in H, one or two players (of weight w_j) use p^u_j. Simplifying the inequality above, we obtain that:

$$\sum_{j:\{k,j\}\in A} w_{kj}(x^u_j - 1) \le \sum_{j:\{k,j\}\in A} w_{kj}(x^l_j - 1) \tag{1}$$

Let $S = \{i \in V : x^u_i = 2\}$. By hypothesis, $k \in S$ and the left-hand side of (1) corresponds to the total weight of the edges in H that are incident to k and do not cross the cut S. Similarly, the right-hand side of (1) corresponds to the total weights of the edges in H that are incident to k and cross the cut S. Therefore, (1) implies that we cannot increase the value of the cut S by moving vertex k from S to $V \setminus S$. Since this or its symmetric condition holds for any vertex k of H, the cut $(S, V \setminus S)$ is locally optimal. To conclude the proof, we argue along the same lines that any locally optimal cut of H corresponds to a PNE in the weighted congestion game on G. ◀

3.2 Weighted Congestion Games with Identity Latency Functions

We next prove that computing a PNE in weighted congestion games on multi-commodity networks with identity latency functions is PLS-complete. Compared to Theorem 1, we allow for a significantly more general strategy space, but we significantly restrict the latency functions, only allowing for the player weights to be exponentially large.

▶ **Theorem 2.** *Computing a pure Nash equilibrium in weighted congestion games on multi-commodity networks with identity latency functions is PLS-complete.*

Proof sketch. We use a reduction from Local-Node-Max-Cut, which as we show in Theorem 4, is PLS-complete. Our construction draws ideas from [1].

Let $H(V, A)$ be an instance of Node-Max-Cut. We construct a weighted congestion game on a multi-commodity network G with identity latency functions $\ell_e(x) = x$ such that equilibria of the congestion game correspond to locally optimal cuts of H.

At the conceptual level, each player i of the congestion game corresponds to vertex $i \in V$ and has weight w_i (i.e., equal to the weight of vertex i in H). The key step is to construct a network G such that for every player $i \in [n]$, there are two paths, say p_i^u and p_i^l, whose cost dominate the cost of any other path for player i. Therefore, in any equilibrium, player i selects either p_i^u or p_i^l (which corresponds to vertex i selecting one of the two sides of a cut). For every edge $\{i, j\} \in A$, paths p_i^u and p_j^u (resp. paths p_i^l and p_j^l) have an edge e_{ij}^u (resp. e_{ij}^l) in common. Intuitively, the cost of p_i^u (resp. p_i^l) for player i is determined by the set of players j, with j connected to i in H, that select path p_j^u (resp. p_j^l).

Let $\vec{s}$ be any equilibrium configuration of the weighted congestion game. Each player $i \in [n]$ selects either p_i^u or p_i^l in $\vec{s}$. Let $S = \{i \in [n] : \text{player } i \text{ selects } p_i^u \text{ in } \vec{s}\}$. Applying the equilibrium condition, we next show that S is a locally optimal cut.

We let $V_i = \{j : \{i, j\} \in A\}$ be the neighborhood of vertex i in H. By the construction of G, the individual cost of a player i on path p_i^u (resp. p_i^l) in $\vec{s}$ is equal to $K+|V_i|w_i+\sum_{j\in S\cap V_i} w_j$ (resp. $K + |V_i|w_i + \sum_{j\in V_i\setminus S} w_j$), where K is a large constant that depends on the network G only. Therefore, for any player $i \in S$, equilibrium condition for $\vec{s}$ implies that

$$K + |V_i|w_i + \sum_{j\in S\cap V_i} w_j \le K + |V_i|w_i + \sum_{j\in V_i\setminus S} w_j \Rightarrow \sum_{j\in S\cap V_i} w_j \le \sum_{j\in V_i\setminus S} w_j$$

Multiplying both sides by w_i, we get that the total weight of the edges that are incident to i and cross the cut S is no less than the total weight of the edges that are incident to i and do not cross the cut. By the same reasoning, we reach the same conclusion for any player $i \notin S$. Therefore, the cut $(S, V \setminus S)$ is locally optimal for the Node-Max-Cut instance $H(V, A)$.

To conclude the proof, we argue along the same lines that any locally optimal cut S for the Node-Max-Cut instance $H(V, A)$ corresponds to an equilibrium in the network G, by letting a player i select path p_i^u if and only if $i \in S$. ◀

4 Computing Approximate Equilibria for Node-Max-Cut

We complement our PLS-completeness proof for Node-Max-Cut, in Section 5, with an efficient algorithm computing $(1 + \varepsilon)$-approximate equilibria for Node-Max-Cut, when the number of different vertex weights is a constant. We note that similar results are not known (and a similar approach fails) for Max-Cut. Investigating if stronger approximation guarantees are possible for efficiently computable approximate equilibria for Node-Max-Cut is beyond the scope of this work and an intriguing direction for further research.

Given a vertex-weighted graph $G(V,E)$ with n vertices and m edges, our algorithm, called BridgeGaps, computes a $(1+\varepsilon)^3$-approximate equilibrium for a Node-Max-Cut, for any $\varepsilon > 0$, in $(m/\varepsilon)(n/\varepsilon)^{O(D_\varepsilon)}$ time, where D_ε is the number of different vertex weights in G, when the weights are rounded down to powers of $1+\varepsilon$. We next sketch the algorithm and the proof of Theorem 3.

For simplicity, we assume that n/ε is an integer and that vertices are indexed in nondecreasing order of weight, i.e., $w_1 \le w_2 \le \cdots \le w_n$. BridgeGaps first rounds down vertex weights to the closest power of $(1+\varepsilon)$. Namely, each weight w_i is replaced by weight $w'_i = (1+\varepsilon)^{\lfloor \log_{1+\varepsilon} w_i \rfloor}$. Clearly, an $(1+\varepsilon)^2$-approximate equilibrium for the new instance G' is an $(1+\varepsilon)^3$-approximate equilibrium for the original instance G. The number of different weights D_ε, used in the analysis, is defined wrt. the new instance G'.

Then, BridgeGaps partitions the vertices of G' into groups $g_1, g_2, \ldots$, so that the vertex weights in each group increase with the index of the group and the ratio of the maximum weight in group g_j to the minimum weight in group g_{j+1} is no less than n/ε. This can be performed by going through the vertices, in nondecreasing order of their weights, and assign vertex $i+1$ to the same group as vertex i, if $w'_{i+1}/w'_i \le n/\varepsilon$. Otherwise, vertex $i+1$ starts a new group. The idea is that for an $(1+\varepsilon)^2$-approximate equilibrium in G', we only need to enforce the $(1+\varepsilon)$-approximate equilibrium condition for each vertex i only for i's neighbors in the highest-indexed group (that includes some neighbor of i). To see this, let g_j be the highest-indexed group that includes some neighbor of i and let ℓ be the lowest indexed neighbor of i in g_j. Then, the total weight of i's neighbors in groups $g_1, \ldots, g_{j-1}$ is less than $\varepsilon w'_\ell$. This holds because i has at most $n-2$ neighbors in these groups and by definition, $w'_q \le (\varepsilon/n) w'_\ell$, for any i's neighbor q in groups $g_1, \ldots, g_{j-1}$. Therefore, we can ignore all neighbors of i in groups $g_1, \ldots, g_{j-1}$, at the expense of one more $1+\varepsilon$ factor in the approximate equilibrium condition.

Since for every vertex i, we need to enforce its (approximate) equilibrium condition only for i's neighbors in a single group, we can scale down vertex weights in the same group uniformly (i.e., dividing all the weights in each group by the same factor), as long as we maintain the key property in the definition of groups (i.e., that the ratio of the maximum weight in group g_j to the minimum weight in group g_{j+1} is no less than n/ε). Hence, we uniformly scale down the weights in each group so that (i) the minimum weight in group g_1 becomes 1; and (ii) for each $j \ge 2$, the ratio of the maximum weight in group g_{j-1} to the minimum weight in group g_j becomes exactly n/ε. This results in a new instance G'' where the minimum weight is 1 and the maximum weight is $(n/\varepsilon)^{D_\varepsilon}$. Therefore, a $(1+\varepsilon)$-approximate equilibrium in G'' can be computed, in a standard way, after at most $(m\varepsilon)(n/\varepsilon)^{2D_\varepsilon}$ ε-best response moves.

Putting everything together and using $\varepsilon' = \varepsilon/7$, so that $(1+\varepsilon')^3 \le 1+\varepsilon$, for all $\varepsilon \in (0,1]$, we obtain the following. We note that the running time of BridgeGaps is polynomial, if $D_\varepsilon = O(1)$ (and quasipolynomial if $D_\varepsilon = \text{poly}(\log n)$).

▶ **Theorem 3.** *For any vertex-weighted graph G with n vertices and m edges and any $\varepsilon > 0$, BridgeGaps computes a $(1+\varepsilon)$-approximate pure Nash equilibrium for Node-Max-Cut on G in $(m/\varepsilon)(n/\varepsilon)^{O(D_\varepsilon)}$ time, where D_ε denotes the number of different vertex weights in G, after rounding them down to the nearest power of $1+\varepsilon$.*

5 PLS-Completeness of Node-Max-Cut

In this section we sketch the proof of Theorem 4 and highlight the main differences of our reduction from known PLS reductions to Max-Cut [9, 20, 33].

▶ **Theorem 4.** *Local-Node-Max-Cut is PLS-complete.*

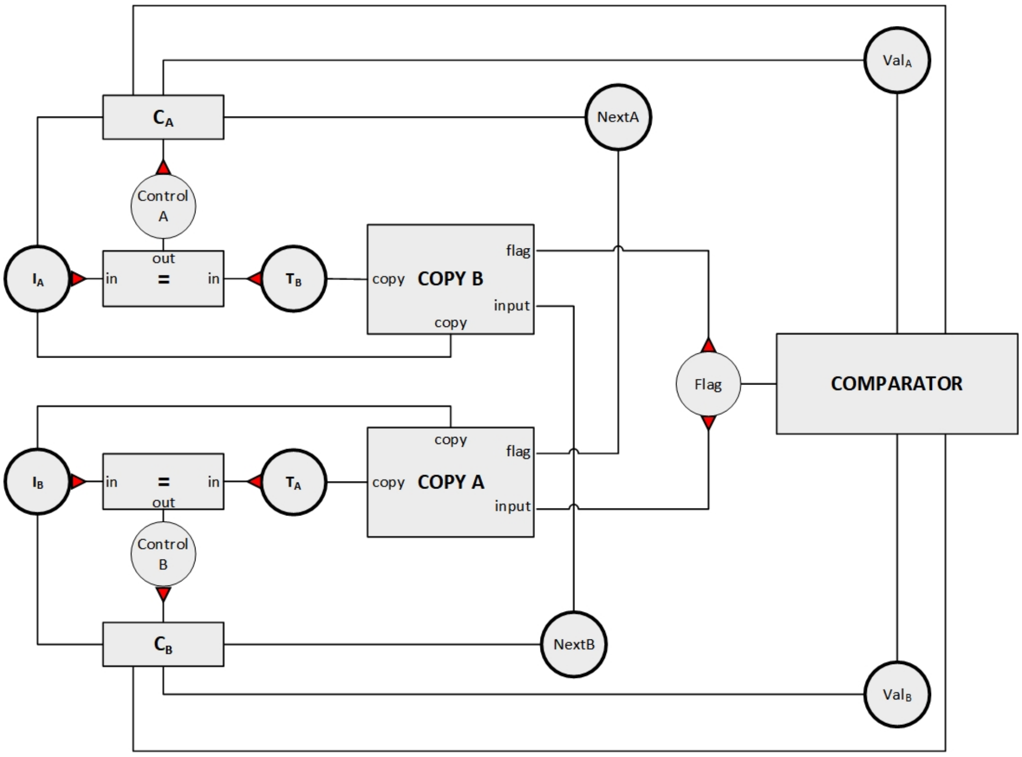


Figure 3 The general structure of the Node-Max-Cut instance constructed in the proof of Theorem 4. Rectangles denote the main gadgets, circles denote vertices that participate in multiple gadgets, and circles with bold border denote groups of n such vertices. The small red triangles are used to indicate the "information flow" through the instance.

As discussed in Section 2, the local search version of Node-Max-Cut is in PLS. To establish the PLS-hardness of Node-Max-Cut, we present a reduction from Circuit-Flip. We start with an outline of our construction and a brief discussion of its technical novelty. Then, in Section 5.1, we discuss our gadget constructions in more detail and present the key technical steps towards the proof of Theorem 4.

An Outline of the Construction. An instance of Circuit-Flip consists of a Boolean circuit C with n inputs and m outputs (and wlog. only NOR gates). The value $C(s)$ of an n-bit input string s is the integer corresponding to the m-bit output string. The neighborhood $N(s)$ of s consists of all n-bit strings s' at Hamming distance 1 to s (i.e., s' is obtained from s by flipping one bit of s). The goal is to find a locally optimal input string s, i.e., an s with $C(s) \geq C(s')$, for all $s' \in N(s)$. Circuit-Flip was the first problem shown to be PLS-complete in [25].

Given an instance C of Circuit-Flip, we construct a vertex-weighted undirected graph $G(V, E)$ so that from any locally optimum cut of G, we can recover, in polynomial time, a locally optimal input of C. The graph G consists of different gadgets (see Figure 3), which themselves might be regarded as smaller instances of Node-Max-Cut. Intuitively, each of these gadgets receives information from its "input" vertices, processes this information, while carrying it through its internal part, and outputs the results through its "output" vertices. Different gadgets are glued together through their "input" and "output" vertices.

Our construction follows the *flip-flop* architecture (Figure 3), previously used e.g., in [9, 20, 33], but requires more sophisticated implementations of several gadgets, so as to conform with the very restricted weight structure of Node-Max-Cut. Next, we outline the functionality of the main gadgets and how the entire construction works.

Given a circuit C, we construct two *Circuit Computing* gadgets C_ℓ (ℓ always stands for either A or B), which are instances of Node-Max-Cut that simulate circuit C in the following sense: Each C_ℓ has a set I_ℓ of n "input" vertices, whose (cut) values correspond to the input string of circuit C, and a set Val_ℓ of m "output" vertices, whose values correspond to the output string of C on input I_ℓ. There is also a set $Next\ell$ of n vertices whose values correspond to a n-bit string in the neighborhood of I_ℓ of circuit value larger than that of I_ℓ (if the values of Nextℓ coincide with the values of I_ℓ, I_ℓ is locally optimal). Lastly, there is a (set of) vertices $Control_\ell$ that control the behavior of the gadget.

The Circuit Computing gadgets operate in two different modes, determined by Control$_\ell$: the *write mode*, when Control$_\ell = 0$, and the *compute mode*, when Control$_\ell = 1$. If C_ℓ operates in write mode, the input values of I_ℓ can be updated to the values of the complementary Next set (i.e., I_A is updated to NextB, and vice versa). When, C_ℓ operates in the compute mode, C_ℓ simulates the computation of circuit C and the values of Nextℓ and Val$_\ell$ are updated to the corresponding values produced by C. Throughout the proof, we let Real-Val(I_ℓ) denote the output string of circuit C on input I_ℓ (i.e., the input of C takes the cut values of the vertices in I_ℓ), and let Real-Next(I_ℓ) denote a neighbor of I_ℓ with circuit value larger than the circuit value of I_ℓ. If I_ℓ is locally optimal, Real-Next(I_ℓ) $= I_\ell$.

Our Circuit Computing gadgets C_A and C_B are based on the gadgets of Schäffer and Yannakakis [33] (see also Figure 4 for an abstract description of them). Their detailed construction is described in the next section and their properties are summarized in Theorem 5.

The *Comparator* gadget compares Val$_A$ with Val$_B$, which are intended to be Real-Val(I_A) and Real-Val(I_B), respectively, and outputs 1, if Val$_A \leq$ Val$_B$, and 0, otherwise. The result of the Comparator is stored in the value of the *Flag* vertex. If Flag $= 1$, the Circuit Computing gadget C_A enters its write mode and the input values in I_A are updated to the neighboring solution of I_B, currently stored in NextB (everything operates symmetrically, if Flag $= 0$). Then, in the next "cycle", the input in I_A leads C_A to a Val$_A >$ Val$_B$ (and to a better neighboring solution at NextA), Flag becomes 0, and the values of I_B are updated to NextA. When we reach a local optimum, I_A and I_B are stabilized to the same values.

The workflow above is implemented by the *Copy* and the *Equality* gadgets. The *CopyB* (resp. *CopyA*) gadget updates the values of I_A (resp. I_B) to the values in NextB (resp. NextA), if Val$_A \leq$ Val$_B$ and Flag $= 1$ (resp. Val$_A >$ Val$_B$ and Flag $= 0$). When Flag $= 1$, the vertices in T_B take the values of the vertices in NextB. If the values of I_A and NextB are different, the Equality gadget sets the value of Control$_A$ to 0. Hence, the Circuit Computing gadget C_A enters its write mode and the vertices in I_A take the values of the vertices in NextB. Next, Control$_A$ becomes 1, because the values of I_A and NextB are now identical, and C_A enters its compute mode. As a result, the vertices in Val$_A$ and NextA take the values of Real-Val(I_A) and Real-Next(I_A), and we proceed to the next cycle.

A key notion used throughout the reduction is the *bias* that a vertex i experiences from a vertex subset. The bias of vertex i from (or wrt.) $V' \subseteq V$ is $\left|\sum_{j \in V_i^1 \cap V'} w_j - \sum_{j \in V_i^0 \cap V'} w_j\right|$, where V_i^1 (resp. V_i^0) denotes the set of i's neighbors on the 1 (resp. 0) side of the cut.

Technical Novelty. Next, we briefly discuss the points where our reduction needs to deviate substantially from known PLS reductions to Max-Cut. Our discussion is unavoidably technical and assumes familiarity with (at least one of) the reductions in [9, 20, 33].

Our Circuit Computing gadgets are based on similar gadgets used e.g., in [33]. A key difference is that our Circuit Computing gadgets are designed to operate with w_{Control} (i.e., weight of the Control_ℓ vertex) arbitrarily smaller than the weight of any other vertex in the Circuit Computing gadget. Hence, the Control vertex can achieve a very small bias wrt. the Circuit Computing gadget (see Theorem 5, Case 3), which in turn, allows us to carefully balance the weights in the Equality gadget. The latter weights should be large enough, so as to control the write and compute modes of C_ℓ, and at the same time, small enough, so as to avoid interference with the values of the input vertices I_ℓ. The second important reason for setting w_{Control} sufficiently small is that we need the Control vertex to follow the "output" of the Equality gadget, and not the other way around.

The discussion above highlights a key difference between Max-Cut and Node-Max-Cut. Previous reductions to Max-Cut (see e.g., the reduction in [33]) implement Control's functionality using different weights for its incident edges. More specifically, Control is connected with edges of appropriately large weight to the vertices of the circuit gadget, so that it can control the gadget's mode, and with edges of appropriately small weight to vertices outside the circuit gadget, so that its does not interfere with its other neighbors.

For Node-Max-Cut, we need to achieve the same desiderata with a single vertex weight. We manage to do so by introducing a *Leverage* gadget. Our *Leverage* gadget reduces the influence of a vertex with large weight to a vertex with small weight and is used internally in the Circuit Computing gadget. Hence, we achieve that Control has small bias wrt. the circuit gadget and weight comparable to the weights of the circuit gadget's internal vertices.

Another important difference concerns the implementation of the *red marks* (denoting the information flow) between Flag and the Copy gadgets, in Figure 3. They indicate that the value of Flag should agree with the output of the Comparator gadget. This part of the information flow is difficult to implement, because the Comparator gadget and the Copy gadgets receive input from NextA, NextB, Val_A and Val_B, where the vertex weights are comparable to the weights of the output vertices in the Circuit Computing gadgets. As a result, the weights of the vertices inside the Comparator gadget cannot become sufficiently larger than the weights of the vertices inside the Copy gadgets. [33, 20, 9] connect Flag to the Copy gadgets with edges of sufficiently small weight, which makes the bias of Flag from the Copy gadgets negligible compared against its bias from Comparator. Again, the *Leverage* gadget comes to our rescue. We use it internally in the Copy gadgets, in order to decrease the influence of the vertices inside the Copy gadgets to Flag. As a result, Flag's bias from the Copy gadgets becomes much smaller than its bias from Comparator (see Lemma 10).

Another key technical difference concerns the design of the Comparator gadget. As stated in Lemma 10 and explained below, the Comparator gadget manages to compute the result of the comparison $\text{Real-Val}(I_A) \leq \text{Real-Val}(I_B)$, even if some input vertices may have incosistent values. In previous work [33, 20, 9], the Comparator guarantees correctness of the values in both NextB and Val_B using appropriately chosen edge weights. With the correctness of the input values guaranteed, the comparison is not hard to implement. It is not clear if this decoupled architecture of the Comparator gadget can be implemented in Node-Max-Cut, due to the special structure of edge weights. Instead, we implement a new *all at once* Comparator, which ensures correctness to a subset of its input values enough to perform the comparison correctly (see also the discussion after Lemma 11 on this point).

5.1 A Technical Overview of the Proof of Theorem 4

In this section, we outline the gadget constructions and the key technical claims used in the proof of Theorem 4. As shown in Figure 3, our Node-Max-Cut instance consists of the following gadgets:

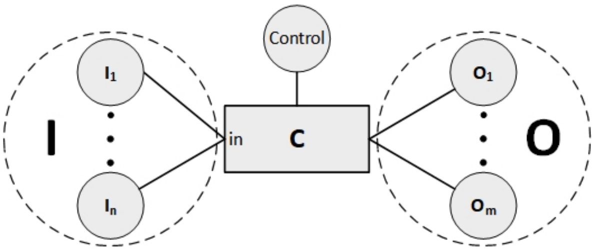


Figure 4 The Circuit Computing gadgets. The dashed circles, labeled I and O, represent all input and output vertices, respectively.

1. Two Circuit Computing gadgets that calculate the values and next neighbors of solutions.
2. Two Copy gadgets that transfer the solution of one circuit to the other, and vice versa.
3. Two Equality gadgets that determine the (write or compute) mode in which the Circuit Computing gadgets operate.
4. A Comparator gadget.

Unlike previous similar reductions in the literature (see e.g.,[33, 20, 9]), we introduce each gadget separately rather than through types of diminishing weight. This is because in NODE-MAX-CUT, weights are associated with vertices and each vertex may be part of multiple gadgets.

The Circuit Computing Gadgets. The Circuit Computing gadgets C_A and C_B are the basic primitives of our reduction. They are based on the gadgets introduced by Schäffer and Yannakakis [33] to establish PLS-completeness of MAX-CUT. This type of Circuit Computing gadgets can be constructed so as to simulate any Boolean circuit C.

The most important vertices are those corresponding to the input and the output of the simulated circuit C. Another important vertex is Control, which allows the gadget to switch between the write and the compute mode of operation. Figure 4 is an abstract depiction of the Circuit Computing gadgets. Theorem 5 describes the local optimum behavior of the input vertices I_ℓ and output vertices Nextℓ, Val$_\ell$ of the Circuit Computing gadgets C_ℓ.

▶ **Theorem 5.** *At any local optimum of the NODE-MAX-CUT instance in Figure 3, the following hold:*

1. *If Control$_\ell = 1$ and the vertices in Nextℓ, Val$_\ell$ experience 0 bias from any other gadget beyond C_ℓ, then Nextℓ = Real-Next(I_ℓ) and Val$_\ell$ = Real-Val(I_ℓ).*
2. *If Control$_\ell = 0$, then each vertex in I_ℓ experiences 0 bias from the internal vertices of C_ℓ.*
3. *Control$_\ell$ experiences w_{Control_ℓ} bias from the internal vertices of C_ℓ.*

Case 1 of Theorem 5 describes the *compute mode* of the Circuit Computing gadgets. At any local optimum with Control$_A = 1$ and with the output vertices of C_A being indifferent wrt. other gadgets, C_A computes its output correctly. Case 2 of Theorem 5 describes the *write mode.* If at a local optimum Control$_A = 0$, the vertices in I_A have 0 bias from the C_A gadget. As a result, their value is determined by the biases of the CopyB gadget and the Equality gadget. Case 3 of Theorem 5 bounds the bias that the Equality gadget poses to the Control vertices, so as to make the computing gadget flip from one mode to the other.

The Copy and Equality Gadgets. The Copy and Equality gadgets are responsible for transferring the output of one Circuit Computing gadget to the other. More specifically, the Equality gadget takes as input the nodes I_A, I_B, T_A, T_B and assures that the Control vertices always have the correct values. Recall that as shown in Theorem 5, Case 3, the bias that the Equality gadget exerts is enough to dominate the values of the Control nodes. Using that, we can prove the following lemma:

▶ **Lemma 6.** *For any local optimum of the* Node-Max-Cut *instance in Figure 3, we have that* $Control_A = (I_A = T_B)$ *and* $Control_B = (I_B = T_A)$.

The Copy gadget transfers the values of the NextB and NextA to I_A and I_B, respectively, when each gadget is in its *write mode*. This behavior is summarized by the following:

▶ **Lemma 7.** *At any local optimum point of the* Node-Max-Cut *instance in Figure 3, the following hold:*

1. *If* $Flag = 1$*, i.e.,* $NextB$ writes *on* I_A*, then (i)* $T_B = NextB$*, and (ii) if* $Control_A = 0$*, then* $I_A = T_B = NextB$.
2. *If* $Flag = 0$ *i.e.,* $NextA$ writes *on* I_B*, then (i)* $T_A = NextA$*, and (ii) if* $Control_B = 0$*, then* $I_B = T_A = NextA$.

Additionally, the Copy gadget has the property of leaving the NextB and NextA vertices unbiased, when Flag has certain values. This is necessary so that whenever a Circuit Computing gadget is about to compute, the second condition of Theorem 5, Case 1, can apply and allow the computation to take place.

▶ **Lemma 8.** *At any local optimum of the* Node-Max-Cut *instance in Figure 3:*

- *If* $Flag = 1$*, then any vertex in* $NextA$ *experiences* 0 *bias from the* $CopyA$ *gadget.*
- *If* $Flag = 0$*, then any vertex in* $NextB$ *experiences* 0 *bias from the* $CopyB$ *gadget.*

Having established the above properties of the Copy and Equality gadgets, we can show the following theorem which asserts that at an local optimum, one of the two circuits has taken the other's output as input.

▶ **Theorem 9.** *At any a local optimum of the* Node-Max-Cut *instance in Figure 3, the following hold:*

- *If* $Flag = 1$*, then* $I_A = NextB$.
- *If* $Flag = 0$*, then* $I_B = NextA$.

Proof of Theorem 9. Let a local optimum in which Flag $= 1$. Let us assume that $I_A \neq$ NextB. Then, by Case 1 of Lemma 7, $T_B =$ NextB. As a result, $I_A \neq T_B$, which implies that $Control_A = 0$, by Lemma 6. Now, by Lemma 7, Case 1.ii, we have that $I_A =$ NextB, which is a contradiction. The same analysis can be applied in case where Flag $= 0$. ◀

The Comparator Gadget. The last important gadget in our reduction is the Comparator, whose construction is technically involved. We first recall that Nextℓ and Val$_\ell$ denote the actual values that the corresponding vertices have in our construction, while Real-Next(I_ℓ) and Real-Val(I_ℓ) denote the values that these vertices are supposed to have, assuming that the Circuit Computing gadgets operate correctly. We also recall that when Flag $= 1$ (resp. Flag $= 0$), the Circuit Computing gadget C_A (resp. C_B) recomputes its output values, based on the (possibly incorrect) outputs of the other Circuit Computing gadget C_B (resp. C_A). The construction of the Comparator gadget and the following lemmas ensure that at any local optimum of the Node-Max-Cut instance in Figure 3, the actual values of the vertices in Nextℓ and Val$_\ell$ are identical to Real-Next(I_ℓ) and Real-Val(I_ℓ).

First, connecting certain internal vertices of the Circuit Computing gadgets with the vertices in NextA and NextB and in the Comparator, we obtain the following pair of lemmas.

▶ **Lemma 10.** *At any local optimum of the* Node-Max-Cut *instance in Figure 3, the following hold:*

- *If* $Flag = 1$*,* $NextA = Real\text{-}Next(I_A)$*,* $Val_A = Real\text{-}Val(I_A)$ *and* $NextB = Real\text{-}Next(I_B)$*, then* $Real\text{-}Val(I_A) \leq Real\text{-}Val(I_B)$*.*
- *If* $Flag = 0$*,* $NextB = Real\text{-}Next(I_B)$*,* $Val_B = Real\text{-}Val(I_B)$ *and* $NextA = Real\text{-}Next(I_A)$*, then* $Real\text{-}Val(I_B) \leq Real\text{-}Val(I_A)$*.*

▶ **Lemma 11.** *At any local optimum of the* Node-Max-Cut *instance in Figure 3:*

- *If* $Flag = 1$*, then* $NextB = Real\text{-}Next(I_B)$*.*
- *If* $Flag = 0$*, then* $NextA = Real\text{-}Next(I_A)$*.*

We should highlight that in the proofs of Lemma 10 and Lemma 11, in the first case where Flag = 1 (the case where Flag = 0 is symmetric), the correctness of values of the output vertices NextB (i.e., that NextB = Real-Next(I_B), in the first case of Lemma 11) is not guaranteed by Theorem 5 (as it happens with the correctness of the values of the output vertices NextA and Val$_A$ in the hypothesis of the first case in Lemma 10), but from the construction of the Comparator gadget. We should also highlight that Lemma 11 does not imply anything about the correctness of the Val$_B$ values, as this cannot be guaranteed in our construction. Hence, (the first case of) Lemma 10 does not assume anything about Val$_B$ (in particular, it does not assume that Val$_B$ = Real-Val(I_B), as one might have expected). However, the Comparator gadget manages to output the right outcome of the comparison Real-Val(I_A) ≤ Real-Val(I_B), even if Val$_B$ ≠ Real-Val(I_B). The connections from internal vertices of the Circuit Computing gadgets to the vertices in NextA and NextB and in the Comparator, in Figure 3, are crucial towards establishing this property. So, Lemma 10 ensures the robustness of the outcome of the Comparator, even with possibly inconsistent values in the output vertices Val$_B$. This property of the Comparator gadget is among the key technical steps (and novelties) in our reduction and is indicative of the difficulty of showing that Local-Node-Max-Cut is PLS-hard.

Furthermore, similarly to the Copy gadget, the Comparator gadget leaves the vertices of the Circuit Computing gadgets unbiased wrt. certain values of the Flag vertex.

▶ **Lemma 12.** *At any local optimum of the instance of* Node-Max-Cut *in Figure 3:*

- *If* $Flag = 1$*, then all vertices of* C_A *experience* 0 *bias from the Comparator gadget.*
- *If* $Flag = 0$*, then all vertices of* C_B *experience* 0 *bias from the Comparator gadget.*

Now using Lemma 8 and Lemma 12, we can prove the correctness of the output vertices NextA and Val$_A$, when Flag = 1, and of the output vertices NextB and Val$_B$, when Flag = 0.

▶ **Lemma 13.** *At any local optimum of the instance of* Node-Max-Cut *of Figure 3:*

- *If* $Flag = 1$*, then* $NextA = Real\text{-}Next(I_A)$ *and* $Val_A = Real\text{-}Val(I_A)$*.*
- *If* $Flag = 0$*, then* $NextB = Real\text{-}Next(I_B)$ *and* $Val_B = Real\text{-}Val(I_B)$*.*

Proof. We only consider the case where Flag = 1 (the same analysis applies to the case where Flag = 0). By Theorem 9, I_A = NextB, and by Lemma 7, T_B = NextB. As a result, $I_A = T_B$, and by Lemma 6, Control$_A$ = 1. Then, Lemmas 8 and 12 ensure that the vertices in NextA and Val$_A$ of the Computing Gadget C_A experience 0 bias from all the other gadgets. Therefore, since Control$_A$ = 1, we can apply Theorem 5, Case 1, and conclude that Val$_A$ = Real-Val(I_A) and that NextA = Real-Next(I_A). ◀

Concluding the Proof of Theorem 4. The technical lemmas above are summarized by the following key technical claim:

▶ **Theorem 14.** *At any local optimum of the instance of* NODE-MAX-CUT *of Figure 3:*

- *If Flag* $= 1$*, then (i) Real-Val*$(I_A) \leq$ *Real-Val*(I_B)*, and (ii) NextB* $=$ *Real-Next*(I_B)*.*
- *If Flag* $= 0$*, then (i) Real-Val*$(I_B) \leq$ *Real-Val*(I_A)*, and (ii) NextA* $=$ *Real-Next*(I_A)*.*

Proof of Theorem 14. We consider a local optimum of the instance in Figure 3 with Flag $= 1$ (the same argument applies when Flag $= 0$). By Lemma 11, NextB $=$ Real-Next(I_B), and thus, (ii) is established. Moreover by Lemma 13, NextA $=$ Real-Next(I_A) and Val$_A$ $=$ Real-Val(I_A). Consequently, by Lemma 10, we conclude Real-Val$(I_A) \leq$ Real-Val(I_B). ◀

With theorems 9 and 14 at hand, the PLS-completeness of LOCAL-NODE-MAX-CUT follows easily. For the sake of completeness, we show how we put everything together.

Proof of Theorem 4. For a given circuit C of CIRCUIT-FLIP, we construct in polynomial time the instance of NODE-MAX-CUT in Figure 3. We next consider any local optimum of this instance. Without loss of generality, we assume that Flag $= 1$. Then, by Theorem 9 and Theorem 14, $I_A =$ NextB, NextB $=$ Real-Next(I_B) and Real-Val$(I_A) \leq$ Real-Val(I_B). Hence, we obtain that

$$\text{Real-Val}(I_B) \geq \text{Real-Val}(I_A) = \text{Real-Val}(\text{NextB}) = \text{Real-Val}(\text{Real-Next}(I_B))\,.$$

If $I_B \neq$ Real-Next(I_B), then Real-Val$(I_B) >$ Real-Val(Real-Next(I_B)), which is a contradiction. Therefore, $I_B =$ Real-Next(I_B), meaning that the binary string defined by the values of I_B is a locally optimal solution for CIRCUIT-FLIP. ◀

6 Conclusions and Future Work

In this work, we showed that equilibrium computation in linear weighted congestion games is PLS-complete either on single-commodity series-parallel networks or on multi-commodity networks with identity latency functions, where computing an equibrium for (unweighted) congestion games is known to be easy. The key step for the latter reduction is to show that local optimum computation for NODE-MAX-CUT, a natural and significant restriction of MAX-CUT, is PLS-complete. The reductions in Section 3 are both *tight* [33], thus preserving the structure of the local search graph. In particular, for the first reduction, we have that (i) there are instances of linear weighted congestion games on single-commodity series-parallel networks such that any best response sequence has exponential length; and (ii) that the problem of computing the equilibrium reached from a given initial state is PSPACE-hard.

However, our reduction of CIRCUIT-FLIP to NODE-MAX-CUT is not tight. Specifically, our *Copy* and *Equality* gadgets allow that the *Circuit Computing* gadget might enter its *compute* mode, before the entire input has changed. Thus, we might “jump ahead” and reach an equilibrium before CIRCUIT-FLIP would allow, preventing the reduction from being tight.

Our work leaves several interesting directions for further research. A natural first step is to investigate the complexity of equilibrium computation for weighted congestion games on series-parallel (or extension-parallel) networks with identity latency functions. An intriguing research direction is to investigate whether our ideas (and gadgets) in the PLS-reduction for NODE-MAX-CUT could lead to PLS-hardness results for approximate equilibrium computation for standard and weighted congestion games (similarly to the results of Skopalik and Vöcking [34], but for latency functions with non-negative coefficients). Finally, it would be interesting to understand better the quality of efficiently computable approximate equilibria for NODE-MAX-CUT and the smoothed complexity of its local optima.

References

1 Heiner Ackermann, Heiko Röglin, and Berthold Vöcking. On the impact of combinatorial structure on congestion games. *J. ACM*, 55(6):25:1–25:22, 2008.

2 Omer Angel, Sébastien Bubeck, Yuval Peres, and Fan Wei. Local Max-Cut in Smoothed Polynomial Time. In *Proc. of the 49th ACM SIGACT Symposium on Theory of Computing (STOC 2017)*, pages 429–437, 2017.

3 Anand Bhalgat, Tanmoy Chakraborty, and Sanjeev Khanna. Approximating pure Nash equilibrium in cut, party affiliation, and satisfiability games. In *Proc. of the 11th ACM Conference on Electronic Commerce (EC 2010)*, pages 73–82, 2010.

4 Ioannis Caragiannis and Angelo Fanelli. On Approximate Pure Nash Equilibria in Weighted Congestion Games with Polynomial Latencies. In *Proc. of the 46th International Colloquium on Automata, Languages, and Programming, (ICALP 2019)*, volume 132 of *LIPIcs*, pages 133:1–133:12, 2019.

5 Ioannis Caragiannis, Angelo Fanelli, Nick Gravin, and Alexander Skopalik. Efficient Computation of Approximate Pure Nash Equilibria in Congestion Games. In *Proc. of the IEEE 52nd Symposium on Foundations of Computer Science, (FOCS 2011)*, pages 532–541, 2011.

6 Ioannis Caragiannis, Angelo Fanelli, Nick Gravin, and Alexander Skopalik. Approximate Pure Nash Equilibria in Weighted Congestion Games: Existence, Efficient Computation, and Structure. *ACM Transactions on Economics and Computation*, 3(1):2:1–2:32, 2015.

7 Xi Chen, Chenghao Guo, Emmanouil-Vasileios Vlatakis-Gkaragkounis, Mihalis Yannakakis, and Xinzhi Zhang. Smoothed complexity of local Max-Cut and binary Max-CSP. In *Proc. of the 52nd Annual ACM Symposium on Theory of Computing (STOC 2020)*, 2020.

8 Steve Chien and Alistair Sinclair. Convergence to approximate Nash equilibria in congestion games. *Games and Economic Behavior*, 71(2):315–327, 2011.

9 Robert Elsässer and Tobias Tscheuschner. Settling the Complexity of Local Max-Cut (Almost) Completely. In *Proc. of the 38th International Colloquium on Automata, Languages and Programming (ICALP 2011)*, pages 171–182, 2011.

10 Michael Etscheid and Heiko Röglin. Smoothed analysis of local search for the maximum-cut problem. *ACM Transactions on Algorithms*, 13(2):25:1–25:12, 2017.

11 Eyal Even-Dar, Alexander Kesselman, and Yishay Mansour. Convergence time to nash equilibria. In *Proc. of the 30th International Colloquium on Automata, Languages and Programming (ICALP 2003)*, pages 502–513, 2003.

12 Alex Fabrikant, Christos H. Papadimitriou, and Kunal Talwar. The Complexity of Pure Nash Equilibria. In *Proc. of the 36th Annual ACM Symposium on Theory of Computing (STOC 2004)*, pages 604–612, 2004.

13 Angelo Fanelli and Luca Moscardelli. On best response dynamics in weighted congestion games with polynomial delays. In *Proc. of the 5th Workshop on Internet and Network Economics (WINE 2009)*, pages 55–66, 2009.

14 Matthias Feldotto, Martin Gairing, Grammateia Kotsialou, and Alexander Skopalik. Computing approximate pure nash equilibria in shapley value weighted congestion games. In *Proc. of the 13th International Conference on Web and Internet Economics (WINE 2017)*, pages 191–204, 2017.

15 Dimitris Fotakis. A selective tour through congestion games. In *Algorithms, Probability, Networks, and Games: Scientific Papers and Essays Dedicated to Paul G. Spirakis on the Occasion of His 60th Birthday*, pages 223–241, 2015.

16 Dimitris Fotakis, Spyros C. Kontogiannis, Elias Koutsoupias, Marios Mavronicolas, and Paul G. Spirakis. The structure and complexity of Nash equilibria for a selfish routing game. *Theoretical Computer Science*, 410(36):3305–3326, 2009.

17 Dimitris Fotakis, Spyros C. Kontogiannis, and Paul G. Spirakis. Selfish unsplittable flows. *Theoretical Computer Science*, 348(2-3):226–239, 2005.

18 Dimitris Fotakis, Spyros C. Kontogiannis, and Paul G. Spirakis. Symmetry in Network Congestion Games: Pure Equilibria and Anarchy Cost. In *Proc. of the 3rd Workshop on Approximation and Online Algorithms, Revised Papers (WAOA 2005)*, pages 161–175, 2005.

19 Martin Gairing, Thomas Lücking, Marios Mavronicolas, and Burkhard Monien. Computing Nash equilibria for scheduling on restricted parallel links. In *Proc. of the 36th Annual ACM Symposium on Theory of Computing (STOC 2004)*, pages 613–622, 2004.

20 Martin Gairing and Rahul Savani. Computing Stable Outcomes in Hedonic Games. In *Proc. of the 3rd Symposium on Algorithmic Game Theory (SAGT 2010)*, pages 174–185, 2010.

21 Yiannis Giannakopoulos, Georgy Noarov, and Andreas S. Schulz. An improved algorithm for computing approximate equilibria in weighted congestion games. *CoRR*, abs/1810.12806, 2018. `arXiv:1810.12806`.

22 Paul W. Goldberg. Bounds for the convergence rate of randomized local search in a multiplayer load-balancing game. In *Proc. of the 23rd Annual ACM Symposium on Principles of Distributed Computing (PODC 2004)*, pages 131–140, 2004.

23 Tobias Harks and Max Klimm. On the Existence of Pure Nash Equilibria in Weighted Congestion Games. In *Proc. of the 37th International Colloquium on Automata, Languages and Programming (ICALP 2010)*, pages 79–89, 2010.

24 Tobias Harks, Max Klimm, and Rolf H. Möhring. Characterizing the existence of potential functions in weighted congestion games. *Theory Computing Systems*, 49(1):46–70, 2011.

25 David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. How easy is local search? *Journal of Computer and System Sciences*, 37(1):79–100, 1988.

26 Richard M. Karp. Reducibility among combinatorial problems. In *Proc. of Symposium on the Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103, 1972.

27 Pieter Kleer and Guido Schäfer. Potential Function Minimizers of Combinatorial Congestion Games: Efficiency and Computation. In *Proc. of the 2017 ACM Conference on Economics and Computation (EC 2017)*, pages 223–240, 2017.

28 Wil Michiels, Emile Aarts, and Jan Korst. *Theoretical Aspects of Local Search.* EATCS Monographs in Theoretical Computer Science. Springer, 2007.

29 Dov Monderer and Lloyd S. Shapley. Potential Games. *Games and economic behavior*, 14(1):124–143, 1996.

30 Panagiota N. Panagopoulou and Paul G. Spirakis. Algorithms for pure Nash equilibria in weighted congestion games. *ACM Journal of Experimental Algorithmics*, 11, 2006.

31 Svatopluk Poljak. Integer Linear Programs and Local Search for Max-Cut. *SIAM Journal on Computing*, 21(3):450–465, 1995.

32 R.W. Rosenthal. A Class of Games Possessing Pure-Strategy Nash Equilibria. *International Journal of Game Theory*, 2:65–67, 1973.

33 Alejandro A. Schäffer and Mihalis Yannakakis. Simple Local Search Problems That are Hard to Solve. *SIAM Journal on Computing*, 20(1):56–87, 1991.

34 Alexander Skopalik and Berthold Vöcking. Inapproximability of Pure Nash Equilibria. In *Proc. of the 40th Annual ACM Symposium on Theory of Computing (STOC 2008)*, pages 355–364, 2008.

35 J. Valdez, R.E. Tarjan, and E.L. Lawler. The Recognition of Series-Parallel Digraphs. *SIAM J. Comput.*, 11(2):298–313, 1982.

The Online Min-Sum Set Cover Problem

Dimitris Fotakis
National Technical University of Athens, Greece
fotakis@cs.ntua.gr

Loukas Kavouras
National Technical University of Athens, Greece
lukaskavouras@gmail.com

Grigorios Koumoutsos
Université libre de Bruxelles, Belgium
gregkoumoutsos@gmail.com

Stratis Skoulakis
Singapore University of Technology and Design, Singapore
efstratios@sutd.edu.sg

Manolis Vardas
ETH Zurich, Switzerland
evardas@student.ethz.ch

Abstract

We consider the online Min-Sum Set Cover (MSSC), a natural and intriguing generalization of the classical list update problem. In Online MSSC, the algorithm maintains a permutation on n elements based on subsets $S_1, S_2, \ldots$ arriving online. The algorithm serves each set S_t upon arrival, using its current permutation π_t, incurring an access cost equal to the position of the first element of S_t in π_t. Then, the algorithm may update its permutation to π_{t+1}, incurring a moving cost equal to the Kendall tau distance of π_t to π_{t+1}. The objective is to minimize the total access and moving cost for serving the entire sequence. We consider the r-uniform version, where each S_t has cardinality r. List update is the special case where $r = 1$.

We obtain tight bounds on the competitive ratio of deterministic online algorithms for MSSC against a static adversary, that serves the entire sequence by a single permutation. First, we show a lower bound of $(r+1)(1-\frac{r}{n+1})$ on the competitive ratio. Then, we consider several natural generalizations of successful list update algorithms and show that they fail to achieve any interesting competitive guarantee. On the positive side, we obtain a $O(r)$-competitive deterministic algorithm using ideas from online learning and the multiplicative weight updates (MWU) algorithm.

Furthermore, we consider efficient algorithms. We propose a memoryless online algorithm, called *Move-All-Equally*, which is inspired by the Double Coverage algorithm for the k-server problem. We show that its competitive ratio is $\Omega(r^2)$ and $2^{O(\sqrt{\log n \cdot \log r})}$, and conjecture that it is $f(r)$-competitive. We also compare Move-All-Equally against the dynamic optimal solution and obtain (almost) tight bounds by showing that it is $\Omega(r\sqrt{n})$ and $O(r^{3/2}\sqrt{n})$-competitive.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases Online Algorithms, Competitive Analysis, Min-Sum Set Cover

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.51

Category Track A: Algorithms, Complexity and Games

Related Version The full version of the paper is available at `https://arxiv.org/abs/2003.02161`.

Funding *Dimitris Fotakis*: Partially supported by the Hellenic Foundation for Research and Innovation (H.F.R.I.) under the "First Call for H.F.R.I. Research Projects to support Faculty members and Researchers' and the procurement of high-cost research equipment grant", project: BALSAM (id: 1424).
Loukas Kavouras: Partially supported by a scholarship from the State Scholarships Foundation.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 51; pp. 51:1–51:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Grigorios Koumoutsos: Supported by Fonds de la Recherche Scientifique-FNRS Grant no MISU F 6001. Part of this work was carried out while visiting the National Technical University of Athens, supported by FNRS Mobility Grant no 35282070.
Stratis Skoulakis: Partially supported by NRF 2018 Fellowship NRF-NRFF2018-07. Part of this research was carried out while the author was a PhD student at the National Technical University of Athens.
Manolis Vardas: This research was carried out while the author was an undergraduate student at the National Technical University of Athens.

1 Introduction

In Min-Sum Set Cover (MSSC), we are given a universe U on n elements and a collection of subsets $\mathcal{S} = \{S_1, \dots, S_m\}$, with $S_t \subseteq U$, and the task is to construct a permutation (or list) π of elements of U. The cost $\pi(S_t)$ of covering a set S_t (a.k.a. the cover time of S_t) with a permutation π is the position of the first element of S_t in π, i.e., $\pi(S_t) = \min\{i \,|\, \pi(i) \in S_t\}$. The goal is to minimize the overall cost $\sum_t \pi(S_t)$ of covering all subsets of $\mathcal{S}$.

The MSSC problem generalizes various NP-hard problems such as Min-Sum Vertex Cover and Min-Sum Coloring and it is well-studied. Feige, Lovasz and Tetali [25] showed that the greedy algorithm, which picks in each position the element that covers the most uncovered sets, is a 4-approximation (this was also implicit in [11]) and that no $(4-\epsilon)$-approximation is possible, unless P = NP. Several generalizations have been considered over the years with applications in various areas (we discuss some of those problems and results in Section 1.2).

Online Min-Sum Set Cover. In this paper, we study the online version of Min-Sum Set Cover. Here, the sets arrive online; at time step t, the set S_t is revealed. An online algorithm is charged the *access cost* of its current permutation $\pi_t(S_t)$; then, it is allowed to change its permutation to π_{t+1} at a *moving cost* equal to the number of inversions between π_t and π_{t+1}, known as the Kendall tau distance $d_{\text{KT}}(\pi_t, \pi_{t+1})$. The goal is to minimize the total cost, i.e., $\sum_t \big(\pi_t(S_t) + d_{\text{KT}}(\pi_t, \pi_{t+1})\big)$. This is a significant generalization of the classic list update problem, which corresponds to the special case where $|S_t| = 1$ for all sets $S_t \in \mathcal{S}$.

Motivation. Consider a web search engine, such as Google. Each query asked might have many different meanings depending on the user. For example, the query "Python" might refer to an animal, a programming language or a movie. Given the pages related to "Python", a goal of the search engine algorithm is to rank them such that for each user, the pages of interest appear as high as possible in the ranking (see e.g., [23]). Similarly, news streams include articles covering different reader interests each. We want to rank the articles so that every reader finds an article of interest as high as possible. The MSSC problem serves as a theoretical model for practical problems of this type, where we want to aggregate disjunctive binary preferences (expressed by the input sets) into a total order. E.g., for a news stream, the universe U corresponds to the available articles and the sets S_t correspond to different user types. The cost of a ranking (i.e., permutation on U) for a user type is the location of the first article of interest. Clearly, in such applications, users arrive online and the algorithm might need to re-rank the stream (i.e., change the permutation) based on user preferences.

Benchmarks. For the most part, we evaluate the performance of online algorithms by comparing their cost against the cost of an optimal offline solution that knows the input in advance and chooses an optimal permutation π. Note that this solution is *static*, in the sense that it does not change permutations over time. This type of analysis, called *static optimality*, is typical in online optimization and online learning. It was initiated in the context of adaptive data structures by the landmark result of Sleator and Tarjan [44], who showed that *splay trees* are asymptotically as fast as any *static* tree. Since then, it has been an established benchmark for various problems in this area (see e.g. [13, 30]); it is also a standard benchmark for several other problems in online optimization (e.g., online facility location [26, 37], minimum metric matching [28, 33, 39], Steiner tree [38], etc.).

A much more general benchmark is the *dynamic Min-Sum Set Cover* problem, where the algorithm is compared against an optimal solution allowed to change permutations over time. This problem has not been studied even in the offline case. In this work, we define the problem formally and obtain first results for the online case.

We remark that the online dynamic MSSC problem belongs to a rich class of problems called *Metrical Task Systems* (MTS) [15]. MTS is a far-reaching generalization of several fundamental online problems and provides a unified framework for studying online problems (we discuss this in more detail in Section 1.2). Indeed, our results suggest that solving the online dynamic MSSC requires the development of powerful generic techniques for online problems, which might have further implications for the broader setting of MTS.

Throughout this paper, whenever we refer to online problems, like Min-Sum Set Cover or list update, we assume the static case, unless stated otherwise.

Previous Work on List Update. Prior to our work, the only version of online MSSC studied is the special case where $|S_t| = 1$ for all sets; this is the celebrated list update problem and it has been extensively studied (an excellent reference is [14]). It is known that the deterministic competitive ratio it least $2 - \frac{2}{n+1}$ and there are several 2-competitive algorithms known; most notably, the Move-to-Front (MTF) algorithm, which moves the (unique) element of S_t to the first position of the permutation, and the Frequency Count algorithm, which orders the elements in decreasing order according to their frequencies.

The dynamic list update problem has also been extensively studied. MTF is known to be 2-competitive [43] and there are several other 2-competitive algorithms [1, 24].

1.1 Our Results

In this work, we initiate a systematic study of the online Min-Sum Set Cover problem. We consider the r-uniform case, where all request sets have the same size $|S_t| = r$. This is without loss of generality, as we explain in Section 1.3.

The first of our main results is a tight bound on the deterministic competitive ratio of Online MSSC. We show that the competitive ratio of deterministic algorithms is $\Omega(r)$.

▶ **Theorem 1.** *Any deterministic online algorithm for the Online Min-Sum Set Cover problem has competitive ratio at least* $(r + 1)(1 - \frac{r}{n+1})$.

Note that for $r = 1$, this bound evaluates to $2 - \frac{2}{n+1}$, which is exactly the best known lower bound for the list update problem.

We complement this result by providing a matching (up to constant factors) upper bound.

▶ **Theorem 2.** *There exists a* $(5r + 2)$*-competitive deterministic online algorithm for the Online Min-Sum Set Cover problem.*

Interestingly, all prior work on the list update problem (case $r = 1$) does not seem to provide us with the right tools for obtaining an algorithm with such guarantees! As we discuss in Section 2, virtually all natural generalizations of successful list update algorithms (e.g., Move-to-Front, Frequency Count) end up with a competitive ratio way far from the desired bound. In fact, even for $r = 2$, most of them have a competitive ratio depending on n, such as $\Omega(\sqrt{n})$ or even $\Omega(n)$.

This suggests that online MSSC has a distinctive combinatorial structure, very different from that of list update, whose algorithmic understanding calls for significant new insights. The main reason has to do with the disjunctive nature of the definition of the access cost $\pi(S_t)$. In list update, where $r = 1$, the optimal solution is bound to serve a request S_t by its unique element. The only question is how fast an online algorithm should upgrade it (and the answer is "as fast as possible"). In MSSC, the hard (and crucial) part behind the design of any competitive algorithm is how to ensure that the algorithm learns fast enough about the element e_t used by the optimal solution to serve each request S_t. This is evident in the highly adaptive nature of the deceptively simple greedy algorithm of [25] and in the adversarial request sequences for generalizations of Move-to-Front, in Section 2.

To obtain the asymptotically optimal ratio of Theorem 2, we develop a rounding scheme and use it to derandomize the multiplicative weights update (MWU) algorithm. Our analysis bounds the algorithm's access cost in terms of the optimal cost, but it does not account for the algorithm's moving cost. We then refine our approach, by performing lazy updates to the algorithm's permutation, and obtain a competitive algorithm for online MSSC.

We also observe (in Section 1.3) that based on previous work of Blum and Burch [12], there exists a (computationally inefficient) randomized algorithm with competitive ratio $1 + \epsilon$, for any $\epsilon \in (0, 1/4)$. This implies that no lower bound is possible, if randomization is allowed, and gives a strong separation between deterministic and randomized algorithms.

Memoryless Algorithms. While the bounds of Theorems 1 and 2 are matching, our algorithm from Theorem 2 is computationally inefficient since it simulates the MWU algorithm, which in turn, maintains a probability distribution over all $n!$ permutations. This motivates the study of trade-offs between the competitive ratio and computational efficiency. To this end, we propose a memoryless algorithm, called *Move-All-Equally* (MAE), which moves all elements of set S_t towards the beginning of the permutation at the same speed until the first reaches the first position. This is inspired by the Double Coverage algorithm from k-server [20, 21]. We believe that MAE achieves the best guarantees among all memoryless algorithms. We show that this algorithm can not match the deterministic competitive ratio.

▶ **Theorem 3.** *The competitive ratio of the Move-All-Equally algorithm is* $\Omega(r^2)$.

Based on Theorem 3, we conjecture that an $O(r)$ guarantee cannot be achieved by a memoryless algorithms. We leave as an open question whether MAE has a competitive ratio $f(r)$, or a dependence on n is necessary. To this end, we show that the competitive ratio of MAE is at most $2^{O(\sqrt{\log n \cdot \log r})}$ (see Section 4 for details).

Dynamic Min-Sum Set Cover. We also consider the dynamic version of online MSSC. Dynamic MSSC is much more general and the techniques developed for the static case do not seem adequately powerful. This is not surprising, since the MWU algorithm is designed to perform well against the best static solution. We investigate the performance of the MAE algorithm. First, we obtain an upper bound on its competitive ratio.

▶ **Theorem 4.** *The competitive ratio of the Move-All-Equally algorithm for the dynamic online Min-Sum Set Cover problem is* $O(r^{3/2}\sqrt{n})$.

Although this guarantee is not very strong, we show that, rather surprisingly, it is essentially tight and no better guarantees can be shown for this algorithm.

▶ **Theorem 5.** *For any $r \geq 3$, the competitive ratio of the Move-All-Equally algorithm for the dynamic online Min-Sum Set Cover problem is $\Omega(r\sqrt{n})$.*

This lower bound is based on a carefully crafted adversarial instance; this construction reveals the rich structure of this problem and suggests that more powerful generic techniques are required in order to achieve any $f(r)$ guarantees. In fact, we conjecture that the lower bound of Theorem 1 is the best possible (ignoring constant factors) even for the dynamic problem and that using a work-function based approach such a bound can be obtained.

1.2 Further Related Work

Multiple Intents Re-ranking. This is a generalization of MSSC where for each set S_t, there is a *covering requirement* $K(S_t)$, and the cost of covering a set S_t is the position of the $K(S_t)$-th element of S_t in π. The MSSC problem is the special case where $K(S_t) = 1$ for all sets S_t. Another notable special case is the Min-Latency Set Cover problem, which corresponds to the other extreme case where $K(S_t) = |S_t|$ [29]. Multiple Intents Re-ranking was first studied by Azar et. al. [5], who presented a $O(\log r)$-approximation; later $O(1)$-approximation algorithms were obtained [10,32,42]. Further generalizations have been considered, such as the Submodular Ranking problem, studied by Azar and Gamzu [4], which generalizes both Set Cover and MSSC, and the Min-Latency Submodular Cover, studied by Im et.al [31].

Prediction from Expert Advice and Randomized MSSC. In prediction from expert advice, there are N experts and expert i incurs a cost c_i^t in each step. A learning algorithm decides which expert i_t to follow (before the cost vector $\mathbf{c}^t$ is revealed) and incurs a cost of $c_{i_t}^t$. The landmark technique for solving this problem is the multiplicative weights update (MWU - a.k.a. Hedge) algorithm. For an in-depth treatment of MWU, we refer to [3,27,35].

In the classic online learning setting, there is no cost for moving probability mass between experts. However, in a breakthrough result, Blum and Burch [12] showed that MWU is $(1+\epsilon)$-competitive against the best expert, even if there is a cost D for moving probability mass between experts. By adapting this result to online MSSC (regarding permutations as experts), we can get an (inefficient) randomized algorithm with competitive ratio $(1+\epsilon)$, for any constant $\epsilon \in (0, 1/4)$. A detailed description is deferred to the full version of this paper.

Metrical Task Systems and Online Dynamic MSSC. The online dynamic Min-Sum Set Cover problem belongs to a rich family of problems called Metrical Task Systems (MTS). In MTS, we are given a set of N states and a metric function d specifying the cost of moving between the states. At each step, a task arrives; the cost of serving the task at state i is c_i. An algorithm has to choose a state to process the task. If it switches from state i to state j and processes the task there, it incurs a cost $d(i,j) + c_j$. Given an initial state and a sequence of requests, the goal is to process all tasks at minimum cost.

It is easy to see that the online version of dynamic MSSC problem is a MTS, where the states correspond to permutations, thus $N = n!$, and the distance between two states is their Kendall tau distance. For a request set S_t, the request is a vector specifying the cost $\pi(S_t)$ for every permutation π.

Several other fundamental online problems (e.g., k-server, convex body chasing) are MTS. Although there has been a lot of work on understanding the structure of MTS problems [2,8,9,15,16,22,34,40,41], there is not a good grasp on how the structure relates to the hardness of MTS problems. Getting a better understanding on this area is a long-term goal, since it would lead to a systematic framework for solving online problems.

1.3 Preliminaries

Notation. Given a request sequence $\mathcal{S} = \{S_1, \dots, S_m\}$, for any algorithm ALG we denote Cost(ALG($\mathcal{S}$)) or simply Cost(ALG) the total cost of ALG on $\mathcal{S}$. Similarly we denote AccessCost(ALG) the total access cost of ALG and MovingCost(ALG) the total movement cost of ALG. For a particular time step t, an algorithm using permutation π_t incurs an access cost AccessCost(ALG(t)) $= \pi_t(S_t)$. We denote by $\pi_t[j]$ the position of element $j \in U$ in the permutation π_t.

Online Min-Sum Set Cover. We focus on the r-uniform case, i.e., when all sets S_t have size $r \ll n$. This is essentially without loss of generality, because we can always let $r = \max_t |S_t|$ and add the $r - |S_t|$ last unrequested elements in the algorithm's permutation to any set S_t with $|S_t| < r$. Assuming that $r \leq n/2$, this modification cannot increase the optimal cost and cannot decrease the online cost by more than a factor of 2.

2 Lower Bounds on the Deterministic Competitive Ratio

We start with a lower bound on the deterministic competitive ratio of online MSSC.

▶ **Theorem 1.** *Any deterministic online algorithm for the Online Min-Sum Set Cover problem has competitive ratio at least* $(r+1)(1 - \frac{r}{n+1})$.

For the proof, we employ an averaging argument, similar to those in lower bounds for list update and k-server [36, 43]. In each step, the adversary requests the last r elements in the algorithm's permutation. Hence, the algorithm's cost is at least $(n - r + 1)$. Using a counting argument, we show that for any fixed set S_t of size r and any $i \in [n - r + 1]$, the number of permutations π with access cost $\pi(S_t) = i$ is $\binom{n-i}{r-1} r!(n-r)!$. Summing up over all permutations and dividing by $n!$, we get that the average access cost for S_t is $\binom{n+1}{r+1}\frac{r!(n-r)!}{n!} = \frac{n+1}{r+1}$. Therefore, the cost of the optimal permutation is a most $\frac{(n+1)}{r+1}$, and the competitive ratio of the algorithm at least $\frac{(n-r+1)(r+1)}{n+1}$. The details can be found in the full version of this paper.

Lower Bounds for Generalizations of Move-to-Front. For list update, where $r = 1$, simple algorithms like Move-to-Front (MTF) and Frequency Count achieve an optimal competitive ratio. We next briefly describe several such generalizations of them and show that their competitive ratio depends on n, even for $r = 2$. Missing details can be found in the full version.

MTF$_{\text{first}}$: Move to the first position (of the algorithm's permutation) the element of S_t appearing first in π_t . This algorithm is $\Omega(n)$-competitive when each request S_t consists of the last two elements in π_t. Then, the last element in the algorithm's permutation never changes and is used by the optimal permutation to serve the entire sequence!

MTF$_{\text{last}}$: Move to the first position the element of S_t appearing last in π_t .

MTF$_{\text{all}}$: Move to the first r positions all elements of S_t (in the same order as in π_t).

MTF$_{\text{random}}$: Move to the first position an element of S_t selected uniformly at random.

MTF$_{\text{last}}$, MTF$_{\text{all}}$ and MTF$_{\text{random}}$ have a competitive ratio of $\Omega(n)$ when each request S_t consists of a fixed element e (always the same) and the last element in π_t, because they all incur an (expected for MTF$_{\text{random}}$) moving cost of $\Theta(n)$ per request.

The algorithms seen so far fail for the opposite reasons: MTF$_{\text{first}}$ cares only about the first element and ignores completely the second, and the others are very aggressive on using the second (rth) element. A natural attempt to balance those two extremes is the following.

$\text{MTF}_{\text{relative}}$: Let i be the position of the first element of S_t in π_t. Move to the first positions of the algorithm's permutation (keeping their relative order) all elements of S_t appearing up to the position $c \cdot i$ in π_t, for some constant c. The bad instance for this algorithm is when each request S_t consists of the last element and the element at position $\lfloor n/c \rfloor - 1$ in π_t; it never uses the nth element and the adversary serves all requests with it at a cost of 1.

All generalizations of MTF above are memoryless and they all fail to identify the element by which optimal serves S_t. The following algorithm tries to circumvent this by keeping memory and in particular the frequencies of reqested elements.

$\text{MTF}_{\text{count}}$: Move to the first position the most frequent element of S_t (i.e., the element of S_t appearing in most requested sets so far).

This algorithm behaves better in easy instances, however with some more work we can show a lower bound of $\Omega(\sqrt{n})$ on its competitive ratio. Let $e_1, \dots, e_n$ be the elements indexed according to the initial permutation π_0 and $b = \sqrt{n}$. The request sequence proceeds in m/n phases of length n each. The first $n - b$ requests of each phase are $\{e_1, e_2\}, \{e_1, e_3\}, \dots, \{e_1, e_{n-b}\}$, and the last b requests consist of e_{n-b+i} and the element at position $n - b$ at the current algorithm's permutation, for $i = 1, \dots, b$. An optimal solution can cover all the requests by the elements $e_1, e_{n-b+1}, \dots, e_n$ with total cost $\Theta(m + n\sqrt{n})$. The elements $e_{n-b+1}, \dots, e_n$ are never upgraded by $\text{MTF}_{\text{count}}$. Hence, the algorithm's cost is $\Theta(m\sqrt{n})$.

3 An Algorithm with Asymptotically Optimal Competitive Ratio

Next, we present algorithm Lazy-Rounding (Algorithm 2) and analyze its competitive ratio. The following is the main result of this section:

▶ **Theorem 2.** *Deterministic online algorithm* Lazy-Rounding*, presented in Algorithm 2, is* $(5r + 2)$*-competitive for the static version of the Online Min-Sum Set Cover problem.*

The remainder of this section is devoted to the proof of Theorem 2. At a high-level, our approach is summarized by the following three steps:

1. We use as black-box the multiplicative weights update (MWU) algorithm with learning rate $1/n^3$. Using standard results from learning theory, we show that its expected access cost is within a factor 5/4 of OPT, i.e., $\text{AccessCost(MWU)} \le \frac{5}{4}\,\text{Cost(OPT)}$ (Section 3.1).
2. We develop an online rounding scheme, which turns any randomized algorithm $\mathcal{A}$ into a deterministic one, denoted $\text{Derand}(\mathcal{A})$, with access cost at most $2r \cdot \mathbb{E}[\text{AccessCost}(\mathcal{A})]$ (Section 3.2). However, our rounding scheme does not provide any immediate guarantee on the moving cost of $\text{Derand}(\mathcal{A})$.
3. Lazy-Rounding is a lazy version of Derand(MWU) that updates its permutation only if MWU's distribution has changed a lot. A *phase* corresponds to a time interval that Lazy-Rounding does not change its permutation. We show that during a phase:
   - **(i)** The upper bound on the access cost increases, compared to Derand(MWU), by a factor of at most 2, i.e., $\text{AccessCost(Lazy-Rounding)} \le 4r \cdot \mathbb{E}[\text{AccessCost(MWU)}]$ (Lemma 11).
   - **(ii)** The (expected) access cost of MWU is at least n^2. Since our algorithm moves only once per phase, its movement cost is at most n^2. Thus we get that (Lemma 12):

$$\text{MovingCost(Lazy-Rounding)} \le \mathbb{E}[\text{AccessCost(MWU)}]\,.$$

For the upper bound on the moving cost above, we relate how much MWU's distribution changes during a phase, in terms of the total variation distance, to the cost of MWU and the cost of our algorithm.

Based on the above properties, we compare the access and the moving cost of Lazy-Rounding against the access cost of MWU and to get the desired competitive ratio:

$$\mathrm{Cost}(\text{Lazy-Rounding}) \leq (4r+1)\,\mathbb{E}[\mathrm{AccessCost}(\mathrm{MWU})] \leq (5r+2)\,\mathrm{Cost}(\mathrm{OPT})\,.$$

Throughout this section we denote by $d_{\mathrm{TV}}(\delta, \delta')$ the total variation distance of two discrete probability distributions $\delta, \delta' : [N] \to [0,1]$, defined as $d_{\mathrm{TV}}(\delta, \delta') = \sum_{i=1}^{N} \max\{0, \delta(i) - \delta'(i)\}$.

3.1 Using Multiplicative Weights Update in Online Min-Sum Set Cover

In this section, we explain how the well-known MWU algorithm [27,35] is used in our context.

The MWU Algorithm. Given $n!$ permutations of elements of U, the algorithm has a parameter $\beta \in [0,1]$ and a weight w_π for each permutation $\pi \in [n!]$, initialized at 1. At each time step the algorithm chooses a permutation according to distribution $\mathrm{P}^t_\pi = w^t_\pi / (\sum_{\pi \in [n!]} w^t_\pi)$. When request S_t arrives, MWU incurs an expected access cost of

$$\mathbb{E}[\mathrm{AccessCost}(\mathrm{MWU}(t))] = \sum_{\pi \in [n!]} \mathrm{P}^t_\pi \cdot \pi(S_t)$$

and updates its weights $w^{t+1}_\pi = w^t_\pi \cdot \beta^{\pi(S_t)}$, where $\beta = e^{-1/n^3}$; this is the so-called *learning rate* of our algorithm. Later on, we discuss the reasons behind choosing this value.

On the Access Cost of MWU. Using standard results from learning theory [27,35] and adapting them to our setting, we get that the (expected) access cost of MWU is bounded by Cost(OPT). This is formally stated in Lemma 6 (and is proven in the full version).

▶ **Lemma 6.** *For any request sequence $\sigma = (S_1, \ldots, S_m)$ we have that*

$$\mathbb{E}[\mathrm{AccessCost}(\mathrm{MWU})] \leq \frac{5}{4} \cdot \mathrm{Cost}(\mathrm{OPT}) + 2n^4 \ln n\,.$$

On the Distribution of MWU. We now relate the expected access cost of the MWU algorithm to the total variation distance among MWU's distributions. More precisely, we show that if the total variation distance between MWU's distributions at times t_1 and t_2 is large, then MWU has incurred a sufficiently large access cost. The proof of the following makes a careful use of MWU's properties and is deferred to the full version of this paper.

▶ **Lemma 7.** *Let P^t be the probability distribution of the MWU algorithm at time t. Then,*

$$d_{\mathrm{TV}}(\mathrm{P}^t, \mathrm{P}^{t+1}) \leq \frac{1}{n^3} \cdot \mathbb{E}[\mathrm{AccessCost}(\mathrm{MWU}(t))].$$

The following is useful for the analysis of Lazy-Rounding. Its proof follows from Lemma 7 and the the triangle inequality and is deferred to the full version of this paper.

▶ **Lemma 8.** *Let t_1 and t_2 two different time steps such that $d_{\mathrm{TV}}(\mathrm{P}^{t_1}, \mathrm{P}^{t_2}) \geq 1/n$. Then,*

$$\sum_{t=t_1}^{t_2-1} \mathbb{E}[\mathrm{AccessCost}(\mathrm{MWU}(t))] \geq n^2\,.$$

3.2 Rounding

Next, we present our rounding scheme. Given as input a probability distribution δ over permutations, it outputs a fixed permutation ρ such that for each possible request set S of size r, the cost of ρ on S is within a $O(r)$ factor of the expected cost of the distribution δ on S. For convenience, we assume that n/r is an integer. Otherwise, we use $\lceil n/r \rceil$.

Algorithm 1 Greedy-Rounding (derandomizing probability distributions over the permutations).

Input: A probability distribution δ over $[n!]$.
Output: A permutation $\rho \in [n!]$.

```
R ← $U$
for $i = 1$ to $n/r$ do
    $S^i \leftarrow \arg\min_{S \in \{R\}^r} \mathbb{E}_{\pi \sim \delta}[\pi(S)]$
    Place the elements of $S^i$ (arbitrarily) from positions $(i-1) \cdot r + 1$ to $i \cdot r$ of $\rho$.
    $R \leftarrow R \setminus S^i$
end for
return $\rho$
```

Our rounding algorithm is described in Algorithm 1. At each step, it finds the request S with minimum expected covering cost under the probability distribution δ and places the elements of S as close to the beginning of the permutation as possible. Then, it removes those elements from set R and iterates. The main claim is that the resulting permutation has the following property: *any request S of size r has covering cost at most $O(r)$ times of its expected covering cost under the probability distribution δ.*

▶ **Theorem 9.** *Let δ be a distribution over permutations and let ρ be the permutation output by Algorithm 1 on δ. Then, for any set S, with $|S| = r$,*

$$\rho(S) \leq 2r \cdot \mathop{\mathbb{E}}_{\pi \sim \delta}[\pi(S)]\,.$$

Proof Sketch. The key step is to show that if the element used by ρ to serve the request S was picked during the kth iteration of the rounding algorithm, then $\mathbb{E}_{\pi \sim \delta}[\pi(S)] \geq k/2$. Clearly, $\rho(S) \leq k \cdot r$ and the theorem follows. Full proof is in the full version. ◀

3.3 The Lazy Rounding Algorithm

Lazy-Rounding, presented in Algorithm 2, is essentially a lazy derandomization of MWU. At each step, it calculates the distribution on permutations maintained by MWU. At the beginning of each *phase*, it sets its permutation to that given by Algorithm 1. Then, it sticks to the same permutation for as long as the total variation distance of MWU's distribution at the beginning of the phase to the current MWU distribution is at most $1/n$. As soon as the total variation distance exceeds $1/n$, Lazy-Rounding starts a new phase.

The main intuition behind the design of our algorithm is the following. In Section 3.2 we showed that Algorithm 1 results in a deterministic algorithm with access cost no larger than $2r\,\mathbb{E}[\text{AccessCost(MWU)}]$. However, such an algorithm may incur an unbounded moving cost; even small changes in the distribution of MWU could lead to very different permutations after rounding. To deal with that, we update the permutation of Lazy-Rounding only if there are substantial changes in the distribution of MWU. Intuitively, small changes in MWU's distribution should not affect much the access cost (this is formalized in Lemma 10). Moreover, Lazy-Rounding switches to a different permutation only if it is really required, which we use to bounds Lazy-Rounding's moving cost.

Bounding the Access Cost. We first show that the access cost of Lazy-Rounding is within a factor of $4r$ from the expected access cost of MWU (Lemma 11). To this end, we first show that if the total variation distance between two distributions is small, then sampling from those distributions yields roughly the same expected access cost for any request S. The proof of the following is based on the optimal coupling lemma and can be found in the full version of this paper.

▶ **Lemma 10.** *Let δ and δ' be two probability distributions over permutations. If that $d_{\mathrm{TV}}(\delta, \delta') \leq 1/n$, for any request set S of size r, we have that*

$$\mathop{\mathbb{E}}_{\pi\sim\delta'}[\pi(S)] \leq 2 \cdot \mathop{\mathbb{E}}_{\pi\sim\delta}[\pi(S)].$$

We are now ready to upper bound the access cost of our algorithm.

▶ **Lemma 11.** $\mathrm{AccessCost(Lazy\text{-}Rounding)} \leq 4r \cdot \mathbb{E}[\mathrm{AccessCost(MWU)}]$.

Proof. Consider a phase of Lazy-Rounding starting at time t_1. We have that at any round $t \geq t_1$, $\pi_t = \mathrm{Greedy\text{-}Rounding}(\mathrm{P}^{t_1})$, as long as $\mathrm{d_{TV}}(\mathrm{P}^t, \mathrm{P}^{t_1}) \leq 1/n$. By Theorem 9 and Lemma 10, we have that,

$$\mathrm{AccessCost}(\mathrm{Lazy\text{-}Rounding}(t)) = \pi_t(S_t) \leq 2r \cdot \mathop{\mathbb{E}}_{\pi\sim\mathrm{P}^{t_1}}[\pi(S_t)] \leq 4r \mathop{\mathbb{E}}_{\pi\sim\mathrm{P}^{t}}[\pi(S_t)].$$

Overall we get, $\mathrm{AccessCost(Lazy\text{-}Rounding)} = \sum_{t=1}^{m} \pi_t(S_t) \leq 4r\, \mathbb{E}[\mathrm{AccessCost(MWU)}]$. ◀

Bounding the Moving Cost. We now show that the moving cost of Lazy-Rounding is upper bounded by the expected access cost of MWU.

▶ **Lemma 12.** $\mathrm{MovingCost(Lazy\text{-}Rounding)} \leq \mathbb{E}[\mathrm{AccessCost(MWU)}]$.

Proof. Lazy-Rounding moves at the end of a phase incurring a cost of at most n^2. Let t_1 and t_2 be the starting times of two consecutive phases. By the definition of Lazy-Rounding, $d_{\mathrm{TV}}(P^{t_1}, P^{t_2}) > 1/n$. By Lemma 8, we have that the access cost of MWU during t_1 and t_2 is at least n^2. We get that

$$\frac{\mathrm{MovingCost(ALG)}}{\mathbb{E}[\mathrm{AccessCost(MWU)}]} \leq \frac{n^2\#\text{ different phases}}{n^2\#\text{different phases}} = 1.$$ ◀

Theorem 2 follows from lemmas 11, 12 and 6. The details can be found in the full version.

Algorithm 2 Lazy Rounding.

Input: Sequence of requests $(S_1, \ldots, S_m)$ and the initial permutation $\pi_0 \in [n!]$.
Output: A permutation π_t at each round t, which serves request S_t.

```
start-phase ← 1
P^1 ← uniform distribution over permutations
for each round $t \geq 1$ do
    if $\mathrm{d_{tv}}(\mathrm{P}^t, \mathrm{P}^{\text{start-phase}}) \leq 1/n$ then
        $\pi_t \leftarrow \pi_{t-1}$
    else
        $\pi_t \leftarrow$ Greedy-Rounding($\mathrm{P}^t$)
        start-phase ← $t$
    end if
    Serve request $S_t$ using permutation $\pi_t$.
    $w_\pi^{t+1} = w_\pi^t \cdot e^{-\pi(S_t)/n^3}$, for all permutations $\pi \in [n!]$.
    $\mathrm{P}^{t+1} \leftarrow$ Distribution on permutations of MWU, $\mathrm{P}_\pi^{t+1} = w_\pi^t / (\sum_{\pi\in[n!]} w_\pi^t)$.
end for
```

Algorithm 3 Move-All-Equally.

```
Input: A request sequence $(S_1, \dots, S_m)$ and the initial permutation $\pi_0 \in [n!]$
Output: A permutation $\pi_t$ at each round $t$.
for each round $t \geq 1$ do
    $k_t \leftarrow \min\{i \,|\, \pi_{t-1}[i] \in S_t\}$
    Decrease the index of all elements of $S_t$ by $k_t - 1$.
end for
```

Remark. Note that to a large extent, our approach is generic and can be used to provide static optimality for a wide range of online problems. The only requirement is that there is a maximum access cost $\mathrm{C}_{\max}$ and a maximum moving cost D; then, we should use MWU with learning rate $1/(D \cdot \mathrm{C}_{\max})$ and move when $d_{\mathrm{TV}} \geq 1/\,\mathrm{C}_{\max}$. Here we used $D = n^2$ and $\mathrm{C}_{\max} = n$. The only problem-specific part is the rounding of Section 3.2. We believe it is an interesting direction to use this technique for generalizations of this problem, like multiple intents re-ranking or interpret known algorithms for other problems like the BST problem using our approach.

4 A Memoryless Algorithm

In this section we focus on memoryless algorithms. We present an algorithm, called Move-All-Equally (MAE), which seems to be the "right" memoryless algorithm for online MSSC. MAE decreases the index of all elements of the request S_t at the same speed until one of them reaches the first position of the permutation (see Algorithm 3). Note that MAE belongs to the *Move-to-Front* family, i.e., it is a generalization of the classic MTF algorithm for the list update problem. MAE admits two key properties that substantially differentiate it from the other algorithms in the Move-to-Front family presented in Section 2.

(i) Let e_t denote the element used by OPT to cover the request S_t. MAE always moves the element e_t towards the beginning of the permutation.

(ii) It balances moving and access costs: if the access cost at time t is k_t, then the moving cost of MAE is roughly $r \cdot k_t$ (see Algorithm 3). The basic idea is that the moving cost of MAE can be compensated by the decrease in the position of element e_t. This is why it is crucial all the elements to be moved with the same speed.

Lower Bound. First, we show that this algorithm, besides its nice properties, fails to achieve a tight bound for the online MSSC problem.

▶ **Theorem 3.** *The competitive ratio of the Move-All-Equally algorithm is $\Omega(r^2)$.*

In the lower bound instance, the adversary always requests the last r elements of the algorithm's permutation. Since MAE moves all elements to the beginning of the permutation, we end up in a request sequence where n/r disjoint sets are repeatedly requested. Thus the optimal solution incurs a cost of $\Theta(n/r)$ per request, while MAE incurs a cost of $\Omega(n \cdot r)$ per request (the details are in the full version) . Note that in such a sequence, MAE loses a factor of r by moving all elements, instead of one. However, this extra movement seems to be the reason that MAE outperforms all other memoryless algorithms and avoids poor performance in trivial instances, like other MTF-like algorithms.

Upper Bounds. Let $\mathcal{L}$ denote the set of elements used by the optimal permutation on a request sequence such that $|\mathcal{L}| = \ell$. That means, OPT has those ℓ elements in the beginning of its permutation, and it never uses the remaining $n - \ell$ elements. Consider a potential function $\Phi(t)$ being the number of inversions between elements of $\mathcal{L}$ and $U \setminus \mathcal{L}$ in the permutation of MAE (an inversion occurs when an element of $\mathcal{L}$ is behind an element of $U \setminus \mathcal{L}$). Consider the request S_t at time t and let k_t be the access cost of MAE.

Let e_t be the element used by OPT to serve S_t. Clearly, in the permutation of MAE, e_t passes (i.e., changes relative order w.r.t) $k_t - 1$ elements. Among them, let L be the set of elements of $\mathcal{L}$ and R the elements of $U \setminus \mathcal{L}$. Clearly, $|L| + |R| = k_t - 1$ and $|L| \leq |\mathcal{L}| = \ell$. We get that the move of e_t changes the potential by $-|R|$. The moves of all other elements increase the potential by at most $(r-1) \cdot \ell$. We get that

$$k_t + \Phi(t) - \Phi(t-1) \quad \leq \quad |L| + |R| - |R| + (r-1) \cdot \ell \leq |L| + (r-1) \cdot \ell \leq r \cdot \ell.$$

Since the cost of MAE at step t is no more than $(r+1) \cdot k_t$, we get that the amortized cost of MAE per request is $O(r^2 \cdot \ell)$. This implies that for all sequences such that OPT uses all elements of $\mathcal{L}$ with same frequencies (i.e, the OPT pays on average $\Omega(\ell)$ per request), MAE incurs a cost within $O(r^2)$ factor from the optimal. Recall that all other MTF-like algorithms are $\Omega(\sqrt{n})$ competitive even in instances where OPT uses only one element!

While this simple potential gives evidence that MAE is $O(r^2)$-competitive, it is not enough to provide satisfactory competitiveness guarantees. We generalize this approach and define the potential function $\Phi(t) = \sum_{j=1}^{n} \alpha_j \cdot \pi_t(j)$, where $\pi_t(j)$ is the position of element j at round t and α_j are some non-negative coefficients. The potential we described before is the special case where $\alpha_j = 1$ for all elements of $\mathcal{L}$ and $\alpha_j = 0$ for elements of $U \setminus \mathcal{L}$.

By refining further this approach and choosing coefficients α_j according to the frequency that OPT uses element j to serve requests (elements of high frequency are "more important" so they should have higher values α_j), we get an improved upper bound.

▶ **Theorem 14.** *The competitive ratio of* MAE *algorithm is at most* $2^{O(\sqrt{\log n \cdot \log r})}$.

Note that this guarantee is $o(n^\epsilon)$ and $\omega(\log n)$. The proof is based on the ideas sketched above but the analysis is quite involved and is deferred to the full version of this paper.

5 Dynamic Online Min-Sum Set Cover

In this section, we turn our attention to the dynamic version of online MSSC. In online dynamic MSSC, the optimal solution maintains a trajectory of permutations $\pi_0^*, \pi_1^*, \ldots, \pi_t^*, \ldots$ and use permutation π_t^* to serve each request S_t. The cost of the optimal dynamic solution is $\mathrm{OPT}_{\mathrm{dynamic}} = \sum_t (\pi_t^*(S_t) + d_{\mathrm{KT}}(\pi_{t-1}^*, \pi_t^*))$, where $\{\pi_t^*\}_t$ denotes the optimal permutation trajectory for the request sequence that minimizes the total access and moving cost.

We remark that the ratio between the optimal static solution and the optimal dynamic solution can be as high as $\Omega(n)$. For example, in the sequence of requests $\{1\}^b\{2\}^b \ldots \{n\}^b$, the optimal static solution pays $\Theta(n^2 b)$, whereas the optimal dynamic solution pays $\Theta(n^2 + n \cdot b)$ by moving the element that covers the next $n \cdot b$ requests to the first position and then incurring access cost 1. The above example also reveals that although Algorithm 2 is $\Theta(r)$-competitive against the optimal static solution, its worst-case ratio against a dynamic solution can be $\Omega(n)$.

MAE Algorithm. As a first study of the dynamic problem, we investigate the competitive ratio of *Move-All-Equally* (MAE) algorithm from Section 4. We begin with an upper bound.

▶ **Theorem 4.** *The competitive ratio of the Move-All-Equally algorithm for the dynamic online Min-Sum Set Cover problem is* $O(r^{3/2}\sqrt{n})$.

The approach for proving Theorem 4 is generalizing that exhibited in Section 4 for the static case. We use a generalized potential function $\Phi(t) = \sum_{j=1}^{n} \alpha_j^t \cdot \pi_t(j)$; i.e, the multipliers α_j may change over time so as to capture the moves of $\text{OPT}_{\text{dynamic}}$. To select coefficients α_j^t we apply a two-level approach. We observe that there is always a 2-approximate optimal solution that moves an element of S_t to the front (similar to classic MTF in list update). We call this MTF_{OPT} . We compare the permutation of the online algorithm with the permutation maintained by this algorithm; at each time, elements the beginning of the offline permutation are considered to be "important" and have higher coefficients α_j^t. The formal proof is in the full version.

Next, we show an almost matching lower bound.

▶ **Theorem 5.** *For any $r \geq 3$, the competitive ratio of the Move-All-Equally algorithm for the dynamic online Min-Sum Set Cover problem is $\Omega(r\sqrt{n})$.*

Sketch of the Construction. The lower bound is based on a complicated adversarial request sequence; we sketch the main ideas. Let k be an integer. During a *phase* we ensure that:

(i) There are $2k$ "important" elements used by OPT; we call them $e_1, \dots, e_{2k}$. In the beginning of the phase, those elements are ordered in the start of the optimal permutation π^*, i.e., $\pi^*[e_j] = j$. The phase contains k consecutive requests to each of them, in order; thus the total number of requests is $\approx 2k^2$. OPT brings each element e_j at the front and uses it for k consecutive requests; thus the access cost of OPT is $2k^2$ (1 per request) and the total movement cost of OPT of order $\Theta(k^2)$. Over a phase of $2k^2$ requests, OPT incurs an overall cost $\Theta(k^2)$, i.e., an average of $O(1)$ per request.

(ii) The first $k + r - 2$ positions of the online permutation will be always occupied by the same set of "not important" elements; at each step the $r - 2$ last of them will be part of the request set and MAE will move them to the front. Thus the access cost will always be $k + 1$ and the total cost more than $(r + 1) \cdot k$.

The two properties above are enough to provide a lower bound $\Omega(r \cdot k)$; the optimal cost is $O(1)$ per request and the online cost $\Omega(r \cdot k)$. The goal of an adversary is to construct a request sequence with those two properties for the largest value of k possible.

The surprising part is that although MAE moves all requested elements towards the beginning of the permutation, it never manages to bring any of the "important" elements in a position smaller than $r + k - 2$. While the full instance is complex and described in the full version, at a high-level, we make sure that whenever a subsequence of k consecutive requests including element e_j begins, e_j is at the end of the online permutation, i.e., $\pi_t[e_j] = n$. Thus, even after k consecutive requests where MAE moves it forward by distance k, it moves by k^2 positions; by making sure that $n - k^2 > r + k - 2$ (which is true for some $k = \Omega(\sqrt{n})$), we can make sure that e_j does not reach the first $r + k - 2$ positions of the online permutation.

6 Concluding Remarks

Our work leaves several intriguing open questions. For the (static version of) Online MSSC, it would be interesting to determine the precise competitive ratio of the MAE algorithm; particularly whether it depends only on r or some dependency on n is really necessary. More generally, it would be interesting to determine the best possible performance of memoryless algorithms and investigate trade-offs between competitiveness and computational efficiency.

For the online dynamic MSSC problem, the obvious question is whether a $f(r)$-competitive algorithm is possible. Here, we showed that techniques developed for the list update problem seem to be too problem-specific and are not helpful in this direction. This calls for the use of more powerful and systematic approaches. For example, the online primal-dual method [18] has been applied successfully for solving various fundamental problems [6,7,17,19]. Unfortunately, we are not aware of a primal-dual algorithm even for the special case of list update; the only attempt we are aware of is in [45], but this analysis basically recovers known (problem-specific) algorithms using dual-fitting. Our work gives further motivation for designing a primal-dual algorithm for list-update: this could be a starting point towards solving the online dynamic MSSC.

In a broader context, the online MSSC is the first among a family of poorly understood online problems such as the multiple intents re-ranking problem described in Section 1.2. In this problem, when a set S_t is requested, we need to cover it using $s \leq r$ elements; MSSC is the special case $s = 1$. It is natural to expect that the lower bound of Theorem 1 can be generalized to $\Omega(r/s)$, i.e., as s grows, we should be able to achieve a better competitive ratio. It will be interesting to investigate this and the applicability of our technique to obtain tight bounds for this problem.

References

1 Susanne Albers. Improved randomized on-line algorithms for the list update problem. *SIAM J. Comput.*, 27(3):682–693, 1998. doi:10.1137/S0097539794277858.

2 C. J. Argue, Anupam Gupta, Guru Guruganesh, and Ziye Tang. Chasing convex bodies with linear competitive ratio. In *SODA*, pages 1519–1524, 2020. doi:10.1137/1.9781611975994.93.

3 Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of Computing*, 8(1):121–164, 2012. doi:10.4086/toc.2012.v008a006.

4 Yossi Azar and Iftah Gamzu. Ranking with submodular valuations. In *SODA*, pages 1070–1079, 2011. doi:10.1137/1.9781611973082.81.

5 Yossi Azar, Iftah Gamzu, and Xiaoxin Yin. Multiple intents re-ranking. In *STOC*, pages 669–678, 2009. doi:10.1145/1536414.1536505.

6 Nikhil Bansal, Niv Buchbinder, Aleksander Madry, and Joseph Naor. A polylogarithmic-competitive algorithm for the *k*-server problem. *J. ACM*, 62(5):40:1–40:49, 2015. doi:10.1145/2783434.

7 Nikhil Bansal, Niv Buchbinder, and Joseph Naor. A primal-dual randomized algorithm for weighted paging. *J. ACM*, 59(4):19:1–19:24, 2012. doi:10.1145/2339123.2339126.

8 Nikhil Bansal, Marek Eliáš, and Grigorios Koumoutsos. Weighted k-server bounds via combinatorial dichotomies. In *FOCS*, pages 493–504, 2017. doi:10.1109/FOCS.2017.52.

9 Nikhil Bansal, Marek Eliáš, Grigorios Koumoutsos, and Jesper Nederlof. Competitive algorithms for generalized *k*-server in uniform metrics. In *SODA*, pages 992–1001, 2018. doi:10.1137/1.9781611975031.64.

10 Nikhil Bansal, Anupam Gupta, and Ravishankar Krishnaswamy. A constant factor approximation algorithm for generalized min-sum set cover. In *SODA*, pages 1539–1545, 2010. doi:10.1137/1.9781611973075.125.

11 Amotz Bar-Noy, Mihir Bellare, Magnús M. Halldórsson, Hadas Shachnai, and Tami Tamir. On chromatic sums and distributed resource allocation. *Inf. Comput.*, 140(2):183–202, 1998. doi:10.1006/inco.1997.2677.

12 Avrim Blum and Carl Burch. On-line learning and the metrical task system problem. *Machine Learning*, 39(1):35–58, 2000.

13 Avrim Blum, Shuchi Chawla, and Adam Kalai. Static optimality and dynamic search-optimality in lists and trees. *Algorithmica*, 36(3):249–260, 2003.

14 Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis.* Cambridge University Press, 1998.

15 Allan Borodin, Nathan Linial, and Michael E. Saks. An optimal on-line algorithm for metrical task system. *J. ACM*, 39(4):745–763, 1992. doi:10.1145/146585.146588.

16 Sébastien Bubeck, Michael B. Cohen, James R. Lee, and Yin Tat Lee. Metrical task systems on trees via mirror descent and unfair gluing. In *SODA*, pages 89–97. SIAM, 2019.

17 Niv Buchbinder and Joseph Naor. Improved bounds for online routing and packing via a primal-dual approach. In *FOCS*, pages 293–304, 2006. doi:10.1109/FOCS.2006.39.

18 Niv Buchbinder and Joseph Naor. The design of competitive online algorithms via a primal-dual approach. *Foundations and Trends in Theoretical Computer Science*, 3(2-3):93–263, 2009. doi:10.1561/0400000024.

19 Niv Buchbinder and Joseph Naor. Fair online load balancing. *J. Scheduling*, 16(1):117–127, 2013. doi:10.1007/s10951-011-0226-0.

20 Marek Chrobak, Howard J. Karloff, T. H. Payne, and Sundar Vishwanathan. New results on server problems. *SIAM J. Discrete Math.*, 4(2):172–181, 1991. doi:10.1137/0404017.

21 Marek Chrobak and Lawrence L. Larmore. An optimal on-line algorithm for k-servers on trees. *SIAM J. Comput.*, 20(1):144–148, 1991. doi:10.1137/0220008.

22 Christian Coester and James R. Lee. Pure entropic regularization for metrical task systems. In *COLT*, volume 99 of *Proceedings of Machine Learning Research*, pages 835–848. PMLR, 2019.

23 Cynthia Dwork, Ravi Kumar, Moni Naor, and D. Sivakumar. Rank aggregation methods for the Web. In *Proceedings of the Tenth International World Wide Web Conference, WWW 10*, pages 613–622. ACM, 2001. doi:10.1145/371920.372165.

24 Ran El-Yaniv. *There are infinitely many competitive-optimal online list accessing algorithms*. Hebrew University of Jerusalem, 1996.

25 Uriel Feige, László Lovász, and Prasad Tetali. Approximating min sum set cover. *Algorithmica*, 40(4):219–234, 2004. doi:10.1007/s00453-004-1110-5.

26 Dimitris Fotakis. On the competitive ratio for online facility location. *Algorithmica*, 50(1):1–57, 2008. doi:10.1007/s00453-007-9049-y.

27 Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.*, 55(1):119–139, 1997. doi:10.1006/jcss.1997.1504.

28 Anupam Gupta, Guru Guruganesh, Binghui Peng, and David Wajc. Stochastic online metric matching. In *ICALP*, pages 67:1–67:14, 2019. doi:10.4230/LIPIcs.ICALP.2019.67.

29 Refael Hassin and Asaf Levin. An approximation algorithm for the minimum latency set cover problem. In *ESA*, pages 726–733, 2005. doi:10.1007/11561071_64.

30 John Iacono and Wolfgang Mulzer. A static optimality transformation with applications to planar point location. *Int. J. Comput. Geometry Appl.*, 22(4):327–340, 2012. doi:10.1142/S0218195912600084.

31 Sungjin Im, Viswanath Nagarajan, and Ruben van der Zwaan. Minimum latency submodular cover. *ACM Trans. Algorithms*, 13(1):13:1–13:28, 2016. doi:10.1145/2987751.

32 Sungjin Im, Maxim Sviridenko, and Ruben van der Zwaan. Preemptive and non-preemptive generalized min sum set cover. *Math. Program.*, 145(1-2):377–401, 2014. doi:10.1007/s10107-013-0651-2.

33 Elias Koutsoupias and Akash Nanavati. The online matching problem on a line. In *WAOA*, pages 179–191, 2003. doi:10.1007/978-3-540-24592-6_14.

34 Elias Koutsoupias and Christos H. Papadimitriou. On the k-server conjecture. *Journal of the ACM*, 42(5):971–983, 1995.

35 Nick Littlestone and Manfred K. Warmuth. The weighted majority algorithm. *Inf. Comput.*, 108(2):212–261, 1994.

36 Mark S. Manasse, Lyle A. McGeoch, and Daniel Dominic Sleator. Competitive algorithms for server problems. *J. Algorithms*, 11(2):208–230, 1990. doi:10.1016/0196-6774(90)90003-W.

37 Adam Meyerson. Online facility location. In *FOCS*, pages 426–431. IEEE Computer Society, 2001.

38 Joseph Naor, Debmalya Panigrahi, and Mohit Singh. Online node-weighted steiner tree and related problems. In *FOCS*, pages 210–219, 2011. doi:10.1109/FOCS.2011.65.

39 Krati Nayyar and Sharath Raghvendra. An input sensitive online algorithm for the metric bipartite matching problem. In *FOCS*, pages 505–515, 2017. doi:10.1109/FOCS.2017.53.

40 Mark Sellke. Chasing convex bodies optimally. In *SODA*, pages 1509–1518, 2020. doi:10.1137/1.9781611975994.92.

41 René Sitters. The generalized work function algorithm is competitive for the generalized 2-server problem. *SIAM J. Comput.*, 43(1):96–125, 2014. doi:10.1137/120885309.

42 Martin Skutella and David P. Williamson. A note on the generalized min-sum set cover problem. *Oper. Res. Lett.*, 39(6):433–436, 2011. doi:10.1016/j.orl.2011.08.002.

43 Daniel Dominic Sleator and Robert Endre Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28(2):202–208, 1985. doi:10.1145/2786.2793.

44 Daniel Dominic Sleator and Robert Endre Tarjan. Self-adjusting binary search trees. *J. ACM*, 32(3):652–686, 1985. doi:10.1145/3828.3835.

45 Erez Timnat. The list update problem, 2016. Master Thesis, Technion- Israel Institute of Technology.

Efficient Diagonalization of Symmetric Matrices Associated with Graphs of Small Treewidth

Martin Fürer
Pennsylvania State University, University Park, PA, USA
furer@cse.psu.edu

Carlos Hoppen
Universidade Federal do Rio Grande do Sul, Porto Alegre, Brazil
choppen@ufrgs.br

Vilmar Trevisan
Universidade Federal do Rio Grande do Sul, Porto Alegre, Brazil
trevisan@mat.ufrgs.br

Abstract

Let $M = (m_{ij})$ be a symmetric matrix of order n and let G be the graph with vertex set $\{1, \ldots, n\}$ such that distinct vertices i and j are adjacent if and only if $m_{ij} \neq 0$. We introduce a dynamic programming algorithm that finds a diagonal matrix that is congruent to M. If G is given with a tree decomposition $\mathcal{T}$ of width k, then this can be done in time $O(k|\mathcal{T}| + k^2 n)$, where $|\mathcal{T}|$ denotes the number of nodes in $\mathcal{T}$.

2012 ACM Subject Classification Computing methodologies → Linear algebra algorithms; Theory of computation → Fixed parameter tractability; Mathematics of computing → Graph theory

Keywords and phrases Treewidth, Diagonalization, Eigenvalues

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.52

Category Track A: Algorithms, Complexity and Games

Funding *Carlos Hoppen*: CNPq 308054/2018-0 and FAPERGS 19/2551-0001727-8.
Vilmar Trevisan: CNPq 409746/2016-9 and 303334/2016-9, CAPES-PRINT 88887.467572/2019-00, and FAPERGS 17/2551-0001.

1 Introduction and main result

Two matrices M and N are said to be *congruent*, which is denoted $M \cong N$, if there exists a nonsingular matrix P for which $N = P^T MP$. Matrix congruence naturally appears when studying Gram matrices associated with a quadratic form on a finite-dimensional vector space; finding a diagonal matrix D that is congruent to a symmetric matrix M allows us to classify the quadratic form.

In a different direction, finding a diagonal matrix that is congruent to M allows us to determine the number of eigenvalues of M in a given real interval. Indeed, fix real numbers $c < d$. Let $D_c \cong N = M - cI$ and $D_d \cong M - dI$ be diagonal matrices. By Sylvester's Law of Inertia [16, p. 568], the number n_1 of eigenvalues of M greater than c equals the number positive entries in D_c. Moreover, the number of eigenvalues equal to c, or less than c, are given by the number of zero diagonal entries, or by the number of negative entries in D_c, respectively. As a consequence, if n_2 is the number of positive entries in D_d, then $n_1 - n_2$ is the number of eigenvalues of M in the interval $(c, d]$.

Given a symmetric matrix $M = (m_{ij})$ of order n, we may associate it with a graph G with vertex set $[n] = \{1, \ldots, n\}$ such that distinct vertices i and j are adjacent if and only if $m_{ij} \neq 0$. We say that G is the *underlying graph* of M. This allows us to employ structural decompositions of graph theory to deal with the nonzero entries of M in an efficient way.

One such decomposition is the tree decomposition, which has been extensively studied since the seminal paper of Robertson and Seymour [18]. The graph parameter associated with this decomposition is the *treewidth*.

A tree decomposition of a graph $G = (V, E)$ is a tree $\mathcal{T}$ with *nodes* $1, \dots, m$, where each node i is associated with a *bag* $B_i \subseteq V$, satisfying the following properties: (1) $\bigcup_{i=1}^{m} B_i = V$; (2) For every edge $\{v, w\} \in E$, there exists B_i containing v and w; (3) For any $v \in V$, the subgraph of $\mathcal{T}$ induced by the nodes that contain v is connected.

The *width* of the tree decomposition $\mathcal{T}$ is defined as $\max_i (|B_i| - 1)$ and the *treewidth* $tw(G)$ of graph G is the smallest k such that G has a tree decomposition of width k. Clearly, it is always the case that $tw(G) < |V|$, and for connected graphs, $tw(G) = 1$ if and only if G is a tree on two or more vertices. Even though computing the treewidth of a graph is hard, this is a widely studied parameter and tree decompositions with bounded width are known for several graph classes. Moreover, there is an extensive literature on approximation algorithms for this problem. For instance, Fomin et al. [9] devised an algorithm such that, given an n-vertex graph and an integer k, runs in time $O(k^7 n \log n)$ and either correctly reports that the treewidth of G is larger than k, or constructs a tree decomposition of G of width $O(k^2)$.

Often, graph widths have been used to design algorithms for NP-complete or even harder problems that are efficient on graphs of bounded width, by which we mean that their running time is equal to $O(f(k)n^c)$ for a constant c and an arbitrary computable function f, where k is the width of the graph. Problems that can be solved with such a running time are called fixed parameter tractable (FPT). For more information, interested readers are referred to [5, 7, 8, 17], and to the references therein.

On the other hand, graph widths may be quite useful for polynomial time solvable problems. Indeed, treewidth has a strong connection to the solution of sparse linear systems and Gaussian elimination. For a long time heuristics have been designed to maintain sparsity throughout Gaussian elimination. The goal is to minimize the *fill-in*, defined as the set of matrix positions that were initially 0, but have received nonzero values during the computation. Fill-in minimization has been analyzed as a graph problem in the case of symmetric [19] as well as asymmetric [20] matrices. As in our paper, a nonzero matrix entry a_{ij} is interpreted as an edge (or arc) from vertex i to vertex j. However, unlike here, it was assumed that the matrix was diagonally dominant, so that all the pivots could be chosen in the diagonal.

A major source of applications are symmetric positive definite matrices due to their prevalence. When the ith diagonal pivot is chosen in the symmetric case, then the fill-in consists of all missing edges between those neighbors of vertex i that have not yet been eliminated. The minimum k over all elimination orders of the maximal number of higher numbered neighbors is precisely the treewidth of the graph associated with a symmetric matrix [2]. When such an optimal elimination order or, equivalently, a tree decomposition of width k is given, then Gaussian elimination can trivially be done in time $O(k^2 n)$, if the pivot can always be chosen in the diagonal. This is clearly the case for symmetric positive definite matrices. On the other hand, it long seemed impossible to get an efficient algorithm when off-diagonal pivots have to be chosen. Note that, if the graph corresponding to a symmetric matrix has treewidth k, then you can always find a diagonal element with at most k off-diagonal nonzero elements in its row and column. But if this diagonal element is zero, and the pivot is chosen somewhere else in its row, then there can be an arbitrary number of nonzero elements in the pivot's column. The main contribution of this paper is an algorithm that deals with these diagonal elements.

The paper of Bodlaender et al. [2] is dedicated to the approximation of treewidth, pathwidth, and minimum elimination tree height. (Pathwidth is defined by tree decompositions where the tree is required to be a path.) In this context, they discuss the important matrix tasks of Cholesky factorization ($A = LL^T$) and Gaussian elimination for sparse matrices. These tasks have efficient FPT algorithms parameterized by treewidth or pathwidth, while their parallel complexity is determined by the minimum elimination tree height.

A recent paper of Fomin et al. [9] explores topics very similar to ours. For a matrix[1] with a given tree decomposition of width k, these authors present various efficient algorithms. In time $O(k^3 n)$ they solve systems of linear equations, compute the determinant and the rank of a matrix. These problems are also solved by our approach, in the case of symmetric matrices. They also solve various matching and flow problems and compute an $O(k^2)$ approximation to treewidth in $k^{O(1)} n \log^{O(1)} n$ time.

Here, we design a dynamic programming algorithm with running time $O(k^2 n)$ to find a diagonal matrix that is congruent to an input symmetric matrix M of order n, provided that a tree decomposition of width k for the underlying graph G is part of the input. In particular, for bounded treewidth, we find a linear-time solution to this problem. To achieve our goal, we were inspired by an algorithm with the same purpose for graphs with small clique-width (*clique-width* is defined based on another structural decomposition introduced by Courcelle and Olariu [4]). This algorithm was proposed by Jacobs and the current authors [12].

Astonishingly, the problem with the treewidth parameter turned out to be more challenging. This is somewhat counterintuitive and unusual, because graphs of bounded treewidth are also of bounded clique-width. Clearly, there is no direct implication. There are graphs whose clique-width is exponentially bigger than their treewidth [3]. However the implication of a running time of $2^{O(k)} n$ for a polynomially solvable problem is rather unimpressive and useless. The conclusion in the current paper is stronger. It is also much more useful, because practical examples of sparse matrices often have small treewidth, while a pattern of large minors with the same entry, typical of bounded clique-width matrices, is rather unusual.

As [9] and [12], the current paper fits into the recent trend "FPT within P", which investigates fundamental problems that are solvable in polynomial time, but for which a lower exponent may be achieved using an FPT algorithm that is also polynomial in terms of the parameter k (see [14]). For our problem, it does not seem that $O(poly(k)n)$ algorithms are possible for graphs of fusion-width or multi-clique-width k [10, 11]. For some problems, like the independent set problem, there are algorithms whose running time as a function of these parameters is not worse than as a function of the clique-width, even though the clique-width can be exponentially larger. An $O(k^2 n)$ algorithm for multi-clique-width k would imply the same time bound when k is the clique-width or the tree-width.

▶ **Theorem 1.** *Given a symmetric matrix M of order n and a tree decomposition $\mathcal{T}$ of width k for the underlying graph of M, algorithm* `CongruentDiagonal` *(see Sect. 3) produces a diagonal matrix D congruent to M in time $O(k|\mathcal{T}| + k^2 n)$.*

Naturally, we assume throughout this paper, that the input matrix M is given in a compact form because, in standard form, just reading M would already take quadratic time. A possible representation could be a list of triples (i, j, m_{ij}) containing all nonzero entries. For convenience, we assume the value m_{uv} for $u \neq v$ is just attached to the edge uv in the given tree decomposition. Likewise the value m_{uu} is attached to the vertex u. This representation could be obtained efficiently from the list representation. We will not discuss

[1] The matrices in [9] are not necessarily symmetric (in fact, not necessarily square), and they associate an $m \times n$-matrix with a bipartite graph whose partition classes have size m and n.

such a transformation even though it is not trivial. An interesting discussion by Bodlaender et al. [1] shows how a data structure of size $O(kn)$ enables an $O(k)$ test whether two given vertices are adjacent, a detail that had previously often been overlooked.

Our paper is organized as follows. In Section 2, we define the concept of an edge-explicit tree decomposition and state auxiliary results about it. We then describe our algorithm in Section 3 and justify why it works as claimed, ending with an example.

2 Edge-explicit tree decomposition

In this paper, we consider graph decompositions based on the concept of *nice tree decomposition*, introduced by Kloks [15]. The current variation is due to Fürer and Yu [13], and to distinguish it from the original version, we call it an *edge-explicit tree decomposition.* (Nice tree decompositions with explicit nodes to introduce edges have been considered by Cygan et al. [6] before.) A rooted tree decomposition $\mathcal{T}$ of a graph G whose nodes are associated with bags $B_1, \ldots, B_m$ is *edge-explicit* if each node i is of one of the following types:

(a) **(Leaf)** The node i is a leaf of $\mathcal{T}$;

(b) **(Introduce Vertex)** The node i introduces vertex v if it has a single child j, $v \notin B_j$ and $B_i = B_j \cup \{v\}$.

(c) **(Introduce Edges)** The node i introduces edges if it has a single child j and $|B_i| = |B_j|$. This node is labelled by a vertex $v \in B_i$ and inserts all edges $\{u, v\}$ of $E(G)$ such that $u \in B_i$.

(d) **(Forget)** The node i forgets vertex v if i has a single child j, $v \notin B_i$ and $B_j = B_i \cup \{v\}$;

(e) **(Join)** The node i is a join if it has two children j and ℓ, where $B_i = B_j = B_\ell$.

Unlike the other operations, the vertex v whose adjacencies are introduced by an Introduce Edges node must be given as part of the operation. We assume that every edge $uv \in E$ is introduced exactly once. In fact, for every edge uv, we will further suppose that, if j is the node that introduces the edge uv and it is labelled by vertex v, then the parent of j forgets v. Another assumption that will simplify our discussion is that $B_r = \emptyset$ for the root r of the tree, so that every vertex will be forgotten.

For constant k, Kloks [15] shows how to construct a nice tree decomposition of size at most $4n$ from a k-tree in time $O(n)$. We want to construct an edge-explicit tree decomposition from an arbitrary tree decomposition efficiently. With the help of the given tree decomposition of G of width k, we could embed G into a k-tree, apply the algorithm of Kloks, and finally add the Introduce Edges nodes. For our application, we want to analyze the dependence of the time on k precisely. For this purpose, we do a direct construction avoiding the k-trees.

▶ **Lemma 2.** *From a tree decomposition of width k and m nodes, an edge-explicit tree decomposition of the same width k with less than $5n$ nodes can be computed in time $O(k(m+n))$.*

Proof. We assume that all bags are given by a sorted list of their vertices. If these lists were unsorted, we could trivially sort them all in time $O((k \log k)m)$, or in a more sophisticated manner in time $O(km)$. As an additional preprocessing step, we produce a new node with empty bag and declare it to be the root. It is connected to an arbitrary node of the original tree decomposition.

Now we modify the tree decomposition in a sequence of depth-first tree traversals. Some of these traversals could be combined, but obviously without improving the asymptotic running time. During the first traversal, every node whose bag is contained in the bag of its parent is merged with the parent. Initially, the number of nodes m is not bounded by any function of the number of vertices n, because many subtrees could represent the same subgraph. From our argument below, it follows that the tree has a size less than $4n$ after this step.

In four more depth-first traversals, we produce an edge-explicit tree decomposition. In the second traversal, whenever the bag of a node has size less than the size of the bag of its parent, we add some nodes from the parent until both bags have the same size. This is a crucial step. Avoiding it, could increase the number of nodes in the fourth depth-first traversal to $\Omega(kn)$.

In the third traversal, all nodes with more than one child are replaced by binary trees with identical bags such that the original children appear as children of their leaves. In the fourth traversal, for any node i with a single child j, if necessary, some new nodes are inserted such that the bags only change by one vertex in each step. This is done from j to i by a sequence of nodes alternating between Forget nodes and Introduce Vertex nodes possibly followed by more Forget nodes. Now we have a nice tree decomposition, which we will show to have at most $4n$ nodes. Finally, in the fifth depth-first traversal, immediately below every Forget node of a vertex v, we insert an Introduce Edges node introducing the edges to v, to make the nice tree decomposition edge-explicit.

Now we count the nodes. Note that the root and the Leaf nodes are incident with a single edge, the Join nodes are incident with three edges and the other nodes are incident with two edges. As a consequence, there is one less Join node than Leaf node. Every vertex can be forgotten only once. Thus the number of Forget nodes is at most n. Between every Leaf node and the first Join node above it in the tree, there is at least one Forget node, otherwise, the the leaf would have been merged with its parent in the first traversal. Thus there are at most n Leaf nodes and at most $n-1$ Join nodes. The single child of every Introduce Vertex node is a Forget node. The same is true for the parent of every Introduce Edges node. Therefore, the number of Introduce Vertex nodes and and the number of Introduce Edges nodes are at most n each. ◀

▶ Remark 3. The somewhat wasteful second traversal could be avoided. Its effect is to push Introduce Vertex nodes down the tree in order to avoid some of them when the corresponding vertices are introduced in leaves. This guarantees a bound of $O(n)$ rather than $O(kn)$ on the number of Introduce Vertex nodes.

Without changing the asymptotic running time nor the treewidth, a tree decomposition with typically many smaller bags could be obtained by allowing Introduce Vertex nodes introducing many vertices at once. For our application, this would work, because handling a node introducing many vertices could still be done in time $O(k^2)$.

For later use, we state an auxiliary result that records facts about an edge-explicit tree decomposition. We do not include a proof, as it follows directly from the definition of this concept. Let $G = (V, E)$ be a graph with tree decomposition $\mathcal{T}$, whose bags are $B_1, \ldots, B_m$. For $v \in V$ and a node j of $\mathcal{T}$, let $\mathcal{T}(v)$ and $\mathcal{T}_j$ be the subtree of $\mathcal{T}$ induced by the nodes containing v and the branch of $\mathcal{T}$ rooted at j, respectively.

▶ **Lemma 4.** *In an edge-explicit tree decomposition $\mathcal{T}$ of a graph $G = (V, E)$, the following statements hold.*

(a) *Every $v \in V$ is forgotten exactly once in $\mathcal{T}$.*

(b) *Let $uv \in E$ and let i be the node that introduces the edge uv. If ℓ is an ancestor of i and ℓ is a join or a node that introduces a vertex, then $\{u, v\} \not\subseteq B_\ell$.*

(c) *For every $v \in V$, the subtree $\mathcal{T}(v)$ of $\mathcal{T}$ is rooted at the child of the node that forgets v. Moreover, the leaves of $\mathcal{T}(v)$ are precisely the leaves of $\mathcal{T}$ that contain v and the nodes of $\mathcal{T}$ that introduce v.*

(d) *Suppose i forgets vertex v and j is its child. If $w \notin B_i$ and $\mathcal{T}(v) \cap \mathcal{T}(w) \neq \emptyset$, then $\mathcal{T}(w)$ is a subtree of $\mathcal{T}_j$. In particular, $\mathcal{T}(v)$ is a subtree of $\mathcal{T}_j$.*

3 The Algorithm

We now describe our diagonalization algorithm, which we call `CongruentDiagonal`. Let M be a symmetric matrix of order n and let $G = (V, E)$ be the underlying graph with vertex set $V = [n]$ associated with it. We wish to find a diagonal matrix congruent to M. Let $\mathcal{T}$ be an edge-explicit tree decomposition of G of width k. The algorithm works bottom-up in the rooted tree $\mathcal{T}$, so we order the nodes $1, \ldots, m$ of $\mathcal{T}$ in post-order and operate on a node i after its children have been processed. It is well-known that two matrices are congruent if we can obtain one matrix from the other by a sequence of *pairs* of elementary operations, each pair consisting of a row operation followed by the *same* column operation. In our algorithm we only use congruence operations that permute rows and columns or add a multiple of a row and column to another row and column respectively. To achieve linear-time we must operate on a sparse representation of the graph associated with M, rather than on the matrix itself.

We start with a high-level description of the algorithm, which is summarized below. Each node i in the tree produces a pair of matrices $(N_i^{(1)}, N_i^{(2)})$, which may be combined into a symmetric matrix N_i of order at most $2(k+1)$. The algorithm traverses the tree decomposition from the leaves to the root so that, at node i, the algorithm either initializes a pair $(N_i^{(1)}, N_i^{(2)})$, or it produces $(N_i^{(1)}, N_i^{(2)})$ based on the matrices produced by its children, transmitting the pair to its parent. During this step, the algorithm may also produce diagonal elements of a matrix congruent to M. These diagonal elements are not transmitted by a node to its parent, but are appended to a global array as they are produced. At the end of the algorithm, the array consists of the diagonal elements of a diagonal matrix D that is congruent to M.

Algorithm 1 High level description of the algorithm `CongruentDiagonal`.

```
CongruentDiagonal(M)
input:  an edge-explicit tree decomposition $\mathcal{T}$ of the underlying
graph $G$ associated with $M$ of width $k$ and the nonzero entries of $M$
output:  diagonal entries in $D \cong M$
Order the nodes of $\mathcal{T}$ as $1, 2, \ldots, m$ in post order
for $i$ from  1 to $m$ do
    if is-Leaf($i$) then construct $(N_i^{(1)}, N_i^{(2)})$=LeafBox($B_i$)
    if is-IntroduceVertex($i$) then construct $(N_i^{(1)}, N_i^{(2)})$=IntroVertexBox($B_i$)
    if is-IntroduceEdge($i$) then construct $(N_i^{(1)}, N_i^{(2)})$=IntroEdgesBox($B_i$)
    if is-Join($i$) then construct $(N_i^{(1)}, N_i^{(2)})$=JoinBox($B_i$)
    if is-Forget($i$) then construct $(N_i^{(1)}, N_i^{(2)})$=ForgetBox($B_i$)
```

In the remainder, we shall describe each operation in detail and justify that the algorithm `CongruentDiagonal` yields the desired output. Step i of the algorithm refers to the ith iteration of the loop above, and we assume that Step i processes the node i. To describe the matrix produced by each node, we need the concept of a matrix $M = (m_{ij})$ in *row echelon form*. This means that $m_{ij} = 0$ for all $j < i$. Moreover, let the *pivot* of row i be the first j such that $m_{ij} \neq 0$, if such an element exists. We require that distinct rows have different pivots.

Each matrix N_i produced by a node on the tree has the form

$$N_i = \begin{array}{|c|c|} \hline N_i^{(0)} & N_i^{(1)} \\ \hline N_i^{(1)T} & N_i^{(2)} \\ \hline \end{array}, \qquad (1)$$

where $N_i^{(0)}$ is a matrix of dimension $k_i' \times k_i'$ whose entries are zero, $N_i^{(2)}$ is a symmetric matrix of dimension $k_i'' \times k_i''$ and $N_i^{(1)}$ is a $k_i' \times k_i''$ matrix in row echelon form. Moreover, $0 \leq k_i' \leq k_i'' \leq k+1$. Observe that k_1' can be zero, in which case we regard $N_i^{(0)}$ and $N_i^{(1)}$ as empty. An important fact about N_i is that each of its rows (and the corresponding column) is associated with a vertex of G (equivalently, a row of M). Let $V(N_i)$ denote the set of vertices of G associated with the rows of N_i. We say that the k_i' rows in $N_i^{(0)}$ have *type-i* and the k_i'' rows of $N_i^{(2)}$ have *type-ii*. This is represented by the partition $V(N_i) = V_1(N_i) \cup V_2(N_i)$, where $V_1(N_i)$ and $V_2(N_i)$ are the vertices of type-i and type-ii, respectively. As it turns out, the vertices of type-ii are precisely the vertices in B_i. When proving facts about the algorithm, we shall often refer to the matrix N_i as the result of processing B_i, even if the actual output is the pair $(N_i^{(1)}, N_i^{(2)})$.

The structure of the matrix N_i is described by the following lemma.

▶ **Lemma 5.** *For all $i \in [m]$, the matrix N_i defined in terms of the pair $(N_i^{(1)}, N_i^{(2)})$ produced by node i satisfies the following properties:*
(a) $0 \leq k_i' \leq k_i'' \leq k+1$.
(b) *$N_i^{(1)}$ is a matrix in row echelon form.*
(c) $V_2(N_i) = B_i$.

To give an intuition about how the algorithm works, consider that we are trying to apply the strategy for Gaussian elimination described in the introduction. Vertices of type-ii would represent the rows that have never been used to eliminate elements of other rows, while vertices of type-i would be the nonzero rows that have already been used to eliminate elements in other rows, but for which the basic strategy of using the diagonal element as a pivot failed because it was equal to 0. The algorithm keeps these rows in a temporary buffer, which is maintained in row echelon form to make sure that its size k' satisfies $k' \leq k+1$. In the process of maintaining row echelon form, some of these rows become diagonalised. In our algorithm, to preserve congruence, we perform the same Gaussian operations on rows and columns. Any row v of the input matrix M begins as a type-ii row. It can either be diagonalized during the application of ForgetBox to the node that forgets v, or it becomes a type-i row at this step, and finally becomes diagonalized in a later application of JoinBox or ForgetBox. Finally, we discuss the content of the boxes. Let $\tilde{M}(i)$ be the matrix that would be obtained by performing all row and column operations performed by the algorithm up to step i to the original matrix M. It turns out that, for a type-i row u in N_i and any row v in the matrix, the entries uv and vu in $\tilde{M}(i)$ and N_i coincide, if $v \in V(N_i)$, and the entries uv and vu in $\tilde{M}(i)$ are equal to 0, if $v \notin V(N_i)$. This is consistent with the intuition that rows of type-i have already been partially diagonalized and that their diagonal elements are 0. However, this connection does not hold in general for the entries of $\tilde{M}(i)$ and $N_i^{(2)}$, as $N_i^{(2)}$ can only capture changes the operations made for nodes in its branch of the tree decomposition, but vertices of type-ii could simultaneously lie in many different branches. This needs to be dealt with when looking at the effect of `JoinBox`.

To record the diagonal entries produced by the algorithm, let D_i be the set of all pairs (v, d_v), where v is a vertex of G (equivalently, a row of M) and d_v is the diagonal entry associated with it, produced up to the end of step i. Let $\pi_1(D_i)$ and $\pi_2(D_i)$ be the projections of D_i onto their first and second coordinates, respectively, so that $\pi_1(D_i)$ is the set of rows that have been diagonalized up to the end of step i and $\pi_2(D_i)$ is the (multi)set of diagonal elements found up to this step. Note that, if we only require the algorithm to produce the diagonal entries of a diagonal matrix that is congruent to the input matrix, it is not necessary to actually keep track of the particular pairs in D_i.

Let M_0 be the input matrix M. Let $\tilde{M}(i)$ be the matrix that is congruent to M obtained by performing the row and column operations performed by the algorithm up to step i. Let M_i be the matrix obtained from M by replacing by 0 any entry m_{uv} such that $u \neq v$ and the edge uv has not been introduced in $\mathcal{T}_i$, or such that $u = v$ and $\mathcal{T}(v) \cap \mathcal{T}_i = \emptyset$. Let $\tilde{M}_i$ be the matrix that is congruent to M_i produced by performing the row and column operations performed by the algorithm for all nodes in $\mathcal{T}_i$, in the order in which they have been performed. We only keep track of the matrices $\tilde{M}(i)$, M_i and $\tilde{M}_i$ to prove the correctness of the algorithm, they are not stored by the algorithm. In what follows, given $S \subset V$ and a matrix Q whose rows and columns are indexed by V, we write $Q[S]$ for the principal submatrix of M indexed by S. Moreover, if $u, v \in V$, $Q[u, v]$ denotes the entry uv in Q.

Our main technical lemmas control the relationship between N_i and the diagonal elements produced in step i with the matrices M_i, $\tilde{M}_i$ and $\tilde{M}(i)$. At the start of the algorithm, we set $\tilde{M}(-1) = \tilde{M}(0) = M$, $D_{-1} = D_0 = \emptyset$, $\mathcal{T}_0 = \emptyset$ and $V(N_0) = \emptyset$. The matrices N_0, M_0 and $\tilde{M}_0$ are empty.

▶ **Lemma 6.** *The following facts hold for all $i \in \{0, \ldots, m\}$.*

(a) *$\tilde{M}(i)$ and $\tilde{M}_i$ are symmetric matrices congruent to M and M_i, respectively.*

(b) *$D_{i-1} \subseteq D_i$.*

(c) *If a multiple of row (or column) v has been added to a row (or column) u in step i, then $v \in \pi_1(D_i \setminus D_{i-1}) \cup V_1(N_i)$ and $u \in \pi_1(D_i \setminus D_{i-1}) \cup V(N_i)$.*

The second lemma relates subtrees $\mathcal{T}(v)$ with the matrices produced by the algorithm.

▶ **Lemma 7.** *The following facts hold for all $i \in \{0, \ldots, m\}$.*

(a) *If $v \in \pi_1(D_i) \cup V_1(N_i)$ and $\mathcal{T}_i \cap \mathcal{T}(v) \neq \emptyset$, then $\mathcal{T}(v)$ is a subtree of $\mathcal{T}_i$.*

(b) *Let v be such that $\mathcal{T}(v) \cap \mathcal{T}_i \neq \emptyset$. Then $v \in V(N_i) \cup \pi_1(D_i)$.*

(c) *If $v \in \pi_1(D_i \setminus D_{i-1}) \cup V(N_i)$, then $\mathcal{T}_i \cap \mathcal{T}(v) \neq \emptyset$.*

The third result relates the entries of the matrices N_i and the set D produced by the algorithm with the entries of $\tilde{M}(i)$ and $\tilde{M}_i$.

▶ **Lemma 8.** *The following facts hold for all $i \in \{0, \ldots, m\}$.*

(a) *If $\mathcal{T}(v) \cap \mathcal{T}(w) = \emptyset$ and $\tilde{M}(i)[v, w] \neq 0$, then $v, w \in V(N_j)$, where $j \leq i$ is the largest index for which $\mathcal{T}(v) \cap \mathcal{T}_j \neq \emptyset$ or $\mathcal{T}(w) \cap \mathcal{T}_j \neq \emptyset$. For $\tilde{M}_i$, $\tilde{M}_i[v, w] \neq 0$ only if $v, w \in V(N_i)$.*

(b) *If $(v, d_v) \in D_i$, the row (and column) associated with v in $\tilde{M}(i)$, consists of zeros, with the possible exception of the vth entry, which is equal to d_v. If $\mathcal{T}_i \cap \mathcal{T}(v) \neq \emptyset$, then the row (and column) associated with v in $\tilde{M}_i$ satisfy the same property.*

(c) *If $v \in V_1(N_i)$, then the row (and column) associated with v in $\tilde{M}_i$ coincides with the row (and column) associated with v (restricted to the elements of $u \in V(N_i)$) in $\tilde{M}(i)$. The entries uv and vu are equal to 0 if $u \notin B_i$ and are equal to the corresponding entries in N_i if $u \in B_i$. Moreover, the entries uv and vu of $\tilde{M}(i)$ for $u \notin V(N_i)$ are equal to 0.*

(d) *Assume that $u, v \in B_i$. The entry uv of $\tilde{M}_i$ is equal to the entry uv of $N_i^{(2)}$.*

The proof of Lemmas 6, 7 and 8 is by induction on i. As $M_0 = M$, Lemma 6(a) is obviously true for $i = 0$, while Lemma 8(a) holds by definition of tree decomposition. The remaining items are vacuously true.

Before detailing each step of the algorithm, we show that, if the above lemmas hold for $i = m$, where m is the the number of nodes in the tree decomposition, then Algorithm `CongruentDiagonal` correctly computes a diagonal matrix congruent to M. To see why this is true, by Lemma 6(a), we know that M is congruent to $\tilde{M}(m)$. Moreover, by Lemma 8(b), if $(v, d_v) \in D_m$, then the row (and column) associated with v in $\tilde{M}(m)$ consists of zeros, with the possible exception of the vth entry, which is equal to d_v. It remains to prove that $\pi_1(D) = V$. To this end, let $v \in V$ and let i be the node that forgets v given by Lemma 4(a). Let j be its child. Since $v \in B_j$, we have $\mathcal{T}(v) \cap \mathcal{T}_i \neq \emptyset$, so that $v \in V_1(N_i) \cup \pi_1(D_i)$ by Lemma 7(b) (we are using that $v \notin B_i = V_2(N_i)$, a consequence of Lemma 5(c)). Then $\mathcal{T}(v) \subset \mathcal{T}_i$ by Lemma 7(a). If $v \in \pi_1(D_i)$ we are done, so assume that $v \in V_1(N_i)$. Let ℓ be the parent of i. By Lemma 7(b), $v \in V_1(N_\ell) \cup \pi_1(D_\ell)$. We would again be done if $v \in \pi_1(D_\ell)$, otherwise we repeat the argument to show that $v \in V(N_p)$, where p is the parent of ℓ. This argument may be repeated inductively. The result now follows from the fact that the root m of the tree satisfies $B_m = \emptyset$, which implies that $V_2(N_m) = \emptyset$ by Lemma 5(c). This implies that $V_1(N_m) = \emptyset$ by Lemma 5(a), as required.

We now describe each step of Algorithm `CongruentDiagonal`. When the node is a leaf corresponding to a bag B_i of size b_i, then we apply procedure `LeafBox`. This procedure only initializes a matrix N_i to be transmitted up the tree. The matrix N_i is such that $k' = 0$ and $k'' = b_i$, where $N_i^{(2)}$ is the diagonal matrix such that, for every $v \in B_i$, the entry vv is given by the element vv in M. Observe that no off-diagonal entries appear in this initialization, as the edges involving vertices in B_i have yet to be introduced.

```
LeafBox($B_i$)
input:  a set $B_i$ of size $b_i$
output:  a matrix $N_i = (N_i^{(1)}, N_i^{(2)})$
   Set $N_i^{(1)} = \emptyset$
   $N_i^{(2)}$ is a diagonal matrix of order $b_i$
   for each vertex $v \in B_i$ set entry $vv$ of $N_i^{(2)}$ as $m_{vv}$.
```

Figure 1 Procedure `LeafBox`.

By construction, the matrix N_i defined by `LeafBox` satisfies the properties of Lemma 5. It is not hard to show that, if Lemmas 6, 7 and 8 hold up to the end of step $i-1$ and step i processes a leaf B_i, the lemmas must also hold at the end of step i.

Next, we explain the procedures associated with nodes of type IntroduceVertex and IntroduceEdge. For vertices, the input is the set B_i, the vertex v that has been introduced and the matrix $N_j = (N_j^{(1)}, N_j^{(2)})$ obtained after processing the child B_j of B_i. The matrix N_i is obtained from N_j by adding a new type-ii row/column corresponding to vertex v (this becomes the last row/column of the matrix). This row is zero everywhere with the exception of the diagonal entry vv, which is equal to m_{vv}.

For edges, the input is the set B_i, a vertex $v \in B_i$, the set $\Gamma_{B_i}(v)$ of neighbors of v in B_i and the matrix $N_j = (N_j^{(1)}, N_j^{(2)})$ produced after processing the child B_j of B_i. The matrix N_i is obtained from N_j by replacing the entries uv and vu in $N_j^{(2)}$, which are equal to some value α, by $\alpha + \beta$, where β is the entry uv in M.

It is obvious that the matrices N_i produced by `IntroVertexBox` and `IntroEdgesBox` satisfy the properties of Lemma 5. In both cases, no row/column operation is performed, $\tilde{M}(i) = \tilde{M}(i-1)$ and $D_i = D_{i-1}$.

We now address the operation associated with nodes of type join. Let i be a node of type join and let N_j and N_ℓ be the matrices transmitted by its children, where $j < \ell < i$. By Lemma 5(c) and the definition of the join operation, we have $V_2(N_j) = V_2(N_\ell)$. By Lemma 7(a), we have $V_1(N_j) \cap V_1(N_\ell) = \emptyset$.

```
IntroVertexBox($B_i, v, N_j$)
input: a node $i$ with bag $B_i$, child $j$, $B_j = B_i - v$, and $N_j = (N_j^{(1)}, N_j^{(2)})$
output: a matrix $N_i = (N_i^{(1)}, N_i^{(2)})$
    $N_i^{(1)} = N_j^{(1)}$
    $N_i^{(2)} = N_j^{(2)}$
    Add zero row and zero column $v$ to $N_i^{(2)}$
    Add diagonal element $vv$ to $N_i^{(2)}$ as $m_{vv}$
    Add zero column $v$ to $N_i^{(1)}$
```

Figure 2 Procedure `IntroVertexBox`.

```
IntroEdgesBox($B_i, v, \Gamma_{B_i}(v), N_j$)
input: $v \in B_i$, $\Gamma_{B_i}(v)$ and a matrix $N_j = (N_j^{(1)}, N_j^{(2)})$
output: a matrix $N_i = (N_i^{(1)}, N_i^{(2)})$
    $N_i^{(1)} = N_j^{(1)}$
    $N_i^{(2)} = N_j^{(1)}$
    For all $u \in \Gamma_{B_i}(v)$, set entries $uv$ and $vu$ of $N_i^{(2)}$ as $N_i^{(2)}[uv] + m_{uv}$
```

Figure 3 Procedure `IntroEdgesBox`.

The `JoinBox` operation first creates a matrix N_i^* whose rows and columns are labelled by $V_1(N_j) \cup V_1(N_\ell) \cup V_2(N_j)$ with the structure below. Assume that $|V_1(N_j)| = r$, $|V_1(N_\ell)| = s$ and $|B_i| = t$.

$$N_i^* = \begin{array}{|c|c|c|}\hline \mathbf{0}_{r\times r} & \mathbf{0}_{r\times s} & N_j^{(1)} \\ \hline \mathbf{0}_{s\times r} & \mathbf{0}_{s\times s} & N_\ell^{(1)} \\ \hline N_j^{(1)T} & N_\ell^{(1)T} & N_i^{*(2)} \\ \hline \end{array}, \qquad (2)$$

where $N_i^{*(2)} = N_j^{(2)} + N_\ell^{(2)} - M_i[B_i]$. We observe that, at this point, $M_i[B_i]$ is a diagonal matrix, as no edges with both endpoints in B_i may have been introduced by Lemma 4(b).

Note that the matrix

$$N_i^{*(1)} = \begin{array}{|c|}\hline N_j^{(1)} \\ \hline N_\ell^{(1)} \\ \hline \end{array}$$

is an $(r+s) \times t$ matrix consisting of two matrices in row echelon form on top of each other. We perform row and column operations on N_i^* involving rows associated with $V_1(N_j)$ (the *left rows*) and $V_1(N_\ell)$ (the *right rows*) to turn N_i^* into a matrix $N_i^{*(1)}$ in row echelon form. To do this, we proceed by steps in which we always add a multiple of a left or right row (and the corresponding column) to a right row (and the corresponding column): to choose the next operation, at each step we look at the pivots of the right rows and select the leftmost such pivot that coincides with a pivot of a left row or with the pivot of another right row (say w and v are the right and left/right rows that satisfy this, and j is the pivot. At the first step, v is always a left row). If R_w and C_w are the row and column corresponding to w, while $\alpha_j = R_w(j) = C_w(j)$ and $\beta_j = R_v(j) = C_v(j)$, we define

$$R_w \leftarrow R_w - \frac{\alpha_j}{\beta_j} R_v, C_w \leftarrow C_w - \frac{\alpha_j}{\beta_j} C_v.$$

```
JoinBox($B_i, N_j, N_\ell$)
input:  a node $i$ with bag $B_i$ and matrices $N_j, N_\ell$ associated with its two children
output:  a matrix $N_i = (N_i^{(1)}, N_i^{(2)})$
   $N_i^{(2)} = N_j^{(2)} + N_\ell^{(2)}$
   For every $v \in B_i$, set the entry $vv$ of $N_i^{(2)}$ as $N_i^{(2)}[v,v] - m_{vv}$
   Construct $N_i^{(1)*} = \begin{bmatrix} N_j^{(1)} \\ N_\ell^{(1)} \end{bmatrix}$
   Do row and column operations on $N_i^{(1)*}$, putting it in row echelon form
   For each zero row of $N_i^{(1)*}$ (indexed by a vertex $u$), add $(u,0)$ to $D_i$
   $N_i^{(1)}$ is $N_i^{(1)*}$ with zero row/columns removed
```

Figure 4 Procedure `JoinBox`.

This eliminates the pivot of row w. Note that the entries in $N_j^{(2)}$ are not affected by these operations. Moreover, for all $u, v \in V_1(N_j) \cup V_1(N_\ell)$, the entry uv in N_i^* is equal to 0.

As we do this, we may create rows and columns (associated with the right rows) whose entries are all zero (for instance, this will certainly happen if $r + s > t$). If Z_i denotes the set of vertices associated with rows whose entries are all zero, where $|Z_i| = z$, we let $D_i = D_{i-1} \cup \{(v, 0) \colon v \in Z_i\}$, we remove the rows and columns associated with vertices in Z_i from $N_i^{*(1)}$ to produce the matrix

$$N_i = \begin{array}{|c|c|}\hline \mathbf{0}_{k' \times k'} & N_i^{(1)} \\ \hline N_i^{(1)T} & N_i^{(2)} \\ \hline \end{array}, \tag{3}$$

where $k' = r + s - z$, $k'' = t$ and $N_i^{(1)}$ is a matrix of dimension $k' \times k''$ in row echelon form and $N_i^{(2)} = N_i^{*(2)}$. We observe that N_i satisfies the properties of Lemma 5. Items (b) and (c) are satisfied by construction. For (a), the inequality $k' \leq k''$ is a consequence of the fact that $N_i^{(1)}$ is in row echelon form, while $k'' \leq k + 1$ follows from $k'' = |B_i|$. Proving that Lemmas 6, 7 and 8 hold after step i uses induction and the properties discussed above.

To conclude the description of the algorithm, we describe `ForgetBox`. Assume that i forgets vertex v and let j be its child, so that $B_i = B_j \setminus \{v\}$. By Lemma 4(c), we know that $\mathcal{T}(v)$ is a subtree of $\mathcal{T}_j$, and therefore all edges incident with v have been introduced. This procedure starts with $N_i^* = N_j$ and produces a new matrix N_i so that $v \in V_1(N_i)$ or $v \in \pi_1(D_i \setminus D_{i-1})$.

We look at N_i^* in the following way:

$$N_i^* = \begin{array}{|c|c|c|}\hline d_v & \mathbf{x}_v & \mathbf{y}_v \\ \hline \mathbf{x}_v^T & \mathbf{0}_{k' \times k'} & N_i^{*(1)} \\ \hline \mathbf{y}_v^T & N_i^{*(1)T} & N_i^{*(2)} \\ \hline \end{array}. \tag{4}$$

Here, the first row and column represent the row and column in N_j associated with v, while $N_i^{*(1)}$ and $N_i^{*(2)}$ are given by the submatrices of $N_j^{(1)}$ and $N_j^{(2)}$ obtained by removing the row and/or column associated with v. In particular $\mathbf{x}_v$ and $\mathbf{y}_v$ are row vectors of size k_j' and $k_j'' - 1$, respectively.

```
ForgetBox($B_i, v, N_j,$)
input:  a node $i$ with bag $B_i$, child $j$ with $B_i = B_j \setminus \{v\}$ and matrix $N_j$
output:  a matrix $N_i = (N_i^{(1)}, N_i^{(2)})$
   $N_i = N_j$
   Perform row/column exchange so that $N_i$ has the form of (4)
   if $\mathbf{x}_v$ is empty or 0 then
      if $\mathbf{y}_v$ is empty or 0 then
         add $(v, d_v)$ to $D$
         remove row $v$ from $N_i$
      else if $d_v \neq 0$ then // Here $\{\mathbf{y}_v \neq 0\}$.
         use $d_v$ to diagonalize row/column $v$
         add $(v, d_v)$ do $D$ and remove row $v$ form $N_i$
      else // Here $d_v = 0$.
         Set $u = \min\{w : y_w \neq 0\}$
         do row and column operations inserting row $v$ to $N_i^{(1)}$
         if a zero row is obtained add $(v, 0)$ to $D$ and remove row from $N_i$
   else // Here $\mathbf{x}_v \neq 0$.
      use operations as in (5) to diagonalize rows $u$ and $v$
      add $(v, d_v)$ and $(u, d_u)$ to $D$ and eliminate rows $v$ and $u$ from $N_i$.
```

Figure 5 Procedure ForgetBox.

Depending on the properties of the vectors $\mathbf{x}_v$ and $\mathbf{y}_v$, we proceed in different ways.

Case 1: $\mathbf{x}_v$ is empty or $\mathbf{x}_v = [0 \cdots 0]$. If $\mathbf{y}_v = [0 \cdots 0]$ (or $\mathbf{y}_v$ is empty), we add (v, d_v) to D_i and remove the row and column associated with v from N_i^* to produce N_i.

If $\mathbf{y}_v \neq [0 \cdots 0]$, there are again two options. If $d_v = 0$, the aim is to turn v into a row of type-i. To do this, we need to insert $\mathbf{y}_v$ into the matrix $N_i^{*(1)}$ in a way that the ensuing matrix is in row echelon form. Note that this may be done by only adding multiples of rows of $V(N_i^{*(1)})$ to the row associated with v. At each step, if the pivot α_j of the (current) row associated with v is in the same position of the pivot β_j of R_u, the row associated with vertex u already in $N_i^{*(1)}$, we use R_u to eliminate the pivot of R_v:

$$R_v \leftarrow R_v - \frac{\alpha_j}{\beta_j} R_u, C_v \leftarrow C_v - \frac{\alpha_j}{\beta_j} C_u.$$

This is done until the pivot of the row associated with v may not be cancelled by pivots of other rows, in which case the row associated with v may be inserted in the matrix (to produce the matrix $N_i^{(1)}$), or until the row associated with v becomes a zero row, in which case $(v, 0)$ is added to D_i and we remove the row and column associated with v from N_i^* to produce N_i. If $d_v \neq 0$, we use d_v to eliminate the nonzero entries in y_v and diagonalize the row corresponding to v. For each element $u \in B_i$ such that the component α_v of y_v associated with u is nonzero, we perform

$$R_u \leftarrow R_u - \frac{\alpha_v}{d_v} R_v, C_u \leftarrow C_u - \frac{\alpha_v}{d_v} C_v.$$

When all such entries have been eliminated, we add (d_v, v) to D_i and we let N_i be the remaining matrix. Observe that, in this case, $N_i^{(1)} = N_i^{*(1)}$, only the elements of $N_i^{*(2)}$ are modified to generate $N_i^{(2)}$.

Case 2: $\mathbf{x}_v$ *is nonempty and* $\mathbf{x}_v \neq [0 \cdots 0]$.

Let u be the vertex associated with the rightmost nonzero component of x_v. Let α_j be this component. We use this element to eliminate all the other nonzero entries in x_v, from right to left. Let w be the vertex associated with the entry α_ℓ. We perform

$$R_w \leftarrow R_w - \frac{\alpha_\ell}{\alpha_j} R_u, \quad C_w \leftarrow C_w - \frac{\alpha_\ell}{\alpha_j} C_u.$$

A crucial fact is that the entries corresponding to the matrix $N_i^{*(1)}$ in the matrix produced by these operations is still in row echelon form and has the same pivots as $N_i^{*(1)}$. If $d_v \neq 0$, we still use R_u to eliminate this element:

$$R_v \leftarrow R_v - \frac{d_v}{2\alpha_j} R_u, \quad C_v \leftarrow C_v - \frac{d_v}{2\alpha_j} C_u.$$

At this point, the only nonzero entries in the $(k'+1) \times (k'+1)$ left upper corner of the matrix obtained after performing these operations are in positions uv and vu (and are equal to α_j). We perform the operations

$$R_u \leftarrow R_u + \frac{1}{2} R_v, \quad C_u \leftarrow C_u + \frac{1}{2} C_v, R_v \leftarrow R_v - R_u, \quad C_v \leftarrow C_v - C_u$$

The relevant entries of the matrix are modified as follows:

$$\begin{pmatrix} 0 & \alpha_j \\ \alpha_j & 0 \end{pmatrix} \to \begin{pmatrix} 0 & \alpha_j \\ \alpha_j & \alpha_j \end{pmatrix} \to \begin{pmatrix} -\alpha_j & 0 \\ 0 & \alpha_j \end{pmatrix}. \tag{5}$$

We are now in the position to use the diagonal elements to diagonalize the rows associated with v and u, as was done in Case 1, when $x_v = [0, \ldots, 0]$ and $d_v \neq 0$. At the end of the step, we add $(v, -\alpha_j)$ and (u, α_j) to D_i.

Finally, it is time to analyze the complexity of Algorithm `CongruentDiagonal`, and prove Theorem 1.

Proof. The correctness of Algorithm `CongruentDiagonal` follows from the Lemmas and the justifications of every step of the algorithm as it is described throughout the paper. By Lemma 2, the time bound of $O(k|\mathcal{T}| + k^2 n)$ is sufficient to transform an arbitrary given tree decomposition into an edge-efficient tree decomposition.

For the running time of the main computation, we have to analyze the procedures done at each type of tree node. `LeafBox` initializes a matrix in $O(k^2)$ trivial steps. `IntroVertexBox` and `IntroEdgesBox` use only $O(k)$ steps. For the other procedures, the main cost comes from row and column operations. As the matrices have order at most $k+1$ each such operation costs $O(k)$. Regarding `ForgetBox`, when v is forgotten, either v is turned into a type-i vertex, or its row and column, and possibly the row and column of another vertex u, are diagonalized. The latter requires at most $O(k)$ row and column operations. If v is turned into a type-i vertex, then inserting it into the matrix $N_i^{(1)}$ in row echelon form takes at most $k+1$ row operations. `JoinBox` can be most time consuming. To insert just one row vector into a matrix of order k in row echelon form, and preserving this property by adding multiples of one vector to another, can require up to $k+1$ row operations. Each such operation can be done in time $O(k)$. To combine two matrices in row echelon form into one such matrix, up to $k+1$ row vectors are inserted. Thus the total time for this operation is $O(k^3)$. This immediately results in an upper bound of $O(k^3 n)$ for the whole computation.

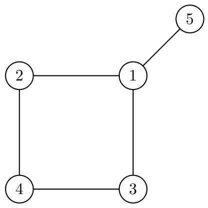


Figure 6 Graph with 5 labeled vertices.

To obtain an $O(k|\mathcal{T}| + k^2n)$ bound for the whole computation, we have to employ a different accounting scheme for the time spent to merge two matrices in row echelon form into one during the `JoinBox` procedure. We notice that every row operation in any $N_i^{(1)}$ creates at least one 0 entry in some row, meaning that its pivot moves at least one position to the right or creates a zero row. For every vertex v, its row is added at most once to some $N_i^{(1)}$, namely when v is forgotten. Its pivot can move at most k times and disappear at most once. Thus for all n vertices together, at most $(k+1)n$ row operations can occur in all $N_i^{(1)}$ together. This only uses time $O(k^2n)$. Thus, while during a single join procedure, $\Omega(k^2)$ row operations might be needed, the average is $O(k)$ such operations per join procedure. ◀

4 Example

In this section, we illustrate how the algorithm acts on a concrete example. To this end, we consider the graph in Figure 6. An edge-explicit tree decomposition representing this graph may be seen in Figure 7.

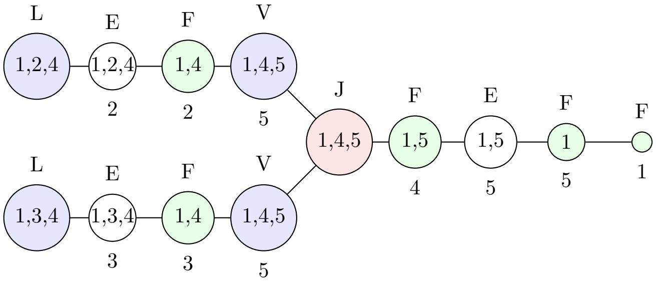


Figure 7 An edge-explicit tree representing the graph, where the root is on the right. A label above describes the type of node, a label below indicates which vertices or edges are introduced or forgotten.

Note that G is the underlying graph of the symmetric matrix

$$M = \begin{pmatrix} 1 & 1 & 1 & 0 & -1 \\ 1 & 0 & 0 & 2 & 0 \\ 1 & 0 & 1 & -1 & 0 \\ 0 & 2 & -1 & 1 & 0 \\ -1 & 0 & 0 & 0 & -1 \end{pmatrix}.$$

Suppose that we want to find the number of eigenvalues greater than 0 (and equal to and less than 0). We apply our algorithm with $c = 0$, that is, originally $M - cI = M$.

Assume that we have ordered the nodes of the tree in Figure 7 in post order so that the first five nodes are in the upper branch of Figure 7, followed by the five nodes on the lower branch and by the five nodes starting from the node of type join. When we start, the node

Leaf calls the LeafBox with bag of vertices $\{1,3,4\}$ producing the matrix $N_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$, whereas the node Introduce Edges labelled by vertex 3 introduces edges 13 and 34, leading to the matrix

$$N_3 = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & -1 \\ 0 & -1 & 1 \end{pmatrix}.$$

The node Forget Vertex $F3$ receives the matrix N_3 and, after exchanging rows and columns 1 and 2 (so that vertex 3 corresponds to first row), processes the matrix

$$\left(\begin{array}{r|rr} 1 & 1 & -1 \\ \hline 1 & 1 & 0 \\ -1 & 0 & 1 \end{array}\right).$$

According to our description, we are in Case 1 of Procedure ForgetBox, with $\mathbf{x}_v$ empty and $\mathbf{y}_v = [1,-1] \neq [0,0]$. Since $d_v = 1$, the algorithm diagonalizes the row/columns corresponding to v. To this end, we perform the operations $R_2 \leftarrow R_2 - R_1$, followed by $C_1 \leftarrow C_1 - C_2$, producing the matrix

$$N_3 = \begin{pmatrix} 1 & 0 & -1 \\ 0 & 0 & 1 \\ -1 & 1 & 1 \end{pmatrix},$$

followed by the operations $R_3 \leftarrow R_3 + R_1$, followed by $C_3 \leftarrow C_3 + C_1$, giving

$$\left(\begin{array}{c|cc} 1 & 0 & 0 \\ \hline 0 & 0 & 1 \\ 0 & 1 & 0 \end{array}\right).$$

We have diagonalized row 1, corresponding to vertex 3. Hence, the node $F3$ sets the diagonal vector $D = (v, d_v) = (3,1)$ and transmits the matrix $N_4 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$, whose rows are indexed by vertices 1 and 4, respectively, to its parent. The node $V5$ introduces vertex 5, producing the matrix

$$N_5 = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -1 \end{pmatrix},$$

indexed by the vertices $1,4$ and 5, respectively. We notice that this matrix N is such that $N_5^{(1)}$ is empty and $N_5^{(2)} = N$.

Working on the lower branch of the tree of Figure 7 in a similar way, we arrive at node $F2$, after exchanging the rows/columns, with the matrix

$$N_9^* = \left(\begin{array}{c|cc} 0 & 1 & 2 \\ \hline 1 & 1 & 0 \\ 2 & 0 & 1 \end{array}\right),$$

indexing vertices 2, 1 and 4, respectively. This corresponds to Case 1 of Procedure ForgetBox, with empty $\mathbf{x}_v$ and $\mathbf{y}_v = [1,2]$, but $d_v = 0$. We notice that, in this particular case, the matrix $N_9^{(1)} = [1,2]$ is already in row echelon form, so that $N_9 = (N_9^{(1)}, N_9^{(2)})$, $N_9^{(2)} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$, is transmitted by F_2 to its parent.

Now the Introduce Vertex node $V5$ processes the matrix N_9 and produces matrix

$$N_{10} = \left(\begin{array}{c:ccc} 0 & 1 & 2 & 0 \\ \hdashline 1 & 1 & 0 & 0 \\ 2 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{array}\right),$$

where the rows are indexed by the vertices 2, 1, 4 and 5, respectively.

Now the JoinBox procedure will process the matrices N_5 and N_{10}. We first do the operation $N_{11}^{(2)} = N_{10}^{(2)} + N_5^{(2)} - M_D[1,4,5]$, where $M_D[1,4,5]$ is the diagonal matrix whose rows and columns are indexed by 1, 4 and 5 such that each entry ii is given by m_{ii}. We then merge $N_{10}^{(1)}$ on top of $N_5^{(1)}$. Since $N_5^{(1)}$ is empty, these operation produce the matrix

$$N_{11} = \left(\begin{array}{c:ccc} 0 & 1 & 2 & 0 \\ \hdashline 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 \end{array}\right),$$

whose rows index the vertices 2, 1, 4 and 5, respectively.

We now process node $F4$ of the tree, where the vertex 4 is forgotten. We first exchange rows and columns so that the first row is indexed by 4. The matrix becomes

$$N_{12}^{*} = \left(\begin{array}{c:c:cc} 0 & 2 & 1 & 0 \\ \hdashline 2 & 0 & 1 & 0 \\ \hdashline 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 \end{array}\right),$$

whose rows index the vertices 4, 2, 1 and 5, respectively. We look at this matrix as in equation (4). We are in case 2 of Procedure ForgetBox with $d_v = 0, \mathbf{x}_v = [2], \mathbf{y}_v = [1,0], N_{12}^{*(1)} = [1,0]$ and $N_{12}^{*(2)} = \begin{pmatrix} 0 & 0 \\ 0 & -1 \end{pmatrix}$. Since $\mathbf{x}_v = [2]$, there is no operation to perform in order to put $\mathbf{x}_v$ in row echelon form. The goal now is to transform the left upper corner of the above matrix $\begin{pmatrix} 0 & 2 \\ 2 & 0 \end{pmatrix}$ into a diagonal matrix. We perform the operations $R_2 \leftarrow R_2 + 1/2R_1, C_2 \leftarrow C_2 + 1/2C_1$ followed by $R_1 \leftarrow R_1 - R_2$ and $C_1 \leftarrow C_1 - C_2$, obtaining the matrix

$$N_{12}^{*} = \begin{pmatrix} -2 & 0 & -1/2 & 0 \\ 0 & 2 & 3/2 & 0 \\ -1/2 & 2 & 2 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}.$$

We now use the nonzero pivots obtained in order to diagonalize rows 1 and 2. To achieve this, we perform the operations $R_3 \leftarrow R_3 - 1/4R_1, C_3 \leftarrow C_3 - 1/4C_1$, followed by $R_3 \leftarrow R_3 - 3/4R_2, C_3 \leftarrow C_3 - 3/4C_2$. This produces the matrix

$$N_{12}^{*} = \begin{pmatrix} -2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}.$$

At this point, the first two rows are diagonalized, corresponding to vertices 4 and 2. To the diagonal vector $D = (v, d_v)$ we added the components $(4,-2)$ and $(2,2)$. Node $F4$ transmits the matrix $N_{12} = \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix}$, corresponding the edges 1 and 5. The Introduce Edges node

$E5$ puts the edge 15, and the matrix returned by IntroduceEdgesBox is $N_{13} = \left(\begin{array}{c:c} -1 & -1 \\ \hdashline -1 & -1 \end{array}\right)$ in the form the Forget node $F5$ receives it. We see that $\mathbf{x}_v$ is empty and $\mathbf{y}_v = [-1]$ and $d_v = -1$, meaning that we are in case 2. Using d_v as pivot we arrive at $N^*_{14} = \begin{pmatrix} -1 & 0 \\ 0 & 0 \end{pmatrix}$, adding to the diagonal vector D the component $(v, d_v) = (5, -1)$, and returning the matrix $N_{14} = [0]$. The final node $F1$ forgets the vertex 1. Since the matrix received is $[0]$ already in diagonal form, it adds to D the component $(v, d_v) = (1, 0)$. The diagonal vector D returned by the algorithm is

$$\begin{pmatrix} v \\ d_v \end{pmatrix} = \begin{pmatrix} 3 & 4 & 2 & 5 & 1 \\ 1 & -2 & 2 & -1 & 0 \end{pmatrix},$$

meaning that M has 2 positive eigenvalues, 2 negative eigenvalues and 0 is an eigenvalue with multiplicity 1.

References

1 Hans L. Bodlaender, Paul S. Bonsma, and Daniel Lokshtanov. The fine details of fast dynamic programming over tree decompositions. In Gregory Z. Gutin and Stefan Szeider, editors, *IPEC*, volume 8246 of *Lecture Notes in Computer Science*, pages 41–53. Springer, 2013.

2 Hans L. Bodlaender, John R. Gilbert, Hjálmtýr Hafsteinsson, and Ton Kloks. Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. *J. Algorithms*, 18(2):238–255, 1995.

3 Derek G. Corneil and Udi Rotics. On the relationship between clique-width and treewidth. *SIAM J. Comput*, 34(4):825–847, 2005.

4 Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discrete Appl. Math.*, 101(1-3):77–114, 2000.

5 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

6 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Joham M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In *Proceedings of the 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 150–159, 2011.

7 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

8 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006.

9 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, Michal Pilipczuk, and Marcin Wrochna. Fully polynomial-time parameterized computations for graphs and matrices of low treewidth. *ACM Trans. Algorithms*, 14(3):34:1–34:45, 2018.

10 Martin Fürer. A natural generalization of bounded tree-width and bounded clique-width. In Alberto Pardo and Alfredo Viola, editors, *LATIN 2014: Theoretical Informatics - 11th Latin American Symposium, Montevideo, Uruguay, March 31 - April 4, 2014. Proceedings*, volume 8392 of *Lecture Notes in Computer Science*, pages 72–83. Springer, 2014. doi:10.1007/978-3-642-54423-1_7.

11 Martin Fürer. Multi-clique-width. In Christos H. Papadimitriou, editor, *8th Innovations in Theoretical Computer Science Conference, ITCS 2017, January 9-11, 2017, Berkeley, CA, USA*, volume 67 of *LIPIcs*, pages 14:1–14:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ITCS.2017.14.

12 Martin Fürer, Carlos Hoppen, David P. Jacobs, and Vilmar Trevisan. Eigenvalue location in graphs of small clique-width. *Linear Algebra and its Applications*, 560:56–85, 2019.

13 Martin Fürer and Huiwen Yu. Space saving by dynamic algebraization based on tree-depth. *Theory of Computing Systems*, 61(2):283–304, 2017.

14 Archontia C. Giannopoulou, George B. Mertzios, and Rolf Niedermeier. Polynomial fixed-parameter algorithms: A case study for longest path on interval graphs. *Theor. Comput. Sci.*, 689:67–95, 2017. doi:10.1016/j.tcs.2017.05.017.

15 T. Kloks. *Treewidth: Computations and Approximations*, volume 842 of *Lecture Notes in Computer Science*. Springer Verlag, 1994.

16 Carl Meyer. *Matrix analysis and applied linear algebra.* Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2000.

17 Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms.* Oxford University Press, 2006.

18 Neil Robertson and Paul D. Seymour. Graph minors II. Algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986.

19 Donald J. Rose, R. Endre. Tarjan, and George S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM Journal on Computing*, 5(2):266–283, 1976.

20 Donald J. Rose and Robert Endre Tarjan. Algorithmic aspects of vertex elimination on directed graphs. *SIAM Journal on Applied Mathematics*, 34(1):176–197, 1978.

Counting Solutions to Random CNF Formulas

Andreas Galanis
Department of Computer Science, University of Oxford, UK
andreas.galanis@cs.ox.ac.uk

Leslie Ann Goldberg
Department of Computer Science, University of Oxford, UK
leslie.goldberg@cs.ox.ac.uk

Heng Guo
School of informatics, University of Edinburgh, UK
hguo@inf.ed.ac.uk

Kuan Yang
Department of Computer Science, University of Oxford, UK
kuan.yang@cs.ox.ac.uk

Abstract

We give the first efficient algorithm to approximately count the number of solutions in the random k-SAT model when the density of the formula scales exponentially with k. The best previous counting algorithm was due to Montanari and Shah and was based on the correlation decay method, which works up to densities $(1+o_k(1))\frac{2\log k}{k}$, the Gibbs uniqueness threshold for the model. Instead, our algorithm harnesses a recent technique by Moitra to work for random formulas with much higher densities. The main challenge in our setting is to account for the presence of high-degree variables whose marginal distributions are hard to control and which cause significant correlations within the formula.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Theory of computation → Randomness, geometry and discrete structures; Mathematics of computing → Discrete mathematics

Keywords and phrases random CNF formulas, approximate counting

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.53

Category Track A: Algorithms, Complexity and Games

Related Version A full version with all proofs is at `https://arxiv.org/abs/1911.07020`. The theorem numbers here match those in the full version. Proof sketches here refer to lemmas in the full version without restating them in detail.

Funding The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) ERC grant agreement no. 334828. The paper reflects only the authors' views and not the views of the ERC or the European Commission. The European Union is not liable for any use that may be made of the information contained therein.

1 Introduction

Let $\Phi = \Phi(k,n,m)$ be a k-CNF formula on n Boolean variables with m clauses chosen uniformly at random where each clause has size $k \geq 3$. The random formula Φ shows an interesting threshold behaviour, where the asymptotic probability that Φ is satisfiable drops dramatically from 1 to 0 when the density $\alpha := m/n$ crosses a certain threshold $\alpha_\star$. There has been tremendous progress on establishing this phase transition and pinpointing the

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 53; pp. 53:1–53:14
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

threshold $\alpha_\star$ [25, 19, 3, 4, 12, 15] guided by elaborate but non-rigorous methods in physics [28, 27]. The exact value of the threshold $\alpha_\star$ is established in [15] for sufficiently large k; it is known that $\alpha_\star = 2^k \ln 2 - \frac{1}{2}(1 + \ln 2) + o_k(1)$ as $k \to \infty$.

In contrast, the "average case" computational complexity of random k-CNF formulas remains elusive. It is a notoriously hard problem to design algorithms that succeed in finding a satisfying assignment when the density of the formula Φ is close to (but smaller than) the satisfiability threshold $\alpha_\star$. The best polynomial-time algorithm to find a satisfying assignment of Φ is due to Coja-Oghlan [8], which succeeds if $\alpha < (1 - o_k(1)) \cdot 2^k \ln k/k$. It is known that beyond this density bound $2^k \ln k/k$ the solution space of the formula undergoes a phase transition and becomes severely more complicated [2], so local algorithms are bound to fail to find a satisfying assignment in polynomial time (see for example [24, 9, 11]).

It is also a natural question to determine the number of satisfying assignments to Φ, denoted by $Z(\Phi)$, when the density is below the satisfying threshold. It has been shown that $\frac{1}{n} \log Z(\Phi)$ is concentrated around its expectation [1, 13] for $\alpha < (1 - o_k(1)) \cdot 2^k \ln k/k$. However, for the k-SAT model, there is no known formula for the expectation $\mathbb{E}\, \frac{1}{n} \log Z(\Phi)$ (though see [35, 14] for progress along these lines for more symmetric models of random formulas). Regarding the algorithmic question, Montanari and Shah [31] have given an efficient algorithm to approximate $\log Z(\Phi)$ if $\alpha \le \frac{2 \log k}{k}(1 + o_k(1))$, based on the correlation decay method and the uniqueness threshold of the Gibbs distribution. Note that this only gives an approximation to $Z(\Phi)$ within an exponential factor. Also, the threshold for α is exponentially lower than the satisfiability threshold. No efficient algorithm was known to give a more precise approximation.

In this paper, we address the algorithmic counting problem by giving the first *fully polynomial-time approximation scheme* (FPTAS) for the number of satisfying assignments to random k-CNF formulas, if the density α is less than 2^{rk}, for sufficiently large k and some constant $r > 0$. Our bound is exponential in k and goes well beyond the uniqueness threshold of $\frac{2 \log k}{k}(1 + o_k(1))$ which is required by the correlation decay method.

Our result is related to other algorithmic counting results on random graphs such as counting colourings, independent sets, and other structures [33, 37, 16, 26] in random graphs. However, previous methods, such as Markov Chain Monte Carlo and Barvinok's method, appear to be difficult to apply to random formulas. Instead, our algorithm is the first adaptation of Moitra's method [30] to the random instance setting. We give a high level overview of the techniques in Section 1.2.

1.1 The model and the main result

For $k \ge 3$, let $\Phi = \Phi(k, n, m)$ denote a k-SAT formula chosen uniformly at random from the set of all k-SAT formulas with n variables and m clauses. Specifically, Φ has n variables $v_1, v_2, \ldots, v_n$ and m clauses $c_1, c_2, \ldots, c_m$. Each clause c_i has k literals $\ell_{i,1}, \ell_{i,2}, \ldots, \ell_{i,k}$ and each literal $\ell_{i,j}$ is chosen uniformly at random from $2n$ literals $\{v_1, v_2, \ldots, v_n, \neg v_1, \neg v_2, \ldots, \neg v_n\}$. Note that each clause has exactly k literals (repetitions allowed), so there are $(2n)^{km}$ possible formulas; we use $\Pr(\cdot)$ to denote the uniform distribution on the set of all such formulas. Throughout, we will assume that $m = \lfloor n\alpha \rfloor$, where $\alpha > 0$ is the density of the formula. We say that an event $\mathcal{E}$ holds *w.h.p.* if $\Pr(\mathcal{E}) = 1 - o(1)$ as $n \to \infty$.

For a k-SAT formula Φ, we let $\Omega = \Omega(\Phi)$ denote the set of satisfying assignments of Φ.

▶ **Theorem 1.** *There is a polynomial-time algorithm $\mathcal{A}$ and there are two constants $r > 0$ and $k_0 \ge 3$ such that, for all $k \ge k_0$ and all $\alpha < 2^{rk}$, the following holds w.h.p. over the choice of the random k-SAT formula $\Phi = \Phi(k, n, \lfloor \alpha n \rfloor)$. The algorithm $\mathcal{A}$, given as input the formula Φ and a rational $\varepsilon > 0$, outputs in time poly$(n, 1/\varepsilon)$ a number Z that satisfies $e^{-\varepsilon}|\Omega(\Phi)| \le Z \le e^{\varepsilon}|\Omega(\Phi)|$.*

Throughout this paper, we will assume that $k \geq k_0$ where k_0 is a sufficiently large constant. We will also assume that the density α of the formula Φ satisfies $\alpha < 2^{k/300}/k^3$, so r can be taken to be $1/301$ in Theorem 1. The constant 300 here is not optimised, but we do not expect to be able to use the current techniques to improve it substantially. Our main point is that for a density which is exponential in k, an FPTAS exists for random k-CNF formulas. Finally, we assume that $k^2\alpha \geq 1$, otherwise it is well-known (see, e.g., Theorem 3.6 in [34]) that w.h.p. every connected component of Φ, viewed as a hypergraph where variables correspond to vertices and clauses correspond to hyperedges, is of size $O(\log n)$. In this case we can count the number of satisfying assignments by brute force.

1.2 Algorithm overview

We give a high-level overview of our algorithm here before giving the details. Approximately counting the satisfying assignments of a k-CNF formula has been a challenging problem using traditional algorithmic techniques, since the solution space (the set of satisfying assignments) is complicated and it is not connected, using the transitions of commonly-studied Markov chains. Recently some new approaches were introduced [30, 20]. Most notably, the breakthrough work of Moitra [30] gives the first (and so far the only) efficient deterministic algorithm that can approximately count the satisfying assignments of k-CNF formulas in which each variable appears in at most d clauses, if, roughly, $d \lesssim 2^{k/60}$. Inspired by this, Feng et al. [18] have also given a MCMC algorithm which applies when $d \lesssim 2^{k/20}$.

As our goal is to count satisfying assignments of sparse random k-CNF formulas, where these degree bounds do not hold, but average degrees are small, it is natural to also choose Moitra's method in the random instance setting. However, the first difficulty is that Moitra's method relies on the fact that the marginal probability of each variable (the probability that the variable is true in a uniformly-chosen satisfying assignment) is nearly 1/2. This is necessary because Moitra's method involves solving a certain linear program (LP) and the size of this LP is polynomially-bounded only if a certain process couples quickly. The proof that the process couples quickly relies on the fact that the marginals are nearly 1/2 (and certainly on the fact that they are bounded away from 0 and 1). In contrast, for a random k-CNF formula, although the *average* degree of variables is low, w.h.p. there are variables with degrees as high as $\Omega(\log n/\log\log n)$. In the presence of these high-degree variables, the marginal probabilities of the variables can be arbitrarily near 0 or 1, instead of 1/2.

Our solution to this issue is to separate out high-degree variables, as well as those that are heavily influenced by high-degree variables. To do this, we define a process to recursively label "bad" variables. At the start, all high-degree variables are bad. Then, all clauses containing more than $k/10$ bad variables are labelled bad, as are all variables that they contain. We run this process until no more bad clauses are found. We call the remaining variables and clauses of the formula "good". A key property is that all good variables have an upper bound on their degree and all good clauses contain at least $9k/10$ good variables; this allows us to show that the marginal probabilities of good variables are close to 1/2.

The next step is to attempt to apply Moitra's method. The goal of Moitra's method is to compute more precise estimates for the marginal probabilities of the variables; given accurate estimates on the marginal probabilities, it is then relatively easy to approximate the number of satisfying assignments using refined self-reducibility techniques.

Of course, we need to modify the method to deal with the bad variables, which still appear in the formula. We first explain Moitra's method and then proceed with our modifications. The first step is to mark variables, so that every clause contains a good fraction of marked variables and a good fraction of unmarked variables. Then, for a particular marked variable

v, we set up an LP. As noted earlier, the variables of the LP correspond to the states of a certain coupling process which couples two distributions on satisfying assignments using the marked variables – the first distribution over satisfying assignments in which v is true, and the second distribution over satisfying assignments in which v is false. Solving the LP recovers the transition probabilities of the coupling process and yields enough information to approximate the marginal probability of v.

In order to guarantee that the size of the LP is bounded by a polynomial in the size of the original CNF formula, we have to restrict the coupling process. The process can be viewed as a tree and it suffices to truncate this tree at a suitable level.

Thus, a crucial part of the proof (both in Moitra's case and in ours) is to show that the error caused by the truncation is sufficiently small. The reason that the error caused by the truncation is small is that, with high probability, branches of the coupling tree "die out" before reaching a large level. The reason for this is that the marginals of marked variables stay near $1/2$, even when conditioning on partial assignments.

In our case where Φ is a random formula, the marginals are not all near $1/2$, even without any conditioning. But the good variables do have marginals near $1/2$. So we only mark/unmark good variables and we "give up" on bad variables. Given that we don't have any control over the bad variables, we have to modify the coupling process. Thus, whenever we meet a bad variable in the coupling process, we have to assume the worst case and treat this variable and all bad variables connected to it as if they all have failed the coupling, meaning that the disagreement spreads quickly over bad components.

The most important part of our analysis is to upper bound the size of connected bad components and how often we encounter them during the coupling processs. Given these upper bounds, we are able to show that the coupling still dies out sufficiently quickly, so the error caused by the truncation is not too large. Solving the LP then allows us to estimate the marginals of the good variables. Given that the bad components have small size, this turns out to be enough information to estimate the number of satisfying assignments of the original formula (containing both good and bad clauses).

We conclude this summary by discussing the prospects for improving our work. Although we have given an efficient algorithm which works for densities that are exponentially large in k, the densities that we can handle are still small compared to the satisfiability threshold or to the threshold under which efficient search algorithms exist. Perhaps a modest start towards obtaining comparable thresholds for approximate counting algorithms would be to consider models whose state spaces are connected. For example, for monotone k-CNF formulas where each variable appears in at most d clauses, Hermon et al. [23] showed that efficient randomised algorithms exist if $d \le c2^{k/2}$ for some constant $c > 0$, which is optimal up to the constant c due to complementing hardness results [6]. They also showed that the same algorithm works for *random regular* monotone k-CNF formulas, if the degree $d \le c2^k/k$ for some $c > 0$. It remains open whether an average case bound of the same order can be achieved for random monotone k-CNF formulas.

2 The coupling tree

2.1 Identifying bad variables

We start by identifying bad variables; the method that we use is inspired by [12].

▶ **Definition 2.** *Let Φ be a k-SAT formula. We say that a variable v of Φ is* high-degree *if Φ contains at least $\Delta := 2^{k/300}$ occurrences of literals involving the variable v.*

The reason that high-degree variables are harmful is that their marginal probabilities (when we sample uniformly from satisfying assignments) are not bounded away from 0 and 1. Also, any variable that shares clauses with high-degree variables may also have biased marginals. In our algorithm, we will not be able to control these high degree variables or other variables that are affected by them. This variables contribute to the "bad" part of the formula Φ. Formally, denote the set of clauses of Φ by $\mathcal{C}$ and the set of variables by $\mathcal{V}$. For each $c \in \mathcal{C}$, let $\mathsf{var}(c)$ denote the set of variables in c. For each subset C of $\mathcal{C}$, let $\mathsf{var}(C) := \cup_{c \in C}\mathsf{var}(c)$. The *bad variables* and *bad clauses* of Φ are identified as follows:

1. $\mathcal{V}_0$ (the initial bad variables) $\leftarrow$ the set of high-degree variables;
2. $\mathcal{C}_0 \leftarrow$ the set of clauses with at least $k/10$ variables in $\mathcal{V}_0$;
3. $i \leftarrow 0$;
4. Do the following until $\mathcal{V}_i = \mathcal{V}_{i-1}$:
   - $i \leftarrow i+1$;
   - $\mathcal{V}_i \leftarrow \mathcal{V}_{i-1} \cup \mathsf{var}(\mathcal{C}_{i-1})$;
   - $\mathcal{C}_i \leftarrow \{c \in \mathcal{C} \mid \mathsf{var}(c) \cap \mathcal{V}_i \geq k/10\}$;
5. $\mathcal{C}_{\text{bad}} \leftarrow \mathcal{C}_i$ and $\mathcal{V}_{\text{bad}} \leftarrow \mathcal{V}_i$;
6. $\mathcal{C}_{\text{good}} \leftarrow \mathcal{C} \setminus \mathcal{C}_i$ and $\mathcal{V}_{\text{good}} \leftarrow \mathcal{V} \setminus \mathcal{V}_i$.

▶ **Observation 3.** *$\forall c \in \mathcal{C}_{good}$, $|\mathsf{var}(c) \cap \mathcal{V}_{bad}| < k/10$. $\forall c \in \mathcal{C}_{bad}$, $|\mathsf{var}(c) \cap \mathcal{V}_{good}| = 0$.*

2.2 Marking good variables and identifying a satisfying assignment

Apart from the fact that we only mark variables in $\mathcal{V}_{\text{good}}$, our marking follows the approach of Moitra [30]. Formally, a "marking" is an assignment from $\mathcal{V}_{\text{good}}$ to $\{\text{marked}, \text{unmarked}\}$. Using Observation 3 and applying the asymmetric version of the Lovász local lemma [17, 36, 22] and the algorithmic version of the local lemma by Moser and Tardos [32] it is easy to prove the following lemma.

▶ **Lemma 8.** *There exists a marking on $\mathcal{V}_{good}$ such that every good clause has at least $3k/10$ marked variables and at least $k/4$ unmarked good variables. It has the property that there is a partial assignment of bad variables that satisfies all bad clauses. Furthermore, such a marking can be found in deterministic polynomial time.*

We also use the Lovász local lemma to identify a partial assignment Λ^* that we will use to apply self-reducibility.

▶ **Lemma 10.** *Let $\Phi = \Phi(k, n, m)$ and let $v_1, v_2, \ldots, v_n$ be the variables of Φ. In each clause, order the literals in the order induced by the indices of their variables. Then there is a partial assignment Λ^* of truth values to some subset of $\mathcal{V}_{\mathsf{marked}}$ with the property that every clause $c \in \mathcal{C}_{good}$ is satisfied by its first $k/20$ literals corresponding to marked variables. Moreover, Λ^* can be found in deterministic polynomial time.*

2.3 The coupling tree

Fix a prefix Λ of the assignment Λ^* from Lemma 10. Let Φ^Λ be the formula produced by simplifying Φ under Λ (remove clauses that are satisfied under Λ and remove all false literals). $\mathcal{C}^\Lambda$ denotes the clauses of Φ^Λ and $\mathcal{V}^\Lambda$ denotes the variables. We also define $\mathcal{V}^\Lambda_{\text{good}} = \mathcal{V}_{\text{good}} \cap \mathcal{V}^\Lambda$ and $\mathcal{C}^\Lambda_{\text{good}} = \mathcal{C}_{\text{good}} \cap \mathcal{C}^\Lambda$. Ω^Λ denotes the set of satisfying assignments of Φ^Λ.

For a variable $v^* \in \mathcal{V}^\Lambda$, let Ω^Λ_1 be the set of assignments in Ω^Λ in which v^* is true, and let Ω^Λ_2 be the set of assignments in Ω^Λ in which v^* is false. The algorithm estimates the marginal probability that v^* is true by solving a certain LP which allows it to estimate the ratio $|\Omega_1^\Lambda|/|\Omega_2^\Lambda|$. The variables of the LP correspond to the states of a coupling process. The process couples the uniform distribution on Ω_1^Λ with the uniform distribution on Ω_2^Λ. We can now describe process via its "coupling tree" $\mathbb{T}^\Lambda$.

For each node ρ there is a partial assignment $\mathcal{A}_1(\rho) \in \Omega_1^\Lambda$ and a partial assignment $\mathcal{A}_2(\rho) \in \Omega_2^\Lambda$. The variables set in these partial assignments are $\Lambda \cup V_{\mathrm{set}}(\rho)$. The set $V_I(\rho)$ contains "problematic" variables. The details will be clear later. Roughly, these include variables in $V_{\mathrm{set}}(\rho)$ on which $\mathcal{A}_1(\rho)$ and $\mathcal{A}_2(\rho)$ disagree, variables contained in clauses that are not satisfied in some $\mathcal{A}_i(\rho)$, even though all marked variables have already been set, and variables "affected" by bad variables during the coupling process. $\mathcal{C}_{\mathrm{rem}}(\rho)$ is the set of remaining clauses to consider at descendants of ρ in the coupling.

The root of the coupling tree is the node ρ^* with $V_{\mathrm{set}}(\rho^*) = V_I(\rho^*) = \{v^*\}$. The assignment $\mathcal{A}_1(\rho^*)$ sets v^* to T and the assignment $\mathcal{A}_2(\rho^*)$ sets v^* to F. $\mathcal{C}_{\mathrm{rem}}(\rho^*) = \mathcal{C}^\Lambda$. Let $n = |\mathcal{V}|$. In order to ensure that the size of the LP is bounded by a polynomial in n we need to ensure that the size of the coupling tree is also bounded by a polynomial in n. To do this, we choose truncation depth $L := C_0(3k^2\Delta)\lceil\log(n/\varepsilon)\rceil$ where C_0 is a sufficiently large constant. We then truncate the tree as follows.

▶ **Definition 12.** *A node ρ of the coupling tree is* a leaf *if $|V_I(\rho)| \le L$ and every $c \in \mathcal{C}_{rem}(\rho)$ has the property that $\mathsf{var}(c) \subseteq V_I(\rho) \cup V_{set}(\rho)$ or $\mathsf{var}(c) \subseteq \mathcal{V}^\Lambda \setminus (V_I(\rho) \cup V_{set}(\rho))$. If $|V_I(\rho)| > L$, then ρ is* a truncating node. *We denote the set of leaves by $\mathcal{L}$, the set of truncating nodes by $\mathcal{T}$, and their union by $\mathcal{L}^* := \mathcal{L} \cup \mathcal{T}$.*

If ρ is not in $\mathcal{L}^*$ then we define its four children as follows. The "first clause" of ρ is the first good clause c with a variable in $V_I(\rho)$ and a variable in $\mathcal{V}^\Lambda \setminus V_I(\rho)$. (The definitions imply that such a clause exists.) The "first variable" u of ρ is the first (good) variable in $\mathsf{marked}(c) \setminus V_{\mathrm{set}}(\rho)$. For each of the four pairs (τ_1, τ_2) where τ_1 and τ_2 are assignments from $\{u\}$ to $\{\mathsf{T}, \mathsf{F}\}$, we create a child ρ_{τ_1,τ_2} of ρ using the following algorithm.

Algorithm 1 Constructing the child ρ_{τ_1,τ_2} of a non-truncating node ρ of the coupling tree, where τ_1, τ_2 are assignments from $\{u\}$ to $\{\mathsf{T}, \mathsf{F}\}$, and u is the first variable of ρ.

```
$V_{\mathrm{set}}(\rho_{\tau_1,\tau_2}) \leftarrow V_{\mathrm{set}}(\rho) \cup \{u\}$;
$\mathcal{A}_1(\rho_{\tau_1,\tau_2}) \leftarrow$ combine $\mathcal{A}_1(\rho)$ with $\tau_1$;
$\mathcal{A}_2(\rho_{\tau_1,\tau_2}) \leftarrow$ combine $\mathcal{A}_2(\rho)$ with $\tau_2$;
$(V_I, \mathcal{C}_{\mathrm{rem}}) \leftarrow (V_I(\rho), \mathcal{C}_{\mathrm{rem}}(\rho))$;
if $\tau_1(u) \neq \tau_2(u)$ then
    $V_I \leftarrow V_I \cup \{u\}$;
end if
for $c' \in \mathcal{C}_{\mathrm{rem}}$ s.t. $c'$ is satisfied by both $\mathcal{A}_1(\rho_{\tau_1,\tau_2})$ and $\mathcal{A}_2(\rho_{\tau_1,\tau_2})$ do
    $\mathcal{C}_{\mathrm{rem}} \leftarrow \mathcal{C}_{\mathrm{rem}} \setminus \{c'\}$;
end for
while $\exists c' \in \mathcal{C}_{\mathrm{rem}}$ with $\mathsf{var}(c') \cap V_I \neq \emptyset$, $\mathsf{var}(c') \cap (\mathcal{V}^\Lambda \setminus V_I) \neq \emptyset$, and $\mathsf{marked}(c') \setminus V_{\mathrm{set}}(\rho_{\tau_1,\tau_2}) = \emptyset$ do
    $V_I \leftarrow V_I \cup (\mathsf{var}(c') \setminus V_{\mathrm{set}}(\rho_{\tau_1,\tau_2}))$;
    $\mathcal{C}_{\mathrm{rem}} \leftarrow \mathcal{C}_{\mathrm{rem}} \setminus \{c'\}$;
end while
while $\exists c' \in \mathcal{C}_{\mathrm{rem}} \cap \mathcal{C}_{\mathrm{bad}}$ with $\mathsf{var}(c') \cap V_I \neq \emptyset$ do
    $V_I \leftarrow V_I \cup (\mathsf{var}(c') \setminus V_{\mathrm{set}}(\rho_{\tau_1,\tau_2}))$;
    $\mathcal{C}_{\mathrm{rem}} \leftarrow \mathcal{C}_{\mathrm{rem}} \setminus \{c'\}$;
end while
$(V_I(\rho_{\tau_1,\tau_2}), \mathcal{C}_{\mathrm{rem}}(\rho_{\tau_1,\tau_2})) \leftarrow (V_I, \mathcal{C}_{\mathrm{rem}})$;
```

2.4 Key property of the coupling tree for a random formula

Recall that the variables of the LP which is used to estimate the marginal of the variable v^* of Φ^Λ correspond to the states of the coupling on the coupling tree $\mathbb{T}^\Lambda$. We will define two LP variables $P_{1,\rho}$ and $P_{2,\rho}$ for each node ρ of $\mathbb{T}^\Lambda$. In order to efficiently solve the LP, we need its size to be bounded by a polynomial in n, so we need the number of nodes of $\mathbb{T}^\Lambda$ to be bounded by a polynomial in n. For a random formula, this follows from the following key lemma, which is a main technical contribution of our work.

▶ **Lemma 14.** *W.h.p. over the choice of Φ, for every prefix Λ of Λ^*, every node ρ in $\mathbb{T}^\Lambda$ has the property that $|V_{set}(\rho)| \leq 3k^3\alpha L + 1$.*

To see that Lemma 14 implies that the size of the coupling tree is at most a polynomial in n, note that the depth of the tree does not exceed $\max_{\rho\in\mathbb{T}^\Lambda} |V_{\text{set}}(\rho)| \leq 3k^3\alpha L + 1 = O(\log \frac{n}{\varepsilon})$. Also, each node has at most 4 children.

In the rest of this section, we sketch the proof of Lemma 14. We start by defining some graphs associated with Φ. The formula Φ naturally corresponds to a bipartite "factor graph" where one side is variables and the other clauses (a variable has an edge to a clause in the factor graph if one its literals is contained in the clause). We also use the following two graphs.

▶ **Definition 3.** *Let G_Φ be the graph with vertex set $\mathcal{C}$ (the clauses of Φ) in which two clauses c and c' are adjacent if and only if $\mathsf{var}(c) \cap \mathsf{var}(c') \neq \emptyset$. We say that a set $C \subseteq \mathcal{C}$ of clauses is connected if the induced subgraph $G_\Phi[C]$ is connected.*

▶ **Definition 4.** *Let H_Φ be the graph with vertex set $\mathcal{V}$ (the variables of Φ) in which two variables v and v' are adjacent if and only if there exists a clause $c \in \mathcal{C}$ such that $v, v' \in \mathsf{var}(c)$. We say that a set $V \subseteq \mathcal{V}$ of variables is connected if the induced subgraph $H_\Phi[V]$ is connected. Let $H_{\Phi,bad}$ be the graph with vertex set $\mathcal{V}_{bad}$ in which two variables v and v' are adjacent if and only if there exists a bad clause $c \in \mathcal{C}_{bad}$ such that $v, v' \in \mathsf{var}(c)$. We say that a set $V \subseteq \mathcal{V}$ of variables is a* bad component *if V is a connected component in $H_{\Phi,bad}$.*

For $V \subseteq \mathcal{V}$, let $\Gamma_{H_\Phi}(V) = \cup_{v\in V}\Gamma_{H_\Phi}(v)$ be the neighbourhood of V in H_Φ. Let $\Gamma^+_{H_\Phi}(V) = V \cup \Gamma_{H_\Phi}(V)$ be the extended neighbourhood. The proof of Lemma 14 relies on the following rather abstract fact about random formulas.[1]

▶ **Lemma 41.** *W.h.p. over the choice of Φ, there do not exist sets Y', Z of clauses and a set V of variables such that $|Y'| \geq \log n$, $|V| \geq |Y'|$, $|Z| \geq 2k^2\alpha|V|$, $Y' \cap Z = \emptyset$, $G_\Phi[Y']$ is connected, $V \subseteq \mathsf{var}(Y')$, and every clause in Z contains at least one variable from V.*

The lemma says that if you take any "large" set of clauses Y' that are connected in G_Φ and any large set V of the variables of Y' then there aren't many clauses outside of Y' that contain variables in V. (There isn't a large set Z of such clauses.) Obviously, the lemma doesn't apply to every Φ, but is highly dependent on the random way in which Φ is chosen. The proof of Lemma 41 relies crucially on upper-bounding the probability that a set of clauses Y' is connected in G_Φ. To do this, we sum over possible trees connecting the clauses in Y'. We use the bound from Lemma 39 of the full version, which shows that the probability that any particular tree T is connected in G_Φ is at most $(k^2/n)^{|V(T)|-1}$.

[1] We need a more general version in the full paper, but this suffices here. The variable names here are chosen to make the (single) application in this short version easy.

Proof of Lemma 14. Let Λ be a prefix of Λ^* and let ρ be a node in $\mathbb{T}^\Lambda$. Our goal is to prove $|V_{\text{set}}(\rho)| \leq 3k^3\alpha L + 1$. We first consider the case in which ρ is not a truncating node, so $|V_I(\rho)| \leq L$ and we show $|V_{\text{set}}(\rho)| \leq 3k^3\alpha L$. The proof has two parts.

Part 1. $V_{\text{set}}(\rho) \subseteq \Gamma^+_{H_\Phi}(V_I(\rho))$.

To prove Part 1, we consider any $u \in V_{\text{set}}(\rho) \setminus V_I(\rho)$ and show that there is a clause c containing u and containing a variable in $V_I(\rho)$.

We first rule out the case that $u = v^*$ by noting (from the construction of the coupling tree) that $v^* \in V_I(\rho) \cap V_{\text{set}}(\rho)$.

So consider $u \in V_{\text{set}}(\rho) \setminus V_I(\rho)$ and let ρ' be the ancestor of ρ in the coupling tree such that u is the first variable of ρ'. The definition of the coupling tree guarantees that ρ' is uniquely defined and that it is a proper ancestor of ρ – the definition of "first variable" guarantees that $u \notin V_{\text{set}}(\rho')$, but for all proper descendants ρ''' of ρ', $u \in V_{\text{set}}(\rho''')$.

Let ρ'' be the child of ρ' on the path to ρ. We will show that there is a clause c containing u and containing a variable in $V_I(\rho')$. Part 1 then follows from the fact that $V_I(\rho)$ contains $V_I(\rho')$. The existence of such a clause c is immediate from the definition of "first variable" – indeed c is the "first clause" of ρ'.

Part 2. W.h.p., the random formula Φ is such that $\forall \rho, |\Gamma^+_{H_\Phi}(V_I(\rho))| \leq 3k^3\alpha L$.

For Part 2, it is important that the set $V_I(\rho)$ is connected in H_Φ – this follows from the construction of the coupling tree. We show (this is Lemma 51) that, w.h.p. over the choice of Φ, *every* connected set of variables $V \subseteq \mathcal{V}$ satisfies

$$|\Gamma^+_{H_\Phi}(V)| \leq 3k^3\alpha \max\{|V|, k \log n\}, \tag{1}$$

which establishes Part 2 since $|V_I(\rho)| \leq L$.

The proof of (1) is as follows. Let V be a connected of variables and let Y be the set of neighbours of V in the factor graph of Φ, i.e., $Y = \{c \in \mathcal{C} \mid \mathsf{var}(c) \cap V \neq \emptyset\}$. Clearly $|\Gamma^+_{H_\Phi}(V)| \leq k|Y|$ and hence it suffices to show that $|Y| \leq 3k^2\alpha \max\{|V|, k \log n\}$. There are two cases depending on the size of V.

- $|V| \geq k \log n$. Since V is a connected set of variables, there exists a set $Y' \subseteq Y$ such that $|V|/k \leq |Y'| \leq |V|$ and $V \cup Y'$ is connected in the factor graph of Φ. Hence, Y' is a connected set of clauses and $|Y'| \geq \log n$. Let $Z = Y \setminus Y'$. If $|Z| \geq 2k^2\alpha |V|$ then we obtain a contradiction to Lemma 41, which holds w.h.p. Thus, w.h.p., $|Z| \leq 2k^2\alpha |V|$ which implies $|Y| = |Y'| + |Z| \leq 3k^2\alpha |V|$, as required.
- Otherwise $|V| < k \log n$. If $|\Gamma^+_{H_\Phi}(V)| < \lceil k \log n \rceil$ then we are finished. Otherwise, consider an arbitrary connected $V' \supset V$ such that $|V'| = \lceil k \log n \rceil$. By the argument of the previous case, the set of neighbours of V' in the factor graph, denoted Y'', satisfies that $|Y''| \leq 3k^2\alpha |V'| \leq 3k^3\alpha \log n$. Thus, $|Y| \leq |Y''| \leq 3k^3\alpha \log n$.

This completes the proof of (1), and hence Part 2.

To finish, we consider the case where ρ is a truncating node. Let ρ' be the parent of ρ. Parts 1 and 2 imply that $|V_{\text{set}}(\rho')| \leq 3k^3\alpha L$. The result follows since $|V_{\text{set}}(\rho)| = |V_{\text{set}}(\rho')| + 1$. ◀

3 The linear program

Here we briefly list the constraints in the LP so that we can discuss its analysis. For a node ρ of the coupling tree, let $\mathcal{C}_I(\rho)$ be the set of clauses $c \in \mathcal{C}^\Lambda$ such that $\mathsf{var}(c) \subseteq V_I(\rho) \cup V_{\text{set}}(\rho)$. For $i \in \{1, 2\}$, let $N_i(\rho)$ be the number of assignments τ to $V_I(\rho) \setminus V_{\text{set}}(\rho)$ such that every clause in $\mathcal{C}_I(\rho)$ is satisfied by $\tau \cup \mathcal{A}_i(\rho)$. It turns out (see Lemma 15) that $N_i(\rho) \neq 0$ for $i \in \{1, 2\}$, so we define $r(\rho) = N_1(\rho)/N_2(\rho)$.

The LP relies on two constants r_{lower} and r_{upper}. The algorithm that uses the LP will move these closer and closer together by binary search. For each node ρ of the coupling tree, we introduce two variables $P_{1,\rho}$ and $P_{2,\rho}$. The constraints are as follows. **Constraint Set 0**: For every node ρ of the coupling tree and every $i \in \{1,2\}$ we add the constraint $0 \leq P_{i,\rho} \leq 1$. **Constraint Set 1**: If $\rho \in \mathcal{L}$ then we add the constraint $r_{\mathsf{lower}}\, P_{2,\rho} \leq P_{1,\rho}\, r(\rho)$ and the constraint $P_{1,\rho}\, r(\rho) \leq r_{\mathsf{upper}}\, P_{2,\rho}$. **Constraint Set 2**: For the root ρ^* of the coupling tree, we add the constraints $P_{1,\rho^*} = 1$ and $P_{2,\rho^*} = 1$. For every node ρ of the coupling tree that is not in $\mathcal{L}^*$, let u be the first variable of ρ. For each $X \in \{\mathsf{T},\mathsf{F}\}$ add the constraints $P_{1,\rho} = P_{1,\rho_{u\to X, u\to \mathsf{T}}} + P_{1,\rho_{u\to X, u\to \mathsf{F}}}$ and $P_{2,\rho} = P_{2,\rho_{u\to \mathsf{T}, u\to X}} + P_{2,\rho_{u\to \mathsf{F}, u\to X}}$. **Constraint Set 3**: For every node ρ of the coupling tree that is not in $\mathcal{L}^*$, every $X \in \{\mathsf{T},\mathsf{F}\}$, and every $i \in \{1,2\}$, let u be the first variable of ρ and add the constraint $P_{i,\rho_{u\to X, u\to\neg X}} \leq \frac{1}{s} P_{i,\rho}$.

4 Analysis of the linear program for a random formula and how it enables us to conclude Theorem 1

The key lemmas demonstrating the purpose of the linear program are as follows.

▶ **Lemma 24.** *Suppose $r_{\mathsf{lower}} \leq |\Omega_1^\Lambda|/|\Omega_2^\Lambda| \leq r_{\mathsf{upper}}$. There is a set of variables $\mathbf{P} = \{P_{i,\rho}\}$ that satisfies all constraints of the LP.*

▶ **Lemma 34.** *Fix $r_{\mathsf{lower}} \leq r_{\mathsf{upper}}$. W.h.p. over the choice of Φ, the following holds. If the LP has a solution $\mathbf{P}$ using r_{lower} and r_{upper}, then $e^{-\varepsilon/(3n)} r_{\mathsf{lower}} \leq |\Omega_1^\Lambda|/|\Omega_2^\Lambda| \leq e^{\varepsilon/(3n)} r_{\mathsf{upper}}$.*

The full version proves Theorem 1 using these two lemmas. Here we just give the main idea. First, consider the sub-goal of estimating $|\Omega_1^\Lambda|/|\Omega_2^\Lambda|$ given Φ and a partial assignment Λ of Λ^*. We can do this with accuracy $\exp(\pm\varepsilon/n)$ using the linear program. The proof of Lemma 57 in the full version uses the Lovász local lemma to establish values for r_{lower} and r_{upper} that meet the conditions in Lemma 24. Then, by binary search we bring r_{lower} and r_{upper} closer together until we achieve the desired accuracy (by Lemma 34). The initial values of r_{lower} and r_{upper} guarantee (see the proof of Lemma 57 for details) that the LP is run at most $O(\log(n/\varepsilon))$ times. Since we have already shown that the size of the LP is bounded by a polynomial in n/ε the algorithm runs in polynomial time.

Now consider the proof of Theorem 1. Using standard self-reducibility, we can use the estimates that we have just established to obtain an accurate estimate (within $\exp(\pm\varepsilon)$) of $|\Omega^{\Lambda^*}|/|\Omega|$, which is the probability that a random satisfying assignment is consistent with Λ^*.

To finish we need one last key ingredient – we need a method to estimate $|\Omega^{\Lambda^*}|$. Since all good clauses are satisfied by Λ^*, the set $\mathcal{C}^{\Lambda^*}$ of clauses of Φ^{Λ^*} consists only of bad clauses. Now we need one more key lemma.

▶ **Lemma 48.** *W.h.p. over the choice of Φ, every bad component S has size at most $21600k\log n$.*

Lemma 48 implies that $\mathcal{C}^{\Lambda^*}$ can be divided into disjoint subsets where each subset of clauses contains $O(\log n)$ variables. The algorithm can then compute the number of satisfying assignments of each subset by brute force in time *poly*(n). Then $|\Omega^{\Lambda^*}|$ is the product of these numbers.

This concludes the sketch of the proof of Theorem 1 – the details are in the full version. In the rest of this short version, we briefly discuss the proof of the remaining key lemmas, Lemmas 48, 34, and 24.

We start with the proof of Lemma 48. This lemma, which bounds the size of bad components, is one of the main technical achievements allowing us to extend Moitra's method to random CNFs with high density. Here we only have room for a very rough sketch. Recall that a bad component is a set S of variables that is connected in $H_{\Phi,\text{bad}}$. Let $\mathsf{HD}(S) = \mathcal{V}_0 \cap S$ be the set of high-degree variables in S. We wish to show that w.h.p., over the choice of Φ, every bad component S has size at most $21600k \log n$. This follows from the following two lemmas, which give a contradiction for large bad components S.

▶ **Lemma 42.** *W.h.p. over the choice of Φ, every connected set U of variables with size at least $21600k \log n$ satisfies that $|\mathsf{HD}(U)| \leq \frac{|U|}{21600}$.*

▶ **Lemma 47.** *W.h.p. over the choice of Φ, for any bad component S, $|S| \leq 60\,|\mathsf{HD}(S)|$.*

The proof of Lemma 42 is deferred to the full version. It uses Lemma 41 and studies trees in the factor graph of Φ. The following proof sketch concludes the proof of Lemma 48.

Proof Sketch of Lemma 47. Consider the following process P which we will use to work with bad compoments. The process, for every set S of variables, defines a set of variables $\mathsf{BC}(S)$.

```
1. Let $\mathsf{BC}(S) = S$.
2. While there is a clause $c$ such that $|\mathsf{var}(c) \cap \mathsf{BC}(S)| \geq k/10$ and $\mathsf{BC}(S) \setminus \mathsf{var}(c) \neq \emptyset$
     $\mathsf{BC}(S) := \mathsf{BC}(S) \cup \mathsf{var}(c)$
```

Note that $\mathcal{V}_{\text{bad}} = \mathsf{BC}(\mathcal{V}_0)$, where $\mathcal{V}_0$ is the set of high-degree variables. We show (Lemma 43) that for every bad component S, we have $S = \mathsf{BC}(\mathsf{HD}(S))$. Thus, the process P can be viewed as a "local" process for identifying bad components.

Let S be a bad component. If S contains only an isolated variable, it must be a high-degree variable and hence $\mathsf{HD}(S) = S$ (so we are finished). Otherwise, since a bad component is a connected component of variables in $H_{\Phi,\text{bad}}$, the definition of $H_{\Phi,\text{bad}}$ ensures that the bad component has at least $k/10$ high-degree variables.

Note that $|\mathsf{HD}(S)| \leq |\mathcal{V}_0|$. In Lemma 35 of the full version we use Poisson estimates for the degrees of the variables to show that, w.h.p., $|\mathcal{V}_0| \leq n/2^{k^{10}}$.

The next step is to apply a counting argument to show that, w.h.p., for *every* set of variables Y such that $2 \leq |Y| \leq n/2^k$, the number of clauses that contain at least $k/10$ variables from Y is at most $\frac{30}{k}|Y|$. This is Corollary 38 of the full version. We apply the corollary with $Y = \mathsf{HD}(S)$, so we find that there are at most $\frac{30}{k}\,|\mathsf{HD}(S)|$ clauses that contain at least $k/10$ variables from $\mathsf{HD}(S)$.

Now, we run the process P starting with $\mathsf{HD}(S)$. Take Z to be the set of clauses that contain at least $k/10$ variables from $\mathsf{HD}(S)$ (so, from above, we have $|Z| \leq \frac{30}{k}\,|\mathsf{HD}(S)| \leq \frac{30}{k}\frac{n}{2^{k^{10}}}$).

The next step is to show that, w.h.p., the number of clauses c such that $\mathsf{var}(c) \subseteq \mathsf{BC}(\mathsf{HD}(S))$ is at most $2|Z|$ (which we have already shown to be at most $60\,|\mathsf{HD}(S)|\,/k$). This analysis is contained in Corollary 45. It is essentially an analysis of the process P which follows easily from a lemma of Coja-Oghlan and Frieze [10, Lemma 2.4]. The high-probability guarantees are universal over Z (hence universal over S).

Since $S = \mathsf{BC}(\mathsf{HD}(S))$ and each variable in S is contained in some bad clause, we have

$$|S| \leq \left| \bigcup_{c \in \mathcal{C}_{\text{bad}}:\ \mathsf{var}(c) \cap S \neq \emptyset} \mathsf{var}(c) \right| \leq 60\,|\mathsf{HD}(S)|\,, \text{ as required.}$$ ◀

We now turn to the proof of Lemma 34. There are two kinds of errors which cause solutions of the LP to differ from the ratio $\left|\Omega_1^\Lambda\right|/\left|\Omega_2^\Lambda\right|$. The first kind comes from so-called "ℓ-wrong assignments" and the second kind comes from the truncation of the coupling tree. To define these more precisely, we need some graph-theoretic notation.

▶ **Definition 25.** *Given a graph G and any positive integer k, let $G^{\leq k}$ be the graph with vertex set $V(G)$ in which vertices u and v are connected iff there is a path from u to v in G of length at most k.*

The main combinatorial structure that we use is a set $\mathcal{D}(G_\Phi)$, which is based on Alon's "2,3-tree" [5]. Similar structures were subsequently used in [30, 21]. The main difference between our definition and previous ones is that we take into account whether clauses are connected via good variables.

▶ **Definition 26.** *Given G_Φ, let $\mathcal{D}(G_\Phi)$ be the set of subsets $T \subseteq V(G_\Phi)$ such that (1) For any $c_1, c_2 \in T$, $\mathsf{var}(c_1) \cap \mathsf{var}(c_2) \cap \mathcal{V}_{good} = \emptyset$; and (2) The graph $G_\Phi^{\leq 4}[T]$, which is the subgraph of $G_\Phi^{\leq 4}$ induced by T, is connected.*

▶ **Definition 29.** *An assignment $\sigma \in \Omega_i^\Lambda$ is ℓ-wrong if $\exists$ a size-ℓ set $T \in \mathcal{D}(G_\Phi)$ such that $c^* \in T$, $|T \cap \mathcal{C}_{good}^\Lambda| \geq 2\,|T|/3$, and there is a size $\lceil \ell/2 \rceil$ subset S of $T \cap \mathcal{C}_{good}^\Lambda$ such that the restriction of σ to marked variables in clauses in S does not satisfy any clause in S. Otherwise σ is ℓ-correct.*

Proof Sketch of Lemma 34. The constraints in **Constraint Set 2** guarantee (see Lemma 18 of the full version) that, for any $i \in \{1,2\}$ and $\sigma \in \Omega_i^\Lambda$, $\sum_{\rho \in \mathcal{L}^*:\sigma\in\Omega^{\mathcal{A}_i(\rho)\cup\Lambda}} P_{i,\rho} = 1$. Thus, $|\Omega_i^\Lambda| = \sum_{\sigma\in\Omega_i^\Lambda} 1 = \sum_{\sigma\in\Omega_i^\Lambda} \sum_{\rho\in\mathcal{L}^*:\sigma\in\Omega^{\mathcal{A}_i(\rho)\cup\Lambda}} P_{i,\rho}$. Let $\ell = L/(3k^2\Delta)$. We start by defining Z_i, Z_i' and Z_i'' as follows for $i \in \{1,2\}$.

$$Z_i = \sum_{\sigma\in\Omega_i^\Lambda} \sum_{\rho\in\mathcal{L}:\sigma\in\Omega^{\mathcal{A}_i(\rho)\cup\Lambda}} P_{i,\rho},$$

$$Z_i' = \sum_{\sigma\in\Omega_i^\Lambda,\ \sigma \text{ is } \ell\text{-wrong}} \sum_{\rho\in\mathcal{L}^*:\sigma\in\Omega^{\mathcal{A}_i(\rho)\cup\Lambda}} P_{i,\rho},$$

$$Z_i'' = \sum_{\sigma\in\Omega_i^\Lambda,\ \sigma \text{ is } \ell\text{-correct}} \sum_{\rho\in\mathcal{T}:\sigma\in\Omega^{\mathcal{A}_i(\rho)\cup\Lambda}} P_{i,\rho}.$$

Thus $Z_i \leq \left|\Omega_i^\Lambda\right| \leq Z_i + Z_i' + Z_i''$. The full version proves

$$r_{\mathsf{lower}} \leq Z_1/Z_2 \leq r_{\mathsf{upper}}. \tag{2}$$

$$Z_i'/|\Omega_i^\Lambda| \leq (1-e^{-\varepsilon/(3n)})/2 \text{ for } i \in \{1,2\}. \tag{3}$$

$$Z_i''/|\Omega_i^\Lambda| \leq (1-e^{-\varepsilon/(3n)})/2 \text{ for } i \in \{1,2\}. \tag{4}$$

The lemma follows easily from these. Combining (3) and (4) with the fact that $Z_i \leq \left|\Omega_i^\Lambda\right| \leq Z_i + Z_i' + Z_i''$, we get $e^{-\varepsilon/(3n)} \leq \frac{Z_i}{|\Omega_i^\Lambda|} \leq 1$. Plugging in (2) we obtain the result.

To prove (2) we exchange the order of summation in the definition of Z_i to get

$$Z_i = \sum_{\rho\in\mathcal{L}} \sum_{\sigma\in\Omega_i^\Lambda:\sigma\in\Omega^{\mathcal{A}_i(\rho)\cup\Lambda}} P_{i,\rho} = \sum_{\rho\in\mathcal{L}} P_{i,\rho} \cdot |\Omega^{\mathcal{A}_i(\rho)\cup\Lambda}|.$$

Since $\rho \in \mathcal{L}$, we prove (see Lemma 17) that $r(\rho) = |\Omega^{\mathcal{A}_1(\rho)\cup\Lambda}|/|\Omega^{\mathcal{A}_2(\rho)\cup\Lambda}|$ (this is actually the point of $r(\rho)$). **Constraint Set 1** then guarantees that

$$r_{\mathsf{lower}} \leq \frac{P_{1,\rho} \cdot \left|\Omega^{\mathcal{A}_1(\rho)\cup\Lambda}\right|}{P_{2,\rho} \cdot \left|\Omega^{\mathcal{A}_2(\rho)\cup\Lambda}\right|} = \frac{P_{1,\rho} \cdot r(\rho)}{P_{2,\rho}\cdot} \leq r_{\mathsf{upper}}, \text{which suffices.}$$

The main ingredient in the proof of (3) is Lemma 30, which shows that the fraction of assignments in Ω_i^Λ that are ℓ-wrong is at most $(k\Delta)^{-9\ell}$. The main ingredient in the proof of (4) is showing that, w.h.p., for every ℓ-correct $\sigma \in \Omega_i^\Lambda$, $\sum_{\rho\in\mathcal{T}:\sigma\in\Omega^{\mathcal{A}_i(\rho)\cup\Lambda}} P_{i,\rho} \le (k\Delta)^{-8\ell}$. This is handled in Lemmas 32 and 33.

To prove these lemmas (say for $i=1$) we consider a sampling procedure for choosing a node $\rho \in \mathcal{L}^*$ conditioned on some $\sigma \in \Omega_1^\Lambda$. The probability that it reaches any node $\rho \in \mathcal{L}^*$ with $\sigma \in \Omega^{\mathcal{A}_1(\rho)\cup\Lambda}$ is designed to be $P_{1,\rho}$ so the goal is to bound the probability that it reaches the set $\Upsilon_\sigma = \{\rho \in \mathcal{T} \mid \sigma \in \Omega^{\mathcal{A}_1(\rho)\cup\Lambda}\}$. This is where the combinatorial structures that we have defined come in. We use $\mathcal{F}(\rho)$ to denote the set of clauses that "fail" in the coupling process, contributing variables to $V_I(\rho)$. Lemma 28 shows that w.h.p., for every node $\rho \in \Upsilon_\sigma$, there is a set $T \subseteq \mathcal{F}(\rho)$ containing the first clause c^* such that $T \in \mathcal{D}(G_\Phi)$, $|T| = \ell$ and $|T \cap \mathcal{C}_{\text{bad}}| \le |T|/3$. This implies that $|T \cap C_{\text{good}}^\Lambda| \ge 2|T|/3$. We therefore need to upper bound the probability that such a T is contained in $\mathcal{F}(\rho)$ when ρ is chosen from the sampling procedure. T has size ℓ and contains c^* and contains enough good clauses. So it turns out that, since σ is ℓ-correct, a lot of these failed clauses in $\mathcal{F}(\rho)$ must have failed due to disagreements in the coupling. Since $T \in \mathcal{D}(G_\Phi)$ these clauses do not share good variables. The constraints in **Constraint Set 3** then imply that the probability of all of these simultaneous disagreements is unlikely.

That concludes the proof, apart from proving the key Lemma 28. This again relies on properties about bad components - in particular on Lemma 50, which says that, w.h.p., for every connected set of clauses Y such that $|\mathsf{var}(Y)| \ge 21600k\log n$, it holds that $|Y \cap \mathcal{C}_{\text{bad}}| \le |Y|/12$. This is somewhat similar to the issues that we discussed regarding the proof of Lemma 48 – we defer the details to the full version. ◀

Proof Sketch of Lemma 24. Suppose $r_{\mathsf{lower}} \le |\Omega_1^\Lambda|/|\Omega_2^\Lambda| \le r_{\mathsf{upper}}$. Our goal is to show that there is a set of variables $\mathbf{P} = \{P_{i,\rho}\}$ that satisfies all constraints of the LP. Here is a suitable assignment. Let ρ be a node of the coupling tree with first variable u. For $X \in \{\mathsf{T}, \mathsf{F}\}$, we use the notation $\psi_{\rho,X,1} := |\Omega^{\mathcal{A}_1(\rho_{u\to X,u\to X})\cup\Lambda}|/|\Omega^{\mathcal{A}_1(\rho)\cup\Lambda}| = |\Omega^{\mathcal{A}_1(\rho_{u\to X,u\to\neg X})\cup\Lambda}|/|\Omega^{\mathcal{A}_1(\rho)\cup\Lambda}|$. This is well-defined since $\mathcal{A}_1(\rho_{u\to X,u\to X}) = \mathcal{A}_1(\rho_{u\to X,u\to\neg X})$. In other words, $\psi_{\rho,X,1}$ is the probability that u is assigned value X under the uniform distribution on $\Omega^{\mathcal{A}_1(\rho)\cup\Lambda}$. We similarly define $\psi_{\rho,X,2} := |\Omega^{\mathcal{A}_2(\rho_{u\to X,u\to X})\cup\Lambda}|/|\Omega^{\mathcal{A}_2(\rho)\cup\Lambda}| = |\Omega^{\mathcal{A}_2(\rho_{u\to\neg X,u\to X})\cup\Lambda}|/|\Omega^{\mathcal{A}_2(\rho)\cup\Lambda}|$.

We will next give an inductive definition of a function Q from nodes of the coupling tree to real numbers in $[0,1]$. The way to think about this is as follows – we will implicitly define a probability distribution over paths from the root of the coupling tree to $\mathcal{L}^*$. For each node ρ, $Q(\rho)$ will be the probability that ρ is included in a path drawn from this distribution.

Any such path starts at the root, so we define $Q(\rho^*) = 1$. Once we have defined $Q(\rho)$ for a node ρ that is not in $\mathcal{L}^*$ we can define $Q(\cdot)$ on the children of ρ as follows. Let u be the first variable of ρ and consider the four children $\rho_{u\to\mathsf{T},u\to\mathsf{T}}, \rho_{u\to\mathsf{T},u\to\mathsf{F}}, \rho_{u\to\mathsf{F},u\to\mathsf{T}}, \rho_{u\to\mathsf{F},u\to\mathsf{F}}$. Define the values of Q as follows: $Q(\rho_{u\to\mathsf{T},u\to\mathsf{T}}) := Q(\rho)\min\{\psi_{\rho,\mathsf{T},1}, \psi_{\rho,\mathsf{T},2}\}$, $Q(\rho_{u\to\mathsf{T},u\to\mathsf{F}}) := Q(\rho)(\psi_{\rho,\mathsf{T},1} - \min\{\psi_{\rho,\mathsf{T},1}, \psi_{\rho,\mathsf{T},2}\})$, $Q(\rho_{u\to\mathsf{F},u\to\mathsf{F}}) := Q(\rho)\min\{1-\psi_{\rho,\mathsf{T},1}, 1-\psi_{\rho,\mathsf{T},2}\}$, and $Q(\rho_{u\to\mathsf{F},u\to\mathsf{T}}) := Q(\rho)((1-\psi_{\rho,\mathsf{T},1}) - \min\{1-\psi_{\rho,\mathsf{T},1}, 1-\psi_{\rho,\mathsf{T},2}\})$. Finally, we define $P_{i,\rho} := Q(\rho)|\Omega_i^\Lambda|/|\Omega^{\mathcal{A}_i(\rho)\cup\Lambda}|$. In the full version, we prove that this assignment satisfies the constraints of the LP. Mostly, the LP is designed to make this true, though for example to establish the constraint $P_{i,\rho_{u\to X,u\to\neg X}} \le \frac{1}{s} P_{i,\rho}$ (Lemma 23) we need to prove that $\psi_{\rho,X,i}$ is around $1/2$. Like Moitra, we prove this using the Lovász local lemma, so this is why it is essential that we restrict the LP to good variables. ◀

References

1 E. Abbe and A. Montanari. On the concentration of the number of solutions of random satisfiability formulas. *Random Struct. Algorithms*, 45(3):362–382, 2014.

2 D. Achlioptas and A. Coja-Oghlan. Algorithmic barriers from phase transitions. In *FOCS*, pages 793–802. IEEE Computer Society, 2008.

3 D. Achlioptas and C. Moore. The asymptotic order of the random k-SAT threshold. In *FOCS*, pages 779–788. IEEE Computer Society, 2002.

4 D. Achlioptas and Y. Peres. The threshold for random k-SAT is $2^k(\ln 2 - o(k))$. In *STOC*, pages 223–231. ACM, 2003.

5 N. Alon. A parallel algorithmic version of the local lemma. *Random Struct. Algorithms*, 2(4):367–378, 1991. doi:10.1002/rsa.3240020403.

6 I. Bezáková, A. Galanis, L. A. Goldberg, H. Guo, and D. Štefankovič. Approximation via correlation decay when strong spatial mixing fails. *SIAM J. Comput.*, 48(2):279–349, 2019.

7 K. Chandrasekaran, N. Goyal, and B. Haeupler. Deterministic algorithms for the Lovász local lemma. *SIAM J. Comput.*, 42(6):2132–2155, 2013.

8 A. Coja-Oghlan. A better algorithm for random k-SAT. *SIAM J. Comput.*, 39(7):2823–2864, 2010.

9 A. Coja-Oghlan. Belief propagation guided decimation fails on random formulas. *J. ACM*, 63(6):Art. 49, 55, 2017.

10 A. Coja-Oghlan and A. Frieze. Analyzing Walksat on random formulas. *SIAM J. Comput.*, 43(4):1456–1485, 2014.

11 A. Coja-Oghlan, A. Haqshenas, and S. Hetterich. Walksat stalls well below satisfiability. *SIAM J. Discrete Math.*, 31(2):1160–1173, 2017.

12 A. Coja-Oghlan and K. Panagiotou. The asymptotic k-SAT threshold. *Adv. Math.*, 288:985–1068, 2016.

13 A. Coja-Oghlan and D. Reichman. Sharp thresholds and the partition function. *Journal of Physics: Conference Series*, 473:012015, 2013.

14 A. Coja-Oghlan and N. Wormald. The number of satisfying assignments of random regular k-SAT formulas. *Combin. Probab. Comput.*, 27(4):496–530, 2018.

15 J. Ding, A. Sly, and N. Sun. Proof of the satisfiability conjecture for large k. In *STOC*, pages 59–68. ACM, 2015.

16 C. Efthymiou, T. P. Hayes, D. Štefankovič, and E. Vigoda. Sampling random colorings of sparse random graphs. In *SODA*, pages 1759–1771. SIAM, 2018.

17 P. Erdős and L. Lovász. Problems and results on 3-chromatic hypergraphs and some related questions. *Infinite and finite sets, volume 10 of Colloquia Mathematica Societatis János Bolyai*, pages 609–628, 1975.

18 W. Feng, H. Guo, Y. Yin, and C. Zhang. Fast sampling and counting k-sat solutions in the local lemma regime, 2019. arXiv:1911.01319.

19 E. Friedgut. Sharp thresholds of graph properties, and the k-sat problem. *J. Amer. Math. Soc.*, 12(4):1017–1054, 1999. With an appendix by Jean Bourgain. doi:10.1090/S0894-0347-99-00305-7.

20 H. Guo, M. Jerrum, and J. Liu. Uniform sampling through the Lovász local lemma. *J. ACM*, 66(3):18:1–18:31, 2019.

21 H. Guo, C. Liao, P. Lu, and C. Zhang. Counting hypergraph colorings in the local lemma regime. *SIAM J. Comput.*, 48(4):1397–1424, 2019.

22 B. Haeupler, B. Saha, and A. Srinivasan. New constructive aspects of the Lovász local lemma. *J. ACM*, 58(6):Art. 28, 28, 2011.

23 J. Hermon, A. Sly, and Y. Zhang. Rapid mixing of hypergraph independent sets. *Random Struct. Algorithms*, 54(4):730–767, 2019.

24 S. Hetterich. Analysing survey propagation guided decimationon random formulas. In *ICALP*, volume 55 of *LIPIcs*, pages 65:1–65:12. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

25 L. M. Kirousis, E. Kranakis, D. Krizanc, and Y. C. Stamatiou. Approximating the unsatisfiability threshold of random formulas. *Random Structures Algorithms*, 12(3):253–269, 1998.

26 C. Liao, J. Lin, P. Lu, and Z. Mao. Counting independent sets and colorings on random regular bipartite graphs. In *APPROX-RANDOM*, volume 145 of *LIPIcs*, pages 34:1–34:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

27 M. Mézard, T. Mora, and R. Zecchina. Clustering of solutions in the random satisfiability problem. *Phys. Rev. Lett.*, 94:197205, 2005.

28 M. Mézard, G. Parisi, and R. Zecchina. Analytic and algorithmic solution of random satisfiability problems. *Science*, 297(5582):812–815, 2002.

29 M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis.* Cambridge University Press, 2005.

30 A. Moitra. Approximate counting, the Lovász local lemma, and inference in graphical models. *J. ACM*, 66(2):Art. 10, 25, 2019.

31 A. Montanari and D. Shah. Counting good truth assignments of random k-SAT formulae. In *SODA*, pages 1255–1264. SIAM, 2007.

32 R. A. Moser and G. Tardos. A constructive proof of the general Lovász local lemma. *J. ACM*, 57(2):Art. 11, 15, 2010.

33 E. Mossel and A. Sly. Exact thresholds for Ising-Gibbs samplers on general graphs. *Ann. Probab.*, 41(1):294–328, 2013.

34 J. Schmidt-Pruzan and E. Shamir. Component structure in the evolution of random hypergraphs. *Combinatorica*, 5(1):81–94, 1985.

35 A. Sly, N. Sun, and Y. Zhang. The number of solutions for random regular NAE-SAT. In *FOCS*, pages 724–731. IEEE Computer Society, 2016.

36 J. Spencer. Asymptotic lower bounds for Ramsey functions. *Discrete Math.*, 20(1):69–76, 1977.

37 Y. Yin and C. Zhang. Sampling in Potts model on sparse random graphs. In *APPROX-RANDOM*, volume 60 of *LIPIcs*, pages 47:1–47:22. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

Robust Algorithms for TSP and Steiner Tree

Arun Ganesh
Department of Electrical Engineering and Computer Sciences,
University of California at Berkeley, CA, USA
arunganesh@berkeley.edu

Bruce M. Maggs
Department of Computer Science, Duke University, Durham, NC, USA
Emerald Innovations, Cambridge, MA, USA
bmm@cs.duke.edu

Debmalya Panigrahi
Department of Computer Science, Duke University, Durham, NC, USA
debmalya@cs.duke.edu

Abstract

Robust optimization is a widely studied area in operations research, where the algorithm takes as input a range of values and outputs a single solution that performs well for the entire range. Specifically, a robust algorithm aims to minimize *regret*, defined as the maximum difference between the solution's cost and that of an optimal solution in hindsight once the input has been realized. For graph problems in **P**, such as shortest path and minimum spanning tree, robust polynomial-time algorithms that obtain a constant approximation on regret are known. In this paper, we study robust algorithms for minimizing regret in **NP**-hard graph optimization problems, and give constant approximations on regret for the classical traveling salesman and Steiner tree problems.

2012 ACM Subject Classification Theory of computation → Routing and network design problems

Keywords and phrases Robust optimization, Steiner tree, traveling salesman problem

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.54

Category Track A: Algorithms, Complexity and Games

Related Version https://arxiv.org/abs/2005.08137

Funding *Arun Ganesh*: Supported in part by NSF Award CCF-1535989.
Bruce M. Maggs: Supported in part by NSF Award CCF-1535972.
Debmalya Panigrahi: Supported in part by NSF grants CCF-1535972, CCF-1955703, an NSF CAREER Award CCF-1750140, and the Indo-US Virtual Networked Joint Center on Algorithms under Uncertainty.

1 Introduction

In many graph optimization problems, the inputs are not known precisely and the algorithm is desired to perform well over a range of inputs. For instance, consider the following situations. Suppose we are planning the delivery route of a vehicle that must deliver goods to n locations. Due to varying traffic conditions, the exact travel times between locations are not known precisely, but a range of possible travel times is available from historical data. Can we design a tour that is nearly optimal for *all* travel times in the given ranges? Consider another situation where we are designing a telecommunication network to connect a set of locations. We are given cost estimates on connecting every two locations in the network but these estimates might be off due to unexpected construction problems. Can we design the network in a way that is nearly optimal for *all* realized construction costs?

These questions have led to the field of *robust* graph algorithms. Given a range of weights $[\ell_e, u_e]$ for every edge e, the goal is to find a solution that minimizes *regret*, defined as the maximum difference between the algorithm's cost and the optimal cost for any edge weights.

In other words, the goal is to obtain: $\min_{\text{SOL}} \max_{\mathbf{I}}(\text{SOL}(\mathbf{I}) - \text{OPT}(\mathbf{I}))$, where $\text{SOL}(\mathbf{I})$ (resp. $\text{OPT}(\mathbf{I})$) denotes the cost of SOL (resp. the optimal solution) in instance $\mathbf{I}$, SOL ranges over all feasible solutions, and $\mathbf{I}$ ranges over all realizable inputs. We emphasize that SOL is a fixed solution (independent of $\mathbf{I}$) whereas the solution determining $\text{OPT}(\mathbf{I})$ is dependent on the input $\mathbf{I}$. The solution that achieves this minimum is called the *minimum regret solution* (MRS), and its regret is the *minimum regret* (MR). In many cases, however, minimizing regret turns out to be **NP**-hard, in which case one seeks an approximation guarantee. Namely, a β-approximation algorithm satisfies, for all input realizations $\mathbf{I}$, $\text{SOL}(\mathbf{I}) - \text{OPT}(\mathbf{I}) \leq \beta \cdot \text{MR}$, i.e., $\text{SOL}(\mathbf{I}) \leq \text{OPT}(\mathbf{I}) + \beta \cdot \text{MR}$.

It is known that minimizing regret is **NP**-hard for shortest path [34] and minimum cut [1] problems, and using a general theorem for converting exact algorithms to robust ones, 2-approximations are known for these problems [12, 23]. In some cases, better results are known for special classes of graphs, e.g., [24]. Robust minimum spanning tree (MST) has also been studied, although in the context of making exponential-time exact algorithms more practical [33]. Moreover, robust optimization has been extensively researched for other (non-graph) problem domains in the operations research community, and has led to results in clustering [5, 3, 6, 27], linear programming [21, 28], and other areas [4, 23]. More details can be found in the book by Kouvelis and Yu [26] and the survey by Aissi *et al.* [2].

To the best of our knowledge, all previous work in polynomial-time algorithms for minimizing regret in robust graph optimization focused on problems in **P**. In this paper, we study robust graph algorithms for minimizing regret in **NP**-hard optimization problems. In particular, we study robust algorithms for the classical traveling salesman (TSP) and Steiner tree (STT) problems, that model e.g. the two scenarios described at the beginning of the paper. As a consequence of the **NP**-hardness, we cannot hope to show guarantees of the form: $\text{SOL}(\mathbf{I}) \leq \text{OPT}(\mathbf{I}) + \beta \cdot \text{MR}$, since for $\ell_e = u_e$ (i.e., $\text{MR} = 0$), this would imply an exact algorithm for an **NP**-hard optimization problem. Instead, we give guarantees: $\text{SOL}(\mathbf{I}) \leq \alpha \cdot \text{OPT}(\mathbf{I}) + \beta \cdot \text{MR}$, where α is (necessarily) at least as large as the best approximation guarantee for the optimization problem. We call such an algorithm an (α, β)-robust algorithm. If both α and β are constants, we call it a constant-approximation to the robust problem. In this paper, our main results are constant approximation algorithms for the robust traveling salesman and Steiner tree problems. We hope that our work will lead to further research in the field of robust approximation algorithms, particularly for other **NP**-hard optimization problems in graph algorithms as well as in other domains.

1.1 Problem Definition and Results

We first define the Steiner tree (STT) and traveling salesman problems (TSP). In both problems, the input is an undirected graph $G = (V, E)$ with non-negative edge costs. In Steiner tree, we are also given a subset of vertices called *terminals* and the goal is to obtain a minimum cost connected subgraph of G that spans all the terminals. In traveling salesman, the goal is to obtain a minimum cost tour that visits every vertex in V[1]. In the robust versions of these problems, the edge costs are ranges $[\ell_e, u_e]$ from which any cost may realize.

Our main results are the following:

▶ **Theorem 1** (Robust Approximations). *There exist constant approximation algorithms for the robust traveling salesman and Steiner tree problems.*

[1] There are two common and equivalent assumptions made in the TSP literature in order to achieve reasonable approximations. In the first assumption, the algorithms can visit vertices multiple times in the tour, while in the latter, the edges satisfy the metric property. We use the former in this paper.

▶ Remark. The constants we are able to obtain for the two problems are very different: $(4.5, 3.75)$ for TSP (in Section 3) and $(2755, 64)$ for STT (in Section 4). While we did not attempt to optimize the precise constants, obtaining small constants for STT comparable to the TSP result requires new ideas beyond our work and is an interesting open problem.

We complement our algorithmic results with lower bounds. Note that if $\ell_e = u_e$, we have $\textsc{mr} = 0$ and thus an (α, β)-robust algorithm gives an α-approximation for precise inputs. So, hardness of approximation results yield corresponding lower bounds on α. More interestingly, we show that hardness of approximation results also yield lower bounds on the value of β (see Section 5 for details):

▶ **Theorem 2** (APX-hardness). *A hardness of approximation of ρ for TSP (resp., STT) under $\mathbf{P} \neq \mathbf{NP}$ implies that it is $\mathbf{NP}$-hard to obtain $\alpha \leq \rho$ (irrespective of β) and $\beta \leq \rho$ (irrespective of α) for robust TSP (resp., robust STT).*

1.2 Our Techniques

We now give a sketch of our techniques. Before doing so, we note that for problems in $\mathbf{P}$ with linear objectives, it is known that running an exact algorithm using weights $\frac{\ell_e + u_e}{2}$ gives a $(1, 2)$-robust solution [12, 23]. One might hope that a similar result can be obtained for $\mathbf{NP}$-hard problems by replacing the exact algorithm with an approximation algorithm in the above framework. Unfortunately, there exists robust TSP instances where using a 2-approximation for TSP with weights $\frac{\ell_e + u_e}{2}$ gives a solution that is **not** (α, β)-robust for *any* $\alpha = o(n), \beta = o(n)$. More generally, a black-box approximation run on a fixed realization could output a solution including edges that have small weight relative to OPT for that realization (so including these edges does not violate the approximation guarantee), but these edges could have large weight relative to MR and OPT in other realizations, ruining the robustness guarantee. This establishes a qualitative difference between robust approximations for problems in $\mathbf{P}$ considered earlier and $\mathbf{NP}$-hard problems being considered in this paper, and demonstrates the need to develop new techniques for the latter class of problems.

LP relaxation. We denote the input graph $G = (V, E)$. For each edge $e \in E$, the input is a range $[\ell_e, u_e]$ where the actual edge weight d_e can realize to any value in this range. The robust version of a graph optimization problem is is then described by the LP

$$\min\{r : \mathbf{x} \in P; \sum_{e \in E} d_e x_e \leq \textsc{opt}(\mathbf{d}) + r, \ \forall \mathbf{d}\},$$

where P is the standard polytope for the optimization problem, and $\textsc{opt}(\mathbf{d})$ denotes the cost of an optimal solution when the edge weights are $\mathbf{d} = \{d_e : e \in E\}$. That is, this is the standard LP for the problem, but with the additional constraint that the fractional solution $\mathbf{x}$ must have regret at most r for any realization of edge weights. We call the additional constraints the *regret constraint set*. Note that setting $\mathbf{x}$ to be the indicator vector of MRS and r to MR gives a feasible solution to the LP; thus, the LP optimum is at most MR.

Solving the LP. We assume that the constraints in P are separable in polynomial time (e.g., this is true for most standard optimization problems including STT and TSP). So, designing the separation oracle comes down to separating the regret constraint set, which requires checking that:

$$\max_{\mathbf{d}} \left[\sum_{e \in E} d_e x_e - \textsc{opt}(\mathbf{d})\right] =$$

$$\max_{\mathbf{d}} \max_{\textsc{sol}} \left[\sum_{e \in E} d_e x_e - \textsc{sol}(\mathbf{d})\right] = \max_{\textsc{sol}} \max_{\mathbf{d}} \left[\sum_{e \in E} d_e x_e - \textsc{sol}(\mathbf{d})\right] \leq r.$$

Thus, given a fractional solution $\mathbf{x}$, we need to find an integer solution $\textsc{sol}$ and a weight vector $\mathbf{d}$ that maximizes the regret of $\mathbf{x}$ given by $\sum_{e \in E} d_e x_e - \textsc{sol}(\mathbf{d})$. Once $\textsc{sol}$ is fixed, finding $\mathbf{d}$ that maximizes the regret is simple: If $\textsc{sol}$ does not include an edge e, then to maximize $\sum_{e \in E} d_e x_e - \textsc{sol}(\mathbf{d})$, we set $d_e = u_e$; else if $\textsc{sol}$ includes e, we set $d_e = \ell_e$. Note that in these two cases, edge e contributes $u_e x_e$ and $\ell_e x_e - \ell_e$ respectively to the regret. The above maximization thus becomes:

$$\max_{\textsc{sol}} \left[\sum_{e \notin \textsc{sol}} u_e x_e + \sum_{e \in \textsc{sol}} (\ell_e x_e - \ell_e) \right] = \sum_{e \in E} u_e x_e - \min_{\textsc{sol}} \sum_{e \in \textsc{sol}} (u_e x_e - \ell_e x_e + \ell_e). \tag{1}$$

Thus, $\textsc{sol}$ is exactly the optimal solution with edge weights $a_e := u_e x_e - \ell_e x_e + \ell_e$. (For reference, we define the *derived* instance of the problem as one with edge weights a_e.)

Now, if we were solving a problem in $\mathbf{P}$, we would simply need to solve the problem on the derived instance. Indeed, we will show later that this yields an alternative technique for obtaining robust algorithms for problems in $\mathbf{P}$, and recover existing results in [23]. However, we cannot hope to find an optimal solution to an $\mathbf{NP}$-hard problem. Our first compromise is that we settle for an *approximate* separation oracle. More precisely, our goal is to show that there exists some fixed constants $\alpha', \beta' \geq 1$ such that if $\sum_e d_e x_e > \alpha' \cdot \textsc{opt}(\mathbf{d}) + \beta' \cdot r$ for some $\mathbf{d}$, then we can find $\textsc{sol}, \mathbf{d}'$ such that $\sum_e d'_e x_e > \textsc{sol}(\mathbf{d}') + r$. Since the LP optimum is at most $\textsc{mr}$, we can then obtain an (α', β')-robust *fractional* solution using the standard ellipsoid algorithm.

For $\textsc{tsp}$, we show that the above guarantee can be achieved by the classic $\textsc{mst}$-based 2-approximation on the derived instance. The details appear in Section 3 and the full paper. Although $\textsc{stt}$ also admits a 2-approximation based on the $\textsc{mst}$ solution, this turns out to be insufficient for the above guarantee. Instead, we use a different approach here. We note that the regret of the fractional solution against any fixed solution $\textsc{sol}$ (i.e., the argument over which Eq. (1) maximizes) can be expressed as the following difference:

$$\sum_{e \notin \textsc{sol}} (u_e x_e - \ell_e x_e + \ell_e) - \sum_{e \in E} (\ell_e - \ell_e x_e) = \sum_{e \notin \textsc{sol}} a_e - \sum_{e \in E} b_e, \text{ where } b_e := \ell_e - \ell_e x_e.$$

The first term is the weight of edges in the derived instance that are *not* in $\textsc{sol}$. The second term corresponds to a new $\textsc{stt}$ instance with different edge weights b_e. It turns out that the overall problem now reduces to showing the following approximation guarantees on these two $\textsc{stt}$ instances (c_1 and c_2 are constants):

$$\text{(i)} \sum_{e \in \textsc{alg} \setminus \textsc{sol}} a_e \leq c_1 \cdot \sum_{e \in \textsc{sol} \setminus \textsc{alg}} a_e \qquad \text{and} \qquad \text{(ii)} \sum_{e \in \textsc{alg}} b_e \leq c_2 \cdot \sum_{e \in \textsc{sol}} b_e.$$

Note that guarantee (i) on the derived instance is an unusual "difference approximation" that is stronger than usual approximation guarantees. Moreover, we need these approximation bounds to *simultaneously* hold, i.e., hold for the same $\textsc{alg}$. Obtaining these dual approximation bounds simultaneously forms the most technically challenging part of our work; a high level overview is given in Section 4 and technical details are deferred to the full paper.

Rounding the fractional solution. After applying our approximate separation oracles, we have a fractional solution $\mathbf{x}$ such that for all edge weights $\mathbf{d}$, we have $\sum_e d_e x_e \leq \alpha' \cdot \textsc{opt}(\mathbf{d}) + \beta' \cdot \textsc{mr}$. Suppose that, ignoring the regret constraint set, the LP we are using has integrality gap at most δ for precise inputs. Then a natural rounding approach is to search for an integer solution $\textsc{alg}$ that has minimum regret with respect to the specific solution $\delta\mathbf{x}$, i.e., $\textsc{alg}$ satisfies:

$$\textsc{alg} = \operatorname*{argmin}_{\textsc{sol}} \max_{\mathbf{d}} \left[\textsc{sol}(\mathbf{d}) - \delta \sum_{e \in E} d_e x_e \right]. \tag{2}$$

Since the integrality gap is at most δ, we have $\delta \cdot \sum_{e\in E} d_e x_e \geq \textsc{opt}(\mathbf{d})$ for any $\mathbf{d}$. This implies that:

$$\textsc{mrs}(\mathbf{d}) - \delta \cdot \sum_{e\in E} d_e x_e \leq \textsc{mrs}(\mathbf{d}) - \textsc{opt}(\mathbf{d}) \leq \textsc{mr}.$$

Hence, the regret of $\textsc{mrs}$ with respect to δx is at most $\textsc{mr}$. Since $\textsc{alg}$ has minimum regret with respect to $\delta\mathbf{x}$, $\textsc{alg}$'s regret is also at most $\textsc{mr}$. Note that $\delta\mathbf{x}$ is a $(\delta\alpha', \delta\beta')$-robust solution. Hence, $\textsc{alg}$ is a $(\delta\alpha', \delta\beta' + 1)$-robust solution.

If we are solving a problem in **P**, finding $\textsc{alg}$ that satisfies Eq. (2) is easy. So, using an integral LP formulation (i.e., integrality gap of 1), we get a $(1, 2)$-robust algorithm overall for these problems. This exactly matches the results in [23], although we are using a different set of techniques. Of course, for **NP**-hard problems, finding a solution $\textsc{alg}$ that satisfies Eq. (2) is **NP**-hard as well. It turns out, however, that we can design a generic rounding algorithm that gives the following guarantee:

▶ **Theorem 3.** *There exists a rounding algorithm that takes as input an (α, β)-robust fractional solution to $\textsc{stt}$ (resp. $\textsc{tsp}$) and outputs a $(\gamma\delta\alpha, \gamma\delta\beta + \gamma)$-robust integral solution, where γ and δ are respectively the best approximation factor and integrality gap for (classical) $\textsc{stt}$ (resp., $\textsc{tsp}$).*

We remark that while we stated this rounding theorem for $\textsc{stt}$ and $\textsc{tsp}$ here, we actually give a more general version (Theorem 4) in Section 2 that applies to a broader class of covering problems including set cover, survivable network design, etc. and might be useful in future research in this domain.

1.3 Related Work

We have already discussed the existing literature in robust optimization for minimizing regret. Other robust variants of graph optimization have also been studied in the literature. In the *robust combinatorial optimization* model proposed by Bertsimas and Sim [7], edge costs are given as ranges as in this paper, but instead of optimizing for all realizations of costs within the ranges, the authors consider a model where at most k edge costs can be set to their maximum value and the remaining are set to their minimum value. The objective is to minimize the maximum cost over all realizations. In this setting, there is no notion of regret and an approximation algorithm for the standard problem translates to an approximation algorithm for the robust problem with the same approximation factor.

In the *data-robust model* [13], the input includes a polynomial number of explicitly defined "scenarios" for edge costs, with the goal of finding a solution that is approximately optimal for all given scenarios. That is, in the input one receives a graph and a polynomial number of scenarios $\mathbf{d}^{(1)}, \mathbf{d}^{(2)} \ldots \mathbf{d}^{(k)}$ and the goal is to find $\textsc{alg}$ whose maximum cost across all scenarios is at most some approximation factor times $\min_{\textsc{sol}} \max_{i\in[k]} \sum_{e\in\textsc{sol}} d_e^{(i)}$. In contrast, in this paper, we have exponentially many scenarios and look at the maximum of $\textsc{alg}(\mathbf{d}) - \textsc{opt}(\mathbf{d})$ rather than $\textsc{alg}(\mathbf{d})$. A variation of this is the *recoverable robust model* [9], where after seeing the chosen scenario, the algorithm is allowed to "recover" by making a small set of changes to its original solution.

Dhamdhere *et al.* [13] also studies the *demand-robust model*, where edge costs are fixed but the different scenarios specify different connectivity requirements of the problem. The algorithm now operates in two phases: In the first phase, the algorithm builds a partial solution T' and then one of the scenarios (sets of terminals) T_i is revealed to the algorithm. In the second phase, the algorithm then adds edges to T' to build a solution T, but must pay a multiplicative cost of σ_k on edges added in the second phase. The demand-robust model was inspired by a two-stage stochastic optimization model studied in, e.g., [30, 29, 31, 13, 14, 25, 18, 19, 20, 8] where the scenario is chosen according to a distribution rather than an adversary.

Another related setting to the data-robust model is that of *robust network design*, introduced to model uncertainty in the demand matrix of network design problems (see the survey by Chekuri [10]). This included the well-known VPN conjecture (see, e.g., [17]), which was eventually settled in [15]. In all these settings, however, the objective is to minimize the maximum cost over all realizations, whereas in this paper, our goal is to minimize the maximum *regret* against the optimal solution.

2 Generalized Rounding Algorithm

We start by giving the rounding algorithm of Theorem 3, which is a corollary of the following, more general theorem:

▶ **Theorem 4.** *Let $\mathcal{P}$ be an optimization problem defined on a set system $\mathcal{S} \subseteq 2^E$ that seeks to find the set $S \in \mathcal{S}$ that minimizes $\sum_{e \in S} d_e$, i.e., the sum of the weights of elements in S. In the robust version of this optimization problem, we have $d_e \in [\ell_e, u_e]$ for all $e \in E$.*

Consider an LP formulation of $\mathcal{P}$ (called $\mathcal{P}$-LP) given by: $\{\min \sum_{e \in E} d_e x_e : \boldsymbol{x} \in X, \boldsymbol{x} \in [0,1]^E\}$, where X is a polytope containing the indicator vector χ_S of all $S \in \mathcal{S}$ and not containing χ_S for any $S \notin \mathcal{S}$. The corresponding LP formulation for the robust version (called $\mathcal{P}_{\text{robust}}$-LP) is given by: $\{\min r : \boldsymbol{x} \in X, \boldsymbol{x} \in [0,1]^E, \sum_{e \in E} d_e x_e \leq \text{OPT}(\boldsymbol{d}) + r \ \forall \boldsymbol{d}\}$.

Now, suppose we have the following properties:

- *There is a γ-approximation algorithm for $\mathcal{P}$.*
- *The integrality gap of $\mathcal{P}$-LP is at most δ.*
- *There is a feasible solution $\boldsymbol{x}^*$ to $\mathcal{P}$-LP that satisfies: $\forall \boldsymbol{d} : \sum_{e \in E} d_e x_e^* \leq \alpha \cdot \text{OPT}(\boldsymbol{d}) + \beta \cdot \text{MR}$.*

Then, there exists an algorithm that outputs an integer solution SOL *for $\mathcal{P}$ that satisfies:*

$$\forall \boldsymbol{d} : \text{SOL}(\boldsymbol{d}) \leq (\gamma\delta\alpha) \cdot \text{OPT}(\boldsymbol{d}) + (\gamma\delta\beta + \gamma) \cdot \text{MR}.$$

Proof. The algorithm is as follows: Construct an instance of $\mathcal{P}$ which uses the same set system $\mathcal{S}$ and where element e has weight $\max\{u_e(1 - \delta x_e^*), \ell_e(1 - \delta x_e^*)\} + \delta \ell_e x_e^*$. Then, use the γ-approximation algorithm for $\mathcal{P}$ on this instance to find an integral solution S, and output it.

Given a feasible solution S to $\mathcal{P}$, note that:

$$\max_{\mathbf{d}}[\sum_{e \in S} d_e - \delta \sum_{e \in E} d_e x_e^*] = \sum_{e \in S} \max\{u_e(1 - \delta x_e^*), \ell_e(1 - \delta x_e^*)\} - \sum_{e \notin S} \delta \ell_e x_e^*$$

$$= \sum_{e \in S}[\max\{u_e(1 - \delta x_e^*), \ell_e(1 - \delta x_e^*)\} + \delta \ell_e x_e^*] - \sum_{e \in E} \delta \ell_e x_e^*.$$

Now, note that since S was output by a γ-approximation algorithm, for any feasible solution S':

$$\sum_{e \in S}[\max\{u_e(1-\delta x_e^*), \ell_e(1-\delta x_e^*)\}+\delta \ell_e x_e^*] \leq \gamma \sum_{e \in S'}[\max\{u_e(1-\delta x_e^*), \ell_e(1-\delta x_e^*)\}+\delta \ell_e x_e^*] \implies$$

$$\sum_{e \in S}[\max\{u_e(1 - \delta x_e^*), \ell_e(1 - \delta x_e^*)\} + \delta \ell_e x_e^*] - \gamma \sum_{e \in E} \delta \ell_e x_e^*$$

$$\leq \gamma[\sum_{e \in S'}[\max\{u_e(1-\delta x_e^*), \ell_e(1-\delta x_e^*)\} + \delta \ell_e x_e^*] - \sum_{e \in E} \delta \ell_e x_e^*]$$

$$= \gamma \max_{\mathbf{d}}[\sum_{e \in S'} d_e - \delta \sum_{e \in E} d_e x_e^*].$$

Since $\mathcal{P}$-LP has integrality gap δ, for any fractional solution $\mathbf{x}$, $\forall \mathbf{d} : \textsc{opt}(\mathbf{d}) \leq \delta \sum_{e \in E} d_e x_e$. Fixing S' to be the set of elements used in the minimum regret solution then gives:

$$\max_{\mathbf{d}}[\sum_{e \in S'} d_e - \delta \sum_{e \in E} d_e x_e^*] \leq \max_{\mathbf{d}}[\textsc{mrs}(\mathbf{d}) - \textsc{opt}(\mathbf{d})] = \textsc{mr}.$$

Combined with the previous inequality, this gives:

$$\sum_{e \in S}[\max\{u_e(1-\delta x_e^*), \ell_e(1-\delta x_e^*)\} + \delta \ell_e x_e^*] - \gamma \sum_{e \in E} \delta \ell_e x_e^* \leq \gamma \textsc{mr} \implies$$

$$\sum_{e \in S}[\max\{u_e(1-\delta x_e^*), \ell_e(1-\delta x_e^*)\} + \delta \ell_e x_e^*] - \sum_{e \in E} \delta \ell_e x_e^* \leq \gamma \textsc{mr} + (\gamma - 1) \sum_{e \in E} \delta \ell_e x_e^* \implies$$

$$\max_{\mathbf{d}}[\sum_{e \in S} d_e - \delta \sum_{e \in E} d_e x_e^*] \leq \gamma \textsc{mr} + (\gamma - 1) \sum_{e \in E} \delta \ell_e x_e^*.$$

This implies:

$$\forall \mathbf{d} : \textsc{sol}(\mathbf{d}) = \sum_{e \in S} d_e \leq \delta \sum_{e \in E} d_e x_e^* + \gamma \textsc{mr} + (\gamma - 1) \sum_{e \in E} \delta \ell_e x_e^*$$

$$\leq \delta \sum_{e \in E} d_e x_e^* + \gamma \textsc{mr} + (\gamma - 1) \sum_{e \in E} \delta d_e x_e^*$$

$$= \gamma\delta \sum_{e \in E} d_e x_e^* + \gamma \textsc{mr} \leq \gamma\delta[\alpha \textsc{opt}(\mathbf{d}) + \beta \textsc{mr}] + \gamma \textsc{mr} = \gamma\delta\alpha \cdot \textsc{opt}(\mathbf{d}) + (\gamma\delta\beta + \gamma) \cdot \textsc{mr}. \blacktriangleleft$$

3 Algorithm for the Robust Traveling Salesman Problem

In this section, we give a robust algorithm for the traveling salesman problem:

▶ **Theorem 5.** *There exists a* $(4.5, 3.75)$*-robust algorithm for the traveling salesman problem.*

Recall that we consider the version of the problem where we are allowed to use edges multiple times in TSP. We present a high level sketch of our ideas here, the details are deferred to the full paper. We recall that any TSP tour must contain a spanning tree, and an Eulerian walk on a doubled MST is a 2-approximation algorithm for TSP (known as the "double-tree algorithm"). One might hope that since we have a $(1, 2)$-robust algorithm for robust MST, one could take its output and apply the double-tree algorithm to get a $(2, 4)$-robust solution to robust TSP. Unfortunately, this algorithm is not (α, β)-robust for any $\alpha = o(n), \beta = o(n)$. Nevertheless, we are able to leverage the connection to MST to arrive at a $(4.5, 3.75)$-robust algorithm for TSP.

$$\textbf{Minimize } \ r \textbf{ subject to}$$

$$\begin{array}{ll}
\forall \emptyset \neq S \subset V: & \sum_{u \in S, v \in V \setminus S} y_{uv} \geq 2 \\
\forall u \in V: & \sum_{v \neq u} y_{uv} = 2 \\
\forall \emptyset \neq S \subset V, u \in S, v \in V \setminus S: & \sum_{e \in \delta(S)} x_{e,u,v} \geq y_{uv} \\
\forall \mathbf{d}: & \sum_{e \in E} d_e x_e \leq \textsc{opt}(\mathbf{d}) + r \\
\forall u, v \in V, u \neq v: & 0 \leq y_{uv} \leq 1 \\
\forall e \in E, u, v \in V, v \neq u: & 0 \leq x_{e,u,v} \leq 1 \\
\forall e \in E: & x_e \leq 2
\end{array} \tag{3}$$

Figure 1 The Robust TSP Polytope.

3.1 Approximate Separation Oracle

We use the LP relaxation of robust traveling salesman in Figure 1. This is the standard subtour LP (see e.g. [32]), but augmented with variables specifying the edges used to visit each new vertex, as well as with the regret constraint set. Integrally, y_{uv} is 1 if splitting the tour into subpaths at each point where a vertex is visited for the first time, there is a subpath from u to v (or vice-versa). That is, y_{uv} is 1 if between the first time u is visited and the first time v is visited, the tour only goes through vertices that were already visited before visiting u. $x_{e,u,v}$ is 1 if on this subpath, the edge e is used. We use x_e to denote $\sum_{u,v \in V} x_{e,u,v}$ for brevity. A discussion of why the constraints other than the regret constraint set in (3) are identical to the standard TSP polytope is included in the full paper.

We now describe the separation oracle RRTSP-Oracle used to separate (3). All constraints except the regret constraint set can be separated in polynomial time by solving a min-cut problem. Recall that exactly separating the regret constraint set involves finding an "adversary" SOL that maximizes $\max_{\mathbf{d}}[\sum_{e \in E} d_e x_e - \textsc{sol}(\mathbf{d})]$, and seeing if this quantity exceeds r. However, since TSP is **NP**-hard, finding this solution in general is **NP**-hard. Instead, we will only consider a solution SOL if it is a walk on some spanning tree T, and find the one that maximizes $\max_{\mathbf{d}}[\sum_{e \in E} d_e x_e - \textsc{sol}(\mathbf{d})]$.

Fix any SOL that is a walk on some spanning tree T. For any e, if e is not in T, the regret of $\mathbf{x}, \mathbf{y}$ against SOL is maximized by setting e's length to u_e. If e is in T, then SOL is paying $2d_e$ for that edge whereas the fractional solution pays $d_e x_e \leq 2d_e$, so to maximize the fractional solution's regret, d_e should be set to ℓ_e. This gives that the regret of fractional solution $\mathbf{x}$ against any SOL that is a spanning tree walk on T is

$$\sum_{e \in T} (\ell_e x_e - 2\ell_e) + \sum_{e \notin T} u_e x_e = \sum_{e \in E} u_e x_e - \sum_{e \in T} (u_e x_e - (\ell_e x_e - 2\ell_e)).$$

The quantity $\sum_{e \in E} u_e x_e$ is fixed with respect to T, so finding the spanning tree T that maximizes this quantity is equivalent to finding T that minimizes $\sum_{e \in T}(u_e x_e - (\ell_e x_e - 2\ell_e))$. But this is just an instance of the minimum spanning tree problem where edge e has weight $u_e x_e - (\ell_e x_e - 2\ell_e)$, and thus we can find T in polynomial time. After finding this spanning tree, RRTSP-Oracle checks if the regret of $\mathbf{x}, \mathbf{y}$ against the walk on T is at least r, and if so outputs this as a violated inequality. If there is some $\textsc{sol}, \mathbf{d}$ such that $\sum_{e \in E} d_e x_e > 2 \cdot \textsc{sol}(\mathbf{d}) + r$, then the regret of the fractional solution against a walk on a spanning tree contained in SOL (which has cost at most $2 \cdot \textsc{sol}(\mathbf{d})$ in realization $\mathbf{d}$) must be at least r, and thus its regret against T must also be at least r. This gives the following lemma:

$$\textbf{Minimize} \quad r \textbf{ subject to}$$

$$\forall S \subset V \text{ such that } \emptyset \subset S \cap T \subset T: \quad \textstyle\sum_{e \in \delta(S)} x_e \geq 1 \tag{4}$$

$$\forall \mathbf{d} \text{ such that } d_e \in [\ell_e, u_e]: \quad \textstyle\sum_{e \in E} d_e x_e \leq \text{OPT}(\mathbf{d}) + r \tag{5}$$

$$\forall e \in E: \quad x_e \in [0,1] \tag{6}$$

Figure 2 The Robust Steiner Tree Polytope.

▶ **Lemma 6.** *For any instance of robust traveling salesman there exists an algorithm RRTSP-Oracle that given a solution* $(\boldsymbol{x}, \boldsymbol{y}, r)$ *to* (3) *either:*

- *Outputs a separating hyperplane for* (3)*, or*
- *Outputs "Feasible", in which case* $(\boldsymbol{x}, \boldsymbol{y})$ *is feasible for the (non-robust) TSP LP and* $\forall \boldsymbol{d}: \sum_{e \in E} d_e x_e \leq 2 \cdot \text{OPT}(\boldsymbol{d}) + r$*.*

The formal description of RRTSP-Oracle and the proof of Lemma 6 are given in the full paper. By using the ellipsoid method with separation oracle RRTSP-Oracle and the fact that (3) has optimum at most MR, we get a $(2, 1)$-robust fractional solution. Applying Theorem 3 as well as the fact that the TSP polytope has integrality gap $3/2$ (see e.g. [32]) and the TSP problem has a $3/2$-approximation gives Theorem 5.

4 Algorithm for the Robust Steiner Tree Problem

In this section, our goal is to find a fractional solution to the LP in Fig. 2 for robust Steiner tree. By Theorem 3 and known approximation/integrality gap results for Steiner Tree, this gives the following theorem:

▶ **Theorem 7.** *There exists a* $(2755, 64)$*-robust algorithm for the Steiner tree problem.*

It is well-known that the standard Steiner tree polytope admits an exact separation oracle (by solving the s, t-min-cut problem using edge weights x_e for all $s, t \in T$) so it is sufficient to find an approximate separation oracle for the regret constraint set. Unlike TSP, we do not know how to leverage the approximation for STT via solving an instance of MST, since this approximation uses information about shortest paths in the STT distance which are not well-defined when the weights are unknown. In turn, a more nuanced separation oracle and analysis is required. We present the main ideas of the separation oracle here, and defer the details to the full paper.

First, we create the derived instance of the Steiner tree problem which is a copy G' of the input graph G with edge weights $u_e x_e + \ell_e - \ell_e x_e$. As noted earlier, the optimal Steiner tree T^* on the derived instance maximizes the regret of the fractional solution $\mathbf{x}$. However, since Steiner tree is **NP**-hard, we cannot hope to exactly find T^*. We need a Steiner tree $\hat{T}$ such that the regret caused by it can be bounded against that caused by T^*. The difficulty is that the regret corresponds to the total weight of edges *not* in the Steiner tree (plus an offset that we will address later), whereas standard Steiner tree approximations give guarantees in terms of the total weight of edges in the Steiner tree. We overcome this difficulty by requiring a stricter notion of "difference approximation" – that the weight of edges $\hat{T} \setminus T^*$ be bounded against those in $T^* \setminus \hat{T}$. Note that this condition ensures that not only is the weight of edges in $\hat{T}$ bounded against those in T^*, but also that the weight of edges *not in* $\hat{T}$ is bounded against that of edges *not in* T^*. We show the following lemma to obtain the difference approximation:

▶ **Lemma 8.** *For any $\epsilon > 0$, there exists a polynomial-time algorithm for the Steiner tree problem such that if $\textsc{opt}$ denotes the set of edges in the optimal solution and $c(S)$ denotes the total weight of edges in S, then for any input instance of Steiner tree, the output solution $\textsc{alg}$ satisfies $c(\textsc{alg} \setminus \textsc{opt}) \leq (4+\epsilon) \cdot c(\textsc{opt} \setminus \textsc{alg})$.*

The algorithm proving Lemma 8 is a local search procedure proposed by [16] (who considered the more general Steiner forest) that considers local moves of the following form: For the current solution $\textsc{alg}$, a local move consists of adding any path f whose endpoints are vertices in $\textsc{alg}$ and whose intermediate vertices are not in $\textsc{alg}$, and then deleting from $\textsc{alg}$ a subpath a in the resulting cycle such that $\textsc{alg} \cup f \setminus a$ remains feasible. We extend the results in [16] by showing that such an algorithm is 4-approximate for Steiner tree. We can further extend this argument to show that such an algorithm, in fact, satisfies the stricter difference approximation requirement in Lemma 8 (see the full paper for details).

Recall that the regret caused by T is not exactly the weight of edges not in T, but includes a fixed offset of $\sum_{e \in E}(\ell_e - \ell_e x_e)$. If $\ell_e = 0$ for all edges, i.e., the offset is 0, then we can recover a robust algorithm from Lemma 8 alone with much better constants than in Theorem 7 (we defer the discussion/proof of this result to the full paper). In general though, the approximation guarantee given in Lemma 8 alone does not suffice because of the offset. We instead rely on a stronger notion of approximation formalized in the next lemma that provides simultaneous approximation guarantees on two sets of edge weights: $c_e = u_e x_e - \ell_e x_e + \ell_e$ and $c'_e = \ell_e - \ell_e x_e$. The guarantee on $\ell_e - \ell_e x_e$ can then be used to handle the offset.

▶ **Lemma 9.** *Let G be a graph with terminals T and two sets of edge weights c and c'. Let $\textsc{sol}$ be any Steiner tree connecting T. Let $\Gamma' > 1$, $\kappa > 0$, and $0 < \epsilon < \frac{4}{35}$ be fixed constants. Then there exists a constant Γ (depending on $\Gamma', \kappa, \epsilon$) and an algorithm that obtains a collection of Steiner trees $\textsc{alg}$, at least one of which (let us call it $\textsc{alg}_i$) satisfies:*

- *$c(\textsc{alg}_i \setminus \textsc{sol}) \leq 4\Gamma \cdot c(\textsc{sol} \setminus \textsc{alg}_i)$, and*
- *$c'(\textsc{alg}_i) \leq (4\Gamma' + \kappa + 1 + \epsilon) \cdot c'(\textsc{sol})$.*

The fact that Lemma 9 generates multiple solutions (but only polynomially many) is fine because as long as we can show that one of these solutions causes sufficient regret, our separation oracle can just iterate over all solutions until it finds one that causes sufficient regret.

We give a high level sketch of the proof of Lemma 9 here, and defer details to the full paper. The algorithm uses the idea of *alternate minimization*, alternating between a "forward phase" and a "backward phase". The goal of each forward phase/backward phase pair is to keep $c'(\textsc{alg})$ approximately fixed while obtaining a net decrease in $c(\textsc{alg})$. In the forward phase, the algorithm greedily uses local search, choosing swaps that decrease c and increase c' at the best "rate of exchange" between the two costs (i.e., the maximum ratio of decrease in c to increase in c'), until $c'(\textsc{alg})$ has increased past some upper threshold. Then, in the backward phase, the algorithm greedily chooses swaps that decrease c' while increasing c at the best rate of exchange, until $c'(\textsc{alg})$ reaches some lower threshold, at which point we start a new forward phase.

We guess the value of $c'(\textsc{sol})$ (we can run many instances of this algorithm and generate different solutions based on different guesses for this purpose) and set the upper threshold for $c'(\textsc{alg})$ appropriately so that we satisfy the approximation guarantee for c'. For c we show that as long as $\textsc{alg}$ is not a 4Γ-difference approximation with respect to c then a forward/backward phase pair reduces $c(\textsc{alg})$ by a non-negligible amount (of course, if $\textsc{alg}$ is a 4Γ-difference approximation then we are done). This implies that after enough iterations,

$\textsc{alg}$ must be a 4Γ-difference approximation as $c(\textsc{alg})$ can only decrease by a bounded amount. To show this, we claim that while $\textsc{alg}$ is not a 4Γ-difference approximation, for sufficiently large Γ the following bounds on rates of exchange hold:

- For each swap in the forward phase, the ratio of decrease in $c(\textsc{alg})$ to increase in $c'(\textsc{alg})$ is at least some constant k_1 times $\frac{c(\textsc{alg} \setminus \textsc{sol})}{c'(\textsc{sol} \setminus \textsc{alg})}$.
- For each swap in the backward phase, the ratio of increase in $c(\textsc{alg})$ to decrease in $c'(\textsc{alg})$ is at most some constant k_2 times $\frac{c(\textsc{sol} \setminus \textsc{alg})}{c'(\textsc{alg} \setminus \textsc{sol})}$.

Before we discuss how to prove this claim, let us see why this claim implies that a forward phase/backward phase pair results in a net decrease in $c(\textsc{alg})$. If this claim holds, suppose we set the lower threshold for $c'(\textsc{alg})$ to be, say, $101c'(\textsc{sol})$. That is, throughout the backward phase, we have $c'(\textsc{alg}) > 101c'(\textsc{sol})$. This lower threshold lets us rewrite our upper bound on the rate of exchange in the backward phase in terms of the lower bound on rate of exchange for the forward phase:

$$k_2 \frac{c(\textsc{sol} \setminus \textsc{alg})}{c'(\textsc{alg} \setminus \textsc{sol})} \leq k_2 \frac{c(\textsc{sol} \setminus \textsc{alg})}{c'(\textsc{alg}) - c'(\textsc{sol})} \leq k_2 \frac{c(\textsc{sol} \setminus \textsc{alg})}{100c'(\textsc{sol})} \leq k_2 \frac{c(\textsc{sol} \setminus \textsc{alg})}{100c'(\textsc{sol} \setminus \textsc{alg})}$$

$$\leq k_2 \frac{1}{4\Gamma} \frac{c(\textsc{alg} \setminus \textsc{sol})}{100c'(\textsc{sol} \setminus \textsc{alg})} = \frac{k_2}{400\Gamma k_1} \cdot k_1 \frac{c(\textsc{alg} \setminus \textsc{sol})}{c'(\textsc{sol} \setminus \textsc{alg})}.$$

In other words, the bound in the claim for the rate of exchange in the forward phase is larger than the bound for the backward phase by a multiplicative factor of $\Omega(1) \cdot \Gamma$. While these bounds depend on $\textsc{alg}$ and thus will change with every swap, let us make the simplifying assumption that through one forward phase/backward phase pair these bounds remain constant. Then, the change in $c(\textsc{alg})$ in one phase is just the rate of exchange for that phase times the change in $c'(\textsc{alg})$, which by definition of the algorithm is roughly equal in absolute value for the forward and backward phase. So this implies that the decrease in $c(\textsc{alg})$ in the forward phase is $\Omega(1) \cdot \Gamma$ times the increase in $c(\textsc{alg})$ in the backward phase, i.e., the net change across the phases is a non-negligible decrease in $c(\textsc{alg})$ if Γ is sufficiently large. Without the simplifying assumption, we can still show that the decrease in $c(\textsc{alg})$ in the forward phase is larger than the increase in $c(\textsc{alg})$ in the backward phase for large enough Γ using a much more technical analysis. In particular, our analysis will show there is a net decrease as long as:

$$\min\left\{\frac{4\Gamma - 1}{8\Gamma}, \frac{(4\Gamma - 1)(\sqrt{\Gamma} - 1)(\sqrt{\Gamma} - 1 - \epsilon)\kappa}{16(1+\epsilon)\Gamma^2}\right\} - (e^{\zeta'(4\Gamma' + \kappa + 1 + \epsilon)} - 1) > 0, \tag{7}$$

where

$$\zeta' = \frac{4(1+\epsilon)\Gamma'}{(\sqrt{\Gamma'} - 1)(\sqrt{\Gamma'} - 1 - \epsilon)(4\Gamma' - 1)(4\Gamma - 1)}.$$

Note that for any positive $\epsilon, \kappa, \Gamma'$, there exists a sufficiently large value of Γ for (7) to hold, since as $\Gamma \to \infty$, we have $\zeta' \to 0$, so that

$$(e^{\zeta'(4\Gamma' + \kappa + 1 + \epsilon)} - 1) \to 0 \text{ and}$$

$$\min\left\{\frac{4\Gamma - 1}{8\Gamma}, \frac{(4\Gamma - 1)(\sqrt{\Gamma} - 1)(\sqrt{\Gamma} - 1 - \epsilon)\kappa}{16(1+\epsilon)\Gamma^2}\right\} \to \min\{1/2, \kappa/(4 + 4\epsilon)\}.$$

So, the same intuition holds: as long as we are willing to lose a large enough Γ value, we can make the increase in $c(\textsc{alg})$ due to the backward phase negligible compared to the decrease in the forward phase, giving us a net decrease.

It remains to argue that the claimed bounds on rates of exchange hold. Let us argue the claim for $\Gamma = 4$, although the ideas easily generalize to other choices of Γ. We do this by generalizing the analysis of the local search algorithm. This analysis shows that if $\textsc{alg}$ is a locally optimal solution, then

$$c(\textsc{alg} \setminus \textsc{sol}) \leq 4 \cdot c(\textsc{sol} \setminus \textsc{alg}),$$

i.e., $\textsc{alg}$ is a 4-difference approximation of $\textsc{sol}$. The contrapositive of this statement is that if $\textsc{alg}$ is not a 4-difference approximation, then there is at least one swap that will improve it by some amount. We modify the approach of [16] by weakening the notion of locally optimal. In particular, suppose we define a solution $\textsc{alg}$ to be "approximately" locally optimal if at least half of the (weighted) swaps between paths a in $\textsc{alg} \setminus \textsc{sol}$ and paths f in $\textsc{sol} \setminus \textsc{alg}$ satisfy $c(a) \leq 2c(f)$ (as opposed to $c(a) \leq c(f)$ in a locally optimal solution; the choice of 2 for both constants here implies $\Gamma = 4$). Then a modification of the analysis of the local search algorithm, losing an additional factor of 4, shows that if $\textsc{alg}$ is approximately locally optimal, then

$$c(\textsc{alg} \setminus \textsc{sol}) \leq 16 \cdot c(\textsc{sol} \setminus \textsc{alg}).$$

The contrapositive of this statement, however, has a stronger consequence than before: if $\textsc{alg}$ is not a 16-difference approximation, then a weighted half of the swaps satisfy $c(a) > 2c(f)$, i.e. reduce $c(\textsc{alg})$ by at least

$$c(a) - c(f) > c(a) - c(a)/2 = c(a)/2.$$

The decrease in $c(\textsc{alg})$ due to all of these swaps together is at least $c(\textsc{alg} \setminus \textsc{sol})$ times some constant. In addition, since a swap between a and f increases $c'(\textsc{alg})$ by at most $c'(f)$, the total increase in c' due to these swaps is at most $c'(\textsc{sol} \setminus \textsc{alg})$ times some other constant. An averaging argument then gives the rate of exchange bound for the forward phase in the claim, as desired. The rate of exchange bound for the backward phase follows analogously.

This completes the algorithm and proof summary, although more detail is needed to formalize these arguments. Moreover, we also need to show that the algorithm runs in polynomial time. These details are given in the full paper.

We now formally define our separation oracle RRST-Oracle in Fig. 3 and prove that it is an approximate separation oracle in the lemma below:

▶ **Lemma 10.** *Fix any $\Gamma' > 1, \kappa > 0, 0 < \epsilon < 4/35$ and let Γ be the constant given in Lemma 9. Let $\alpha = (4\Gamma' + \kappa + 2 + \epsilon)4\Gamma + 1$ and $\beta = 4\Gamma$. Then there exists an algorithm RRST-Oracle that given a solution $(\boldsymbol{x}, r)$ to the LP in Fig. 2 either:*

- *Outputs a separating hyperplane for the LP in Fig. 2, or*
- *Outputs "Feasible", in which case $\boldsymbol{x}$ is feasible for the (non-robust) Steiner tree LP and*

$$\forall \boldsymbol{d} : \sum_{e \in E} d_e x_e \leq \alpha \cdot \textsc{opt}(\boldsymbol{d}) + \beta \cdot r.$$

Proof. It suffices to show that if there exists $\mathbf{d}, \textsc{sol}$ such that

$$\sum_{e \in E} d_e x_e > \alpha \cdot \textsc{sol}(\mathbf{d}) + \beta \cdot r, \text{ i.e., } \sum_{e \in E} d_e x_e - \alpha \cdot \textsc{sol}(\mathbf{d}) > \beta \cdot r$$

then RRST-Oracle outputs a violated inequality on line 6, i.e., the algorithm finds a Steiner tree T' such that

$$\sum_{e \notin T'} u_e x_e + \sum_{e \in T'} \ell_e x_e - \sum_{e \in T'} \ell_e > r.$$

```
RRST-Oracle($G(V,E)$, $\{[\ell_e, u_e]\}_{e\in E}$, $(\mathbf{x}, r)$)
   Data: Undirected graph $G(V,E)$, lower and upper bounds on edge lengths
         $\{[\ell_e, u_e]\}_{e\in E}$, solution $(\mathbf{x} = \{x_e\}_{e\in E}, r)$ to the LP in Fig. 2
   Check all constraints of the LP in Fig. 2 except regret constraint set, return any
     violated constraint that is found;
   $G' \leftarrow$ copy of $G$ where $c_e = u_e x_e - \ell_e x_e + \ell_e$, $c'_e = \ell_e - \ell_e x_e$;
   $\textsc{alg} \leftarrow$ output of algorithm from Lemma 9 on $G'$;
   for $\textsc{alg}_i \in \textsc{alg}$ do
      if $\sum_{e\notin \textsc{alg}_i} u_e x_e + \sum_{e\in \textsc{alg}_i} \ell_e x_e - \sum_{e\in \textsc{alg}_i} \ell_e > r$ then
         return $\sum_{e\notin \textsc{alg}_i} u_e x_e + \sum_{e\in \textsc{alg}_i} \ell_e x_e - \sum_{e\in \textsc{alg}_i} \ell_e \le r$;
      end
   end
   return "Feasible";
```

Figure 3 Separation Oracle for LP in Fig. 2.

Notice that in the inequality

$$\sum_{e\in E} d_e x_e - \alpha \cdot \textsc{sol}(\mathbf{d}) > \beta \cdot r,$$

replacing $\mathbf{d}$ with $\mathbf{d}'$ where $d'_e = \ell_e$ when $e \in \textsc{sol}$ and $d'_e = u_e$ when $e \notin \textsc{sol}$ can only increase the left hand side. So replacing $\mathbf{d}$ with $\mathbf{d}'$ and rearranging terms, we have

$$\sum_{e\in\textsc{sol}} \ell_e x_e + \sum_{e\notin\textsc{sol}} u_e x_e > \alpha \sum_{e\in\textsc{sol}} \ell_e + \beta\cdot r = \sum_{e\in\textsc{sol}} \ell_e + \left[(\alpha-1)\sum_{e\in\textsc{sol}} \ell_e + \beta\cdot r\right].$$

In particular, the regret of the fractional solution against $\textsc{sol}$ is at least $(\alpha-1)\sum_{e\in\textsc{sol}} \ell_e + \beta\cdot r$.

Let T' be the Steiner tree satisfying the conditions of Lemma 9 with $c_e = u_e x_e - \ell_e x_e + \ell_e$ and $c'_e = \ell_e - \ell_e x_e$. Let $E_0 = E \setminus (\textsc{sol} \cup T')$, $E_S = \textsc{sol} \setminus T'$, and $E_T = T' \setminus \textsc{sol}$. Let $c(E')$ for $E' = E_0, E_S, E_T$ denote $\sum_{e\in E'}(u_e x_e - \ell_e x_e + \ell_e)$, i.e., the total weight of the edges E' in G'. Now, note that the regret of the fractional solution against a solution using edges E' is:

$$\sum_{e\notin E'} u_e x_e + \sum_{e\in E'} \ell_e x_e - \sum_{e\in E'} \ell_e = \sum_{e\notin E'} (u_e x_e - \ell_e x_e + \ell_e) - \sum_{e\in E} (\ell_e - \ell_e x_e)$$

$$= c(E \setminus E') - \sum_{e\in E} (\ell_e - \ell_e x_e).$$

Plugging in $E' = \textsc{sol}$, we then get that:

$$c(E_0) + c(E_T) - \sum_{e\in E} (\ell_e - \ell_e x_e) > (\alpha-1)\sum_{e\in\textsc{sol}} \ell_e + \beta\cdot r.$$

Isolating $c(E_T)$ then gives:

$$c(E_T) > (\alpha-1)\sum_{e\in\textsc{sol}} \ell_e + \beta\cdot r - \sum_{e\in E_0} (u_e x_e - \ell_e x_e + \ell_e) + \sum_{e\in E} (\ell_e - \ell_e x_e)$$

$$= (\alpha-1)\sum_{e\in\textsc{sol}} \ell_e + \beta\cdot r - \sum_{e\in E_0} u_e x_e + \sum_{e\notin E_0} (\ell_e - \ell_e x_e).$$

Since $\beta = 4\Gamma$, Lemma 9 along with an appropriate choice of ϵ gives that $c(E_T) \leq \beta c(E_S)$, and thus:

$$c(E_S) > \frac{1}{\beta}\left[(\alpha-1)\sum_{e\in\text{SOL}} \ell_e + \beta\cdot r - \sum_{e\in E_0} u_e x_e + \sum_{e\notin E_0}(\ell_e - \ell_e x_e)\right].$$

Recall that our goal is to show that $c(E_0) + c(E_S) - \sum_{e\in E}(\ell_e - \ell_e x_e) > r$, i.e., that the regret of the fractional solution against T' is at least r. Adding $c(E_0) - \sum_{e\in E}(\ell_e - \ell_e x_e)$ to both sides of the previous inequality, we can lower bound $c(E_0) + c(E_S) - \sum_{e\in E}(\ell_e - \ell_e x_e)$ as follows:

$$\begin{aligned}
&c(E_0) + c(E_S) - \sum_{e\in E}(\ell_e - \ell_e x_e)\\
&> \frac{1}{\beta}\left[(\alpha-1)\sum_{e\in\text{SOL}} \ell_e + \beta\cdot r - \sum_{e\in E_0} u_e x_e + \sum_{e\notin E_0}(\ell_e - \ell_e x_e)\right]\\
&+ \sum_{e\in E_0}(u_e x_e - \ell_e x_e + \ell_e) - \sum_{e\in E}(\ell_e - \ell_e x_e)\\
&= r + \frac{\alpha-1}{\beta}\sum_{e\in\text{SOL}} \ell_e + \frac{1}{\beta}\sum_{e\notin E_0}(\ell_e - \ell_e x_e) + \frac{\beta-1}{\beta}\sum_{e\in E_0} u_e x_e - \sum_{e\notin E_0}(\ell_e - \ell_e x_e)\\
&\geq r + \frac{\alpha-1-\beta}{\beta}\sum_{e\in\text{SOL}} \ell_e + \frac{1}{\beta}\sum_{e\notin E_0}(\ell_e - \ell_e x_e) + \frac{\beta-1}{\beta}\sum_{e\in E_0} u_e x_e - \sum_{e\in E_T}(\ell_e - \ell_e x_e) \geq r.
\end{aligned}$$

Here, the last inequality holds because by our setting of α, we have

$$\frac{\alpha-1-\beta}{\beta} = 4\Gamma' + \kappa + 1 + \epsilon,$$

and thus Lemma 9 gives that

$$\sum_{e\in E_T}(\ell_e - \ell_e x_e) \leq \frac{\alpha-1-\beta}{\beta}\sum_{e\in\text{SOL}}(\ell_e - \ell_e x_e) \leq \frac{\alpha-1-\beta}{\beta}\sum_{e\in\text{SOL}} \ell_e. \qquad ◀$$

By using Lemma 10 with the ellipsoid method and the fact that the LP optimum is at most MR, we get an (α,β)-robust fractional solution. Then, Theorem 3 and known approximation/integrality gap results give us the following theorem, which with appropriate choice of constants gives Theorem 7:

▶ **Theorem 11.** *Fix any $\Gamma' > 1, \kappa > 0, 0 < \epsilon < 4/35$ and let Γ be the constant given in Lemma 9. Let $\alpha = (4\Gamma' + \kappa + 2 + \epsilon)4\Gamma + 1$ and $\beta = 4\Gamma$. Then there exists a polynomial-time $(2\alpha\ln 4 + \epsilon, 2\beta\ln 4 + \ln 4 + \epsilon)$-robust algorithm for the Steiner tree problem.*

5 Lower Bounds

To contextualize our approximation guarantees, we give the following generalized hardness result for a family of problems which includes many graph optimization problems:

▶ **Theorem 12.** *Let $\mathcal{P}$ be any robust covering problem whose input includes a weighted graph G where the lengths d_e of the edges are given as ranges $[\ell_e, u_e]$ and for which the non-robust version of the problem, $\mathcal{P}'$, has the following properties:*

- *A solution to an instance of $\mathcal{P}'$ can be written as a (multi-)set S of edges in G, and has cost $\sum_{e \in S} d_e$.*
- *Given an input including G to $\mathcal{P}'$, there is a polynomial-time approximation-preserving reduction from solving $\mathcal{P}'$ on this input to solving $\mathcal{P}'$ on some input including G', where G' is the graph formed by taking G, adding a new vertex v^*, and adding a single edge from v^* to some $v \in V$ of weight 0.*
- *For any input including G to $\mathcal{P}'$, given any spanning tree T of G, there exists a feasible solution only including edges from T.*

Then, if there exists a polynomial time (α, β)-robust algorithm for $\mathcal{P}$, there exists a polynomial-time β-approximation algorithm for $\mathcal{P}$.

Before proving Theorem 12, we note that robust traveling salesman and robust Steiner tree are examples of problems that Theorem 12 implicitly gives lower bounds for. For both problems, the first property clearly holds.

For traveling salesman, given any input G, any solution to the problem on input G' as described in Theorem 12 can be turned into a solution of the same cost on input G by removing the new vertex v^* (since v^* was distance 0 from v, removing v^* does not affect the length of any tour), giving the second property. For any spanning tree of G, a walk on the spanning tree gives a valid TSP tour, giving the third property.

For Steiner tree, for the input with graph G' and the same terminal set, for any solution containing the edge (v, v^*) we can remove this edge and get a solution for the input with graph G that is feasible and of the same cost. Otherwise, the solution is already a solution for the input with graph G that is feasible and of the same cost, so the second property holds. Any spanning tree is a feasible Steiner tree, giving the third property.

We now give the proof of Theorem 12.

Proof of Theorem 12. Suppose there exists a polynomial time (α, β)-robust algorithm A for $\mathcal{P}$. The β-approximation algorithm for $\mathcal{P}'$ is as follows:

1. From the input instance $\mathcal{I}$ of $\mathcal{P}$ where the graph is G, use the approximation-preserving reduction (that must exist by the second property of the theorem) to construct instance $\mathcal{I}'$ of $\mathcal{P}'$ where the graph is G'.
2. Construct an instance $\mathcal{I}''$ of $\mathcal{P}$ from $\mathcal{I}'$ as follows: For all edges in G', their length is fixed to their length in $\mathcal{I}'$. In addition, we add a "special" edge from v^* to all vertices besides v with length range $[0, \infty]$[2].
3. Run A on $\mathcal{I}''$ to get a solution $\textsc{sol}$. Treat this solution as a solution to $\mathcal{I}'$ (we will show it only uses edges that appear in $\mathcal{I}$). Use the approximation-preserving reduction to convert $\textsc{sol}$ into a solution for $\mathcal{I}$ and output this solution.

Let O denote the cost of the optimal solution to $\mathcal{I}'$. Then, $\textsc{mr} \leq O$. To see why, note that the optimal solution to $\mathcal{I}'$ has cost O in all realizations of demands since it only uses edges of fixed cost, and thus its regret is at most O. This also implies that for all $\mathbf{d}$, $\textsc{opt}(\mathbf{d})$ is finite. Then for all $\mathbf{d}$, $\textsc{sol}(\mathbf{d}) \leq \alpha \cdot \textsc{opt}(\mathbf{d}) + \beta \cdot \textsc{mr}$, i.e. $\textsc{sol}(\mathbf{d})$ is finite in all realizations of demands, so $\textsc{sol}$ does not include any special edges, as any solution with a special edge has infinite cost in some realization of demands.

Now consider the realization of demands $\mathbf{d}$ where all special edges have length 0. The special edges and the edge (v, v^*) span G', so by the third property of $\mathcal{P}'$ in the theorem statement there is a solution using only cost 0 edges in this realization, i.e. $\textsc{opt}(\mathbf{d}) = 0$.

[2] We use ∞ to simplify the proof, but it can be replaced with a sufficiently large finite number. For example, the total weight of all edges in G suffices and has small bit complexity.

Then in this realization, $\textsc{sol}(\mathbf{d}) \leq \alpha \cdot \textsc{opt}(\mathbf{d}) + \beta \cdot \textsc{mr} \leq \beta \cdot O$. But since $\textsc{sol}$ does not include any special edges, and all edges besides special edges have fixed cost and their cost is the same in $\mathcal{I}''$ as in $\mathcal{I}'$, $\textsc{sol}(\mathbf{d})$ also is the cost of $\textsc{sol}$ in instance $\mathcal{I}'$, i.e. $\textsc{sol}(\mathbf{d})$ is a β-approximation for $\mathcal{I}'$. Since the reduction from $\mathcal{I}$ to $\mathcal{I}'$ is approximation-preserving, we also get a β-approximation for $\mathcal{I}$. ◀

From [11, 22] we then get the following hardness results:

▶ **Corollary 13.** *Finding an (α, β)-robust solution for Steiner tree where $\beta < 96/95$ is NP-hard.*

▶ **Corollary 14.** *Finding an (α, β)-robust solution for TSP where $\beta < 121/120$ is NP-hard.*

6 Conclusion

In this paper, we designed constant approximation algorithms for the robust Steiner tree and traveling salesman problems. To the best of our knowledge, this is the first instance of robust polynomial-time algorithms being developed for **NP**-complete graph problems. While our approximation bounds for TSP are small constants, that for STT are very large constants. A natural question is whether these constants can be made smaller, e.g. of the same scale as classic approximation bounds for STT. While we did not seek to optimize our constants, obtaining truly small constants for STT appears to be beyond our techniques, and is an interesting open question. More generally, robust algorithms are a key component in the area of optimization under uncertainty that is of much practical and theoretical significance. We hope that our work will lead to more research in robust algorithms for other fundamental problems in combinatorial optimization, particularly in graph algorithms.

References

1 H. Aissi, C. Bazgan, and D. Vanderpooten. Complexity of the min–max (regret) versions of min cut problems. *Discrete Optimization*, 5(1):66–73, 2008. doi:10.1016/j.disopt.2007.11.008.

2 Hassene Aissi, Cristina Bazgan, and Daniel Vanderpooten. Min–max and min–max regret versions of combinatorial optimization problems: A survey. *European Journal of Operational Research*, 197(2):427–438, 2009. doi:10.1016/j.ejor.2008.09.012.

3 I. Averbakh and Oded Berman. Minimax regret p-center location on a network with demand uncertainty. *Location Science*, 5(4):247–254, 1997. doi:10.1016/S0966-8349(98)00033-3.

4 Igor Averbakh. On the complexity of a class of combinatorial optimization problems with uncertainty. *Mathematical Programming*, 90(2):263–272, April 2001. doi:10.1007/PL00011424.

5 Igor Averbakh. The minmax relative regret median problem on networks. *INFORMS Journal on Computing*, 17(4):451–461, 2005. doi:10.1287/ijoc.1040.0080.

6 Igor Averbakh and Oded Berman. Minmax regret median location on a network under uncertainty. *INFORMS Journal on Computing*, 12(2):104–110, 2000. doi:10.1287/ijoc.12.2.104.11897.

7 Dimitris Bertsimas and Melvyn Sim. Robust discrete optimization and network flows. *Mathematical Programming*, 98(1):49–71, September 2003. doi:10.1007/s10107-003-0396-4.

8 Moses Charikar, Chandra Chekuri, and Martin Pál. Sampling bounds for stochastic optimization. In *Proceedings of the 8th International Workshop on Approximation, Randomization and Combinatorial Optimization Problems, and Proceedings of the 9th International Conference on Randamization and Computation: Algorithms and Techniques*, APPROX'05/RANDOM'05, pages 257–269, Berlin, Heidelberg, 2005. Springer-Verlag. doi:10.1007/11538462_22.

9 André Chassein and Marc Goerigk. On the recoverable robust traveling salesman problem. *Optimization Letters*, 10, September 2015. doi:10.1007/s11590-015-0949-5.

10 Chandra Chekuri. Routing and network design with robustness to changing or uncertain traffic demands. *SIGACT News*, 38(3):106–129, 2007. doi:10.1145/1324215.1324236.

11 Miroslav Chlebík and Janka Chlebíková. Approximation hardness of the Steiner tree problem on graphs. In Martti Penttonen and Erik Meineche Schmidt, editors, *Algorithm Theory — SWAT 2002*, pages 170–179, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg. doi:10.1007/3-540-45471-3_18.

12 Eduardo Conde. On a constant factor approximation for minmax regret problems using a symmetry point scenario. *European Journal of Operational Research*, 219(2):452–457, 2012. doi:10.1016/j.ejor.2012.01.005.

13 Kedar Dhamdhere, Vineet Goyal, R. Ravi, and Mohit Singh. How to pay, come what may: Approximation algorithms for demand-robust covering problems. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2005), 23-25 October 2005, Pittsburgh, PA, USA, Proceedings*, pages 367–378, 2005. doi:10.1109/SFCS.2005.42.

14 Uriel Feige, Kamal Jain, Mohammad Mahdian, and Vahab S. Mirrokni. Robust combinatorial optimization with exponential scenarios. In *Integer Programming and Combinatorial Optimization, 12th International IPCO Conference, Ithaca, NY, USA, June 25-27, 2007, Proceedings*, pages 439–453, 2007. doi:10.1007/978-3-540-72792-7_33.

15 Navin Goyal, Neil Olver, and F. Bruce Shepherd. The VPN conjecture is true. *J. ACM*, 60(3):17:1–17:17, 2013. doi:10.1145/2487241.2487243.

16 Martin Groß, Anupam Gupta, Amit Kumar, Jannik Matuschke, Daniel R. Schmidt, Melanie Schmidt, and José Verschae. A local-search algorithm for Steiner forest. In *9th Innovations in Theoretical Computer Science Conference, ITCS 2018, January 11-14, 2018, Cambridge, MA, USA*, pages 31:1–31:17, 2018. doi:10.4230/LIPIcs.ITCS.2018.31.

17 Anupam Gupta, Jon M. Kleinberg, Amit Kumar, Rajeev Rastogi, and Bülent Yener. Provisioning a virtual private network: a network design problem for multicommodity flow. In *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece*, pages 389–398, 2001. doi:10.1145/380752.380830.

18 Anupam Gupta, Viswanath Nagarajan, and R. Ravi. Thresholded covering algorithms for robust and max-min optimization. *Math. Program.*, 146(1-2):583–615, 2014. doi:10.1007/s10107-013-0705-5.

19 Anupam Gupta, Viswanath Nagarajan, and R. Ravi. Robust and maxmin optimization under matroid and knapsack uncertainty sets. *ACM Trans. Algorithms*, 12(1):10:1–10:21, 2016. doi:10.1145/2746226.

20 Anupam Gupta, Martin Pál, R. Ravi, and Amitabh Sinha. Boosted sampling: Approximation algorithms for stochastic optimization. In *Proceedings of the Thirty-sixth Annual ACM Symposium on Theory of Computing*, STOC '04, pages 417–426, New York, NY, USA, 2004. ACM. doi:10.1145/1007352.1007419.

21 Masahiro Inuiguchi and Masatoshi Sakawa. Minimax regret solution to linear programming problems with an interval objective function. *European Journal of Operational Research*, 86(3):526–536, 1995. doi:10.1016/0377-2217(94)00092-Q.

22 Marek Karpinski, Michael Lampis, and Richard Schmied. New inapproximability bounds for TSP. In Leizhen Cai, Siu-Wing Cheng, and Tak-Wah Lam, editors, *Algorithms and Computation*, pages 568–578, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. doi:10.1016/j.jcss.2015.06.003.

23 Adam Kasperski and PawełZieliński. An approximation algorithm for interval data minmax regret combinatorial optimization problems. *Inf. Process. Lett.*, 97(5):177–180, March 2006. doi:10.1016/j.ipl.2005.11.001.

24 Adam Kasperski and Pawel Zieliński. On the existence of an FPTAS for minmax regret combinatorial optimization problems with interval data. *Oper. Res. Lett.*, 35:525–532, 2007. doi:10.1016/j.orl.2006.09.007.

25 Rohit Khandekar, Guy Kortsarz, Vahab S. Mirrokni, and Mohammad R. Salavatipour. Two-stage robust network design with exponential scenarios. *Algorithmica*, 65(2):391–408, 2013. doi:10.1007/s00453-011-9596-0.

26 P. Kouvelis and G. Yu. *Robust Discrete Optimization and Its Applications*. Springer US, 1996.

27 Panos Kouvelis and Gang Yu. *Robust 1-Median Location Problems: Dynamic Aspects and Uncertainty*, pages 193–240. Springer US, Boston, MA, 1997. doi:10.1007/978-1-4757-2620-6_6.

28 Helmut E. Mausser and Manuel Laguna. A new mixed integer formulation for the maximum regret problem. *International Transactions in Operational Research*, 5(5):389–403, 1998. doi:10.1016/S0969-6016(98)00023-9.

29 David B. Shmoys and Chaitanya Swamy. An approximation scheme for stochastic linear programming and its application to stochastic integer programs. *J. ACM*, 53(6):978–1012, November 2006. doi:10.1145/1217856.1217860.

30 Chaitanya Swamy and David B. Shmoys. Approximation algorithms for 2-stage stochastic optimization problems. *SIGACT News*, 37(1):33–46, 2006. doi:10.1145/1122480.1122493.

31 Chaitanya Swamy and David B. Shmoys. Sampling-based approximation algorithms for multistage stochastic optimization. *SIAM J. Comput.*, 41(4):975–1004, 2012. doi:10.1137/100789269.

32 Jens Vygen. New approximation algorithms for the tsp.

33 H. Yaman, O. E. Karaşan, and M. Ç. Pinar. The robust spanning tree problem with interval data. *Operations Research Letters*, 29(1):31–40, 2001. doi:10.1016/S0167-6377(01)00078-5.

34 P. Zieliński. The computational complexity of the relative robust shortest path problem with interval data. *European Journal of Operational Research*, 158(3):570–576, 2004. doi:10.1016/S0377-2217(03)00373-4.

Cryptographic Reverse Firewalls for Interactive Proof Systems

Chaya Ganesh
Department of Computer Science and Automation, Indian Institute of Science, Bangalore, India
chaya@iisc.ac.in

Bernardo Magri
Department of Computer Science, Aarhus University, Denmark
magri@cs.au.dk

Daniele Venturi
Department of Computer Science, Sapienza University of Rome, Italy
venturi@di.uniroma1.it

Abstract

We study interactive proof systems (IPSes) in a strong adversarial setting where the machines of *honest parties* might be corrupted and under control of the adversary. Our aim is to answer the following, seemingly paradoxical, questions:

- Can Peggy convince Vic of the veracity of an NP statement, without leaking any information about the witness even in case Vic is malicious and Peggy does not trust her computer?
- Can we avoid that Peggy fools Vic into accepting false statements, even if Peggy is malicious and Vic does not trust her computer?

At EUROCRYPT 2015, Mironov and Stephens-Davidowitz introduced cryptographic reverse firewalls (RFs) as an attractive approach to tackling such questions. Intuitively, a RF for Peggy/Vic is an external party that sits between Peggy/Vic and the outside world and whose scope is to sanitize Peggy's/Vic's incoming and outgoing messages in the face of subversion of her/his computer, e.g. in order to destroy subliminal channels.

In this paper, we put forward several natural security properties for RFs in the concrete setting of IPSes. As our main contribution, we construct efficient RFs for different IPSes derived from a large class of Sigma protocols that we call *malleable*.

A nice feature of our design is that it is completely transparent, in the sense that our RFs can be directly applied to already deployed IPSes, without the need to re-implement them.

2012 ACM Subject Classification Security and privacy → Cryptography

Keywords and phrases Subversion, Algorithm substitution attacks, Cryptographic reverse firewalls, Interactive proofs, Zero knowledge

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.55

Category Track A: Algorithms, Complexity and Games

Related Version A full version of the paper is available at `https://eprint.iacr.org/2020/204`.

Funding *Bernardo Magri*: This work was supported by Concordium Blockchain Research Center, Aarhus University, Denmark.

1 Introduction

An interactive proof system (IPS) $\Pi = (\mathsf{P}, \mathsf{V})$ allows a prover P to convince a verifier V about the veracity of a public statement $x \in \mathcal{L}$, where $\mathcal{L}$ is an NP language and where both P and V are modeled as interactive PPT machines. The prover is facilitated by possessing a witness w to the fact that, indeed, $x \in \mathcal{L}$, and the interaction with the verifier may consist of several rounds of communication, at the end of which the verifier outputs a verdict on the membership of x in $\mathcal{L}$.

In order to be useful, an IPS should satisfy the following properties:

- *Completeness:* If $x \in \mathcal{L}$, the honest prover (almost) always convinces the honest verifier.
- *Soundness:* If $x \notin \mathcal{L}$, no (computationally bounded) malicious prover can convince the honest verifier that $x \in \mathcal{L}$. An even stronger guarantee, known as *knowledge soundness* [9], is to require that the only way a prover can convince the honest verifier that $x \in \mathcal{L}$ is to "know" a valid witness w corresponding to x. Such proofs[1] are called *proofs of knowledge* (PoKs).
- *Zero Knowledge (ZK):* A valid proof reveals nothing beyond the fact that $x \in \mathcal{L}$, and thus in particular it leaks no information about the witness w, even in case the proof is conducted in the presence of a (computationally bounded) malicious verifier [36]. A weaker guarantee, known as *witness indistinguishability* (WI) [24], is that, whenever there are multiple witnesses attesting that $x \in \mathcal{L}$, no (computationally bounded) malicious verifier can distinguish whether a proof is conducted using either of two witnesses.

One of the motivations for studying IPSes with the above properties is that they are ubiquitous in cryptography, with applications ranging from identification protocols [24], blind digital signatures [42], and electronic voting [16], to general-purpose maliciously secure multi-party computation [35].

1.1 Sigma Protocols

While WI/ZK PoKs exist for all of NP, based on minimal cryptographic assumptions [23, 34, 33], efficiency is a different story. Fortunately, it is possible to design practical interactive proofs for specific languages, typically in the form of so-called Sigma protocols. Briefly, a Sigma protocol is a special type of IPS consisting of just three rounds, where the prover sends a first message α (the commitment), the verifier sends a random string β (the challenge), and finally the prover forwards a last message γ (the response). Sigma protocols satisfy two main properties: The first one, known as *special soundness*, is a strong form of knowledge soundness; the second one, known as *honest-verifier zero knowledge* (HVZK), is a weak form of the zero knowledge property that only holds against honest-but-curious verifiers.

The applications of Sigma protocols to cryptographic constructions are countless (see, e.g., [25, 17, 48, 22, 43]). These results are perhaps surprising, as Sigma protocols only satisfy HVZK and thus guarantee no security in the presence of malicious verifiers. In some cases, the solution to this apparent paradox is due to a beautiful technique put forward by Cramer, Damgård, and Schoenmakers [15], which allows to add WI to any Sigma protocol. Moreover, it is relatively easy to transform any Sigma protocol into an interactive ZK PoK at the cost of adding a single round of interaction [33].

1.2 Our Question

The standard definitions of security for IPSes (implicitly) rely on the assumption that honest parties can fully trust their machines. In practice, however, such an assumption may just be too optimistic, as witnessed by the revelations of Edward Snowden about subversion of cryptographic standards [45, 7], and in light of the numerous (seemingly accidental) bugs in widespread pieces of cryptographic software [38, 1, 2].

[1] Sometimes, the term "proof" is used to refer to statistically sound IPSes, while computationally sound IPSes are typically called "arguments".

Motivated by the above incidents, we ask the following question which constitutes the main source of inspiration for this work:

> *Can we design practical interactive proofs that remain secure even if the machines of the honest parties running them have been tampered with?*

In order to see why the above question is well motivated and not trivial, let us analyze the dramatic consequences of subverting the prover of ZK IPSes. Clearly, the problem of subversion-resistant interactive zero knowledge is just impossible in its utmost generality, as a subverted prover could just reveal the witness to the verifier. However, one may argue that these kind of attacks are easily detectable, and thus can be avoided.

The problem becomes more interesting if we restrict the subversion to be *undetectable*, as suggested by Bellare, Paterson, and Rogaway [11] in their seminal work on subversion of symmetric encryption, where the authors show how to subvert any sufficiently randomized cipher in an undetectable manner, using rejection sampling. A moment of reflection shows that their attack can be adapted to the case of IPSes.[2] The solution proposed by [11] is to rely on deterministic symmetric encryption. Unfortunately, this approach is not viable for the case of IPSes, as it is well-known that interactive proofs with deterministic provers can be zero knowledge only for trivial languages [32, §4.5].

Reverse firewalls

The above described undetectable attacks show that the problem of designing IPSes that remain secure even when run on untrusted machines is simply impossible if we are not willing to make any further assumption. In this paper, we study how to tackle subversion attacks against interactive proofs in the framework of "cryptographic reverse firewalls (RFs)", introduced by Mironov and Stephens-Davidowitz [40]. In such a setting, both the prover and the verifier are equipped with their own RF W, also modeled as an interactive PPT machine, whose scope is solely to sanitize the parties' incoming and outgoing messages in the face of subversion.

Importantly, neither the prover nor the verifier put any trust in the RF, meaning that they are not allowed to share secrets with the firewall itself. The hope is that an uncorrupted[3] RF can provide meaningful security guarantees even in case the honest prover's and/or verifier's machines have been tampered with. Note that a RF can never "create security", as it does not even know the inputs to the protocol, but at best can preserve the security guarantees satisfied by the initial IPS. At the same time, the RF should not ruin the functionality of the underlying IPS, in the sense that the sanitized IPS should still work in case no subversion takes place.

Mironov and Stephens-Davidowitz construct general-purpose RFs that can be used in order to preserve both functionality and security of any two-party protocol. It is important to note that since ZK/WI IPSes are a special case of secure two-party computation, their RF constructions already seem to solve our problem.[4] However, the solutions in [40] are not

[2] In particular, a subverted prover with a hardwired secret key k for a pseudorandom function $F_k(\cdot)$, could sample the random coins $r^{(i)}$ needed to generate the honest prover's message $m^{(i)}$ (for round $i \in \mathbb{N}$) multiple times, until $F_k(m^{(i)})$ leaks one bit of the witness. This attack works provided that at least one of the prover's messages has high-enough min-entropy.

[3] Clearly, if both the machine of the honest party and the firewall are corrupted, there is no hope for security. On the other hand, in case the machine is honest and the firewall is corrupt, the underlying protocol is still secure, since we can simply think of the RF as being part of the adversary [21].

[4] At least to some extent, since, strictly speaking, their results for IPSes are incomparable to ours. We refer the reader to §5.1 for more details.

practical. In particular, one of their RFs increases the round complexity of the initial IPS, and, more importantly, it requires to carry out the underlying IPS in the encrypted domain, thus requiring to completely change the original protocol. In contrast, we seek constructions of RFs that can be applied directly to existing IPSes, without adding any overhead, and without the need to re-implement them.

2 Reverse Firewalls for Interactive Proofs

In this section, we give security definitions for RFs applied to IPSes. Our notions can be seen as special cases of the generic framework by Mironov and Stephens-Davidowitz [40], who defined security of RFs for the more general case of arbitrary two-party protocols.

Let $\Pi = (\mathsf{P}, \mathsf{V})$ be an IPS for a relation $\mathcal{R}$. A cryptographic reverse firewall is an external party W that can be attached either to the prover P or to the verifier V, whose scope is to sanitize incoming and outgoing messages in the face of parties' subversion. Importantly, the RF is allowed to keep its own state but cannot share state with any of the parties. Similarly to [40], we model an interactive Turing machine M as a triple of algorithms $\mathsf{M} := (\mathsf{M}_{\mathsf{nxt}}, \mathsf{M}_{\mathsf{rec}}, \mathsf{M}_{\mathsf{out}})$ specified as follows: (i) Algorithm $\mathsf{M}_{\mathsf{nxt}}$ takes as input the current state and outputs the next message to be sent; (ii) Algorithm $\mathsf{M}_{\mathsf{rec}}$ takes as input an incoming message, and updates the state; (iii) Algorithm $\mathsf{M}_{\mathsf{out}}$ takes as input the final state at the completion of the protocol, and returns a bit.

▶ **Definition 1** (RF for IPSes). *Let $\Pi = (\mathsf{P}, \mathsf{V})$ be an IPS for a relation $\mathcal{R}$. A cryptographic reverse firewall (RF) for Π is a stateful algorithm W that takes as input a message, its state, and outputs a sanitized message, together with an updated state. For an interactive Turing machine $\mathsf{M} = (\mathsf{M}_{\mathsf{nxt}}, \mathsf{M}_{\mathsf{rec}}, \mathsf{M}_{\mathsf{out}}) \in \{\mathsf{P}, \mathsf{V}\}$, and RF W, the sanitized machine $\mathsf{W} \circ \mathsf{M} := \widehat{\mathsf{M}} = (\widehat{\mathsf{M}}_{\mathsf{nxt}}, \widehat{\mathsf{M}}_{\mathsf{rec}}, \widehat{\mathsf{M}}_{\mathsf{out}})$ is specified as follows:*

$$\begin{aligned}\widehat{\mathsf{M}}_{\mathsf{nxt}}(\sigma) &:= \mathsf{W}(\mathsf{M}_{\mathsf{nxt}}(\sigma))\\ \widehat{\mathsf{M}}_{\mathsf{rec}}(\sigma, m) &:= \mathsf{M}_{\mathsf{rec}}(\sigma, \mathsf{W}(m))\\ \widehat{\mathsf{M}}_{\mathsf{out}}(\sigma) &:= \mathsf{M}_{\mathsf{out}}(\sigma).\end{aligned}$$

As our first contribution, we put forward several natural properties that a RF for an IPS might satisfy. In particular, we consider the following notions (see the full version [29] for more formal definitions).

- *Completeness preservation:* The sanitized IPS (i.e., the IPS obtained by sanitizing both the honest prover's and the honest verifier's messages) still satisfies completeness.
- *Strong soundness preservation:* Whenever $x \notin \mathcal{L}$, no malicious prover can convince the verifier that $x \in \mathcal{L}$, even if the verifier's implementation has been arbitrarily subverted.
- *Strong ZK preservation:* A valid proof reveals nothing beyond the fact that $x \in \mathcal{L}$, even in case the proof is conducted in the presence of a malicious verifier talking to a prover whose implementation has been arbitrarily subverted.
- *Strong WI preservation:* Whenever there are multiple witnesses attesting that $x \in \mathcal{L}$, no malicious verifier talking to a prover whose implementation has been arbitrarily subverted can distinguish whether a proof is conducted using either of two witnesses.
- *Strong exfiltration resistance for the prover (resp. verifier):* Transcripts produced by running the sanitized IPS in the presence of a malicious verifier (resp. prover) talking to a prover (resp. verifier) whose implementation has been arbitrarily subverted are indistinguishable to transcripts produced by running the sanitized IPS in the presence of a malicious verifier (resp. prover) talking to the honest prover (resp. verifier).

For each of the above properties (except for completeness), we also consider a weak variant which only holds w.r.t. *functionality-maintaining* provers/verifiers. Intuitively, a prover is functionality maintaining if, upon input a valid statement/witness pair, it still convinces the honest verifier with overwhelming probability. Similarly, a verifier is functionality maintaining if, upon input a valid statement, it still accepts with overwhelming probability in a protocol run with the honest prover.

What is possible and what is impossible

A moment of reflection shows that soundness preservation is impossible to achieve. In fact, an arbitrarily subverted verifier might always[5] output one, thus automatically accepting both true and false statements. Such a verifier is still functionality maintaining,[6] and thus this simple attack even rules out *weak* soundness preservation. One way to circumvent this impossibility would be to only consider *partial subversion*, i.e. split the verifier into two components, one for computing the next messages in the protocol, and the other one for determining the final verdict on the veracity of a statement; hence, assume the latter component to be untamperable.

Turning to subversion of the prover, consider the subverted prover that always outputs the all-zero string. The soundness property of the underlying IPS implies that, for any RF and for any *false* statement $x \notin \mathcal{L}$, a sanitized transcript in this case can never be accepting. Moreover, assuming the language $\mathcal{L}$ is non-trivial, the latter holds true even in case x is a *true* statement, which in turn rules out strong exfiltration resistance. For similar reasons, strong ZK/WI preservation are also impossible to achieve.

3 Firewall Constructions from Malleable Sigma Protocols

As our second contribution, we formalize a class of Sigma protocols which admit simple, and very efficient, RFs for the prover. (See the full version [29] for similar constructions dealing with functionality-maintaining subversion of the verifier.) The main idea is to use the RF to re-randomize the prover's messages, in order to destroy any potential subliminal channel signaling information about the witness. The difficulty, though, is that such re-randomization must be carried out without knowing a witness, and while at the same time preserving the completeness property of the underlying IPS.

For the sake of concreteness, let us describe our firewall applied to the classical Sigma protocol for proving knowledge of a discrete logarithm [49]. Here, the statement consists of a description of a cyclic group $\mathbb{G}$ with generator g and prime order q, together with a value $x \in \mathbb{G}$ such that $x = g^w$ for some $w \in \mathbb{Z}_q$. The prover's first message is a random group element $\alpha = g^a \in \mathbb{G}$. Finally, the prover's last message is $\gamma = a - w \cdot \beta$, where $\beta \in \mathbb{Z}_q$ is the verifier's challenge; the verifier accepts (α, β, γ) if and only if $g^\gamma = \alpha \cdot x^{-\beta}$. Our RF sanitizes the messages α and γ from a possibly subverted implementation of the prover as follows:

$$\begin{aligned} \widehat{\alpha} &= \alpha \cdot g^\sigma \\ \widehat{\gamma} &= \gamma + \sigma, \end{aligned}$$

for random $\sigma \in \mathbb{Z}_q$. Note that $g^{\widehat{\gamma}} = g^a \cdot g^\sigma \cdot x^{-\beta} = \widehat{\alpha} \cdot x^{-\beta}$, and thus the RF preserves completeness.

[5] If one insists on undetectability, the subverted verifier may output 1 upon some hard-wired, randomly chosen, false statement $\overline{x} \notin \mathcal{L}$.

[6] The latter is because completeness is a guarantee that only concerns true statements.

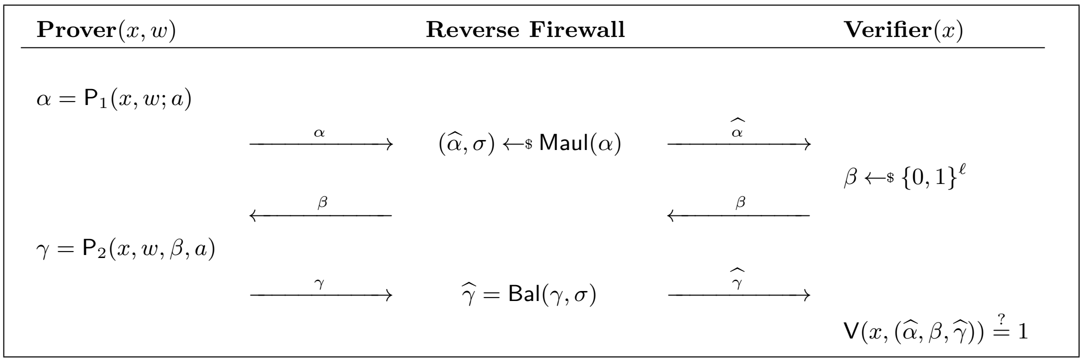

Prover(x, w)
Reverse Firewall
Verifier(x)
$\alpha = \mathsf{P}_1(x, w; a)$
α
$(\widehat{\alpha}, \sigma) \leftarrow\!\!\$\ \mathsf{Maul}(\alpha)$
$\widehat{\alpha}$
$\beta \leftarrow\!\!\$\ \{0,1\}^\ell$
β
β
$\gamma = \mathsf{P}_2(x, w, \beta, a)$
γ
$\widehat{\gamma} = \mathsf{Bal}(\gamma, \sigma)$
$\widehat{\gamma}$
$\mathsf{V}(x, (\widehat{\alpha}, \beta, \widehat{\gamma})) \overset{?}{=} 1$

Figure 1 Cryptographic reverse firewall for a malleable Sigma protocol.

We now sketch the proof of weak HVZK preservation. Observe that for any $\widetilde{\alpha} = g^{\widetilde{a}}$ sent by a functionality-maintaining subverted prover, the distribution of $\widehat{\alpha} = g^{\widetilde{a}+\sigma}$ is uniform over $\mathbb{G}$ and independent of $\widetilde{\alpha}, \widetilde{a}$, and in fact it is identical to the distribution of α in an honest run of the original Sigma protocol (without the firewall). As for $\widehat{\gamma}$, note that if there would be two possible values γ, γ' which make both $\tau = (\alpha, \beta, \gamma)$ and $\tau' = (\alpha, \beta, \gamma')$ valid transcripts, the choice of which response to pick could be used by a functionality-maintaining subverted prover as a subliminal channel signaling information about the witness. Hence, we exploit the fact that for any prefix α, β, there exists a unique response γ such that the verifier accepts upon input x and (α, β, γ).

It follows that the distribution of $\widehat{\gamma}$ is identical to that of γ in an honest run of the original Sigma protocol (without the firewall). Putting it all together, we have shown that the distribution of a sanitized transcript $\widehat{\tau} = (\widehat{\alpha}, \beta, \widehat{\gamma})$ is identical to the distribution of an honest transcript $\tau = (\alpha, \beta, \gamma)$. Thus, weak HVZK preservation follows by the fact that Schnorr's Sigma protocol is HVZK.

3.1 HVZK Preservation

Let us now explain how to generalize the above idea to a large class of Sigma protocols that we call *malleable*. In what follows, given a Sigma protocol $\Sigma = (\mathsf{P}, \mathsf{V})$, we denote by P_1 and P_2 the algorithms that compute, respectively, the first prover's message α, and the last prover's message (response) γ. The challenge space is represented[7] as $\{0,1\}^\ell$, so that there are 2^ℓ possible challenges, and we write V for the algorithm that the verifier runs upon statement x and transcript τ to make its final decision. Let $\mathcal{A}$ be the space of all possible prover's first messages; we assume that membership in $\mathcal{A}$ can be tested efficiently, so that V always outputs $\perp$ whenever $\alpha \notin \mathcal{A}$.

As for the case of Schnorr's Sigma protocol, an additional requirement that we need is that the prover's responses are unique, meaning that for all $x \in \mathcal{L}$, and for any $\alpha \in \mathcal{A}$ and $\beta \in \{0,1\}^\ell$, there exists at most one[8] value γ such that $\mathsf{V}(x, (\alpha, \beta, \gamma)) = 1$.

Intuitively, a Sigma protocol is malleable if there exists an efficient algorithm Maul for randomizing the prover's first message α into a value $\widehat{\alpha}$ which is distributed identically to the first message of an honest prover. Moreover, for any challenge β, given the coins used to randomize α and any response γ yielding a valid transcript $\tau = (\alpha, \beta, \gamma)$, there exists an

[7] In the case of Schnorr's Sigma protocol, the challenge space is a cyclic group. However, we can embed such group in $\{0,1\}^\ell$ for some $\ell \in \mathbb{N}$.

[8] This property is met by many natural Sigma protocols, and was already considered in several previous works [26, 22, 51].

Prover(x, w)		**Verifier**(x)
$pk \leftarrow\$ \mathsf{Gen}(1^\lambda)$	$\xrightarrow{pk}$	
		$\beta \leftarrow\$ \{0,1\}^\ell$ $(c, d) \leftarrow\$ \mathsf{Com}(pk, \beta)$
	$\xleftarrow{c}$	
$\alpha = \mathsf{P}_1(x, w; a)$		
	$\xrightarrow{\alpha}$	
	$\xleftarrow{d}$	
$\beta = \mathsf{Open}(pk, c, d)$ If $\beta \neq \bot$, then $\gamma = \mathsf{P}_2(x, w, \alpha, \beta, a)$ Else, $\gamma = \bot$		
	$\xrightarrow{\gamma}$	
		$\mathsf{V}(x, (\alpha, \beta, \gamma)) \overset{?}{=} 1$

Figure 2 Sigma protocol compiled with standard techniques to obtain full zero knowledge.

efficient algorithm Bal for computing a balanced response $\widehat{\gamma}$ such that $(\widehat{\alpha}, \beta, \widehat{\gamma})$ is also valid. As we show in the full version [29], many natural Sigma protocols are already malleable. In particular, the latter holds true for Maurer's unifying protocol [39], which includes the protocols by Fiat-Shamir [25], Guillou-Quisquater [37], Schnorr [49], Okamoto [41], and many others as special cases.

Our RF construction is depicted in Fig. 1. Intuitively, the firewall uses the malleability property of the underlying Sigma protocol in order to re-randomize the prover's first and last messages, in such a way that a functionality-maintaining subverted prover cannot signal information about the witness through them. The theorem below, whose proof appears in the full version [29], establishes its security.

▶ **Theorem 2.** *Let $\Sigma = (\mathsf{P} = (\mathsf{P}_1, \mathsf{P}_2), \mathsf{V})$ be a malleable Sigma protocol with unique responses, for a relation $\mathcal{R}$. The RF W of Fig. 1 preserves completeness, and is weakly HVZK preserving for the prover.*

3.2 ZK Preservation

As Sigma protocols are not in general zero knowledge, there is no hope to prove that the above firewall weakly preserves ZK. However, a standard trick [33] allows to transform any Sigma protocol into a 5-round IPS satisfying ZK. The idea is to let the prover send the public key pk of a commitment scheme $(\mathsf{Gen}, \mathsf{Com}, \mathsf{Open})$ during the first round. Then, during the second round, the verifier forwards to the prover a commitment c to the challenge β. Finally, the Sigma protocol is executed as before with the difference that the verifier also needs to open the commitment, with the prover aborting if the opening is invalid. We depict such a modified protocol in Fig. 2.

In order to build a RF for this IPS, we need to sanitize the additional messages from the (possibly subverted, but functionality-maintaining) prover.[9] We do so by relying on a special type of *key-malleable* commitment, which intuitively allows to maul any public key pk (via an algorithm $\mathsf{MaulKey}$) into a uniformly random public key $\widehat{pk}$, in such a way that, given a commitment c with opening d w.r.t. $\widehat{pk}$, it is possible to map (c, d) into a commitment

[9] The other messages are sanitized as before, i.e. we still exploit the fact that the underlying Sigma protocol is malleable.

Prover(x,w)		**ZK Reverse Firewall**		**Verifier**(x)
$pk \leftarrow\$ \mathsf{Gen}(1^\lambda)$	$\xrightarrow{pk}$	$(\widehat{pk},\rho) \leftarrow\$ \mathsf{MaulKey}(pk)$	$\xrightarrow{\widehat{pk}}$	
				$\beta \leftarrow\$ \{0,1\}^\ell$ $(c,d) \leftarrow\$ \mathsf{Com}(\widehat{pk},\beta)$
	$\xleftarrow{\widehat{c}}$	$\widehat{c} \leftarrow\$ \mathsf{MaulCom}(pk,c,\rho)$	$\xleftarrow{c}$	
$\alpha = \mathsf{P}_1(x,w;a)$				
	$\xrightarrow{\alpha}$	$(\widehat{\alpha},\sigma) \leftarrow\$ \mathsf{Maul}(\alpha)$	$\xrightarrow{\widehat{\alpha}}$	
	$\xleftarrow{\widehat{d}}$	$\widehat{d} = \mathsf{BalOpen}(d,\rho)$	$\xleftarrow{d}$	
$\beta = \mathsf{Open}(pk,\widehat{c},\widehat{d})$ If $\beta \neq \perp$, then $\gamma = \mathsf{P}_2(x,w,\beta,a)$				
	$\xrightarrow{\gamma}$			
		$\beta = \mathsf{Open}(pk,\widehat{c},\widehat{d})$ If $\beta = \perp$, then $\widehat{\gamma} = \perp$ Else, $\widehat{\gamma} = \mathsf{Bal}(\gamma,\sigma)$		
			$\xrightarrow{\widehat{\gamma}}$	
				$\mathsf{V}(x,(\widehat{\alpha},\beta,\widehat{\gamma})) \overset{?}{=} 1$

Figure 3 Prover's RF for the protocol in Fig. 2.

$\widehat{c}$ with opening $\widehat{d}$ w.r.t. pk, without changing the message inside the commitment. We denote by $\mathsf{MaulCom}$ and $\mathsf{BalOpen}$, respectively, the algorithms for mauling the commitment c and the opening d, and additionally require that the distribution of mauled public keys and commitments is identical, respectively, to that of honestly computed public keys and commitments. As we show in the full version [29], the standard Pedersen's commitment [44] is easily seen to be key malleable, thus yielding a concrete instantiation under the Discrete Logarithm assumption.

Our RF for the protocol of Fig. 2 is depicted in Fig. 3. The theorem below, whose proof appears in the full version [29], establishes its security.

▶ **Theorem 3.** *Let $\Sigma = (\mathsf{P} = (\mathsf{P}_1, \mathsf{P}_2), \mathsf{V})$ be a malleable Sigma protocol with unique responses, for a relation $\mathcal{R}$. Let $\Gamma = (\mathsf{Gen}, \mathsf{Com}, \mathsf{Open})$ be a key-malleable commitment scheme with message space $\{0,1\}^\ell$. The RF W of Fig. 3 preserves completeness, and moreover is weakly exfiltration resistant and weakly zero-knowledge preserving for the prover.*

▶ Remark 4 (On knowledge soundness). The IPS of Fig. 2 satisfies soundness, but is not in general a proof of knowledge. However, we would like to note that the prover's firewall still works for the standard transformation of a Sigma protocol into a zero-knowledge proof of knowledge. In such a transformation, a *trapdoor* commitment scheme is used to commit to the verifier's challenge. Then, after the verifier decommits, the prover sends the trapdoor to the verifier. This allows an extractor to learn the trapdoor, rewind the prover, and open the commitment to a different challenge, thus learning the response for two different challenges, which allows it to obtain a witness using special soundness.

The prover's RF for this protocol stays the same, except that it additionally needs to provide a trapdoor for the mauled public key $\widehat{pk}$ given a trapdoor for the original public key pk. This is possible, for instance, using Pedersen's commitment, where given a public key $pk = (g, h = g^k)$ with trapdoor k, we can maul the key to $(\widehat{g} = g^{t_1}, \widehat{h} = h^{t_2})$ for random t_1, t_2. Given the trapdoor k for the key pk, the trapdoor for the mauled key $\widehat{pk}$ can be computed as $t_2 t_1^{-1} k$.

4 Firewalls for Proving Compound Statements

In this section, we show how to construct firewalls for Sigma protocols that prove compound statements.

Given two Sigma protocols Σ_0 and Σ_1 for NP languages $\mathcal{L}_0$ and $\mathcal{L}_1$, it is easy to obtain a Sigma protocol Σ_{AND} for the NP language $\mathcal{L}_{\mathsf{AND}} = \{(x_0, x_1) : \ x_0 \in \mathcal{L}_0 \wedge x_1 \in \mathcal{L}_1\}$ by simply running Σ_0 and Σ_1 in parallel, with the verifier sending a single challenge. In a similar vein, the OR technique by Cramer, Damgård, and Schoenmakers [15] allows to obtain a Sigma protocol Σ_{OR} for the NP language $\mathcal{L}_{\mathsf{OR}} = \{(x_0, x_1) : \ x_0 \in \mathcal{L}_0 \vee x_1 \in \mathcal{L}_1\}$. Importantly, if Σ_0 and Σ_1 are both perfect HVZK, Σ_{OR} satisfies perfect WI. On the other hand, Garay et al. [30] showed that if Σ_0 and Σ_1 are computational HVZK, Σ_{OR} satisfies computational WI, although the latter holds only in case both statements x_0, x_1 in the definition of language $\mathcal{L}_{\mathsf{OR}}$ are true (but the prover knows either a witness for x_0 or for x_1).

As long as Σ_0 and Σ_1 are malleable, it is easy to build RFs for Σ_{AND} and Σ_{OR} using our techniques. The RF for Σ_{AND} weakly preserves HVZK, whereas the RF for Σ_{OR} weakly preserves both HVZK and WI.

4.1 AND Composition

Given x_0, x_1, a prover wishes to prove to a verifier that $x_0 \in \mathcal{L}_0$ and $x_1 \in \mathcal{L}_1$. More precisely, consider the derived relation:

$$\mathcal{R}_{\mathsf{AND}} = \{((x_0, x_1), (w_0, w_1)) : \ (x_0, w_0) \in \mathcal{R}_0 \wedge (x_1, w_1) \in \mathcal{R}_1\}.$$

Let $\Sigma_0 = ((\mathsf{P}_1^0, \mathsf{P}_2^0), \mathsf{V}^0)$ (resp. $\Sigma_1 = ((\mathsf{P}_1^1, \mathsf{P}_2^1), \mathsf{V}^1)$) be a Sigma protocol for language $\mathcal{L}_0$ (resp. $\mathcal{L}_1$). A Sigma protocol Σ_{AND} for the relation $\mathcal{R}_{\mathsf{AND}}$ can be obtained by running the two provers of Σ_0 and Σ_1 in parallel, with the verifier sending a single challenge for both executions. Fig. 4 shows a RF for the prover of Σ_{AND}, assuming that both Σ_0 and Σ_1 are malleable. We prove the following result, whose proof appears in the full version [29].

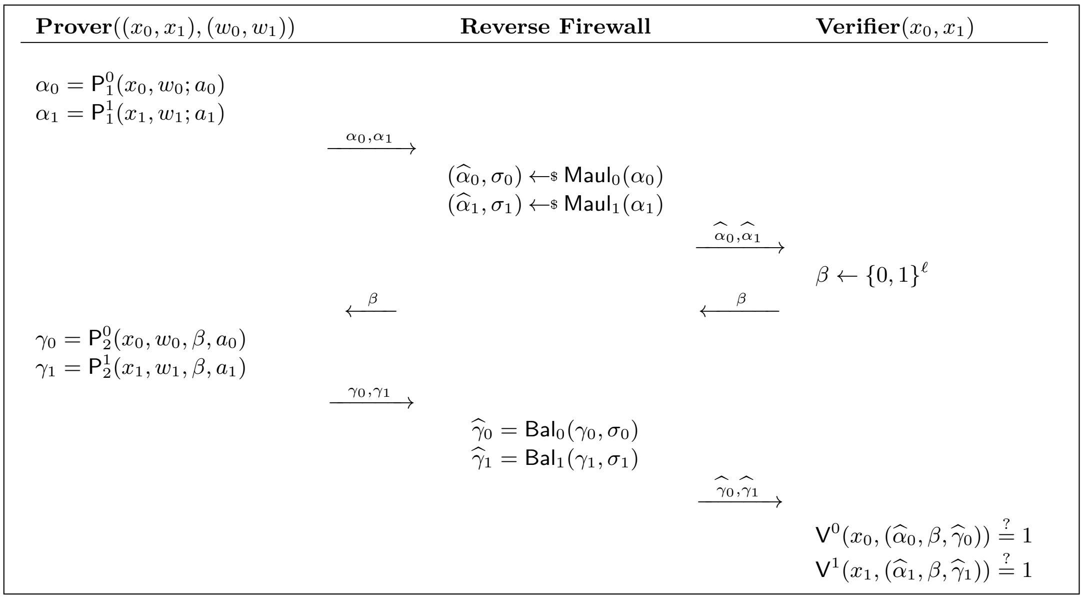

Prover$((x_0, x_1), (w_0, w_1))$
Reverse Firewall
Verifier(x_0, x_1)
$\alpha_0 = \mathsf{P}_1^0(x_0, w_0; a_0)$
$\alpha_1 = \mathsf{P}_1^1(x_1, w_1; a_1)$
$\xrightarrow{\alpha_0, \alpha_1}$
$(\widehat{\alpha}_0, \sigma_0) \leftarrow\$ \mathsf{Maul}_0(\alpha_0)$
$(\widehat{\alpha}_1, \sigma_1) \leftarrow\$ \mathsf{Maul}_1(\alpha_1)$
$\xrightarrow{\widehat{\alpha}_0, \widehat{\alpha}_1}$
$\beta \leftarrow \{0,1\}^\ell$
$\xleftarrow{\beta}$
$\xleftarrow{\beta}$
$\gamma_0 = \mathsf{P}_2^0(x_0, w_0, \beta, a_0)$
$\gamma_1 = \mathsf{P}_2^1(x_1, w_1, \beta, a_1)$
$\xrightarrow{\gamma_0, \gamma_1}$
$\widehat{\gamma}_0 = \mathsf{Bal}_0(\gamma_0, \sigma_0)$
$\widehat{\gamma}_1 = \mathsf{Bal}_1(\gamma_1, \sigma_1)$
$\xrightarrow{\widehat{\gamma}_0, \widehat{\gamma}_1}$
$\mathsf{V}^0(x_0, (\widehat{\alpha}_0, \beta, \widehat{\gamma}_0)) \overset{?}{=} 1$
$\mathsf{V}^1(x_1, (\widehat{\alpha}_1, \beta, \widehat{\gamma}_1)) \overset{?}{=} 1$

Figure 4 Reverse firewall for the AND composition of Sigma protocols.

▶ **Theorem 5.** *Let* $\Sigma_0 = (\mathsf{P}^0 = (\mathsf{P}_1^0, \mathsf{P}_2^0), \mathsf{V}^0)$ *and* $\Sigma_1 = (\mathsf{P}^1 = (\mathsf{P}_1^1, \mathsf{P}_2^1), \mathsf{V}^1)$ *be malleable Sigma protocols with unique responses, for relations* $\mathcal{R}_0$ *and* $\mathcal{R}_1$*. The RF* W *of Fig. 4 preserves completeness, and is weakly HVZK preserving for the prover of the Sigma protocol* Σ_{AND} *for relation* $\mathcal{R}_{\mathsf{AND}}$*.*

4.2 OR Composition

Given x_0, x_1, a prover wishes to prove to a verifier that either $x_0 \in \mathcal{L}_0$ or $x_1 \in \mathcal{L}_1$ (without revealing which one is the case). More precisely, consider the derived relation

$$\mathcal{R}_{\mathsf{OR}} = \{((x_0, x_1), w) : \ (x_0, w) \in \mathcal{R}_0 \vee (x_1, w) \in \mathcal{R}_1\}.$$

Let $\Sigma_0 = ((\mathsf{P}_1^0, \mathsf{P}_2^0), \mathsf{V}^0)$ (resp. $\Sigma_1 = ((\mathsf{P}_1^1, \mathsf{P}_2^1), \mathsf{V}^1)$) be a Sigma protocol for language $\mathcal{L}_0$ (resp. $\mathcal{L}_1$); we denote by S^0 (resp. S^1) the HVZK simulator for Σ_0 (resp. Σ_1). A Sigma protocol Σ_{OR} for the relation $\mathcal{R}_{\mathsf{OR}}$ has been constructed for the first time in [15], where the authors showed that Σ_{OR} satisfies both (perfect) special HVZK and (perfect) WI. We describe the protocol Σ_{OR} in Fig. 5, and depict our RF for the prover in Fig. 6.

Prover$((x_0, x_1), w)$		**Verifier**(x_0, x_1)
$\alpha_b = \mathsf{P}_1^b(x_b, w; a)$ $(\alpha_{1-b}, \beta_{1-b}, \gamma_{1-b}) \leftarrow\$ \mathsf{S}^{1-b}(x_{1-b})$		
	$\xrightarrow{\alpha_0, \alpha_1}$	
		$\beta \leftarrow\$ \{0,1\}^\ell$
	$\xleftarrow{\beta}$	
$\beta_b = \beta \oplus \beta_{1-b}$ $\gamma_b = \mathsf{P}_2^b(x_b, w, \beta_b, a)$		
	$\xrightarrow{\beta_0, \beta_1, \gamma_0, \gamma_1}$	
		$\beta \overset{?}{=} \beta_0 \oplus \beta_1$ $\mathsf{V}^0(x_0, (\alpha_0, \beta_0, \gamma_0)) \overset{?}{=} 1$ $\mathsf{V}^1(x_1, (\alpha_1, \beta_1, \gamma_1)) \overset{?}{=} 1$

Figure 5 OR composition of Sigma protocols, where $b \in \{0,1\}$ is s.t. $(x_b, w) \in \mathcal{R}_b$.

As in the case of AND composition, we still rely on the fact that the input Sigma protocols Σ_0, Σ_1 are malleable. An additional difficulty, however, stems from the fact that a functionality maintaining prover could now try to change the distribution of the challenges β_0, β_1 in such a way that, even if $\beta_0 \oplus \beta_1 = \beta$, the pair (β_0, β_1) signals some information about the witness w or about the hidden bit b. Intuitively, the RF in Fig. 6 tackles this attack by randomizing the challenges β, β_0, β_1. The latter requires a different form of malleability from the underlying Sigma protocols, which we dub *instance-dependent malleability*, where it should be possible to maul the prover's first message in such a way that we can later balance the prover's last message as well as the verifier's challenge.

For the RF in Fig. 6, we prove the following result, whose proof appears in the full version [29] of this paper.

▶ **Theorem 6.** *Let* $\Sigma_0 = (\mathsf{P}^0 = (\mathsf{P}_1^0, \mathsf{P}_2^0), \mathsf{V}^0)$ *and* $\Sigma_1 = (\mathsf{P}^1 = (\mathsf{P}_1^1, \mathsf{P}_2^1), \mathsf{V}^1)$ *be instance-dependent malleable Sigma protocols with unique responses, for relations* $\mathcal{R}_0$ *and* $\mathcal{R}_1$*. The RF* W *of Fig. 6 preserves completeness, and is weakly HVZK/WI preserving for the prover of the Sigma protocol* Σ_{OR} *for relation* $\mathcal{R}_{\mathsf{OR}}$*.*

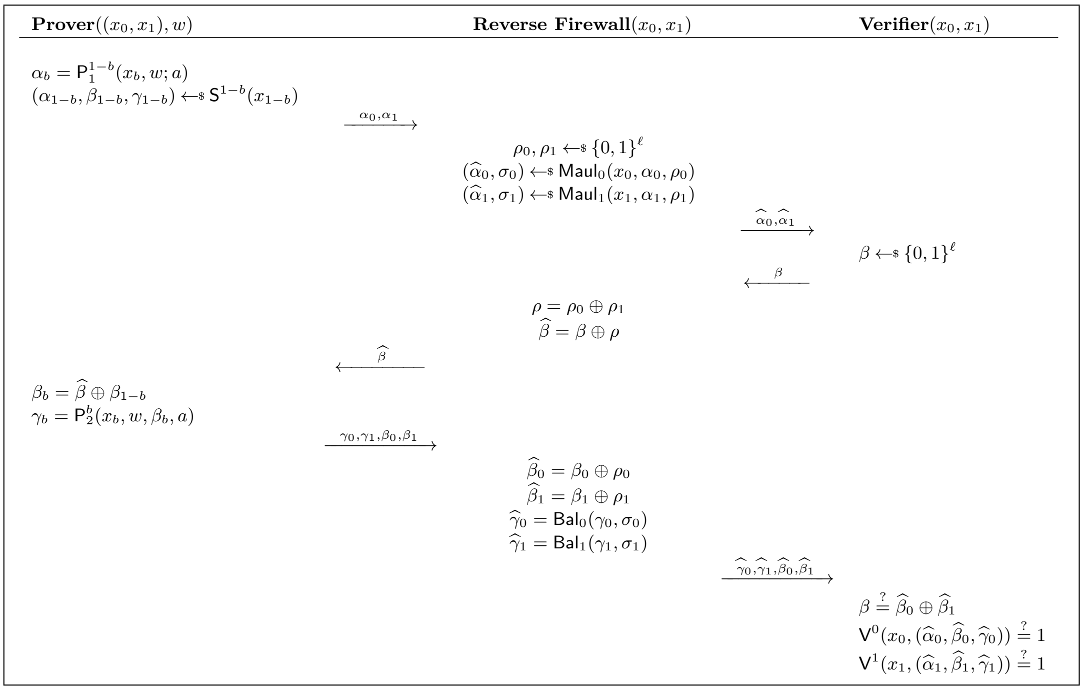

Prover$((x_0, x_1), w)$
Reverse Firewall(x_0, x_1)
Verifier(x_0, x_1)
$\alpha_b = \mathsf{P}_1^{1-b}(x_b, w; a)$
$(\alpha_{1-b}, \beta_{1-b}, \gamma_{1-b}) \leftarrow\!\!\$\ \mathsf{S}^{1-b}(x_{1-b})$
$\xrightarrow{\alpha_0, \alpha_1}$
$\rho_0, \rho_1 \leftarrow\!\!\$\ \{0,1\}^\ell$
$(\widehat{\alpha}_0, \sigma_0) \leftarrow\!\!\$\ \mathsf{Maul}_0(x_0, \alpha_0, \rho_0)$
$(\widehat{\alpha}_1, \sigma_1) \leftarrow\!\!\$\ \mathsf{Maul}_1(x_1, \alpha_1, \rho_1)$
$\xrightarrow{\widehat{\alpha}_0, \widehat{\alpha}_1}$
$\beta \leftarrow\!\!\$\ \{0,1\}^\ell$
$\xleftarrow{\beta}$
$\rho = \rho_0 \oplus \rho_1$
$\widehat{\beta} = \beta \oplus \rho$
$\xleftarrow{\widehat{\beta}}$
$\beta_b = \widehat{\beta} \oplus \beta_{1-b}$
$\gamma_b = \mathsf{P}_2^b(x_b, w, \beta_b, a)$
$\xrightarrow{\gamma_0, \gamma_1, \beta_0, \beta_1}$
$\widehat{\beta}_0 = \beta_0 \oplus \rho_0$
$\widehat{\beta}_1 = \beta_1 \oplus \rho_1$
$\widehat{\gamma}_0 = \mathsf{Bal}_0(\gamma_0, \sigma_0)$
$\widehat{\gamma}_1 = \mathsf{Bal}_1(\gamma_1, \sigma_1)$
$\xrightarrow{\widehat{\gamma}_0, \widehat{\gamma}_1, \widehat{\beta}_0, \widehat{\beta}_1}$
$\beta \overset{?}{=} \widehat{\beta}_0 \oplus \widehat{\beta}_1$
$\mathsf{V}^0(x_0, (\widehat{\alpha}_0, \widehat{\beta}_0, \widehat{\gamma}_0)) \overset{?}{=} 1$
$\mathsf{V}^1(x_1, (\widehat{\alpha}_1, \widehat{\beta}_1, \widehat{\gamma}_1)) \overset{?}{=} 1$

Figure 6 Reverse Firewall for the basic OR composition of Sigma protocols, where $b \in \{0,1\}$ is s.t. $(x_b, w) \in \mathcal{R}_b$.

5 Previous Work

5.1 Comparison with Mironov and Stephens-Davidowitz

In their original paper, Mironov and Stephens-Davidowitz [40] build RFs for arbitrary two-party protocols. While their results are related to ours, since IPSes are just a special case of two-party computation, there are some crucial differences which we highlight below.

The first RF construction sanitizes a specific combination of re-randomizable garbled circuits and oblivious transfer, for obtaining general-purpose private function evaluation. The second RF construction sanitizes any two-party protocol, at the price of encrypting the full transcript under public keys that are broadcast at the beginning of the protocol. Both constructions can be instantiated based on (variants of) the DDH assumption. When cast to IPSes, their results yield:

(i) A RF for the prover that weakly preserves ZK. This is comparable to our RF achieving weak ZK preservation using malleable Sigma protocols and key-malleable commitments. However, our constructions have the advantage that we do not need to change the initial IPS, and thus our RF can be applied directly to already existing implementations in a fully transparent manner (and without introducing any overhead).

(ii) A RF for the prover satisfying a property called strong exfiltration resistance *against an eavesdropper*, which means that exfiltration resistance holds w.r.t. an arbitrarily subverted prover talking to the *honest verifier*. Note that the latter does not contradict our impossibility result ruling out strong ZK preservation, as our attacks crucially rely on the fact that the distinguisher can (passively) corrupt the verifier.

(iii) A RF for the verifier satisfying both strong exfiltration resistance and the following weak guarantee: No malicious prover can find statements x_0, x_1 such that it can distinguish transcripts obtained by talking to an arbitrarily subverted verifier holding either input x_0 or input x_1. Note that the latter does not contradict our impossibility result that rules out weak soundness preservation, since none of the above guarantees imply soundness preservation.

We observe that the above results have at least one of the following drawbacks: (i) The RF is not transparent, i.e. it cannot be applied to the initial protocol as is; (ii) The resulting sanitized protocol is not efficient, as we first need to encode the function being computed as a circuit.

Our techniques allow to overcome these limitations in the concrete case of IPSes, as our RFs are both transparent (i.e. they can be applied directly to already deployed protocols) and efficient (i.e. the sanitized IPSes have exactly the same efficiency as the original, both in terms of round and communication complexity). We see this as the main novelty of our work.

5.2 Additional Related Works

Besides the already mentioned constructions, RFs have also been realized in other settings including digital signatures [5], secure message transmission and key exchange [21, 12], and oblivious transfer [40, 12].

Moreover, a few other lines of research recently[10] emerged to tackle the challenge of protecting cryptographic algorithms against (different forms of) subversion. We review the main ones below.

Algorithm substitution attacks

Bellare, Patterson, and Rogaway [11] studied subversion of symmetric encryption schemes in the form of algorithm substitution attacks (ASAs). In particular, they show that *undetectable* subversion of the encryption algorithm is possible, and may lead to severe security breaches; moreover, they prove that deterministic, stateful, ciphers are secure against the same type of ASAs. Follow-up works improved the original paper in several aspects [18, 10], and explored the power of ASAs in other contexts, e.g. digital signatures [5], secret sharing [31], and message authentication codes [3].

Backdoors

Another form of subversion consists of all those attacks that surreptitiously generate public parameters (primes, curves, etc.) together with secret backdoors that allow to bypass security. The study of this type of subversion is motivated by the DUAL_EC_DRBG PRG incident.

A formal study of parameters subversion has been considered for several primitives, including pseudorandom generators [20, 19], hash functions [27], non-interactive zero knowledge [8], and public-key encryption [6].

Cliptography

Russell et al. [46] (see also [47, 4]) consider a different approach to the immunization of cryptosystems against complete subversion (i.e., when all algorithms can be subverted by the attacker): offline/online black-box testing. This amounts to introducing an external entity, called the watchdog, whose goal is to test, either in an online or in an offline fashion, whether a given cryptographic implementation is compliant with its specification.

[10] All these research directions have their roots in the seminal works of Young and Yung [52] and Simmons [50], in the settings of kleptography and subliminal channels.

Hence, a cryptosystem is deemed secure against complete subversion if there exists a universal watchdog such that, for every attacker subverting all algorithms, either the watchdog detects subversion with high probability, or the cryptoscheme remains secure even when using its subverted implementation.

Self-guarding

Yet another approach towards thwarting subversion is that of self-guarding [28]. The idea here is to assume a trusted initialization phase in which the honest parties possess a genuine implementation of the cryptosystem, before subversion takes place. This phase is used in order to generate samples that will be exploited later, together with additional simple operations that need to be implemented from scratch, to prevent leakage in the face of subversion attacks.

6 Conclusion

We showed how to design cryptographic reverse firewalls allowing to preserve security of interactive proof systems in the face of subversion. Our firewalls apply to a large class of Sigma protocols meeting a natural malleability property, and can be extended to cover classical applications of Sigma protocols for designing zero-knowledge proofs and for proving compound statements.

We leave it as an intriguing open problem to design a reverse firewall for the OR composition of Sigma protocols that are delayed input, as considered in [13, 14].

References

1 Vulnerability summary for cve-2014-6271 (shellshock), September 2014. URL: `http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-6271`.

2 Juniper vulnerability, 2015. URL: `https://kb.juniper.net/InfoCenter/index?page=content&id=JSA10713`.

3 Marcel Armour and Bertram Poettering. Substitution attacks against message authentication. *IACR Trans. Symmetric Cryptol.*, 2019(3):152–168, 2019.

4 Giuseppe Ateniese, Danilo Francati, Bernardo Magri, and Daniele Venturi. Public immunization against complete subversion without random oracles. In Robert H. Deng, Valérie Gauthier-Umaña, Martín Ochoa, and Moti Yung, editors, *ACNS 19*, volume 11464 of *LNCS*, pages 465–485. Springer, Heidelberg, June 2019. `doi:10.1007/978-3-030-21568-2_23`.

5 Giuseppe Ateniese, Bernardo Magri, and Daniele Venturi. Subversion-resilient signature schemes. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015*, pages 364–375. ACM Press, October 2015. `doi:10.1145/2810103.2813635`.

6 Benedikt Auerbach, Mihir Bellare, and Eike Kiltz. Public-key encryption resistant to parameter subversion and its realization from efficiently-embeddable groups. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018, Part I*, volume 10769 of *LNCS*, pages 348–377. Springer, Heidelberg, March 2018. `doi:10.1007/978-3-319-76578-5_12`.

7 James Ball, Julian Borger, and Glenn Greenwald. Revealed: How US and UK spy agencies defeat internet privacy and security. *Guardian Weekly*, September 2013.

8 Mihir Bellare, Georg Fuchsbauer, and Alessandra Scafuro. NIZKs with an untrusted CRS: Security in the face of parameter subversion. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 777–804. Springer, Heidelberg, December 2016. `doi:10.1007/978-3-662-53890-6_26`.

9 Mihir Bellare and Oded Goldreich. On defining proofs of knowledge. In Ernest F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 390–420. Springer, Heidelberg, August 1993. doi:10.1007/3-540-48071-4_28.

10 Mihir Bellare, Joseph Jaeger, and Daniel Kane. Mass-surveillance without the state: Strongly undetectable algorithm-substitution attacks. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015*, pages 1431–1440. ACM Press, October 2015. doi:10.1145/2810103.2813681.

11 Mihir Bellare, Kenneth G. Paterson, and Phillip Rogaway. Security of symmetric encryption against mass surveillance. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 1–19. Springer, Heidelberg, August 2014. doi:10.1007/978-3-662-44371-2_1.

12 Rongmao Chen, Yi Mu, Guomin Yang, Willy Susilo, Fuchun Guo, and Mingwu Zhang. Cryptographic reverse firewall via malleable smooth projective hash functions. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 844–876. Springer, Heidelberg, December 2016. doi:10.1007/978-3-662-53887-6_31.

13 Michele Ciampi, Giuseppe Persiano, Alessandra Scafuro, Luisa Siniscalchi, and Ivan Visconti. Improved OR-composition of sigma-protocols. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A, Part II*, volume 9563 of *LNCS*, pages 112–141. Springer, Heidelberg, January 2016. doi:10.1007/978-3-662-49099-0_5.

14 Michele Ciampi, Giuseppe Persiano, Alessandra Scafuro, Luisa Siniscalchi, and Ivan Visconti. Online/offline OR composition of sigma protocols. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 63–92. Springer, Heidelberg, May 2016. doi:10.1007/978-3-662-49896-5_3.

15 Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Yvo Desmedt, editor, *CRYPTO'94*, volume 839 of *LNCS*, pages 174–187. Springer, Heidelberg, August 1994. doi:10.1007/3-540-48658-5_19.

16 Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi-authority election scheme. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 103–118. Springer, Heidelberg, May 1997. doi:10.1007/3-540-69053-0_9.

17 Ivan Damgård and Jens Groth. Non-interactive and reusable non-malleable commitment schemes. In *35th ACM STOC*, pages 426–437. ACM Press, June 2003. doi:10.1145/780542.780605.

18 Jean Paul Degabriele, Pooya Farshim, and Bertram Poettering. A more cautious approach to security against mass surveillance. In Gregor Leander, editor, *FSE 2015*, volume 9054 of *LNCS*, pages 579–598. Springer, Heidelberg, March 2015. doi:10.1007/978-3-662-48116-5_28.

19 Jean Paul Degabriele, Kenneth G. Paterson, Jacob C. N. Schuldt, and Joanne Woodage. Backdoors in pseudorandom number generators: Possibility and impossibility results. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 403–432. Springer, Heidelberg, August 2016. doi:10.1007/978-3-662-53018-4_15.

20 Yevgeniy Dodis, Chaya Ganesh, Alexander Golovnev, Ari Juels, and Thomas Ristenpart. A formal treatment of backdoored pseudorandom generators. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 101–126. Springer, Heidelberg, April 2015. doi:10.1007/978-3-662-46800-5_5.

21 Yevgeniy Dodis, Ilya Mironov, and Noah Stephens-Davidowitz. Message transmission with reverse firewalls—secure communication on corrupted machines. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 341–372. Springer, Heidelberg, August 2016. doi:10.1007/978-3-662-53018-4_13.

22 Sebastian Faust, Markulf Kohlweiss, Giorgia Azzurra Marson, and Daniele Venturi. On the non-malleability of the Fiat-Shamir transform. In Steven D. Galbraith and Mridul Nandi, editors, *INDOCRYPT 2012*, volume 7668 of *LNCS*, pages 60–79. Springer, Heidelberg, December 2012. doi:10.1007/978-3-642-34931-7_5.

23 Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple non-interactive zero knowledge proofs based on a single random string (extended abstract). In *31st FOCS*, pages 308–317. IEEE Computer Society Press, October 1990. doi:10.1109/FSCS.1990.89549.

24 Uriel Feige and Adi Shamir. Witness indistinguishable and witness hiding protocols. In *22nd ACM STOC*, pages 416–426. ACM Press, May 1990. doi:10.1145/100216.100272.

25 Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987. doi:10.1007/3-540-47721-7_12.

26 Marc Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 152–168. Springer, Heidelberg, August 2005. doi:10.1007/11535218_10.

27 Marc Fischlin, Christian Janson, and Sogol Mazaheri. Backdoored hash functions: Immunizing HMAC and HKDF. In *31st IEEE Computer Security Foundations Symposium, CSF 2018, Oxford, United Kingdom, July 9-12, 2018*, pages 105–118, 2018.

28 Marc Fischlin and Sogol Mazaheri. Self-guarding cryptographic protocols against algorithm substitution attacks. In *31st IEEE Computer Security Foundations Symposium, CSF 2018, Oxford, United Kingdom, July 9-12, 2018*, pages 76–90, 2018.

29 Chaya Ganesh, Bernardo Magri, and Daniele Venturi. Cryptographic reverse firewalls for interactive proof systems. *IACR Cryptology ePrint Archive*, 2020:204, 2020. URL: https://eprint.iacr.org/2020/204.

30 Juan A. Garay, Philip D. MacKenzie, and Ke Yang. Strengthening zero-knowledge protocols using signatures. *Journal of Cryptology*, 19(2):169–209, April 2006. doi:10.1007/s00145-005-0307-3.

31 Irene Giacomelli, Ruxandra F. Olimid, and Samuel Ranellucci. Security of linear secret-sharing schemes against mass surveillance. In Michael Reiter and David Naccache, editors, *CANS 15*, LNCS, pages 43–58. Springer, Heidelberg, Dec. 2015. doi:10.1007/978-3-319-26823-1_4.

32 Oded Goldreich. *Foundations of Cryptography: Basic Tools*, volume 1. Cambridge University Press, Cambridge, UK, 2001.

33 Oded Goldreich and Ariel Kahan. How to construct constant-round zero-knowledge proof systems for NP. *Journal of Cryptology*, 9(3):167–190, June 1996.

34 Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM*, 38(3):691–729, 1991.

35 Shafi Goldwasser and Silvio Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *14th ACM STOC*, pages 365–377. ACM Press, May 1982. doi:10.1145/800070.802212.

36 Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.

37 Louis C. Guillou and Jean-Jacques Quisquater. A practical zero-knowledge protocol fitted to security microprocessor minimizing both trasmission and memory. In C. G. Günther, editor, *EUROCRYPT'88*, volume 330 of *LNCS*, pages 123–128. Springer, Heidelberg, May 1988. doi:10.1007/3-540-45961-8_11.

38 Arjen K. Lenstra, James P. Hughes, Maxime Augier, Joppe W. Bos, Thorsten Kleinjung, and Christophe Wachter. Public keys. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 626–642. Springer, Heidelberg, August 2012. doi:10.1007/978-3-642-32009-5_37.

39 Ueli M. Maurer. Unifying zero-knowledge proofs of knowledge. In Bart Preneel, editor, *AFRICACRYPT 09*, volume 5580 of *LNCS*, pages 272–286. Springer, Heidelberg, June 2009.

40 Ilya Mironov and Noah Stephens-Davidowitz. Cryptographic reverse firewalls. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 657–686. Springer, Heidelberg, April 2015. doi:10.1007/978-3-662-46803-6_22.

41 Tatsuaki Okamoto. Provably secure and practical identification schemes and corresponding signature schemes. In Ernest F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 31–53. Springer, Heidelberg, August 1993. doi:10.1007/3-540-48071-4_3.

42 Tatsuaki Okamoto and Kazuo Ohta. Divertible zero knowledge interactive proofs and commutative random self-reducibility. In Jean-Jacques Quisquater and Joos Vandewalle, editors, *EUROCRYPT'89*, volume 434 of *LNCS*, pages 134–148. Springer, Heidelberg, April 1990. doi:10.1007/3-540-46885-4_16.

43 Rafail Ostrovsky, Vanishree Rao, and Ivan Visconti. On selective-opening attacks against encryption schemes. In Michel Abdalla and Roberto De Prisco, editors, *SCN 14*, volume 8642 of *LNCS*, pages 578–597. Springer, Heidelberg, September 2014. doi:10.1007/978-3-319-10879-7_33.

44 Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 129–140. Springer, Heidelberg, August 1992. doi:10.1007/3-540-46766-1_9.

45 Nicole Perlroth, Jeff Larson, and Scott Shane. N.S.A. able to foil basic safeguards of privacy on web. *The New York Times*, September 2013.

46 Alexander Russell, Qiang Tang, Moti Yung, and Hong-Sheng Zhou. Cliptography: Clipping the power of kleptographic attacks. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 34–64. Springer, Heidelberg, December 2016. doi:10.1007/978-3-662-53890-6_2.

47 Alexander Russell, Qiang Tang, Moti Yung, and Hong-Sheng Zhou. Generic semantic security against a kleptographic adversary. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 907–922. ACM Press, October / November 2017. doi:10.1145/3133956.3133993.

48 Alessandra Scafuro and Ivan Visconti. On round-optimal zero knowledge in the bare public-key model. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 153–171. Springer, Heidelberg, April 2012. doi:10.1007/978-3-642-29011-4_11.

49 Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 239–252. Springer, Heidelberg, August 1990. doi:10.1007/0-387-34805-0_22.

50 Gustavus J. Simmons. The prisoners' problem and the subliminal channel. In David Chaum, editor, *CRYPTO'83*, pages 51–67. Plenum Press, New York, USA, 1983.

51 Dominique Unruh. Quantum proofs of knowledge. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 135–152. Springer, Heidelberg, April 2012. doi:10.1007/978-3-642-29011-4_10.

52 Adam Young and Moti Yung. Kleptography: Using cryptography against cryptography. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 62–74. Springer, Heidelberg, May 1997. doi:10.1007/3-540-69053-0_6.

Robust Algorithms Under Adversarial Injections

Paritosh Garg
EPFL, Lausanne, Switzerland
paritosh.garg@epfl.ch

Sagar Kale
University of Vienna, Austria
sagar.kale@univie.ac.at

Lars Rohwedder
EPFL, Lausanne, Switzerland
lars.rohwedder@epfl.ch

Ola Svensson
EPFL, Lausanne, Switzerland
ola.svensson@epfl.ch

Abstract

In this paper, we study streaming and online algorithms in the context of randomness in the input. For several problems, a random order of the input sequence – as opposed to the worst-case order – appears to be a necessary evil in order to prove satisfying guarantees. However, algorithmic techniques that work under this assumption tend to be vulnerable to even small changes in the distribution. For this reason, we propose a new *adversarial injections* model, in which the input is ordered randomly, but an adversary may inject misleading elements at arbitrary positions. We believe that studying algorithms under this much weaker assumption can lead to new insights and, in particular, more robust algorithms. We investigate two classical combinatorial-optimization problems in this model: Maximum matching and cardinality constrained monotone submodular function maximization. Our main technical contribution is a novel streaming algorithm for the latter that computes a 0.55-approximation. While the algorithm itself is clean and simple, an involved analysis shows that it emulates a subdivision of the input stream which can be used to greatly limit the power of the adversary.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms; Theory of computation → Adversary models

Keywords and phrases Streaming algorithm, adversary, submodular maximization, matching

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.56

Category Track A: Algorithms, Complexity and Games

Related Version A full version of the paper is available at `http://arxiv.org/abs/2004.12667`.

Funding This research was supported in part by the Swiss National Science Foundation project 200021-184656 "Randomness in Problem Instances and Randomized Algorithms."

1 Introduction

In the streaming model, an algorithm reads the input sequentially from the input stream while using limited memory. In particular, the algorithm is expected to use memory that is much smaller than the input size, ideally, linear in the size of a solution. We consider the most fundamental setting in which the algorithm is further restricted to only read the input stream once. In this case, the algorithm cannot remember much of the input along the way, and part of the input is irrevocably lost. Something similar happens for online algorithms: Here, the input is given to the algorithm one element at a time and the algorithm has to decide whether to take it into its solution or to discard it. This decision is irrevocable.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 56; pp. 56:1–56:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The most common approach to analyze the quality of an algorithm in these models is worst-case analysis. Here, an adversary has full knowledge of the algorithm's strategy and presents a carefully crafted instance to it, trying to make the ratio between the value of the algorithm's solution and that of an optimum solution (the approximation ratio; for online algorithms called the competitive ratio) as small as possible[1]. While worst-case analysis gives very robust guarantees, it is also well-known that such an analysis is often very pessimistic. Not only are good guarantees not possible for many problems, but in many cases worst-case instances appear quite artificial. Hence, the worst-case approximation/competitive ratio does not necessarily represent the quantity that we want to optimize.

One way to remedy this is to weaken the power of the adversary and a popular model to achieve that is the random-order model. Here, an adversary may pick the instance as before, but it is presented in a uniformly-random order to the algorithm. This often allows for significantly better provable guarantees. A prime example is the secretary problem: For the worst-case order it is impossible to get a bounded competitive ratio whereas for the random-order a very simple stopping rule achieves a competitive ratio of $1/e$. Unfortunately, in this model, algorithms tend to overfit and the assumption of a uniformly-random permutation of the input is a strong one. To illustrate this point, it is instructive to consider two examples of techniques that break apart when the random-order assumption is slightly weakened:

Several algorithms in the random-order model first read a small fraction of the input, say, the first 1% of the input. Such an algorithm relies on the assumption that around 1% of the elements from an optimum solution are contained in this first chunk. It computes some statistics, summaries, or initial solutions using this chunk in order to estimate certain properties of the optimum solution. Then in the remaining 99% of the input it uses this knowledge to build a good solution for the problem. For examples of such streaming algorithms, see Norouzi-Fard et al. [29] who study submodular maximization and Gamlath et al. [13] who study maximum matching. Also Guruganesh and Singla's [16] online algorithm for maximum matching for bipartite graphs is of this kind. Note that these algorithms are very sensitive to noise at the beginning of the stream.

Another common technique is to split the input into fixed parts and exploit that with high probability the elements of the optimum solution are distributed evenly among these parts, e.g., each part has at most one optimum element. These methods critically rely on the assumption that each part is representative for the whole input or that the parts are in some way homogeneous (properties of the parts are the same in expectation). Examples of such algorithms include the streaming algorithm for maximum matching [23], and the streaming algorithm for submodular maximization [1] that achieves the tight competitive ratio $1 - 1/e$ in the random-order model.

The motivation of this work is to understand whether the strong assumption of uniformly-random order is necessary to allow for better algorithms. More specifically, we are motivated by the following question:

> Can we achieve the same guarantees as in the uniform-random order but by algorithms that are more robust against some distortions in the input?

In the next subsection, we describe our proposed model that is defined so as to avoid overfitting to the random-order model, and, by working in this model, our algorithms for submodular maximization and maximum matching are more robust while maintaining good guarantees.

[1] We assume that the problem is a maximization problem.

1.1 The Adversarial Injections Model

Our model – that we call the *adversarial-injections* model – lies in between the two extremes of random-order and adversarial-order. In this model, the input elements are divided into two sets E_{NOISE} and E_{GOOD}. An adversary first picks all elements, puts each element in either E_{NOISE} or E_{GOOD}, and chooses the input order. Then the elements belonging to E_{GOOD} are permuted uniformly at random among themselves. The algorithm does not know if an element is good or noise. We judge the quality of the solution produced by an algorithm by comparing it to the best solution in E_{GOOD}.

An equivalent description of the model is as follows. First, a set of elements is picked by the adversary and is permuted randomly. Then, the adversary injects more elements at positions of his choice without knowing the random permutation of the original stream[2]. Comparing with the previous definition, the elements injected by the adversary correspond to E_{NOISE} and the elements of the original stream correspond to E_{GOOD}.

We denote by $E_{\text{OPT}} \subseteq E_{\text{GOOD}}$ the elements of a fixed optimum solution of the elements in E_{GOOD}. We can assume without loss of generality that $E_{\text{GOOD}} = E_{\text{OPT}}$, because otherwise elements in $E_{\text{GOOD}} \setminus E_{\text{OPT}}$ can be treated as those belonging to E_{NOISE} (which only strengthens the power of the adversary).

1.2 Related Models

With a similar motivation, Kesselheim, Kleinberg, Niazadeh [21] studied the robustness of algorithms for the secretary problem from a slightly different perspective: They considered the case when the order of the elements is not guaranteed to be uniformly-at-random but still contains "enough" randomness with respect to different notions such as entropy. Recently, Esfandiari, Korula, Mirrokni [8] introduced a model where the input is a combination of stochastic input that is picked from a distribution and adversarially ordered input. Our model is different in the sense that the input is a combination of randomly ordered (instead of stochastic input) and adversarially ordered elements.

Two models that are more similar to ours in the sense that the input is initially ordered in a uniformly-random order and then scrambled by an adversary in a limited way are [15] and [3]. First, in the streaming model, Guha and McGregor [15] introduced the notion of a t-bounded adversary that can disturb a uniformly-random stream but has memory to remember and delay at most t input elements at a time. Second, Bradac et al. [3] very recently introduced a new model that they used to obtain robust online algorithms for the secretary problem. Their model, called the *Byzantine* model, is very related to ours: the input is split into two sets which exactly correspond to E_{GOOD} and E_{NOISE} in the adversarial-injections model. The adversary gets to pick the elements in both of them, but an algorithm will be compared against only E_{GOOD}. Then – this is where our models differ – the adversary chooses an arrival time in $[0, 1]$ for each element in E_{NOISE}. He has no control over the arrival times of the elements in E_{GOOD}, which are chosen independently and uniformly at random in $[0, 1]$. The algorithm does not know to which set an element belongs, but it knows the timestamp of each element, as the element arrives. While the Byzantine model prevents certain kinds of

[2] We remark that the assumption that the adversary does not know the order of the elements is important. Otherwise, the model is equivalent to the adversarial order model for "symmetric" problems such as the matching problem. To see this, let E_{OPT} correspond to an optimum matching in any hard instance under the adversarial order. Since a matching is symmetric, the adversary can inject appropriately renamed edges depending on the order of the edges (which he without this assumption knows) and obtain exactly the hard instance.

overfitting (e.g., of the classical algorithm for the secretary problem), it does not tackle the issues of the two algorithmic techniques we discussed earlier: Indeed, by time $t = 0.01$, we will see around 1% of the elements from E_{OPT}. Hence, we can still compute some estimates based on them, but do not lose a lot when dismissing them. Likewise, we may partition the timeline, and thereby the input, into parts such that in each part at most one element of E_{OPT} appears.

Hence, even if our model appears very similar to the Byzantine model, there is this subtle, yet crucial, difference. The adversarial-injections model does not add the additional dimension of time, and hence, does not allow for the kind of overfitting that we discussed earlier. To further emphasize this difference, we now describe why it is strictly harder to devise algorithms in the adversarial-injections model compared to the Byzantine model. It is at least as hard as the Byzantine model, because any algorithm for the former also works for the latter. This holds because the adversarial-injections model can be thought of as the Byzantine model with additional power to the adversary and reduced power for the algorithm: The adversary gets the additional power of setting the timestamps of elements in E_{GOOD}, but not their identities, whereas the algorithm is not allowed to see the timestamp of any element.

To show that it is strictly harder, consider online bipartite matching. We show that one cannot beat 1/2 in the adversarial-injections model (for further details, see Section 2.2) whereas we observe that the $(1/2 + \delta)$-approximation algorithm [16] for bipartite graphs and its analysis generalizes to the Byzantine model as well. This turns out to be the case because the algorithm in [16] runs a greedy algorithm on the first small fraction, say 1% of the input and "augments" this solution using the remaining 99% of the input. The analysis crucially uses the fact that 99% of the optimum elements are yet to arrive in the augmentation phase. This can be simulated in the Byzantine model using timestamps in the online setting as one sees 1% of E_{OPT} in expectation.

1.3 Our Results

We consider two benchmark problems in combinatorial optimization under the adversarial-injections model in both the streaming and the online settings, namely maximum matching and monotone submodular maximization subject to a cardinality constraint. As we explain next, the study of these classic problems in our new model gives interesting insights: for many settings we can achieve more robust algorithms with similar guarantees as in the random-order model but, perhaps surprisingly, there are also natural settings where the adversarial-injection model turns out to be as hard as the adversarial order model.

The maximum matching problem. We first discuss the (unweighted) *maximum matching* problem. Given a graph $G = (V, E)$, a matching M is a subset of edges such that every vertex has at most one incident edge in M. A matching of maximum cardinality is called a maximum matching, whereas a *maximal* matching is one in which no edge can be added without breaking the property of it being a matching. The goal in the maximum matching problem is to compute a matching of maximum cardinality. Note that a maximal matching is 1/2-approximate. Work on maximum matching has led to several important concepts and new techniques in theoretical computer science [26, 24, 6, 19]. The combination of streaming and random-order model was first studied by Konrad, Magniez and Mathieu [23], where edges of the input graph arrive in the stream. We allow a streaming algorithm to have memory $O(n \operatorname{polylog}(n))$, which is called the semi-streaming setting. This is usually significantly less than the input size, which can be as large as $O(n^2)$. This memory usage is justified, because even storing a solution can take $\Omega(n \log(n))$ space ($\Omega(\log(n))$ for each edge identity). The question that Konrad et al. answered affirmatively was whether the trivial 1/2-approximation algorithm that computes a maximal matching can be improved in the random-order model. Since then, there has been some work on improving the constant [14, 9]. The state-of-the-art is an approximation ratio of $6/11 \approx 0.545$ proved by Farhadi, Hajiaghayi, Mah, Rao, and Rossi [9]. We show that beating the ratio of 1/2 is possible also in the adversarial-injections model by building on the techniques developed for the random-order model.

▶ **Theorem 1.** *There exists an absolute constant $\gamma > 0$ such that there is a semi-streaming algorithm for maximum matching under adversarial-injections with an approximation ratio of $1/2 + \gamma$ in expectation.*

We note that beating 1/2 in adversarial-order streams is a major open problem. In this regard, our algorithm can be viewed as a natural first step towards understanding this question.

Now we move our attention to the online setting, where the maximum matching problem was first studied in the seminal work of Karp, Vazirani, and Vazirani [20]. They gave a tight $(1 - 1/e)$-competitive algorithm for the so-called one-sided vertex arrival model which is an important special case of the edge-arrival model considered here. Since then, the online matching problem has received significant attention (see e.g. [4, 7, 11, 17, 14]). Unlike the adversarial streaming setting, there is a recent hardness result due to [14] in the adversarial online setting that the trivial ratio of 1/2 cannot be improved. We also know by [16] that one can beat 1/2 for bipartite graphs in the random-order online setting. Hence, one might hope at least for bipartite graphs to use existing techniques to beat 1/2 in the online adversarial-injections setting and get a result analogous to Theorem 1. But surprisingly so, this is not the case. We observe that the construction used in proving Theorem 3 in [14] also implies that there does not exist an algorithm with a competitive ratio of $1/2 + \varepsilon$ for any $\varepsilon > 0$ in the adversarial-injections model.

Maximizing a monotone submodular function subject to a cardinality constraint. In this problem, we are given a ground set E of n elements and a monotone submodular set function $f : 2^E \to \mathbb{R}_{\geqslant 0}$. A function is said to be submodular, if for any $S, T \subseteq E$ it holds that $f(S) + f(T) \geqslant f(S \cup T) + f(S \cap T)$. It is monotone if $f(S) \leqslant f(T)$ for all $S \subseteq T \subseteq E$. The problem we consider is to find a set $S \subseteq E$ with $|S| \leqslant k$ that maximizes $f(S)$. We assume that access to f is via an oracle.

In the offline setting, a simple greedy algorithm that iteratively picks the element with the largest marginal contribution to f with respect to the current solution is $(1 - 1/e)$-approximate [28]. This is tight: Any algorithm that achieves an approximation ratio of better than $(1 - 1/e)$ must make $\Omega(n^k)$ oracle calls [27], which is enough to brute-force over all k-size subsets. Even for maximum coverage (which is a special family of monotone submodular functions), it is NP-hard to get an approximation algorithm with ratio better than $1 - 1/e$ [10].

In the random-order online setting, this problem is called the *submodular secretary* problem, and an exponential time $1/e$-approximation and polynomial-time $(1 - 1/e)/e$-approximation algorithms are the state-of-the-art [22]. In the adversarial online setting, it is impossible to get any bounded approximation ratio for even the very special case of picking a maximum weight element. In this case, $|E_{\text{OPT}}| = 1$ and adversarial and adversarial-injections models coincide; hence the same hardness holds. In light of this negative result, we focus on adversarial-injections in the streaming setting. Note that to store a solution we only need the space for k element identities. We think of k to be much smaller than n. Hence, it is natural to ask, whether the number of elements in memory can be independent of n.

Table 1 : Comparison of different models for the two studied problems. Here, $\gamma > 0$ is a fixed absolute constant and $\varepsilon > 0$ is any constant.

Maximum matching

	Random order	Adversarial Injections	Adversarial order
Streaming	$\geqslant 6/11$ [9]	$\geqslant 1/2 + \gamma$	$\leqslant 1 - 1/e + \varepsilon$ [18]
Online	$\geqslant 1/2$ (folklore)	$\leqslant 1/2$	$\leqslant 1/2$ [14]

Submodular function maximization

	Random order	Adversarial Injections	Adversarial order
Streaming	$\geqslant 1 - 1/e - \varepsilon$ [1] $\leqslant 1 - 1/e + \varepsilon$ [25]	$\geqslant 0.55$	$\geqslant 1/2 - \varepsilon$ [2] $\leqslant 1/2$ [12]

For streaming algorithms in the adversarial order setting, the problem was first studied by Chakrabarti and Kale [5] where they gave a 1/4-approximation algorithm. This was subsequently improved to $1/2 - \varepsilon$ by Badanidiyuru et al. [2]. Later, Norouzi-Fard et al. [29] observed that in the random-order model this ratio can be improved to beyond 1/2. Finally, Agrawal et al. [1] obtained a tight $(1 - 1/e)$-approximation guarantee in the random-order model.

The algorithm of Agrawal et al. [1] involves as a crucial step a partitioning the stream in order to isolate the elements of the optimum solution. As discussed earlier, this approach does not work under adversarial-injections. However, we note that the algorithm and analysis by Norouzi-Fard et al. [29] can be easily modified to work under adversarial-injections as well. Their algorithm, however, has an approximation ratio of $1/2 + 8 \cdot 10^{-14}$. In this paper, we remedy this weak guarantee.

▶ **Theorem 2.** *There exists a* 0.55*-approximation algorithm that stores a number of elements that is independent of n for maximizing a monotone submodular function with a cardinality constraint k under adversarial-injections in the streaming setting.*

We summarize and compare our results with random-order and adversarial-order models for the problems we study in Table 1. It is interesting to see that in terms of beating 1/2, our model in the streaming setting agrees with the random-order model and in the online setting agrees with the adversarial-order model.

2 Matching

In this section, we consider the problem of maximum unweighted matching under adversarial injections in both streaming and online settings where the edges of the input graph arrive one after another.

2.1 Streaming Setting

We show that the trivial approximation ratio of 1/2 can be improved upon. We provide a robust version of existing techniques and prove a statement about robustness of the greedy algorithm to achieve this.

First, let us introduce some notation which we will use throughout this section. We denote the input graph by $G = (V, E)$, and let M^* be a maximum matching. For any matching M, the union $M \cup M^*$ is a collection of vertex-disjoint paths and cycles. When M is clear from the context, a path of length $i \geqslant 3$ in $M \cup M^*$ which starts and ends with an edge of M^* is called an i-augmenting path. Notice that an i-augmenting path alternates between edges of M^* and M and that we can increase the size of M by one by taking all edges from M^* and removing all edges from M along this path. We say that an edge in M is 3-augmentable if it belongs to some 3-augmenting path. Otherwise, we say it is non-3-augmentable. Also, let $M^* = E_{\text{OPT}}$; as described in the introduction, this is without loss of generality.

As a subroutine for our algorithm we need the following procedure.

▶ **Lemma 3** (Lemma 3.1 in [13]). *There exists a streaming algorithm 3-Aug-Paths with the following properties:*

1. *The algorithm is initialized with a matching M and a parameter $\beta > 0$. Then a set E of edges is given to the algorithm one edge at a time.*
2. *If $M \cup E$ contains at least $\beta|M|$ vertex disjoint 3-augmenting paths, the algorithm returns a set A of at least $(\beta^2/32)|M|$ vertex disjoint 3-augmenting paths. The algorithm uses space $O(|M|)$.*

2.1.1 The Algorithm

We now describe our algorithm Match. It runs two algorithms in parallel and selects the better of the two outputs. The first algorithm simply constructs a maximal matching greedily by updating the variable M_1. The second algorithm also constructs a matching $M_2^{(1)}$ greedily, but it stops once $M_2^{(1)}$ has $|M^*|(1/2 - \varepsilon)$ edges. We call this Phase 1. Then, it finds 3-augmentations using the 3-Aug-Paths algorithm given by Lemma 3. Finally, it augments the paths found to obtain a matching M_2. The constant β used in 3-Aug-Paths is optimized for the analysis and will be specified there.

Notice that here we assumed that the algorithm knows $|M^*|$. This assumption can be removed using geometric guessing at a loss of an arbitrary small factor in the approximation ratio. We refer the reader to the full version for details.

2.1.2 Overview of the Analysis

We discuss only the intuition here and refer the reader to the full version for a formal proof. Consider the first portion of the stream until we have seen a small constant fraction of the elements in E_{OPT}. If the greedy matching up to this point is already close to a 1/2-approximation, this is good for the second algorithm as we are able to augment the matching using the remaining edges of M^*. The other case is good for the first algorithm: We will show that the greedy matching formed so far must contain a significant fraction of the edges in M^* which we have seen so far. If this happens, the first algorithm outputs a matching of size a constant fraction more than $|M^*|/2$.

A technical challenge and novelty comes from the fact that the two events above are not independent of the random order of E_{OPT}. Hence, when conditioning on one event, we can no longer assume that the order of E_{OPT} is uniformly at random. We get around this by showing that the greedy algorithm is robust to small changes in streams. The intuition is that in the first part of the stream the greedy solution either is large for all permutations of E_{OPT} or it is small for all permutations. Hence, these are not random events depending on the order, but two cases in which we can assume a uniform distribution.

2.2 Online Setting

Since we can improve 1/2 for the streaming setting, it is natural to hope that the existing techniques (e.g., the approach of the previous subsection) can be applied in the online setting as well. Surprisingly, this is not the case. In other words, the competitive ratio of 1/2 is optimal even for bipartite graphs. The technique from the previous subsection breaks apart, because the algorithm constructs several candidate solutions in parallel by guessing $|M^*|$. This is not a problem for a streaming algorithm, however, an online algorithm can only build one solution.

For a formal proof, we rely on the bipartite construction used in the proof of Theorem 3 from [14]. The authors show that there is no (randomized) algorithm with a competitive ratio of $1/2 + \varepsilon$ for any $\varepsilon > 0$. More precisely, they show that not even a good fractional matching can be constructed online. For fractional matchings, randomization does not help and therefore we can assume the algorithm is deterministic. The original proof is with respect to adversarial order, but it is not hard to see that it transfers to adversarial injections.

The authors construct a bipartite instance that arrives in (up to) N rounds. In round i, a matching of size i arrives. The algorithm does not know whether the current round is the last one or not. Hence, it has to maintain a good approximation after each round. This forces the algorithm to take edges that do not belong to the optimal matching and eventually leads to a competitive ratio of $1/2$. The same construction works in our model: The edges from the optimal matching arrive in the last round and their internal order does not affect the proof. In fact, the construction works for any order of the elements within a round. Thus, an algorithm cannot exploit the fact that their order is randomized and therefore also cannot do better than $1/2$.

3 Submodular Maximization

In this section, we consider the problem of submodular maximization subject to a cardinality constraint. The algorithm has query access to a monotone, submodular function $f : 2^E \to \mathbb{R}$ over a ground set E. Moreover, f is normalized with $f(\emptyset) = 0$. The goal is to compute a set S of size at most k that maximizes $f(S)$. We present a 0.55-approximate streaming algorithm in the adversarial-injections model which only needs the memory to store $(O(k))^k$ many elements. In particular, this number is independent of the length of the stream.

3.1 Notation

For $e \in E$ and $S \subseteq E$ we write $S + e$ for the set $S \cup \{e\}$ and $f(e \mid S)$ for $f(S + e) - f(S)$. Similarly, for $A, B \subseteq E$ let $f(A \mid B) := f(A \cup B) - f(B)$. An equivalent definition of submodularity to the one given in the introduction states that for any two sets $S \subseteq T \subseteq E$, and $e \in E \setminus T$ it holds that $f(e \mid S) \geqslant f(e \mid T)$.

We denote by σ the stream of elements E, by $-\infty$ and ∞ the start and end of the stream. For elements a and b, we write $\sigma[a, b]$ for the interval including a and b and $\sigma(a, b)$ for the interval excluding them. Moreover, we may assume that $f(\emptyset) = 0$, since otherwise, we may replace the submodular function by $f' : 2^E \to \mathbb{R}_{\geqslant 0}, T \mapsto f(T) - f(\emptyset)$.

Denote the permutation of E_{OPT} by π. Let o_i^π be the i'th element of E_{OPT} in the stream according to the order given by π. Let $O_0^\pi = \emptyset$ and $O_i^\pi = \{o_1^\pi, \dots, o_i^\pi\}$ for all i; hence, $E_{\text{OPT}} = O_k^\pi$ for any π. Finally, let $\text{OPT} = f(O_k^\pi)$.

3.2 The Algorithm

For simplicity we present an algorithm with the assumption that it knows the value OPT. Moreover, for the set of increases in f, that is $I = \{f(e \mid S) : e \in E, S \subseteq E\}$, we assume that $|I| \leqslant O(k)$. These two assumptions can be made at a marginal cost in the approximation ratio and an insignificant increase in memory. This follows from standard techniques. We refer the reader to the full version for details.

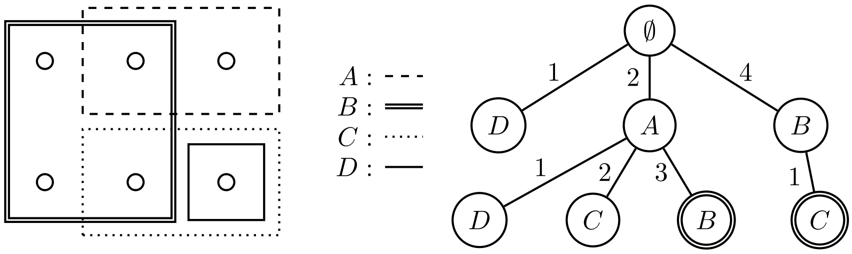


Figure 1 In this example, function f counts the dots covered by a set of rectangles. On the right, the tree for stream $\sigma = (A, B, C, D)$ and $k = 2$ is depicted. The labels on the edges correspond to the increase in f. The maximal leaves are highlighted.

As a central data-structure, the algorithm maintains a rooted tree T of height at most k. Every node except for the root stores a single element from E. The structure resembles a prefix tree: Each node is associated with the solution, where the elements on the path from the root to it is selected. The nodes can have at most $|I|$ children, that is, one for each increase. The basic idea is that for some partial solution $S \subseteq E$ (corresponding to a node) and two elements e, e' with $f(e \mid S) = f(e' \mid S)$ we only consider one of the solutions $S \cup \{e\}$ and $S \cup \{e'\}$. More precisely, the algorithm starts with a tree consisting only of the root. When it reads an element e from the stream, it adds e as a child to every node where (1) the distance of the node to the root is smaller than k and (2) the node does not have a child with increase $f(e \mid S)$, where S is the partial solution corresponding to this particular node.

Because of (1), the solutions are always of cardinality at most k. When the stream is read completely, the algorithm selects the best solution among all leaves. An example of the algorithm's behavior is given in Figure 1.

3.3 Overview of the Analysis

For analyzing the algorithm, we will use a sophisticated strategy to select one of the leaves and only compare this leaf to the optimum. We emphasize that this selection does not have to be computed by the algorithm. In particular, it does not need to be computable by a streaming algorithm and it can rely on knowledge of E_{OPT} and E_{NOISE}, which the algorithm does not have. Since the algorithm always takes the best leaf, we only need to give a lower bound for one of them. Before we describe this strategy, we analyze the tree algorithm in two educational corner cases.

The first one shows that by a careful selection of a leaf the algorithm appears to take elements based on the location of the E_{OPT}, although it does not know them. Let $r_i^\pi = \text{argmax}_{e \in \sigma(-\infty, o_1^\pi]} f(e)$, that is, the most valuable element until the arrival of the first element from E_{OPT}. Here argmax breaks ties in favor of the first element in σ. We do not know when o_1^π arrives, but we know that the algorithm will have created a node (with the root as its parent) for r_1^π by then. We define iteratively $R_i^\pi = \{r_1^\pi, \dots, r_i^\pi\}$ and $r_{i+1}^\pi = \text{argmax}_{e \in \sigma(r_i^\pi, o_{i+1}^\pi]} f(e \mid R_i^\pi)$ for all i. Again, we can be sure that r_{i+1}^π, which yields the best increase for R_i^π until the arrival of o_{i+1}^π, is a appended to the path $r_1^\pi \to \cdots \to r_i^\pi$.

This selection is inspired by the following idea. Suppose we could partition the stream into k intervals such that in each exactly one elements from E_{OPT} appears. Then a sensible approach would be to start with an empty solution and greedily add the element that yields the maximal increase to our partial solution in each interval. Clearly one such partition would be $\sigma(o_i^\pi, o_{i+1}^\pi]$, $i = 1, \dots, k$. We note that while the selection above is similar, it does not completely capture this. Although r_{i+1}^π is an element that arrives before o_{i+1}^π, we cannot be certain that it arrives after o_i^π. We only know that it arrives after r_i^π.

Next, we prove that the solution R_k^π is a 1/2-approximation. This already shows that the tree algorithm is 1/2-approximate even in the adversarial order model. By definition of R_i^π and r_i^π, we have

$$\begin{aligned} f(R_k^\pi) = \sum_{i=1}^{k} f(r_i^\pi \mid R_{i-1}^\pi) \geqslant \sum_{i=1}^{k} f(o_i^\pi \mid R_{i-1}^\pi) \\ = \sum_{i=1}^{k} [f(o_i^\pi \mid R_{i-1}^\pi) - f(o_i^\pi \mid R_k^\pi)] + \sum_{i=1}^{k} f(o_i^\pi \mid R_k^\pi). \end{aligned}$$

Notice that due to submodularity the term $f(o_i^\pi \mid R_{i-1}^\pi) - f(o_i^\pi \mid R_k^\pi)$ is always non-negative. Moreover, if $o_i^\pi = r_i^\pi \in R_k^\pi$, it collapses to $f(o_i^\pi \mid R_{i-1}^\pi)$. Thus, we can bound the right term of the equation and thereby $f(R_k^\pi)$ with

$$f(R_k^\pi) \geqslant \sum_{\substack{i=1 \\ r_i^\pi = o_i^\pi}}^{k} f(o_i^\pi \mid R_{i-1}^\pi) + \sum_{i=1}^{k} f(o_i^\pi \mid R_k^\pi).$$

From submodularity and monotonicity of f it follows that

$$\sum_{i=1}^{k} f(o_i^\pi \mid R_k^\pi) \geqslant f(O_k^\pi \mid R_k^\pi) = f(O_k^\pi \cup R_k^\pi) - f(R_k^\pi) \geqslant f(O_k^\pi) - f(R_k^\pi).$$

Hence, we conclude that

$$2f(R_k^\pi) \geqslant f(O_k^\pi) + \sum_{\substack{i=1 \\ r_i^\pi = o_i^\pi}}^{k} f(o_i^\pi \mid R_{i-1}^\pi).$$

This shows that R_k^π is 1/2-approximate, because $O_k^\pi = E_{\text{OPT}}$. Indeed, if a significant value of the elements in E_{OPT} are taken, then R_k^π is even better than 1/2-approximate.

Recall that the elements E_{OPT} are ordered randomly in the adversarial-injections model. Hence, the worst-case in the analysis above is that R_k^π is disjoint from E_{OPT} for all realizations of π. However, by a different analysis we can see that this case is in fact well-behaved. This is because the algorithm would select the same elements $r_1^\pi, \dots, r_k^\pi$ for every realization of π. Hence, we can safely drop the superscript π in R_i^π and r_i^π. Since for every element $o \in E_{\text{OPT}}$ there is some realization of π where $o_i^\pi = o$, yet the algorithm does not pick o_i^π, we can bound the increase of each r_i by

$$f(r_i \mid R_{i-1}) \geqslant \max_{o \in E_{\text{OPT}}} f(o \mid R_{i-1}) \geqslant \frac{1}{k} \sum_{o \in E_{\text{OPT}}} f(o \mid R_{i-1}).$$

By submodularity and monotonicity we get

$$\frac{1}{k} \sum_{o \in E_{\text{OPT}}} f(o \mid R_{i-1}) \geqslant \frac{1}{k} f(E_{\text{OPT}} \mid R_{i-1}) \geqslant \frac{1}{k}(\text{OPT} - f(R_{i-1})).$$

This is the same recurrence formula as in the classic greedy algorithm and by simple calculations we get the closed form

$$f(R_k) \geqslant \left(1 - \left(1 - \frac{1}{k}\right)^k\right) \text{OPT} \geqslant \left(1 - \frac{1}{e}\right) \text{OPT}.$$

In other words, the algorithm is even $(1 - 1/e)$-approximate in this case. In our main proof we will use a more involved strategy for selecting a leaf. This is to be able to combine the two approaches discussed above.

3.4 Analysis

Let us first define the selection of the leaf we are going to analyze. The elements on the path to this leaf will be denoted by $s_1^\pi, \dots, s_k^\pi$ and we write S_i^π for $\{s_1^\pi, \dots s_i^\pi\}$. The elements are defined inductively, but as opposed to the previous section we need in addition indices $n_1, \dots, n_k$. Recall, previously we defined the $(i+1)$'th element r_{i+1}^π as the best increase in $\sigma(r_i^\pi, o_{i+1}^\pi]$. Here, we use n_{i+1} to describe the index of the element from E_{OPT} which constitutes the end of this interval. It is not necessarily o_{i+1}^π anymore. We always start with $n_1 = 1$, but based on different cases we either set $n_{i+1} = n_i + 1$ or $n_{i+1} = n_i$. We underline that n_i is independent of the realization of π. In the following, $t \in [0,1]$ denotes a parameter that we will specify later.

The element s_i^π will be chosen from two candidates u_i^π and v_i^π. The former is the best increase of elements excluding $o_{n_i}^\pi$, that is,

$$u_i^\pi = \begin{cases} \operatorname{argmax}_{e \in \sigma(-\infty, o_{n_1}^\pi)} f(e) & \text{if } i = 1, \\ \operatorname{argmax}_{e \in \sigma(s_i^\pi, o_{n_i}^\pi)} f(e \mid S_{i-1}^\pi) & \text{otherwise.} \end{cases}$$

The latter is defined in the same way, except it includes $o_{n_i}^\pi$ in the choices, that is,

$$v_i^\pi = \begin{cases} \operatorname{argmax}_{e \in \sigma(-\infty, o_{n_1}^\pi]} f(e) & \text{if } i = 1, \\ \operatorname{argmax}_{e \in \sigma(s_{i-1}^\pi, o_{n_i}^\pi]} f(e \mid S_{i-1}^\pi) & \text{otherwise.} \end{cases}$$

We now define the choice of s_i^π and n_{i+1} based on the following two cases. Note that the cases are independent from the realization of π.

Case 1: $\mathbb{E}_\pi\, f(u_i^\pi \mid S_{i-1}^\pi) \geqslant t \cdot \mathbb{E}_\pi\, f(o_{n_i}^\pi \mid S_{i-1}^\pi)$. In this case, we set $s_i^\pi = u_i^\pi$ and $n_{i+1} = n_i$. Notice that this means s_i^π is chosen independently from $o_{n_i}^\pi$. In other words, we did not see $o_{n_i}^\pi$, yet. The element $o_{n_i}^\pi$ is still each of the remaining elements in E_{OPT} with equal probability. In the analysis this is beneficial, because the distribution of $o_{n_i}^\pi, \dots, o_k^\pi$ remains unchanged. This is similar to the second case in the previous section.

Case 2: $\mathbb{E}_\pi\, f(u_i^\pi \mid S_{i-1}^\pi) < t \cdot \mathbb{E}_\pi\, f(o_{n_i}^\pi \mid S_{i-1}^\pi)$. Here, set $s_i^\pi = v_i^\pi$ and $n_{i+1} = n_i + 1$. Now the distribution of $o_i^\pi, \dots, o_k^\pi$ can change. However, a considerable value of s_i^π over different π comes from taking $o_{n_i}^\pi$. As indicated by the first case in the previous section this will improve the guarantee of the algorithm.

The solution S_k^π corresponds to a leaf in the tree algorithm. Clearly, u_1^π and v_1^π are children of the root. Hence, s_1^π is also a child. Then for induction we assume s_i^π is a node, which implies u_{i+1}^π and v_{i+1}^π are also nodes: The elements u_{i+1}^π and v_{i+1}^π are the first elements after s_i^π with the respective gains $(f(u_{i+1}^\pi \mid S_i^\pi)$ and $f(v_{i+1}^\pi \mid S_i^\pi))$. Hence, s_{i+1}^π is a child of s_i^π.

In order to bound $\mathbb{E}_\pi\, f(S_k^\pi)$, we will study more broadly all values of $\mathbb{E}_\pi\, f(S_h^\pi)$ where $h \leqslant k$. To this end, we define a recursive formula $R(k,h)$ and prove that it bounds $\mathbb{E}_\pi\, f(S_h^\pi)/\operatorname{OPT}$ from below. Then using basic calculus we will show that $R(k,k) \geqslant 0.5506$ for all k. Initialize $R(k,0) = 0$ for all k. Then let $R(k,h)$, $h \leqslant k$, be defined by

$$R(k,h) = \min\left\{ \frac{t}{k} + \left(1 - \frac{t}{k}\right) R(k, h-1)\,,\ \frac{1}{k} + \left(1 - \frac{1+t}{k}\right) R(k-1, h-1)\,,\ \frac{1}{1+t} \right\}.$$

▶ **Lemma 4.** *For all instances of the problem and $h \leqslant k$, the solution S_h^π as defined above satisfies $\mathbb{E}_\pi\, f(S_h^\pi) \geqslant R(k,h)\operatorname{OPT}$.*

Proof. The proof is by induction over h. For $h = 0$, the statement holds as $R(k, 0)\,\mathrm{OPT} = 0 = \mathbb{E}_\pi\, f(S_0^\pi)$. Let $h > 0$ and suppose the statement of the lemma holds with $h - 1$ for all instances of the problem. Suppose we are given an instance with $k \geqslant h$. We distinguish the two cases $s_1^\pi = u_1^\pi$ and $s_1^\pi = v_1^\pi$.

First, consider $\mathbb{E}_\pi\, f(u_1^\pi) \geqslant t \cdot \mathbb{E}_\pi\, f(o_1^\pi)$, which implies that $s_1^\pi = u_1^\pi$. Note that u_1^π is the best element in $\sigma(-\infty, o_1^\pi)$, consequently, its choice is independent from the realization of π. Let us drop the superscript in u_1^π and s_1^π for clarity. We construct a new instance mimicking the subtree of s_1. Formally, our new instance still has the same k elements E_{OPT}, i.e., $k' = k$. The stream is $\sigma' = \sigma(s_1^\pi, \infty)$ and, the submodular function $f' : 2^U \to \mathbb{R}$, $f'(T) \mapsto f(T \mid s_1)$. In this instance we have $\mathrm{OPT}' = f'(E_{\mathrm{OPT}}) = f(E_{\mathrm{OPT}} \mid s_1) \geqslant \mathrm{OPT} - f(s_1)$. It is easy to see that the elements $s_1'^\pi, \ldots, s_{h-1}'^\pi$ chosen in the new instance correspond exactly to the elements $s_2^\pi, \ldots, s_h^\pi$. Hence, with the induction hypothesis we get

$$\mathbb{E}_\pi\, f(S_h^\pi) = f(s_1) + \mathbb{E}_\pi\, f(S_h^\pi \mid s_1) = f(s_1) + \mathbb{E}_\pi\, f'(S_{h-1}'^\pi) \geqslant f(s_1) + R(k, h-1)(\mathrm{OPT} - f(s_1)).$$

By assumption we have $f(s_1) \geqslant t \cdot \mathbb{E}_\pi\, f(o_i^\pi) \geqslant t \cdot \mathrm{OPT}/k$. Together with $R(k, h-1) \leqslant 1/(1+t) \leqslant 1$ we calculate

$$f(s_1) + R(k, h-1)(\mathrm{OPT} - f(s_1)) \geqslant \frac{t}{k}\,\mathrm{OPT} + R(k, h-1)\left(1 - \frac{t}{k}\right)\mathrm{OPT}\,.$$

The right-hand side is by definition at least $R(k, h)\,\mathrm{OPT}$.

Now we turn to the case $\mathbb{E}_\pi\, f(u_1^\pi) < t \cdot \mathbb{E}_\pi\, f(o_1^\pi)$, which means $s_1^\pi = v_1^\pi$ is chosen. Similar to the previous case, we construct a new instance. After taking s_1^π, our new instance has $k' = k - 1$ elements $E'_{\mathrm{OPT}} = E_{\mathrm{OPT}} \setminus \{o_1^\pi\}$, stream $\sigma' = \sigma(s_1, \infty)$, and submodular function $f' : 2^E \to \mathbb{R}$, $f(T) \mapsto f(T \mid s_1^\pi)$. Thus, $\mathrm{OPT}' = f'(E'_{\mathrm{OPT}}) = f(E_{\mathrm{OPT}} \setminus \{o_1^\pi\} \mid s_1^\pi) \geqslant \mathrm{OPT} - f(s_1^\pi \cup o_1^\pi)$. We remove o_1^π from E_{OPT}, because $s_1^\pi = v_1^\pi$ depends on it. The distribution of $o_2^\pi, \ldots, o_k^\pi$ when conditioning on the value of o_1^π (and thereby the choice of s_1^π) is still a uniformly random permutation of E'_{OPT}. Like in the previous case, we can see that $S_{h-1}'^\pi = S_h^\pi \setminus \{s_1^\pi\}$ and we can apply the induction hypothesis. First, however, let us examine $\mathbb{E}_\pi\, f(s_1^\pi \cup o_1^\pi)$. Since we know that whenever $s_1^\pi \neq o_1^\pi$ we have $s_1^\pi = u_1^\pi$, it follows that

$$\mathbb{P}_\pi[s_1^\pi \neq o_1^\pi] \cdot \mathbb{E}_\pi[f(s_1) \mid s_1^\pi \neq o_1^\pi] \leqslant \mathbb{E}_\pi\, f(u_1^\pi) < t \cdot \mathbb{E}_\pi\, f(o_1^\pi) \leqslant t \cdot \mathbb{E}_\pi\, f(s_1^\pi).$$

Hence, we deduce

$$\mathbb{E}_\pi\, f(s_1^\pi \cup o_1^\pi) \leqslant \mathbb{E}_\pi\, f(o_1^\pi) + \mathbb{P}_\pi[s_1^\pi \neq o_1^\pi] \cdot \mathbb{E}_\pi[f(s_1) \mid s_1^\pi \neq o_1^\pi] \leqslant \mathbb{E}_\pi\, f(o_1^\pi) + t \cdot \mathbb{E}_\pi\, f(s_1^\pi).$$

We are ready to prove the bound on $\mathbb{E}_\pi\, f(S_h^\pi)$. By induction hypothesis, we get

$$\begin{aligned}\mathbb{E}_\pi\, f(S_h^\pi) &= \mathbb{E}_\pi\, f(s_1^\pi) + \mathbb{E}_\pi\, f'(S_{h-1}'^\pi)\\ &\geqslant \mathbb{E}_\pi\, f(s_1^\pi) + R(k-1, h-1)(\mathrm{OPT} - \mathbb{E}_\pi\, f(s_1^\pi \cup o_1^\pi)).\end{aligned}$$

Inserting the bound on $\mathbb{E}_\pi\, f(s_1^\pi \cup o_1^\pi)$ we know that the right-hand side is at least

$$\mathbb{E}_\pi\, f(s_1^\pi) + R(k-1, h-1)(\mathrm{OPT} - \mathbb{E}_\pi\, f(o_1^\pi) - t \cdot \mathbb{E}_\pi\, f(s_1^\pi)).$$

Using that $f(s_1^\pi) \geqslant f(o_1^\pi)$ for all π and $R(k-1, h-1) \cdot t \leqslant t/(1+t) \leqslant 1$ we bound the previous term from below by

$$\mathbb{E}_\pi\, f(o_1^\pi) + R(k-1, h-1)(\mathrm{OPT} - (1+t)\,\mathbb{E}_\pi\, f(o_1^\pi)).$$

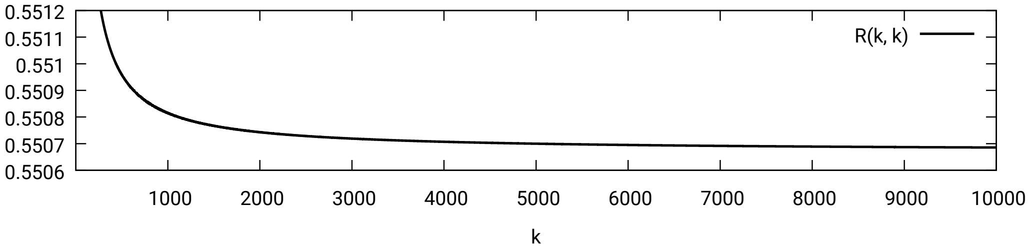


Figure 2 Values of the recurrence formula for $t = 0.8$.

Finally, we use that $\mathbb{E}_\pi f(o_1^\pi) \geqslant \mathrm{OPT}/k$ and $R(k-1, h-1)(1+t) \leqslant 1$ to arrive at

$$\frac{1}{k}\,\mathrm{OPT} + R(k-1,h-1)\left(\mathrm{OPT} - \frac{1+t}{k}\,\mathrm{OPT}\right) \geqslant R(k,h)\,\mathrm{OPT},$$

which concludes the proof. ◀

With $t = 0.8$ we are able to show that for sufficiently large k the minimum in the definition of $R(k,k)$ is always attained by the first term. Then, after calculating a lower bound on $R(k,k)$ for small values, we can easily derive a general bound.

▶ **Lemma 5.** *With $t = 0.8$ for all positive integers k it holds that $R(k,k) \geqslant 0.5506$.*

Figure 2 contains a diagram (generated by computer calculation), which shows that the formula tends to a value between 0.5506 and 0.5507 for $k \in \{0, \ldots, 10000\}$. The proof requires tedious and mechanical calculations and hence is omitted here. We refer the reader to the full version for complete details.

4 Conclusion and Open Problems

In this paper, we introduced a semi-random model called adversarial-injections with the motivation of eliminating algorithms that overfit to random-order streams while still being easier than adversarial-order streams. We studied two classical problems in combinatorial optimization in this model.

For unweighted matching, we could beat 1/2 in the streaming setting whereas we observed from [14] that we could not beat 1/2 in the online setting. This also makes our model non-trivial as there is a separation between the online and streaming setting.

For monotone submodular maximization with cardinality constraint k, we obtained a 0.55 approximation algorithm albeit with a huge memory footprint but importantly independent of n (universe size). The obvious open question is whether one can design a $(1-1/e)$-approximation algorithm which stores number of elements that is independent of n. Does our algorithm have an approximation ratio of $1-1/e$? We observed that the algorithm in [29] is a $1/2+\varepsilon$ approximation for a very small $\varepsilon > 0$. The algorithm stores $\mathrm{poly}(k)$ elements. Can one design an algorithm that stores only $\mathrm{poly}(k)$ elements and beats 1/2 by a significant constant or, even better, gets $1-1/e$?

References

1 Shipra Agrawal, Mohammad Shadravan, and Cliff Stein. Submodular secretary problem with shortlists. In *10th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10-12, 2019, San Diego, California, USA*, pages 1:1–1:19, 2019. doi:10.4230/LIPIcs.ITCS.2019.1.

2 Ashwinkumar Badanidiyuru, Baharan Mirzasoleiman, Amin Karbasi, and Andreas Krause. Streaming submodular maximization: massive data summarization on the fly. In *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, pages 671–680, 2014. doi:10.1145/2623330.2623637.

3 Domagoj Bradac, Anupam Gupta, Sahil Singla, and Goran Zuzic. Robust algorithms for the secretary problem. In *11th Innovations in Theoretical Computer Science Conference, ITCS 2020, January 12-14, 2020, Seattle, Washington, USA*, pages 32:1–32:26, 2020. doi:10.4230/LIPIcs.ITCS.2020.32.

4 Niv Buchbinder, Danny Segev, and Yevgeny Tkach. Online algorithms for maximum cardinality matching with edge arrivals. *Algorithmica*, 81(5):1781–1799, 2019. doi:10.1007/s00453-018-0505-7.

5 Amit Chakrabarti and Sagar Kale. Submodular maximization meets streaming: matchings, matroids, and more. *Math. Program.*, 154(1-2):225–247, 2015. doi:10.1007/s10107-015-0900-7.

6 Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17:449–467, 1965.

7 Leah Epstein, Asaf Levin, Danny Segev, and Oren Weimann. Improved bounds for randomized preemptive online matching. *Inf. Comput.*, 259(1):31–40, 2018. doi:10.1016/j.ic.2017.12.002.

8 Hossein Esfandiari, Nitish Korula, and Vahab Mirrokni. Online allocation with traffic spikes: Mixing adversarial and stochastic models. In *Proceedings of the Sixteenth ACM Conference on Economics and Computation*, pages 169–186, 2015.

9 Alireza Farhadi, Mohammad Taghi Hajiaghayi, Tung Mai, Anup Rao, and Ryan A. Rossi. Approximate maximum matching in random streams. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1773–1785, 2020. doi:10.1137/1.9781611975994.108.

10 Uriel Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, 1998. doi:10.1145/285055.285059.

11 Uriel Feige. Tighter bounds for online bipartite matching. *CoRR*, abs/1812.11774, 2018. arXiv:1812.11774.

12 Moran Feldman, Ashkan Norouzi-Fard, Ola Svensson, and Rico Zenklusen. The one-way communication complexity of submodular maximization with applications to streaming and robustness. In *Proceedings of the Fifty-Second Annual ACM on Symposium on Theory of Computing, STOC (to appear)*, 2020.

13 Buddhima Gamlath, Sagar Kale, Slobodan Mitrovic, and Ola Svensson. Weighted matchings via unweighted augmentations. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*, pages 491–500, 2019. doi:10.1145/3293611.3331603.

14 Buddhima Gamlath, Michael Kapralov, Andreas Maggiori, Ola Svensson, and David Wajc. Online matching with general arrivals. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 26–37, 2019. doi:10.1109/FOCS.2019.00011.

15 Sudipto Guha and Andrew McGregor. Approximate quantiles and the order of the stream. In *Proceedings of the Twenty-Fifth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 26-28, 2006, Chicago, Illinois, USA*, pages 273–279, 2006. doi:10.1145/1142351.1142390.

16 Guru Prashanth Guruganesh and Sahil Singla. Online matroid intersection: Beating half for random arrival. In *Integer Programming and Combinatorial Optimization - 19th International Conference, IPCO 2017, Waterloo, ON, Canada, June 26-28, 2017, Proceedings*, pages 241–253, 2017. doi:10.1007/978-3-319-59250-3_20.

17 Zhiyi Huang, Binghui Peng, Zhihao Gavin Tang, Runzhou Tao, Xiaowei Wu, and Yuhao Zhang. Tight competitive ratios of classic matching algorithms in the fully online model. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 2875–2886, 2019. doi:10.1137/1.9781611975482.178.

18 Michael Kapralov. Better bounds for matchings in the streaming model. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 1679–1697, 2013. doi:10.1137/1.9781611973105.121.

19 Richard M. Karp, Eli Upfal, and Avi Wigderson. Constructing a perfect matching is in random NC. *Combinatorica*, 6(1):35–48, 1986. doi:10.1007/BF02579407.

20 Richard M. Karp, Umesh V. Vazirani, and Vijay V. Vazirani. An optimal algorithm for on-line bipartite matching. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 352–358, 1990. doi:10.1145/100216.100262.

21 Thomas Kesselheim, Robert D. Kleinberg, and Rad Niazadeh. Secretary problems with non-uniform arrival order. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 879–888, 2015. doi:10.1145/2746539.2746602.

22 Thomas Kesselheim and Andreas Tönnis. Submodular secretary problems: Cardinality, matching, and linear constraints. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2017, August 16-18, 2017, Berkeley, CA, USA*, pages 16:1–16:22, 2017. doi:10.4230/LIPIcs.APPROX-RANDOM.2017.16.

23 Christian Konrad, Frédéric Magniez, and Claire Mathieu. Maximum matching in semi-streaming with few passes. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques – 15th International Workshop, APPROX 2012, and 16th International Workshop, RANDOM 2012, Cambridge, MA, USA, August 15-17, 2012. Proceedings*, pages 231–242. Springer, 2012. doi:10.1007/978-3-642-32512-0_20.

24 László Lovász. On determinants, matchings, and random algorithms. In *Fundamentals of Computation Theory, FCT 1979, Proceedings of the Conference on Algebraic, Arthmetic, and Categorial Methods in Computation Theory, Berlin/Wendisch-Rietz, Germany, September 17-21, 1979*, pages 565–574, 1979.

25 Andrew McGregor and Hoa T. Vu. Better streaming algorithms for the maximum coverage problem. *Theory Comput. Syst.*, 63(7):1595–1619, 2019. doi:10.1007/s00224-018-9878-x.

26 Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 1987. doi:10.1007/BF02579206.

27 George L. Nemhauser and Laurence A. Wolsey. Best algorithms for approximating the maximum of a submodular set function. *Math. Oper. Res.*, 3(3):177–188, 1978. doi:10.1287/moor.3.3.177.

28 George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. An analysis of approximations for maximizing submodular set functions – I. *Mathematical programming*, 14(1):265–294, 1978.

29 Ashkan Norouzi-Fard, Jakub Tarnawski, Slobodan Mitrovic, Amir Zandieh, Aidasadat Mousavifar, and Ola Svensson. Beyond 1/2-approximation for submodular maximization on massive data streams. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 3826–3835, 2018. URL: http://proceedings.mlr.press/v80/norouzi-fard18a.html.

Minimum Cut in $O(m \log^2 n)$ Time

Paweł Gawrychowski
University of Wrocław, Poland
gawry@cs.uni.wroc.pl

Shay Mozes
The Interdisciplinary Center Herzliya, Israel
smozes@idc.ac.il

Oren Weimann
University of Haifa, Israel
oren@cs.haifa.ac.il

Abstract

We give a randomized algorithm that finds a minimum cut in an undirected weighted m-edge n-vertex graph G with high probability in $O(m \log^2 n)$ time. This is the first improvement to Karger's celebrated $O(m \log^3 n)$ time algorithm from 1996. Our main technical contribution is a deterministic $O(m \log n)$ time algorithm that, given a spanning tree T of G, finds a minimum cut of G that 2-respects (cuts two edges of) T.

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

Keywords and phrases Minimum cut, Minimum 2-respecting cut

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.57

Category Track A: Algorithms, Complexity and Games

Related Version A full version of the paper is available at `https://arxiv.org/abs/1911.01145`.

Funding *Shay Mozes*: Supported in part by Israel Science Foundation grant 592/17.
Oren Weimann: Supported in part by Israel Science Foundation grant 592/17.

Acknowledgements We thank Daniel Anderson and Guy Blelloch for drawing our attention to an inaccuracy in a prior version of Section 3.1.

1 Introduction

The minimum cut problem is one of the most fundamental and well-studied optimization problems in theoretical computer science. Given an undirected edge-weighted graph $G = (V, E)$, the problem asks to find a subset of vertices S such that the total weight of all edges between S and $V \setminus S$ is minimized. The vast literature on the minimum cut problem can be classified into three main approaches:

The maximum-flow approach. The minimum cut problem was originally solved by computing the maximum st-flow [3] for all pairs of vertices s and t. In 1961, Gomory and Hu [10] showed that only $O(n)$ maximum st-flow computations are required, and in 1994 Hao and Orlin [11] showed that in fact a single maximum st-flow computation suffices. A maximum st-flow can be found in $O(mn \log(n^2/m))$ time using the Goldberg-Tarjan algorithm [9], and the fastest algorithm to date takes $O(mn)$ time [33, 22]. Faster (though not near-linear time) algorithms are known (see e.g [8, 25, 27] and references within) when the graph is unweighted or when the maximum edge weight W is not extremely large.

The edge-contraction approach. An alternative method is edge contraction. If we can identify an edge that does not cross the minimum cut, then we can contract this edge without affecting the minimum cut. Nagamochi and Ibaraki [30, 31] showed how to deterministically find a contractible edge in $O(m)$ time, leading to an $O(mn + n^2 \log n)$-time minimum cut algorithm. Karger [14] showed that randomly choosing the edge to contract works well with high probability. In particular, Karger and Stein [20] showed that this leads to an improved $O(n^2 \log^3 n)$ Monte Carlo algorithm.

The tree-packing approach. In 1961, Nash-Williams [32] proved that, in unweighted graphs, any graph with minimum cut c contains a set of $c/2$ edge-disjoint spanning trees. Gabow's algorithm [4] can be used to find such a tree-packing with $c/2$ trees in $O(mc \log n)$ time. Karger [18] observed that the c edges of a minimum cut must be partitioned among these $c/2$ spanning trees, hence the minimum cut *1-* or *2-respects* some tree in the packing. That is, one of the trees is such that at most two of its edges cross the minimum cut (these edges are said to *determine* the cut). We can therefore find the minimum cut by examining each tree and finding the minimum cut that 1- or 2-respects it.

Several obstacles need be overcome in order to translate this idea into an efficient minimum cut algorithm for weighted graphs: (1) we need a weighted version of tree-packing, (2) finding the packing (even in unweighted graphs) takes time proportional to c (and c may be large), (3) checking all trees takes time proportional to c, and (4) one needs an efficient algorithm that, given a spanning tree T of G, finds the minimum cut in G that 2-respects T (finding a minimum cut that 1-respects T can be easily done in $O(m+n)$ time, see e.g [18, Lemma 5.1]).

In a seminal work, Karger [18] overcame all four obstacles: First, he converts G into an unweighted graph by conceptually replacing an edge of weight w by w parallel edges. Then, he uses his random sampling from [16, 14] combined with Gabow's algorithm [4] to reduce the packing time to $O(m + n \log^3 n)$ and the number of trees in the packing to $O(\log n)$. Finally, he designs a deterministic $O(m \log^2 n)$ time algorithm that given a spanning tree T of G finds the minimum cut in G that 2-respects T. Together, this gives an $O(m \log^3 n)$ time randomized algorithm for minimum cut. Until the present work, this was the fastest known algorithm for undirected weighted graphs.

Karger's $O(m \log^2 n)$ algorithm for the 2-respecting problem finds, for each edge $e \in T$, the edge $e' \in T$ that minimizes the cut determined by $\{e, e'\}$. He used link-cut trees [35] to efficiently keep track of the sizes of cuts as the candidate edges e of T are processed in a certain order (bough decomposition), consisting of $O(\log n)$ iterations, and guarantees that the number of dynamic tree operations is $O(m)$ per iteration. Since each link-cut tree operation takes $O(\log n)$ time, the total running time for solving the 2-respecting problem is $O(m \log^2 n)$.

In a very recent paper, Lovett and Sandlund [26] proposed a simplified version of Karger's algorithm. Their algorithm has the same $O(m \log^3 n)$ running time as Karger's. To solve the 2-respecting problem they use top trees [1] rather than link-cut trees, and use heavy path decomposition [12, 35] to guide the order in which edges of T are processed. A property of heavy path decomposition is that, for every edge $(u, v) \notin T$, the u-to-v path in T intersects $O(\log n)$ paths of the decomposition. This property implies that the contribution of each non-tree edge to the cut changes $O(\log n)$ times along the the entire process. See also [6] who use the fact that the bough decomposition, implicitly used by Karger, also satisfies the above property. The idea of traversing a tree according to a heavy path decomposition, i.e., by first processing a smaller subtree and then processing the larger subtree has been used quite a few times in similar problems on trees. See e.g., [2]. While the ideas of Lovett and Sandlund [26] do not improve on Karger's bound, their paper has drawn our attention to this problem.

1.1 Our result and techniques

In this paper, we present a deterministic $O(m \log n)$ time algorithm that, given a spanning tree T of G, finds the minimum cut in G that 2-respects T. Using Karger's framework, this implies an $O(m \log^2 n)$ time randomized algorithm for minimum cut in weighted graphs.

Like prior algorithms for this problem, our algorithm finds, for each edge $e \in T$ the edge $e' \in T$ that minimizes the cut determined by $\{e, e'\}$. The difficult case to handle is when e and e' are such that neither of them is an ancestor of the other. In Karger's solution, handling each edge $e = (u, v) \in T$ was done using amortized $O(d \log n)$ operations on Sleator and Tarjan's link-cut trees [35] where d is the number of non-tree edges incident to u. Since operations on link-cut trees require $O(\log n)$ amortized time, the time to handle all edges is $O(m \log^2 n)$ (implying an $O(m \log^3 n)$ time algorithm for the minimum cut problem). As an open problem, Karger [18] asked (more than 20 years ago) whether the required link-cut tree operations can be done in constant amortized time per operation (implying an $O(m \log^2 n)$ time algorithm for the minimum cut problem). Karger even pointed out why one could perhaps achieve this: "*We are not using the full power of dynamic trees (in particular, the tree we are operating on is static, and the sequence of operations is known in advance).*" In this paper, we manage to achieve exactly that. We show how to order the link cut tree operations so that they can be handled efficiently in batches. We call such a batch a *bipartite problem* (see Definition 9).

Perhaps a reason that the running time of Karger's algorithm has not been improved in more than two decades is that it is not at all apparent that these bipartite problems can indeed be solved more efficiently. Coming up with an efficient solution to the bipartite problem requires a combination of several additional ideas. Like [26], we use heavy path decomposition, but in a different way. We develop a new decomposition of a tree that combines heavy path decomposition with biased divide and conquer, and use this decomposition in conjunction with a compact representation which we call topologically induced subtrees (see Definition 4). This compact representation turns out to be crucial not only for solving the bipartite problem, but also to the reduction from the original problem to a collection of bipartite problems.

1.2 Application to unweighted graphs

Karger's method is inherently randomized and obtaining a deterministic (or at least Las Vegas) near-linear time algorithm for the minimum cut in undirected weighted graphs is an interesting open problem. For unweighted graphs, such a deterministic algorithm was provided by Kawarabayashi and Thorup [21]. Later, Henzinger, Rao, and Wang [13] designed a faster $O(m \log^2 n (\log \log n)^2)$ time algorithm. Very recently Ghaffari, Nowicki and Thorup [7] introduced a new technique of random 2-out contractions and applied it to design an $O(\min\{m + n \log^3 n, m \log n\})$ time randomized algorithm that finds a minimum cut with high probability. We stress that the faster algorithms of Henzinger et al. and Ghaffari et al. work only for unweighted graphs, that is, for edge connectivity. Interestingly, the latter uses Karger's $O(m \log^3 n)$ time algorithm as a black box, and by plugging in our faster method one immediately obtains an improved running time of $O(\min\{m + n \log^2 n, m \log n\})$ for unweighted graphs.

1.3 Independent work

Independently to our work[1], Mukhopadhyay and Nanongkai [29] came up with an $O(m \log n + n \log^4 n)$ time algorithm for finding a minimal 2-respecting cut. While this improves Karger's bound for sufficiently dense graphs, it does not improve it for all graphs, and is randomized. Our algorithm uses a different (deterministic and simple) approach and strictly dominates both Karger's and Mukhopadhyay and Nanongkai's running time for all graphs. There are however benefits to the approach of [29] in other settings. Namely, they use it to obtain an algorithm that requires $\tilde{O}(n)$ cut queries to compute the min-cut, and a streaming algorithm that requires $\tilde{O}(n)$ space and $O(\log n)$ passes to compute the min-cut.

2 Preliminaries

2.1 Karger's algorithm

At a high level, Karger's algorithm [18] has two main steps. The input is a weighted undirected graph G. The first step produces a set $\{T_1, \ldots, T_s\}$ of $s = O(\log n)$ spanning trees such that, with high probability, the minimum cut of G 2-respects at least one of them. The second step deterministically computes for each T_i the minimum cut in G that 2-respects T_i. The minimum cut in G is the minimum among the cuts found in the second step.

Karger shows [18, Theorem 4.1] that producing the trees $\{T_1, \ldots, T_s\}$ in the first step can be done in $O(m + n \log^3 n)$ time, and that finding the minimum 2-respecting cut for all the T_i's in the second step can be done in $O(m \log^3 n)$ time. We will show that each of the steps can be implemented in $O(m \log^2 n)$ time. Showing this for the second step is the main result of the paper, and is presented in Section 3. For the first step, we essentially use Karger's proof. Since the first step was not the bottleneck in Karger's paper, proving a bound of $O(m + n \log^3 n)$ was sufficient for his purposes. Karger's concluding remarks suggest that he knew that the first step could be implemented in $O(m \log^2 n)$ time. For completeness, we prove the $O(m \log^2 n)$ bound by slightly modifying Karger's arguments and addressing a few issues that were not important in his proof. Readers proficient with Karger's algorithm can safely skip the proof.

▶ **Definition 1** (2-respecting and 2-constraining). *Given a spanning tree T and a cut $(S, \bar{S})$, we say that the cut 2-respects T and that T 2-constrains the cut if at most 2 edges of T cross the cut.*

▶ **Definition 2** (weighted tree packing). *Let G be an unweighted undirected graph. Let $\mathcal{T}$ be a set of spanning trees of G, where each tree $T \in \mathcal{T}$ is assigned a weight $w(T)$. We say that the* load *of an edge e of G (w.r.t. $\mathcal{T}$) is $\ell(e) = \sum_{T \in \mathcal{T} : e \in T} w(T)$. We say that $\mathcal{T}$ is a* weighted tree packing *if no edge has load exceeding 1. The* weight *of the packing $\mathcal{T}$ is $\tau = \sum_{T \in \mathcal{T}} w(T)$.*

▶ **Theorem 3.** *Given a weighted undirected graph G, in $O(m \log^2 n)$ time, we can construct a set $\mathcal{T}$ of $O(\log n)$ spanning trees such that, with high probability, the minimum cut 2-respects at least one of the trees in $\mathcal{T}$.*

Before proving the above theorem, we first describe how to precompute (in $O(m \log^2 n)$ time using Matula's algorithm [28]) a constant factor approximation of the weight c of the minimum cut in G.

[1] To be accurate, their work appeared on arXiv one day after ours.

Matula's algorithm [28]. Matula gave an $O(m/\epsilon)$ time algorithm that finds a $(2+\epsilon)$ approximation of the minimum cut in an unweighted graph G. The algorithm proceeds in iterations, where each iteration takes $O(m)$ time, and either finds a $(2+\epsilon)$ approximate cut, or produces a subgraph G' that contains the minimum cut of G, but has only a constant fraction of the edges of G. Hence there are $O(\log n)$ iterations, and the total running time is $O(m)$ for any fixed ϵ.

Matula's algorithm can be easily extended to the weighted setting. Each iteration can be implemented in $O(m \log n)$ time (cf. [15]), and produces a subgraph G' with a constant factor of the total edge weight of G. Thus, the algorithm produces a $(2+\epsilon)$ approximation of the minimum cut in a weighted graph G in $O(m \log n \log W)$ time, where W is the sum of edge weights in G.

The running time can be decreased to $O(m \log^2 n)$ at the expense of a worse constant factor approximation as follows. Let G be a weighted graph. Let c denote the weight of the minimum cut in G. Compute a maximum spanning tree T of G, and let w^* be the minimum weight of an edge in T. It is easy to see [17] that $w^* \leq c \leq n^2 w^*$. Contract all edges e with $w(e) > n^2 w^*$. Clearly, this does not affect the minimum cut. If $w^* \leq n^3$, then all edge weights are now bounded by n^5, and we can run Matula's algorithm in $O(m \log^2 n)$ time, and obtain a $(2+\epsilon)$-approximate minimum cut. Otherwise, set $\tilde{w}(e) \leftarrow \lfloor w(e)/\frac{w^*}{n^3} \rfloor$, and delete all edges with $\tilde{w} = 0$. Call the resulting graph $\tilde{G}$. Observe that $\tilde{G}$ has integer edge weights bounded by n^5, so we can find a $(2+\epsilon)$-approximate minimum cut in $\tilde{G}$ in $O(m \log^2 n)$ time. Now scale up the edge weights in $\tilde{G}$ by $\frac{w^*}{n^3}$. The weight of each edge in $\tilde{G}$ is now off from its original weight in G by at most $\frac{w^*}{n^3} \leq \frac{c}{n^3}$. Thus, the weights of any cut in $\tilde{G}$ and in G differ by at most c/n. Hence, an $(2+\epsilon)$-approximate minimum cut in $\tilde{G}$ is an $O(1)$-approximate minimum cut in G.

Proof of Theorem 3. Let $(S, \bar{S})$ be the partition of the vertices of G that forms a minimum cut. We assume that all weights are integers, each fitting in a single memory word. Since edges with weight greater than c never cross the minimum cut, we contract all edges with weight greater than our estimate for c, so that now the total weight of edges of G is $O(mc)$.

For the sake of presentation we think of an unweighted graph $\tilde{G}$, obtained from G by replacing an edge of weight w by w parallel edges. We stress that $\tilde{G}$ is never actually constructed by the algorithm. Let $\tilde{m}$ denote the number of edges of $\tilde{G}$. By the argument above, $\tilde{m} = O(mc)$. Let $p = \Theta(\log n/c)$. Let H be the unweighted multigraph obtained by sampling $\lceil p\tilde{m} \rceil$ edges of $\tilde{G}$ ($H = \tilde{G}$ if $c < \log n$). Clearly, the expected value of every cut in H is p times the value of the same cut in G. By [16, Lemma 5.1], choosing the appropriate constants in the sampling probability p guarantees that, with high probability, the value of every cut in H is at least $64/65$ times its expected value, and no more than $66/65$ times its expected value. It follows that, with high probability, (i) the minimum cut in H has value $c' = \Theta(\log n)$, and that (ii) the value of the cut in H defined by $(S, \bar{S})$ is at most $33c'/32$.

The conceptual process for constructing H can be carried out by randomly selecting $\lceil p\tilde{m} \rceil$ edges of G (with replacement) with probability proportional to their weights. Since the total edge weight of G is $O(mc)$, each selection can be easily performed in $O(\log(mc))$ time. Using a standard technique [24], each selection can actually be done with high probability in $O(\log m)$ time. Thus, the time to construct H is $O(p\tilde{m} \log m) = O(m \log^2 n)$. We emphasize that H is an unweighted multigraph with $m' = O(m \log n)$ edges, and note that we can assume no edge of H has multiplicity greater than c' (we can just delete extra copies).

Next, we apply the following specialized instantiation [36, Theorem 2] of Young's variant [38] of the Lagrangian packing technique of Plotkin, Shmoys, and Tardos [34]. It is shown [36, 38] that for an unweighted graph H with m' edges and minimum cut of size c', the following algorithm finds a weighted tree packing of weight $3c'/8 \leq \tau \leq c'$.

```
$\ell(e) := 0$ for all $e \in E(H)$
while there is no $e$ with $\ell(e) \geq 1$ do
    find a minimum spanning tree $T$ w.r.t. $\ell(\cdot)$
    $w(T) = w(T) + 1/(96 \ln m')$
    $\ell(e) = \ell(e) + 1/(96 \ln m')$ for all $e \in T$
end while
```

Karger [18, Lemma 2.3] proves that for a graph H with minimum cut c', any tree packing of weight at least $3c'/8$, and any cut $(S, \bar{S})$ of H of value at most $33c'/32$, at least a $1/8$ fraction of the trees (by weight) 2-constrain the cut $(S, \bar{S})$. Thus, a tree chosen at random from the packing according to the weights 2-constrains the cut $(S, \bar{S})$ with probability at least $1/8$. Choosing $O(\log n)$ trees guarantees that, with high probability, one of them 2-constrains the cut $(S, \bar{S})$, which is the minimum cut in G.

It remains to bound the running time of the packing algorithm. Observe that the algorithm increases the weight of some tree by $1/(96 \ln m')$ at each iteration. Since the weight τ of the resulting packing is bounded by c', there are at most $96c' \ln m' = O(\log^2 n)$ iterations. The bottleneck in each iteration is the time to compute a minimum spanning tree in H. We argue that this can be done in $O(m)$ time even though $m' = O(n \log n)$. To see this, first note that since H is a subgraph of G, H has at most m edges (ignoring multiplicities of parallel edges). Next note that it suffices to invoke the MST algorithm on a subgraph of H that includes just the edge with minimum load among any set of parallel edges. Since the algorithm always increases the load of edges by a fixed amount, the edge with minimum load in each set of parallel edges can be easily maintained in $O(1)$ time per load increase by maintaining a cyclic ordered list of each set of parallel edges and moving to choose the next element in this cyclic list whenever the load of the current element is incremented. Hence, we can invoke the randomized linear time MST algorithm [19] on a simple subgraph of H of size $O(m)$. It follows that the running time of the packing algorithm, and hence of the entire procedure, is $O(m \log^2 n)$. ◀

2.2 Link-cut trees

In our algorithm we will repeatedly use a structure that maintains a rooted tree T with costs on the edges under the following operations:

1. T.ADD(u, Δ) adds Δ to the cost of every edge on the path from u to the root,
2. T.PATH(u) finds the minimum cost of an edge on the path from u to the root,
3. T.SUBTREE(u) finds the minimum cost of an edge in the subtree rooted at u.

All three operations can be supported with a link-cut tree [35] in amortized $O(\log |T|)$ time.[2] We note that we only require these three operations and do not actually use the link and cut functionality of link-cut trees. Other data structures might also be suitable. See, e.g., the use of top-trees in [26].

[2] The original paper [35] did not include the third operation. However, as shown in [23, Appendix 17], it is not difficult to add it.

2.3 Topologically induced subtrees

For a rooted tree T and a node v we denote by T_v the subtree of T rooted at v. For an edge e of T we denote by T_e the subtree of T rooted at the lower endpoint of e.

Let T be a binary tree with edge-costs and n nodes, equipped with a data structure that can answer lowest common ancestor (LCA) queries on T in constant time [12]. Let $\Lambda = \{w_1, w_2, \ldots, w_s\}$ be a subset of nodes of T. We define a smaller tree T^Λ that is equivalent to T in the following sense:

▶ **Definition 4** (topologically induced tree). *We say that a tree T^Λ is topologically induced on T by Λ if for every $S \subseteq \Lambda$, the minimum cost edge $f \in T^\Lambda$ with $T^\Lambda_f \cap \Lambda = S$ has the same cost as the minimum cost edge $e \in T$ with $T_e \cap \Lambda = S$.*

To be clear, the above definition implies that, for any $S \subseteq \Lambda$, there is an edge $e \in T$ with $T_e \cap \Lambda = S$, if and only if there is an edge $f \in T^\Lambda$ with $T^\Lambda_f \cap \Lambda = S$. The term *topologically induced tree* will be justified by the construction in the following lemma.

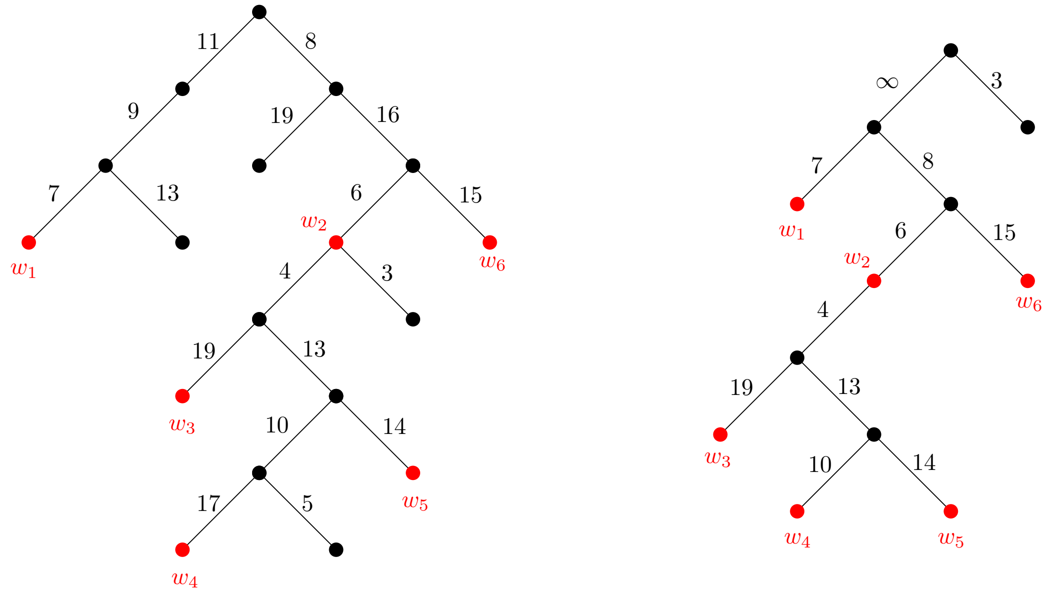


Figure 1 On the left: a tree and (in red) a set $\Lambda = \{w_1, w_2, \ldots, w_6\}$ sorted according to their preorder numbers. On the right: the corresponding topologically induced tree.

▶ **Lemma 5.** *There exists an algorithm that, given a binary tree T with edge costs, equipped with a link-cut data structure, and a list $\Lambda = \{w_1, w_2, \ldots, w_s\}$ of nodes of T, ordered according to their visit time in a preorder traversal of T, constructs in $O(\min\{|T|, s \log |T|\})$ time, a tree T^Λ of size $O(s)$ that is topologically induced on T by Λ.*

Proof. We define the tree T^Λ to be a tree over all nodes $w_i \in \Lambda$, together with the root and the lowest common ancestor in T of every pair of nodes w_i and w_j. For any two nodes $u, v \in T^\Lambda$, u is an ancestor of v in T^Λ if and only if u is an ancestor of v in T. Thus, each edge (u, v) of T^Λ corresponds to the u-to-v path in T. The edges on this path in T are exactly the edges e of T with $T_e \cap \Lambda = T^\Lambda_{(u,v)} \cap \Lambda$. Hence, the paths of T corresponding to distinct edges of T^Λ are edge disjoint. We define the cost of the edge (u, v) of T^Λ to be the minimum cost of an edge on the corresponding path in T. It follows that for every $\emptyset \neq S \subseteq \Lambda$, the minimum cost edge $f \in T^\Lambda$ with $T^\Lambda_f \cap \Lambda = S$ has the same cost as the minimum cost edge $e \in T$ with $T_e \cap \Lambda = S$. To guarantee that this condition holds for $S = \emptyset$ as well, we choose the edge e of T with the minimum cost such that $T_e \cap \Lambda = \emptyset$ and proceed as follows if such an edge exists. We create a new node v and change the root of T^Λ to v by making the old root of T^Λ a child of v via an edge with infinite cost. We then add a new edge incident to v, whose cost is set to the cost of e. This transformation does not change $T_e \cap \Lambda$ for any edge e of T^Λ, but now that condition with $S = \emptyset$ is satisfied for the new edge incident to the root.

We now turn to proving the construction time. We first prove that T^Λ consists of at most $2s$ nodes. This is because T^Λ consists only of the root, the nodes w_i and $\text{LCA}(w_i, w_{i+1})$. To see this, consider two nodes w_i and w_j with $i < j$ such that their lowest common ancestor u is different than w_i and w_j. Let u_ℓ (u_r) be the left (right) child of u. Then, w_i is a descendant of u_ℓ and w_j a descendant of u_r. Let i' be the largest index such that $w_{i'}$ is in the subtree rooted at u_ℓ. Then $u =$LCA$(w_{i'}, w_{i'+1})$.

We next prove that T^Λ can be constructed in $O(s)$ time. We use a method similar to constructing the Cartesian tree [37, 5] of a sequence: we scan $w_1, w_2, \ldots, w_s$ from the left to right while maintaining the subtree of T^Λ induced by $w_0 = \text{LCA}(w_1, w_s)$, and $w_1, w_2, \ldots, w_i$. initially, the subtree of T^Λ induced by w_0 and w_1 is just a single edge (w_0, w_1). We keep the rightmost path of the subtree of T^Λ induced by $w_0, w_1, \ldots, w_i$ on a stack, with the bottommost edge on the top. To process w_{i+1}, we first find $x = \text{LCA}(w_i, w_{i+1})$. Then, we pop from the stack all edges (u, v) such that u and v are both below (or equal to) x in T. Finally, we possibly split the edge on the top of the stack into two and push a new edge onto the stack. The amortized complexity of every step is constant, so the total time is $O(s)$.

Once T^Λ is constructed, we set the cost of every edge (u, v) in T^Λ to be the minimum cost of an edge on the u-to-v path in T. This can be done in $O(\log|T|)$ time per edge of T^Λ by first calling $T.\textsc{add}(u)(\infty)$, then $T.\textsc{path}(v)$ to retrieve the answer, and finally $T.\textsc{add}(u)(-\infty)$, for a total of $O(s \log|T|)$ time. Alternatively, we can explicitly go over the edges of the corresponding paths of T for every edge of T^Λ. We had argued above that these paths are disjoint so this takes $O(|T|)$ in total.

We also need to compute the cost of the edge e of T with minimum cost such that $T_e \cap \Lambda = \emptyset$. To this end, for each $v \in \Lambda$ we add ∞ to the cost of all edges on the path from v to the root of T. This takes $O(\min(|T|, s \log|T|)$ by either a bottom up computation on T, or using $T.\textsc{add}(u, \infty)$ for every $v \in \Lambda$. We then retrieve the edge with minimum cost in the entire tree in $O(\log|T|)$ time by a call to SUBTREE for the root of T, and then subtract ∞ from the cost of all edges on the path from v to the root of T for every $v \in \Lambda$. ◀

We will use the fact that the operation of taking the topologically induced subtree is composable in the following sense.

▶ **Proposition 6.** *Let T be a binary tree with edge-costs. Let $\Lambda_2 \subseteq \Lambda_1$ be subsets of nodes of T. Let T_1 be topologically induced on T by Λ_1 and T_2 be topologically induced on T_1 by Λ_2. Then T_2 is topologically induced on T_1 by Λ_2.*

3 Finding a Minimum 2-respecting Cut

Given a graph G and a spanning tree T of G, a cut in G is said to *2-respect* the tree T if at most two edges e, e' of T cross the cut (these edges are said to *determine* the cut). In this section we prove the main theorem of this paper:

▶ **Theorem 7.** *Given an edge-weighted graph G with n vertices and m edges and a spanning tree T, the minimum (weighted) cut in G that 2-respects T can be found in $O(m \log n)$ time.*

The minimum cut determined by every single edge can be easily found in $O(m+n)$ time [18, Lemma 5.1]. We therefore focus on finding the minimum cut determined by two edges. Observe that the cut determined by $\{e, e'\}$ is unique and consists of all edges $(u, v) \in G$ such that the u-to-v path in T contains exactly one of $\{e, e'\}$.

We begin by transforming T (in linear time) into a binary tree. This is standard and is done by replacing every node of degree d with a binary tree of size $O(d)$ where internal edges have weight ∞ and edges incident to leaves have their original weight. We also add an artificial root to T and connect it to the original root with an edge of weight ∞. From now we will be working with binary trees only.

3.1 Descendant edges

We first describe an $O(m \log n)$ time algorithm for finding the minimum cut determined by all pairs of edges $\{e, e'\}$ where e' is a descendant of e in T (i.e. e' is in the subtree of T rooted at the lower endpoint of e). To this end we shall efficiently find, for each edge e of T, the descendant edge e' that minimizes the weight of the cut determined by $\{e, e'\}$, and return the pair minimizing the weight of the cut.

For a given edge e of T, let T_e denote the subtree of T rooted at the lower endpoint of e. We associate with every node x a list of all edges (u, v) such that x is the lowest common ancestor of u and v. Note that all these lists can be computed in linear time, and form a partition of the edges of G. We also compute in $O(m)$ time, for every edge e of T, the total weight $A(e)$ of all edges with exactly one endpoint in T_e (in fact, this is the information computed by Karger's algorithm for the 1-respecting case). Note that $A(e)$ includes the weight of e.

Using a link-cut tree we maintain a *score* for every edge e of T. All scores are first initialized to zero. Then, for every edge (u, v) of G, we increase the score of all edges on the u-to-v path in T by the weight $w(u, v)$ of (u, v). This takes $O(\log n)$ time per edge (u, v) by calling $T.\textsc{add}(u, w(u, v))$, $T.\textsc{add}(v, (u, w(u, v))$ and $T.\textsc{add}(\mathrm{LCA}(u, v), -2(u, w(u, v))$. This initialization takes $O(m \log n)$ time. We then perform an Euler tour of T. When the tour first descends below a node x, for every edge (u, v) in the list of x, we decrease the score of all edges on the u-to-v path in T by $2w(u, v)$. Note that, at any point during this traversal, each edge (u, v) either contributes $w(u, v)$ or $-w(u, v)$ to the score of every edge on the u-to-v path in T, depending on whether the tour is yet to descend below $\mathrm{LCA}(u, v)$ or has already done so. As above, each update can be implemented in $O(\log n)$ time. Since every edge appears in exactly one list, the total time to perform all the updates is $O(m \log n)$.

▶ **Lemma 8.** *Consider the point in time when the Euler tour had just encountered an edge e for the first time. At that time, for every descendant edge e' of e, the weight of the cut determined by $\{e, e'\}$ is $A(e)$ plus the score of e'.*

Proof. Observe that the weight of the cut determined by $\{e, e'\}$ is the sum of weights of all edges with (1) one endpoint in $T_e - T_{e'}$ and the other not in T_e, or (2) one endpoint in $T_e - T_{e'}$ and the other in $T_{e'}$. Note that the edges satisfying (1) have exactly one endpoint in T_e, and hence their weight is accounted for in $A(e)$. However, $A(e)$ also counts the weight of edges (u, v) with one endpoint in $T_{e'}$ and the other not in T_e. Such edges do not cross the cut. Note that for such edges both e and e' are on the u-to-v path in T. The fact that e is on the u-to-v path implies that the traversal has already descended below $\mathrm{LCA}(u, v)$. Hence, (u, v) currently contributes $-w(u, v)$ to the score of e', offseting its contribution to

$A(e)$. Next note that the edges satisfying (2) are edges (u, v) with both u and v in T_e, which means that they are not accounted for in $A(e)$, and that the traversal did not yet descend below $\mathrm{LCA}(u, v)$. Hence the contribution of such an edge (u, v) to the score of e' is indeed the weight of (u, v). ◀

By the lemma, the descendant edge e' of e that minimizes the weight of the cut determined by $\{e, e'\}$ is the edge with minimum score in the subtree of e at that time. The score of this edge e' can be found in $O(\log n)$ time by calling $T.\textsc{subtree}(x)$, where x is the lower endpoint of e.

3.2 Independent edges

We now describe an $O(m \log n)$ time algorithm for finding the minimum cut determined by all pairs of edges $\{e, e'\}$ where e is *independent* of e' in T (i.e. e is not a descendant of e' and e' is not a descendant of e). We begin by showing that the problem can be reduced to the following *bipartite* problem:

▶ **Definition 9** (The bipartite problem). *Given two trees T_1 and T_2 with costs on the edges and a list of non-tree edges $L = \{(u, v) : u \in T_1, v \in T_2\}$ where each non-tree edge has a cost, find a pair of edges $e \in T_1$ and $e' \in T_2$ that minimize the sum of costs of e, of e', and of all non-tree edges $(u, v) \in L$ where u is in ${T_1}_e$, and v is in ${T_2}_{e'}$. The size of such a problem is defined as the number of non-tree edges in L plus the sizes of T_1 and T_2.*

▶ **Lemma 10.** *Given an edge-weighted graph G with n vertices and m edges and a spanning tree T, finding the minimum cut among those determined by a pair of independent edges $\{e, e'\}$ can be reduced in $O(m \log n)$ time to multiple instances of the bipartite problem of total size $O(m)$.*

Proof. Recall that every node w of T has at most two children. We create a separate bipartite problem for every node w of T that has exactly two children (x and y). This bipartite problem will be responsible for finding the minimum cut determined by all pairs of independent edges $\{e, e'\}$ where e is in T_x and e' is in T_y.

Throughout our description, note the distinction between edge weights and edge costs. The input graph G has edge weights, and the goal is to find the cut with minimum weight. The bipartite problems we define have edge costs, which are derived from the weights of edges in the input graph.

We initialize the cost of every edge of G to be zero. Then, for every edge $f = (u, v)$ of G, we add the weight of f to the cost of every edge on the u-to-v path. We maintain the costs in a link-cut tree so each f is handled in $O(\log n)$ time. Now consider any node w with exactly two children x and y, and any pair of independent edges $\{e, e'\}$ where e is in T_x and e' is in T_y. Observe that the edges crossing the cut determined by $\{e, e'\}$ are exactly the edges $f = (u, v)$ with one endpoint in T_e or in $T_{e'}$, and the other endpoint not in T_e nor in $T_{e'}$. Hence, the weight of the cut determined by $\{e, e'\}$ equals the sum of the cost of e plus the cost of e' minus twice the total weight of all non-tree edges $f = (u, v)$ such that u is in T_e and v is in $T_{e'}$.

We therefore define the bipartite problem for w as follows: (1) T_1 is composed of the edge (w, x) and the subtree rooted at x with costs as described above, (2) T_2 is composed of the edge (w, y) and the subtree rooted at y with the costs as described above, and (3) for every non-tree edge $f = (u, v)$ with weight c such that $\mathrm{LCA}(u, v) = w$ the list of non-tree edges L includes (u, v) with cost $-2c$. By construction, the solution to this bipartite problem is the pair of independent edges e, e' with $e \in T_x$ and $e' \in T_y$ that minimize the weight of the cut in G defined by e and e'.

The only issue with the above bipartite problem is that the overall size of all bipartite problems (over all nodes w) might not be $O(m)$. This is because the edges of T might appear in the bipartite problems defined for more than a single node w. In order to guarantee that the overall size of all bipartite problems is $O(m)$, we construct a compact bipartite problem using the topologically induced trees of Lemma 5.

We construct in $O(m)$ time a constant-time LCA data structure [12] for T. In overall $O(m \log n)$ time, we construct, for each node $w \in T$ with exactly two children x and y:

1. A list L_w of all non-tree edges (u, v) with $\mathrm{LCA}(u, v) = w$.
2. A list $\Lambda_x = \{w, x\} \cup \{u : (u, v) \in L_w \text{ and } u \in T_x\}$, sorted according to their visit time in a preorder traversal of T.
3. A list $\Lambda_y = \{w, y\} \cup \{v : (u, v) \in L_w \text{ and } v \in T_y\}$, sorted according to their visit time in a preorder traversal of T.

These lists require $O(m)$ space and can be easily computed in $O(m \log n)$ time by going over the non-tree edges, because each non-tree edge is in the list L_w of a unique node w.

The list L for the compact bipartite problem is identical to the list L for the non-compact problem. The tree T_1° (T_2°) for the compact bipartite problem of w is the topologically tree induced on $(w, x) \cup T_x$ ($(w, y) \cup T_y$) by Λ_x (Λ_y). This is done in $O(|\Lambda_x| \log n)$ ($O(|\Lambda_y| \log n)$) time by invoking Lemma 5. It follows that the total time for constructing all compact bipartite problems is $O(m \log n)$ and the their total space is $O(m)$.

It remains to argue that the solution to the compact bipartite problem is identical to the solution to the non-compact one. Observe that the cost of a solution e, e' for the non-compact bipartite problem is the cost of e plus the cost of e' plus the cost of all edges in L with one endpoint in ${T_1}_e \cap \Lambda$ and the other endpoint in ${T_2}_{e'} \cap \Lambda$.

Consider now any pair of edges e and e' for the non-compact bipartite problem. By definition of topologically induced trees, the minimum cost edge $f \in T_1^\circ$ with ${T_1^\circ}_f \cap \Lambda = {T_1}_e \cap \Lambda$, has cost not exceeding that of e. An analogous argument holds for e' and an edge f' of T_2°. Hence, the cost of the optimal solution for the compact problem is not greater than that of the non-compact problem. Conversely, for any edge f in T_1°, there exists an edge e in T_1 with cost not exceeding that of f and ${T_1^\circ}_f \cap \Lambda = {T_1}_e \cap \Lambda$. Hence, the cost of the optimal solution for the compact problem is not less than that of the non-compact problem. It follows that the two solutions are the same. ◀

The proof of Theorem 7 follows from the above reduction and the following solution to the bipartite problem:

▶ **Lemma 11.** *A bipartite problem of size m can be solved in $O(m \log m)$ time.*

Proof. Recall that in the bipartite problem we are given two trees T_1 and T_2 with edge-costs and a list of non-tree edges $L = \{(u, v) : u \in T_1, v \in T_2\}$ where each non-tree edge has a cost. To prove the lemma, we describe a recursive $O(m \log m)$ time algorithm that finds, for every edge $e \in T_1$, the best edge $e' \in T_2$ (i.e. the edge e' that minimizes the sum of costs of e' and of all non-tree edges $(u, v) \in L$ where u in ${T_1}_e$ and $v \in {T_2}_{e'}$).

We begin by applying a standard heavy path decomposition [12] to T_1, guided by the number of non-tree edges: The *heavy* edge of a node of T_1 is the edge leading to the child whose subtree has the largest number of incident non-tree edges in L (breaking ties arbitrarily). The other edges are called *light*. The maximal sets of connected heavy edges define a decomposition of the nodes into *heavy paths*.

We define a *fragment* of the tree T_1 to be a subtree of T_1 formed by a contiguous subpath $u_1 - u_2 - \cdots - u_k$ of some heavy path of T_1, together with all subtrees hanging from this subpath via light edges. Given a fragment f, let $L(f) = \{(x_1, y_1), (x_2, y_2) \ldots, (x_\ell, y_\ell)\}$ be the set of edges (x, y) of L with $x \in f$. We define the induced subtree $T_2(f)$ to be the tree topologically induced on T_2 by the root of T_2 and $\{y_1, y_2, \ldots, y_\ell\}$. The size of $T_2(f)$ is $|T_2(f)| = O(|L(f)|)$. We also define a modified induced subtree $T'_2(f)$ as follows. Let $L_\downarrow(f)$ be the set of edges $(x, y) \in L$ with x in the subtree rooted at the heavy child of the last node u_k of the fragment f (if such a heavy child exists). Consider the tree T_2, where the cost of each edge e' of T_2 is increased by the total cost of all edges $(x, y) \in L_\downarrow(f)$, where y is in $T_{2e'}$. The modified induced subtree $T'_2(f)$ is defined as the tree topologically induced by the root of T_2 and $\{y_1, y_2, \ldots, y_\ell\}$ on this modified T_2.

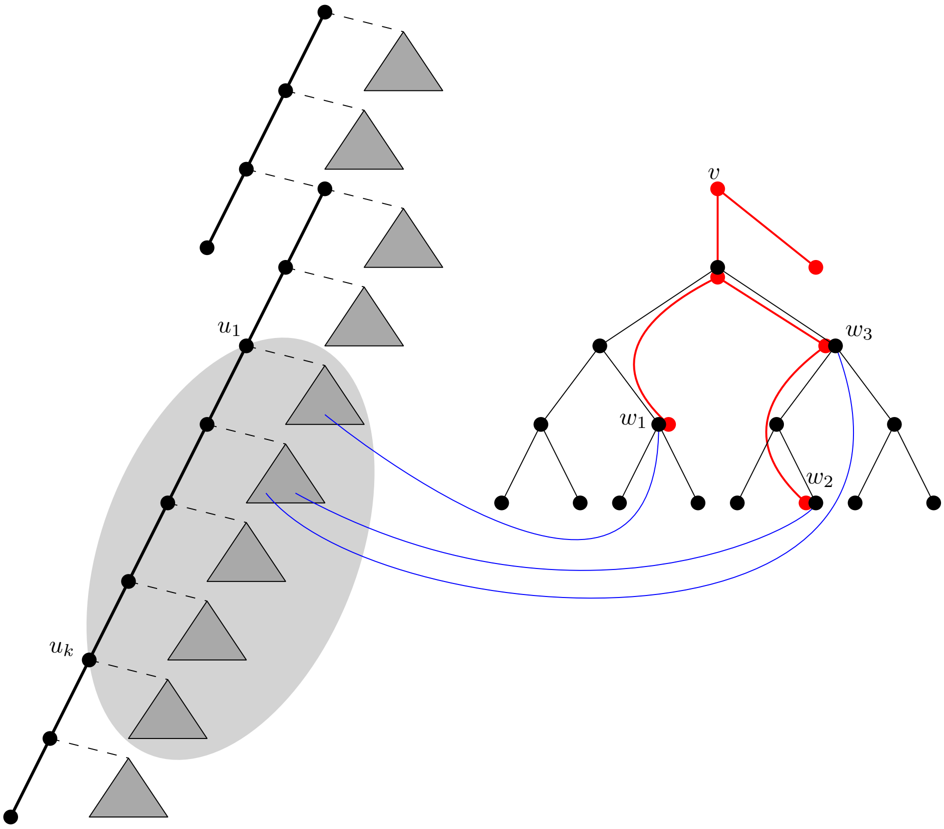


Figure 2 On the left: A fragment (in light gray) in the tree T_1, defined by the top node u_1 and the bottom node u_k, both laying on the same heavy path (solid edges). The triangles (in dark gray) are the subtrees hanging from the heavy paths via light edges (dashed). On the right: The tree T_2 (black) connected to the fragment via three non-tree edges (blue). The endpoints w_1, w_2, w_3 of these edges define the topologically induced tree (in red).

We are now ready to describe the recursion. The input to a recursive call is a fragment f of T_1 and the list $(x_1, y_1), (x_2, y_2), \ldots, (x_\ell, y_\ell)$ of all non-tree edges in L with x_i in f, together with $T_2(f)$ and $T'_2(f)$. A fragment f is specified by the *top* node (u_1) and the *bottom* node (u_k) of the corresponding subpath of a heavy path of T_1. In the first call, f is specified by the root of T_1 and the leaf ending the heavy path of T_1 that contains the root. That is, in the first call f is the entire tree T_1. The list of non-tree edges for the first call is the entire list L. The recursion works by selecting the *middle* node of the subpath, defined as follows: We define the *light size* s_i of node u_i as the number of non-tree edges $(x, y) \in L$ where either $x = u_i$ or x is in the subtree rooted at the light child of u_i. Note that $s_1 + s_2 + \ldots + s_k = |L(f)|$. If

$s_1 > |L(f)|/2$ then the middle node is defined as u_1. Otherwise, the middle node is defined as the node u_i such that $s_1 + \ldots + s_{i-1} \leq |L(f)|/2$ but $s_1 + \ldots + s_i > |L(f)|/2$. We keep, for every heavy path P of T_1 a list of the nodes of P with non-zero light size, ordered according to their order on P. We find the middle node u_i in $O(|L(f)|)$ time by going over the nodes in this list one after the other until we encounter the middle node.

After identifying the middle node u_i we apply recursion on the following three fragments: the fragment defined by subpath $u_1 - \cdots - u_{i-1}$, the fragment defined by subpath $u_{i+1} - \cdots - u_k$, and the fragment consisting of the entire subtree rooted at the light child of u_i. Before a recursive call to fragment g, we construct the appropriate $T_2(g)$ and $T_2'(g)$. The induced tree $T_2(g)$ can be computed from $T_2(f)$ in $O(|T_2(f)|) = O(|L(f)|)$ time by invoking Lemma 5 on $T_2(f)$ with $\Lambda_g = r \cup \{y : (x, y) \in L(g)\}$, where r is the root of T_2. Note that we had defined $T_2(g)$ as the topologically induced tree on T_2 by Λ_g, not on $T_2(f)$ by Λ_g. However, since $\Lambda_g \subseteq \Lambda_f$, by Proposition 6, the two definitions are equivalent.

For constructing $T_2'(g)$ from $T_2'(f)$, we first need to increase the cost of each edge $\tilde{e}$ of $T_2'(f)$ by the total cost of edges in $L_\downarrow(g) \setminus L_\downarrow(f)$ that are incident to $T_2'(f)_{\tilde{e}}$. This can be done in a single bottom up traversal of $T_2'(f)$ in $O(|T_2'(f)|)$ time. Then, we invoke Lemma 5 on $T_2'(f)$ with Λ_g to obtain $T_2'(g)$. To summarize, constructing the trees $T_2(g)$ and $T_2'(g)$ for all three recursive subproblems takes $O(|L(f)|)$ time.

The three recursive calls will take care of finding the best edge $e' \in T_2$ for every edge $e \in T_1$ included in one of the recursive problems. It only remains to handle the three edges that do not belong to any of the recursive problems; the edge between u_i and its light child, the edge (u_{i-1}, u_i), and the edge (u_i, u_{i+1}). For each such edge e we describe a procedure that finds its best $e' \in T_2(f)$ in time $O(|T_2(f)|)$.

Recall that, by definition of the bipartite problem, the best edge e' for e is the edge e' of T_2 minimizing the cost of e' plus the cost of all non-tree edges $(x, y) \in L$ with $x \in T_{1e}$, and $y \in T_{2e'}$. For the case where e is the edge between u_i and its light child, $T_{1e} = T_1(f)_e$. We therefore mark all non-tree edges $(x, y) \in L(f)$, where x is in $T_1(f)_e$. A non-efficient solution would work directly on T_2 by propagating, in a bottom up traversal of T_2, the cost of all marked edges so that, after the propagation, the cost of every edge e' in T_2 has been increased by the total cost of all non-tree edges $(x, y) \in L$ with $x \in T_1(f)_e$, and with $y \in T_{2e'}$. Then we can take the edge $e' \in T_2$ with minimum cost. However, this would take $O(|T_2|) = O(m)$, which is too slow. Instead, we perform the propagation in $T_2(f)$. Namely, in a bottom up traversal of $T_2(f)$, we propagate the cost of all marked edges so that, after the propagation, the cost of every edge e' in $T_2(f)$ has been increased by the total cost of all non-tree edges $(x, y) \in L(f)$ with $x \in T_1(f)_e$, and with $y \in T_2(f)_{e'}$. This takes $O(|T_2(f)|) = O(|L(f)|)$ time. Since the propagation process affects all the edges $\tilde{e}$ with the same $T_{2\tilde{e}} \cap \Lambda_f$ in the same way, the definition of topologically induced tree guarantees that the edges with minimum cost in T_2 and in $T_2(f)$ have the same cost, so using $T_2(f)$ instead of T_2 is correct.

The procedure for the cases where e is the edge (u_{i-1}, u_i) or (u_i, u_{i+1}) is identical, except that we apply it with $T_2'(f)$ instead of $T_2(f)$. This difference stems from the fact that applying the above procedure on $T_2(f)$ only considers the costs of the non-tree edges in $L(f)$, but not the costs of the non-tree edges in $L_\downarrow(f)$, which might also cross cuts involving the edges (u_{i-1}, u_i) or (u_i, u_{i+1}). The definition of the costs of edges in $T_2'(f)$ takes into account the contribution of costs of non-tree edges in $L_\downarrow(f)$. The rest of the propagation procedure and the proof of its correctness remain unchanged.

To analyze the overall running time, let $T(m)$ be the time to handle a fragment f corresponding to a whole heavy path, and $T'(m)$ be the time to handle a fragment f corresponding to a proper subpath of some heavy path, where $m = |L(f)|$. Then $T(m) =$

$T'(m_1)+T(m_2)+T'(m_3)$ for some m_1, m_2, m_3, where $m_1+m_2+m_3=m$ since the subproblems are disjoint, $m_1, m_3 \le m/2$ by the choice of the middle node, and $m_2 \le m/2$ by the definition of a heavy path. This is since the light child of u_i does not have more incident non-tree edges in its subtree than the number of non-tree edges incident to the subtree of the heavy child u_{i+1}. When f corresponds to a whole heavy path the number of non-tree edges incident to the subtree of u_{i+1} is exactly m_3. Similarly, for the case where f does not correspond to a whole heavy path, $T'(m)=T'(m_1)+T(m_2)+T'(m_3)$ for some m_1, m_2, m_3, where $m_1+m_2+m_3=m$ and $m_1, m_3 \le m/2$ (but now we cannot guarantee that $m_2 \le m/2$). Considering the tree describing the recursive calls, on any path from the root (corresponding to the fragment consisting of the whole T_1) to a leaf, we have the property that the value of m decreases by at least a factor of 2 every two steps. Hence, the depth of the recursion is $O(\log m)$. It follows that the total time to handle a bipartite problem of size m is $O(m\log m)$. ◀

References

1 Stephen Alstrup, Jacob Holm, Kristian De Lichtenberg, and Mikkel Thorup. Maintaining information in fully dynamic trees with top trees. *ACM Trans. Algorithms*, 1(2):243–264, 2005.

2 Gerth Stølting Brodal, Rolf Fagerberg, and Christian N. S. Pedersen. Computing the quartet distance between evolutionary trees in time $O(n\log^2 n)$. In *12th ISAAC*, pages 731–742, 2001.

3 Lester R. Ford and Delbert R. Fulkerson. Flows in network. *Princeton Univ. Press*, 1962.

4 Harold N. Gabow. A matroid approach to finding edge connectivity and packing arborescences. *J. Comput. Syst. Sci.*, 50(2):259–273, 1995. Announced at STOC 1991.

5 Harold N. Gabow, Jon Louis Bentley, and Robert Endre Tarjan. Scaling and related techniques for geometry problems. In *16th STOC*, pages 135–143, 1984.

6 Barbara Geissmann and Lukas Gianinazzi. Parallel minimum cuts in near-linear work and low depth. In *30th SPAA*, pages 1–11, 2018.

7 Mohsen Ghaffari, Krzysztof Nowicki, and Mikkel Thorup. Faster algorithms for edge connectivity via random 2-out contractions. *CoRR*, abs/1909.00844, 2019. arXiv:1909.00844.

8 Andrew V. Goldberg and Satish Rao. Beyond the flow decomposition barrier. *J. ACM*, 45(5):783–797, 1998.

9 Andrew V. Goldberg and Robert Endre Tarjan. A new approach to the maximum-flow problem. *J. ACM*, 35(4):921–940, 1988.

10 Ralph E. Gomory and Tien Chung Hu. Multi-terminal network flows. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):551–570, 1961.

11 Jianxiu Hao and James B. Orlin. A faster algorithm for finding the minimum cut in a directed graph. *J. Algorithms*, 17(3):424–446, 1994.

12 David Harel and Robert E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM J. Comput.*, 13(2):338–355, 1984.

13 Monika Henzinger, Satish Rao, and Di Wang. Local flow partitioning for faster edge connectivity. In *28th SODA*, pages 1919–1938, 2017.

14 David R. Karger. Global min-cuts in RNC, and other ramifications of a simple min-cut algorithm. In *4th SODA*, pages 21–30, 1993.

15 David R. Karger. *Random Sampling in Graph Optimization Problems.* PhD thesis, Stanford University, Stanford, CA 94305, 1994.

16 David R. Karger. Random sampling in cut, flow, and network design problems. *Math. Oper. Res.*, 24(2):383–413, 1999. Announced at STOC 1994.

17 David R. Karger. Random sampling in cut, flow, and network design problems. *Math. Oper. Res.*, 24(2):383–413, 1999.

18 David R. Karger. Minimum cuts in near-linear time. *J. ACM*, 47(1):46–76, 2000. Announced at STOC 1996.

19 David R. Karger, Philip N. Klein, and Robert Endre Tarjan. A randomized linear-time algorithm to find minimum spanning trees. *J. ACM*, 42(2):321–328, 1995.

20 David R. Karger and Clifford Stein. A new approach to the minimum cut problem. *J. ACM*, 43(4):601–640, 1996.

21 Ken-ichi Kawarabayashi and Mikkel Thorup. Deterministic edge connectivity in near-linear time. *J. ACM*, 66(1):4:1–4:50, 2019.

22 Valerie King, S. Rao, and Robert Endre Tarjan. A faster deterministic maximum flow algorithm. *J. Algorithms*, 17(3):447–474, 1994. Announced at SODA 1992.

23 Philip N. Klein and Shay Mozes. Optimization algorithms for planar graphs. `http://planarity.org`. Book draft.

24 Donald Knuth and Andrew Yao. *Algorithms and Complexity: New Directions and Recent Results*, chapter The complexity of nonuniform random number generation, pages 357–428. Academic Press, 1976.

25 Yin Tat Lee and Aaron Sidford. Path finding methods for linear programming: Solving linear programs in $\tilde{O}(\sqrt{rank})$ iterations and faster algorithms for maximum flow. In *55th FOCS*, pages 424–433, 2014.

26 Antonio Molina Lovett and Bryce Sandlund. A simple algorithm for minimum cuts in near-linear time. *CoRR*, abs/1908.11829, 2019. `arXiv:1908.11829`.

27 Aleksander Madry. Computing maximum flow with augmenting electrical flows. In *57th FOCS*, pages 593–602, 2016.

28 David W. Matula. A linear time $2 + \epsilon$ approximation algorithm for edge connectivity. In *4th SODA*, pages 500–504, 1993.

29 Sagnik Mukhopadhyay and Danupon Nanongkai. Weighted min-cut: Sequential, cut-query and streaming algorithms. In *52nd STOC*, 2020. To appear. See also `arXiv:1911.01651`.

30 Hiroshi Nagamochi and Toshihide Ibaraki. Computing edge-connectivity in multigraphs and capacitated graphs. *SIAM J. Discrete Math.*, 5(1):54–66, 1992.

31 Hiroshi Nagamochi and Toshihide Ibaraki. A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph. *Algorithmica*, 7(5&6):583–596, 1992.

32 Crispin St. J. A. Nash-Williams. Edge disjoint spanning trees of finite graphs. *Journal of the London Mathematical Society*, 36:445–450, 1961.

33 James B. Orlin. Max flows in $O(nm)$ time, or better. In *45th STOC*, pages 765–774, 2013.

34 Serge A. Plotkin, David B. Shmoys, and Éva Tardos. Fast approximation algorithms for fractional packing and covering problems. *Math. Oper. Res.*, 20(2):257–301, 1995. `doi:10.1287/moor.20.2.257`.

35 Daniel D. Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26(3):362–391, 1983.

36 Mikkel Thorup and David R. Karger. Dynamic graph algorithms with applications. In *7th SWAT*, pages 1–9, 2000.

37 Jean Vuillemin. A unifying look at data structures. *Communications of the ACM*, 23(4):229–239, 1980.

38 Neal E. Young. Randomized rounding without solving the linear program. In *6th SODA*, pages 170–178, 1995.

Sparse Recovery for Orthogonal Polynomial Transforms

Anna Gilbert
Department of Mathematics, Yale University, New Haven, CT, USA
anna.gilbert@yale.edu

Albert Gu
Department of Computer Science, Stanford University, CA, USA
albertgu@stanford.edu

Christopher Ré
Department of Computer Science, Stanford University, CA, USA
chrismre@cs.stanford.edu

Atri Rudra
Department of Computer Science and Engineering, University at Buffalo, NY, USA
atri@buffalo.edu

Mary Wootters
Departments of Computer Science and Electrical Engineering, Stanford University, CA, USA
marykw@stanford.edu

Abstract

In this paper we consider the following *sparse recovery* problem. We have query access to a vector $\mathbf{x} \in \mathbb{R}^N$ such that $\hat{\mathbf{x}} = \mathbf{F}\mathbf{x}$ is k-sparse (or nearly k-sparse) for some orthogonal transform $\mathbf{F}$. The goal is to output an approximation (in an ℓ_2 sense) to $\hat{\mathbf{x}}$ in sublinear time. This problem has been well-studied in the special case that $\mathbf{F}$ is the Discrete Fourier Transform (DFT), and a long line of work has resulted in sparse Fast Fourier Transforms that run in time $O(k \cdot \text{polylog} N)$. However, for transforms $\mathbf{F}$ other than the DFT (or closely related transforms like the Discrete Cosine Transform), the question is much less settled.

In this paper we give sublinear-time algorithms – running in time $\text{poly}(k \log(N))$ – for solving the sparse recovery problem for orthogonal transforms $\mathbf{F}$ that arise from *orthogonal polynomials*. More precisely, our algorithm works for any $\mathbf{F}$ that is an orthogonal polynomial transform derived from *Jacobi polynomials*. The Jacobi polynomials are a large class of classical orthogonal polynomials (and include *Chebyshev* and *Legendre* polynomials as special cases), and show up extensively in applications like numerical analysis and signal processing. One caveat of our work is that we require an assumption on the sparsity structure of the sparse vector, although we note that vectors with random support have this property with high probability.

Our approach is to give a very general reduction from the k-sparse sparse recovery problem to the 1-sparse sparse recovery problem that holds for any flat orthogonal polynomial transform; then we solve this one-sparse recovery problem for transforms derived from Jacobi polynomials. Frequently, sparse FFT algorithms are described as implementing such a reduction; however, the technical details of such works are quite specific to the Fourier transform and moreover the actual implementations of these algorithms do not use the 1-sparse algorithm as a black box. In this work we give a reduction that works for a broad class of orthogonal polynomial families, and which uses any 1-sparse recovery algorithm as a black box.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms; Theory of computation → Algorithm design techniques

Keywords and phrases Orthogonal polynomials, Jacobi polynomials, sublinear algorithms, sparse recovery

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.58

Category Track A: Algorithms, Complexity and Games

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 58; pp. 58:1–58:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Related Version https://arxiv.org/abs/1907.08362

Funding *Anna Gilbert*: ACG is partially funded by a Simons Foundation Fellowship.
Albert Gu: AG and CR gratefully acknowledge the support of DARPA under Nos. FA87501720095 (D3M) and FA86501827865 (SDH), NIH under No. U54EB020405 (Mobilize), NSF under Nos. CCF1763315 (Beyond Sparsity) and CCF1563078 (Volume to Velocity), ONR under No. N000141712266 (Unifying Weak Supervision), the Moore Foundation, NXP, Xilinx, LETI-CEA, Intel, Google, NEC, Toshiba, TSMC, ARM, Hitachi, BASF, Accenture, Ericsson, Qualcomm, Analog Devices, the Okawa Foundation, and American Family Insurance, Google Cloud, Swiss Re, Stanford Bio-X SIG Fellowship, and members of the Stanford DAWN project: Intel, Microsoft, Teradata, Facebook, Google, Ant Financial, NEC, SAP, VMWare, and Infosys. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views, policies, or endorsements, either expressed or implied, of DARPA, NIH, ONR, or the U.S. Government.
Atri Rudra: AR is partially funded by NSF grant CCF-1763481.
Mary Wootters: MW is partially funded by NSF CAREER award CCF-1844628.

Acknowledgements We would like to thank Mark Iwen for useful conversations. Thanks also to Stefan Steinerberger for showing us the proof of a number theory lemma we used in the 1-sparse Jacobi solver and graciously allowing us to use it, and to Clément Canonne for helpful comments on our manuscript. We also thank Eric Winsor for pointing out an error in a lemma in an earlier version of this paper. Thanks also to anonymous reviewers who pointed out some missing references and whose comments improved the presentation of the paper.

1 Introduction

In this paper, we consider the following *sparse recovery* problem. Suppose that we have query access to a vector $\mathbf{x} \in \mathbb{R}^N$, which has the property that for a fixed *orthogonal transform matrix* $\mathbf{F}$, $\hat{\mathbf{x}} = \mathbf{F}\mathbf{x}$ is k-sparse (or approximately k-sparse, in the sense that $\hat{\mathbf{x}}$ is close in ℓ_2 distance to a k-sparse vector). The goal is to recover an approximation $\hat{\mathbf{z}}$ to $\hat{\mathbf{x}}$, so that $\|\hat{\mathbf{x}} - \hat{\mathbf{z}}\|_2$ is small with high probability, as quickly as possible.

Variants of this problem have been studied extensively over several decades – we refer the reader to the book [16] for many examples and references. One particularly well-studied example is the *sparse Fast Fourier Transform* (sFFT) – see the survey [18] and the references therein. In this case, the matrix $\mathbf{F}$ is taken to be the Discrete Fourier Transform (DFT) and a long line of work has produced near-optimal results: algorithms with running time $O(k \operatorname{polylog}(N))$ and sample complexity $O(k \log N)$ [8, 9, 19, 23–27]– though not all of these works achieve both the claimed sample complexity and runtime at the same time.

We study the sparse recovery problem for a more general class of transforms $\mathbf{F}$ called *orthogonal polynomial transforms,* and in particular those that arise from *Jacobi polynomials,* a broad class of orthogonal polynomials (OPs). Jacobi polynomials include as special cases many familiar families of OPs, including Gegenbauer and in particular Chebyshev, Legendre, and Zernike[1] polynomials, and the corresponding OP transforms appear throughout numerical analysis and signal processing.

Despite the progress on the sFFT described above, much remains unknown for general orthogonal polynomial transforms. As discussed more in Section 1.2 below, the *sample complexity* of the sparse recovery problem is well understood, and the "correct" answer

[1] To be more precise the *radial* component of a Zernike polynomial is a Gegenbauer and hence, a Jacobi polynomial.

is known to be $\Theta(k\,\mathrm{polylog}(N))$ queries to $\mathbf{x}$. However, the algorithmic results that go along with these sample complexity bounds result in $\mathrm{poly}(N)$ time algorithms. Our goal in this work will be *sublinear* time algorithms as well as sublinear sample complexity. There are sublinear-time algorithms available for the special cases of Chebyshev and Legendre polynomials that work by essentially reducing to the Fourier case [21]. For general Jacobi polynomials, such reductions are not available. We elaborate in the full version of the paper why reducing general Jacobi polynomials to the Fourier case does not seem easy. There are also algorithms based on Prony's method, some of which work for quite general families of OPs [29]. However these general results require exact sparsity; to the best of our knowledge versions of Prony's method that are provably robust to noise are restricted to classes of OPs similar to the Fourier transform.

Results

In this work, we give the first (to the best of our knowledge) sublinear-time algorithms with provable guarantees for the (approximately-)sparse recovery problem for general orthogonal transforms derived from Jacobi polynomials. We discuss our results in more detail in Section 3 and briefly summarize them here. Our algorithms run in time $\mathrm{poly}(k\log(N))$ and given query access to $\mathbf{v} = \mathbf{F}^{-1}\hat{\mathbf{v}}$, can find approximations to $\hat{\mathbf{v}}$ when $\hat{\mathbf{v}}$ is approximately k-sparse of an appropriate form. More precisely, we can handle vectors $\hat{\mathbf{v}} = \hat{\mathbf{x}} + \hat{\mathbf{w}}$ where $\hat{\mathbf{x}}$ is k-sparse with a "spread-out" support (made precise in Definition 2.3), and $\hat{\mathbf{w}}$ is an adversarial noise vector with sufficiently small ℓ_2 norm. We obtain guarantees of the following flavor: for any such vector $\mathbf{v}$, we can find $\hat{\mathbf{z}}$ such that $\|\hat{\mathbf{z}} - \hat{\mathbf{x}}\|_2 \leq 0.01\|\hat{\mathbf{x}}\|_2$ with high probability.

We note that these results are weaker than the results for the sFFT: our sample complexity and running time are polynomially larger, and we need stronger assumptions on the sparse signals. However, we also note that the decade or so of work on the sFFT culminating in the results above began with similar results (see [17], for example, in which the dependence on k is an unspecified polynomial) and we hope that this work will similarly be a first step towards near-optimal algorithms for general orthogonal polynomial transforms.

Techniques

Our techniques follow the outline of existing algorithms for the sFFT, although as we elaborate on in Section 1.3, the situation for general Jacobi polynomials is substantially more complicated. More precisely, we first give a very general reduction, which reduces the k-sparse case to the 1-sparse case. The idea of such a reduction was implicit in the sFFT literature, but previous work has relied heavily on the structure of the DFT. Our reduction applies to a broad class of OPs including Jacobi polynomials. Next, we show how to solve the 1-sparse recovery problem for general Jacobi polynomials. The basic idea is to use known approximations of Jacobi polynomial evaluations by certain cosine evaluations [35] in order to iteratively narrow down the support of the unknown 1-sparse vector. We give a more detailed overview of our techniques in Section 1.3.

Organization

For the rest of the introduction, we briefly introduce orthogonal polynomial transforms, discuss previous work, and give a high-level overview of our approach. After that we introduce the formal notation and definitions we need in Section 2, after which we state our results more formally in Section 3.

Due to space constraints, proofs of our main results are in the full version of the paper, including the reduction from k to 1-sparse recovery, the 1-sparse recovery algorithm for Jacobi polynomials, and the resulting k-sparse recovery algorithm for Jacobi polynomials.

1.1 Orthogonal Polynomial Transforms

Orthogonal polynomials (OPs) play an important role in classical applied mathematics, mathematical physics, and the numerical analysis necessary to simulate solutions to such problems. We give more precise definitions in Section 2; briefly, a family of orthogonal polynomials $p_0(X), p_1(X), \ldots$ is a collection of polynomials defined on an interval $\mathcal{D}$ of $\mathbb{R}$, that are pairwise orthogonal with respect to a (non-negative) weight function w.

Orthogonal polynomials naturally give rise to (discrete) orthogonal polynomial transforms. In particular, we define the transform as follows– $\mathbf{F}$ is an $N \times N$ matrix, with each column corresponding to an orthogonal polynomial $p_0, \ldots, p_{N-1}$ and each row an evaluation point $\lambda_0, \ldots, \lambda_{N-1}$ in a suitable domain and suitably normalized so that it is an orthogonal matrix (Definition 2.1). A familiar example might be the DFT: in this language, the DFT matrix is defined by the polynomials $1, X, X^2, \ldots, X^{N-1}$, evaluated at points $\lambda_j = \omega^j$ where ω is the Nth root of unity.[2] Like the Fourier Transform, it is known that all OP transforms admit "fast" versions, allowing matrix-vector multiplication in time $O(N \log^2(N))$ [15].[3] Thus, our problem of sparse recovery for OP transforms is a natural extension of the sFFT problem, with applications to several areas mentioned below.

In this work we study *Jacobi polynomials* (defined formally in Section 2), which are a very general class of orthogonal polynomials. These include Chebyshev polynomials, Legendre polynomials, Zernike polynomials and more generally Gegenbauer polynomials. These OP families show up in many places. For example, Zernike polynomials are a family of orthogonal polynomials on the unit disk that permit an analytic expression of the 2D Fourier transform on the disk. They are used in optics and interferometry [36]. They can be utilized to extract features from images that describe the shape characteristics of an object and were recently used for improved cancer imaging [39]. Different families of orthogonal polynomials give rise to different quadrature rules for numerical integration [12, 33]. Specifically, Chebyshev polynomials are used for numerical stability (see e.g. the ChebFun package [3]) as well as approximation theory (see e.g. Chebyshev approximation [1]). Chebyshev polynomials also have certain optimal *extremal* properties, which has resulted in many uses in theoretical computer science, including in learning theory, quantum complexity theory, linear systems solvers, eigenvector computation, optimization, and more [28]. Further, Jacobi polynomials form solutions of certain differential equations [2].

More recently, orthogonal polynomials and orthogonal polynomial transforms have found applications in various facets of machine learning. For example, Dao et al. [13] leverage the connection between orthogonal polynomials and quadrature to derive rules for computing

[2] We note that in this work we consider a setting slightly different than this example, where $\mathcal{D} = [-1, 1]$ rather than $\mathbf{S}^1$.

[3] We note that even though the work of [15] has in some sense solved the problem of computing any OP transform in near-linear time, many practical issues still remain to be resolved and the problem of computing OP transforms in near-linear time has seen a lot of research activity recently. We just mention two recent works [6, 7] that present near-linear time algorithms for the Jacobi polynomial transforms (and indeed their notion of *uniform Jacobi transform* corresponds exactly to the Jacobi polynomial transform that we study in this paper). However, these algorithms inherently seem to require at least linear-time and it is not clear how to convert them into sub-linear algorithms, which is the focus of our work.

kernel features in machine learning. The Legendre Memory Unit [38] augments recurrent neural networks by orthogonalizing the history of features on a sliding Legendre basis; mathematically, this is essentially an online update of the discrete Legendre Transform. More directly, Thomas et al. [37] apply parametrized families of structured matrices directly inspired by orthogonal polynomial transforms (De Sa et al. [14]) as layers in neural networks. In this context, *any* form of structured matrix that admits fast operations is valuable, such as those considered in this work. Although not directly applied yet, all of these applications have a natural way of incorporating *sparsity* if the appropriate sparse transforms exist, which is a particular focus of modern ML in the face of sharply increasing trends in computation.

1.2 Related Work

As previously described, there has been a great deal of work on the sFFT; we refer the reader to the survey [18] for an overview. There has also been work on non-Fourier OP transforms. We break up our discussion below into discussion on the *sample complexity* (which as mentioned above is largely settled) and the *algorithmic complexity* (which remains largely open).

Sample complexity

The sample complexity of OP transforms $\mathbf{F}$ has been largely pinned down by the *compressed sensing* literature. For example, suppose that $\mathbf{F} \in \mathbb{R}^{N \times N}$ is any orthogonal and sufficiently flat matrix, in the sense that none of the entries of $\mathbf{F}$ are too large. Then a result of Haviv and Regev (sharpening of results by Bourgain, and Rudelson and Vershynin) shows that $m = O(k \log^2 k \log N)$ samples suffice to establish that the matrix $\Phi \in \mathbb{R}^{m \times N}$ (which is made up of m sampled rows from $\mathbf{F}^T$) has the *Restricted Isometry Property* (RIP) [5,20,34]. Finding $\hat{\mathbf{x}} = \mathbf{F}\mathbf{x}$ from samples of $\mathbf{F}$ of corresponds to the problem of finding an (approximately) k-sparse vector $\hat{\mathbf{x}}$ from the linear measurements $\Phi\hat{\mathbf{x}}$, which is precisely the compressed sensing problem. It is known that if Φ satisfies the RIP, then this can be solved (for example with ℓ_1 minimization) in time $N^{O(1)}$. On the other hand, recent results by Błasiok et al. show that this is essentially tight when $\mathbf{F}$ are certain Fourier matrices over constant sized prime finite fields, such as the Hadamard matrix, in that $O(k \log k \log N)$ queries (for a certain range of k) to $\mathbf{x}$ are needed to compute a k-sparse approximation of $\mathbf{F}\mathbf{x}$ [4].

Foucart and Rauhut [16] show that if the orthogonal polynomials satisfy a Bounded Orthogonal System (BOS) that are suitably flat, then if the m evaluation points λ_j are chosen uniformly at random proportional to the weight function w, then the $m \times N$ matrix Φ defined by normalizing $\mathbf{P}_N[i,j] = p_j(\lambda_i)$ appropriately satisfies the RIP with high probability provided that m has an appropriate dependence on N, k, ϵ, and the flatness of the matrix, and this again gives an $N^{O(1)}$-time algorithm to solve the sparse recovery problem.

Rauhut and Ward [32] show that for Jacobi polynomial transforms if the evaluation points were picked according to the *Chebyshev measure*, then with $O(k\,\mathrm{polylog} N)$ random measurements, the corresponding matrix has the RIP (note that the Foucart and Rauhut sample the evaluation points according to the measure of orthogonality for the Jacobi polynomials, which in general is *not* the Chebyshev measure). This result again does not give a sub-linear time algorithm but was used in the result of [21] which we describe below.

While these approaches can give near-optimal sample complexity, they do not give sublinear-time algorithms. In fact, it is faster to compute $\hat{\mathbf{x}}$ exactly by computing $\mathbf{F}\mathbf{x}$, if we care only about the running time and not about sample complexity [15]. Thus, we turn our attention to sublinear-time algorithms.

Sublinear-time algorithms for OP transforms

There have been several works generalizing and building on the sFFT results mentioned above. One direction is to the multi-dimensional DFT (for example in [23,27]). Another direction is to apply the sFFT framework to orthogonal polynomials with similar structure. One example is Chebyshev polynomials and the Discrete Cosine Transform (DCT). It was observed in [21] that this can be reduced to sFFT in a black box manner, solving the sparse recovery problem for Chebyshev polynomials and the DCT. A second example of OP transforms which can essentially be reduced to the sFFT is *Legendre polynomials.* Hu et al. [21] seek to recover an unknown k-term Legendre polynomial (with highest magnitude degree limited to be $N/2$), defined on $[-1,1]$, from samples. They give a sublinear two-phase algorithm: in the first phase, they reduce k-sparse-Legendre to sFFT to identify a set of candidate Legendre polynomials. The second phase uses the RIP result for BOS to produce a matrix that is used to estimate the coefficients of the candidate Legendre polynomials. We note that in this work the setting is naturally continuous, while ours is discrete.

Choi et al. [10,11] study higher dimensions and obtain sublinear-time algorithms for more general harmonic expansions in multiple dimensions. These results complement our work. More precisely, that work shows how to use any algorithm for a univariate polynomial transform (the work in [11] needs these algorithms to have certain specific properties) to design an algorithm for a multi-variate polynomial transform where the multi-variate polynomials are products of univariate polynomials in the individual variables. Thus our improvements for univariate polynomial transforms can (potentially) be used with [10,11].

Finally, there are sparse OP transforms based on Prony's method. The work [29] extends Prony's method to a very general setting, including Jacobi polynomials, and gives an algorithm that requires only $O(k)$ queries to recover exactly k-sparse polynomials. However, these general results work only for exact sparsity and are in general not robust to noise. There has been work extending and modifying these techniques to settings with noise (for example, [22,30]), but to the best of our knowledge the only provable results for noise are for either the sFFT or closely related polynomial families. We note that [31] presents a Prony-like algorithm for Legendre and Gegenbauer polynomials and demonstrates empirically that this algorithm is robust to noise, although they do not address the question theoretically.

1.3 Technical overview

Our technical results have two main parts. First, inspired by existing approaches to the sFFT, we provide a general reduction from the k-sparse recovery problem to the 1-sparse recovery algorithm, which works for any family of OPs that is sufficiently "flat": that is, no entry of the matrix $\mathbf{F}$ is too large. Second, we provide a 1-sparse recovery algorithm for Jacobi polynomials. We give an overview of both parts below.

For what follows, let $\mathbf{F}$ be an orthogonal matrix. For simplicity in this overview we will assume that there is no noise. That is, we want to compute the exactly k-sparse $\hat{\mathbf{x}} = \mathbf{F}\mathbf{x}$ given query access to $\mathbf{x}$. However, we note that our final results do work for approximately k-sparse vectors $\hat{\mathbf{v}} = \hat{\mathbf{x}} + \hat{\mathbf{w}}$ provided that $\|\hat{\mathbf{w}}\|_2$ is sufficiently small.

1.3.1 Reduction to one-sparse recovery

We give a general reduction from the k-sparse recovery problem to the one-sparse recovery problem, which works for a broad class of OP families defined on a finite interval.[4] At a high level, the idea is as follows. Suppose that $\hat{\mathbf{x}} = \mathbf{Fx}$ is k-sparse and $\mathbf{b} \in \mathbb{R}^N$ is a "filter": at this stage it is helpful to think of it like a boxcar filter, so $\mathbf{b}$ is 1 on some interval I and zero outside of that interval. If we choose this interval randomly, we might hope to isolate a single "spike" of $\hat{\mathbf{x}}$ with $\mathbf{b}$: that is, we might hope that $\mathbf{D_b}\hat{\mathbf{x}}$ is one-sparse, where $\mathbf{D_b}$ is the diagonal matrix with $\mathbf{b}$ on the diagonal. Suppose that this occurs, so $\mathbf{y} = \mathbf{D_b}\hat{\mathbf{x}}$ is one sparse. In order to take advantage of this with a black-box solution to the one-sparse recovery problem $\hat{\mathbf{y}} = \mathbf{Fy}$, we would need query access to the vector $\mathbf{y} = \mathbf{F}^{-1}\mathbf{D_b}\hat{\mathbf{x}} = \mathbf{F}^{-1}\mathbf{D_b}\mathbf{Fx}$, while what we have is query access to $\mathbf{x}$. Thus, we would like to design $\mathbf{b}$ so that $\mathbf{F}^{-1}\mathbf{D_b}\mathbf{F}$ is *row-sparse*. This would allow us to query a position of $\mathbf{y} = \mathbf{F}^{-1}\mathbf{D_b}\hat{\mathbf{x}}$ using only a few queries from $\mathbf{x}$.

One of our main technical contributions is showing how to design such a vector $\mathbf{b}$, so that $\mathbf{b}$ approximates a boxcar filter and so that $\mathbf{F}^{-1}\mathbf{D_b}\mathbf{F}$ is row-sparse for *any* OP transform $\mathbf{F}$.

Then, given this filter, we can iteratively identify and subtract off "spikes" in $\hat{\mathbf{x}}$ until we have recovered the whole thing. Of course, the actual details are much more complicated than the sketch above. First, the one-sparse solver might have a bit of error, which will get propagated through the algorithm. Second, in our analysis the vector $\hat{\mathbf{x}}$ need not be exactly k-sparse. Third, $\mathbf{b}$ will only approximate a boxcar filter, and this is an additional source of error that needs to be dealt with. Complete details are in the full version of the paper.

For the reader familiar with the sFFT, this approach might look familiar: most sFFT algorithms work by using some sort of filter to isolate single spikes in an approximately sparse signal. Below, we highlight some of the challenges in extending this idea beyond the Fourier transform. Some of these challenges we have overcome, and one we have not (yet) overcome. We mention this last open challenge both because it explains the assumption we have to make on the sparsity structure of $\hat{\mathbf{x}}$, and also because we hope it will inspire future work.

Challenge 1: Choice of filter

One key difficulty in extending sFFT algorithms to general orthogonal polynomials is that the filters used in the sFFT approach are very specific to the Fourier transform. Indeed, much of the progress that has been made on that problem has been due to identifying better and better choices of filter specialized to the Fourier transform. In order to find filters that work for *any* OP family, we take a different approach and construct a filter out of low-degree Chebyshev polynomials. Then we use the orthogonality properties of the OP family to guarantee that $\mathbf{F}^{-1}\mathbf{D_b}\mathbf{F}$ has the desired sparsity properties.

Challenge 2: Explicit black-box reduction

Because our goal is generality (to as broad a class of OPs as possible), we give an explicit reduction that uses a 1-sparse solution as a black box. To the best of our knowledge, existing work on the sFFT does not explicitly do this: a reduction of this flavor is certainly implicit in many of these works, and even explicitly given as intuition, but we are not aware of an sFFT algorithm which actually uses a 1-sparse recovery algorithm as a black box.

[4] We note that our results do not (yet) work for the case when the orthogonality is defined over an infinite interval. In particular, our reduction does not work for the Hermite and Laguerre polynomials.

Challenge 3: Equi-spaced evaluation points

The evaluations points in DFT and the DCT are equispaced (in the angular space). This fact is crucially exploited in sFFT algorithms (as well as the reduction of DCT to DFT – see the full version of this paper for more details on the reduction). Unfortunately, the roots of Jacobi polynomials are no longer equally spaced. However, it is known that the roots of Jacobi polynomials are "spread out" (in a sense made below precise in Definition 2.2), and we show that this property is enough for our reduction. In fact, our reduction from k-sparse recovery to 1-sparse recovery works generally for any "flat" OP family with "spread out" roots.

(Open) Challenge 4: Permuting the coordinates of $\hat{\mathbf{x}}$

In the approach described above, we hoped that an interval I would "isolate" a single spike. In the sFFT setting, this can be achieved through a permutation of the coordinates of $\hat{\mathbf{x}}$. In our language, in the sFFT setting it is possible to define a random (enough) permutation matrix $\mathbf{P}$ so that $\mathbf{P}\hat{\mathbf{x}}$ has permuted coordinates, and so that $\mathbf{F}^{-1}\mathbf{D_b}\mathbf{P}\mathbf{F}$ is row-sparse – this argument crucially exploits the fact that the roots of unity are equispaced in the angle space. This means that not only can we sample from the one-sparse vector $\mathbf{D_b}\hat{\mathbf{x}}$, but also we can sample from $\mathbf{D_b}\mathbf{P}\hat{\mathbf{x}}$, and then there is some decent probability that any given spike in $\hat{\mathbf{x}}$ is isolated by $\mathbf{b}$. However, we have not been able to come up with (an approximation to) such a $\mathbf{P}$ that works in the general OP setting. This explains why we require the assumption that the support of $\hat{\mathbf{x}}$ be reasonably "spread out," so that we can hope to isolate the spikes by $\mathbf{b}$. This assumption is made precise in Definition 2.3. We note that if such a $\mathbf{P}$ were found in future work, this would immediately lead to an improved k-sparse recovery result for Jacobi polynomials, which would work for arbitrary sparse signals $\hat{\mathbf{x}}$.

1.3.2 A one-sparse recovery algorithm for Jacobi polynomials

With the reduction complete, to obtain a k-sparse recovery algorithm for general Jacobi polynomials we need to solve the one-sparse case. We give an overview of the basic idea here, with full details in the full version of the paper. First, we note that via well-known approximations of Jacobi polynomials [35], one can approximate the evaluation of any Jacobi polynomial at a point in $(-1, 1)$ by evaluating the cosine function at an appropriate angle. Using some standard local error-correcting techniques (for example, computing $\cos(A)$ via $\cos A = \frac{\cos(A+B)+\cos(A-B)}{2\cos B}$ for a random B), we reduce the 1-sparse recovery problem to computing some unknown value θ, corresponding to the index of the spike, from noisy values of $\cos(w\theta)$ for some integers $w \geq 1$, corresponding to evaluations of the Jacobi polynomials for this index. Since the reduction is approximate, some care has to be taken to handle some corner cases where the approximation does not hold. In particular, we have to figure out for which real numbers $y \in [0, N)$ does its orbit $\langle xy \rangle$ for $x \in \mathbb{Z}_N$ have small order. We give a result to handle this, which to the best of our knowledge (and somewhat surprisingly) seems to be new.[5] With this out of the way, our algorithm to compute the value of θ from the evaluations $\cos(w\theta)$ is based on the following idea. Assuming we already know $\cos(\theta)$ up to $\pm\epsilon$, we get a noisy estimate of θ (which lives in the range $\arccos(\cos(\theta) \pm \epsilon)$) and then use the evaluations at $w > 1$ to "dilate" the range where we know θ lies, reducing ϵ. We

[5] We thank Stefan Steinerberger for showing us a much simpler proof than our original more complicated proof, which also gave worse parameters.

proceed iteratively until the region of uncertainty is small enough that there are only $O(1)$ possibilities remaining, which we then prune out using the fact that $\mathbf{F}$ is orthogonal and flat, in the sense that none of its entries are too large. (We note that proving $\mathbf{F}$ is flat needs a bit of care. In particular, we need a sharper bound on Jacobi polynomials (than the cosine approximation mentioned above) in terms of Bessel functions to prove that *all* entries of $\mathbf{F}$ are small.) Similar ideas have been used for 1-sparse recovery for the DFT (for example, in [19]), although our situation is more complicated than the DFT because working with cosines instead of complex exponentials means that we lose sign information about θ along the way (though it is similar in spirit to the one-sparse recovery algorithm for DFT in [19]).

2 Background and Preliminaries

2.1 Notation

We use bold lower-case letters $(\mathbf{x}, \mathbf{y})$ for vectors and bold upper-case letters $(\mathbf{P}, \mathbf{F})$ for matrices. Non-bold notation x, y, U is used for scalars in $\mathbb{R}$. In general, if there is a given transform $\mathbf{F}$ we are considering, then the notation $\hat{\mathbf{x}} \in \mathbb{R}^N$ indicates $\mathbf{F} \cdot \mathbf{x}$. We use the notation $\mathbf{x}[i]$ or $\mathbf{X}[i,j]$ to index into a vector or matrix, respectively. All of our vectors and matrices are 0-indexed, i.e. the entries of a vector $\mathbf{x} \in \mathbb{R}^N$ are $\mathbf{x}[0], \dots, \mathbf{x}[N-1]$. We use $[N]$ to denote the set $\{0, \dots, N-1\}$. Given a subset $S \subset [N]$, we will denote the complement set (i.e. $[N] \setminus S$) by S^c.

Given a vector $\mathbf{x} \in \mathbb{R}^N$ and an integer $1 \leq s \leq N$, we define $\textsc{large}(s, \mathbf{x})$ to be the magnitude of the sth largest value in $\mathbf{x}$ (by absolute value).

For any vector $\mathbf{u} \in \mathbb{R}^N$, we define $\mathbf{D}_\mathbf{u} \in \mathbb{R}^{N \times N}$ as the diagonal matrix with $\mathbf{u}$ on its diagonal. For a diagonal matrix $\mathbf{D}$, and any real α we denote $\mathbf{D}^\alpha$ to denote the diagonal matrix with the (i,i) entry being $(\mathbf{D}[i,i])^\alpha$. Given a vector $\mathbf{x} \in \mathbb{R}^N$ and set $S \subseteq [N]$, $\mathbf{x}_S$ denotes the vector $\mathbf{x}$ where all entries out of S are masked to 0. For $\mathbf{x} \in \mathbb{R}^N$, $\mathrm{supp}(\mathbf{x}) \subseteq [N]$ denotes the support (i.e. the set of non-zero positions) of $\mathbf{x}$.

We use $x \pm h$ to refer to either the interval $[x-h, x+h]$ or a point in this interval, whichever is clear from context. Similarly, if S is an interval $[a,b]$ then $S \pm h$ is the interval $[a-h, b+h]$.

When stating algorithms, we use superscript notation to denote query access. That is $\mathcal{A}^{(\mathbf{x})}(\mathbf{z})$ takes input $\mathbf{z}$ and has query access to $\mathbf{x}$.

We use the notation $f(n) \lesssim g(n)$ to mean that there is some constant C so that, for sufficiently large $n \geq n_0$, $f(n) \leq Cg(n)$.

The notation $\mathcal{J}_\square$ means $\mathcal{J}_j$ for all indices j.

2.2 Orthogonal Polynomials

For the remainder of this paper, we consider polynomials $p_0(X), p_1(X), \dots$ that form a normalized orthogonal polynomial family with respect to some compactly supported measure $w(X)$. By suitably scaling and translating X, we can ensure that the orthogonality is on $[-1,1]$.[6] In particular $\deg(p_i) = i$ and for any $i, j \geq 0$,

$$\int_{-1}^{1} p_i(X) p_j(X) w(X) dX = \delta_{i,j}, \tag{1}$$

where $\delta_{i,j} = 1$ if $i = j$ and 0 otherwise.

[6] See footnote 4.

Then for given N evaluation points $\lambda_0, \dots \lambda_{N-1}$, define the orthogonal polynomial transform $\mathbf{P}_N$ as follows. For any $0 \le i, j < N$, we have

$$\mathbf{P}_N[i, j] = p_j(\lambda_i).$$

In other words, the rows of $\mathbf{P}_N$ are indexed by the evaluation points and the columns are indexed by the polynomials.

For the rest of the paper, assume $\lambda_0 \le \lambda_1 \le \cdots \le \lambda_{N-1}$ are the roots of $p_N(X)$. Then it is well-known (see e.g. [35]) that

- The roots lie in the support of the measure (i.e. $\lambda_i \in [-1, 1]$) and are distinct (i.e. $\lambda_0 < \lambda_1 < \cdots < \lambda_{N-1}$).
- There exists Gaussian quadrature weights $w_\ell = \frac{1}{\sum_{j=0}^{N-1} p_j(\lambda_\ell)^2}, i = 0, \dots, N-1$ such that for any polynomial $f(X)$ of degree at most $2N-1$,

$$\int_{-1}^{1} f(X)w(X)dX = \sum_{\ell=0}^{N-1} f(\lambda_\ell) \cdot w_\ell. \tag{2}$$

We are now ready to define the orthogonal matrix corresponding to $\mathbf{P}_N$ that we deal with in this paper:

▶ **Definition 2.1.** *Let $p_0(X), \dots, p_{N-1}(X), \dots$ be an orthogonal polynomial family, $\lambda_0, \dots,$ λ_{N-1} be the roots of $p_N(X)$, and $w_0, \dots, w_{N-1}$ be the Gaussian quadrature weights. Define $\mathbf{D_w}$ to be the diagonal matrix with $w_0, \dots, w_{N-1}$ on its diagonal, and*

$$\mathbf{F}_N = \mathbf{D}_{\mathbf{w}}^{\frac{1}{2}} \mathbf{P}_N.$$

Note that by (1) and (2),

$$\mathbf{F}_N^T \mathbf{F}_N = \mathbf{P}_N^T \mathbf{D_w} \mathbf{P}_N = \mathbf{I}_N,$$

so $\mathbf{F}_N$ is an orthogonal matrix. In particular,

$$\mathbf{P}_N^T \mathbf{D_w} \mathbf{P}_N[i, j] = \sum_{k=0}^{N-1} p_i(\lambda_k) w_k p_j(\lambda_k) = \int_{-1}^{1} p_i(X) p_j(X) w(X) \, dX = \delta_{i,j}.$$

Note that since $\mathbf{F}_N$ is orthogonal, by definition we have

$$\mathbf{F}_N^{-1} = \mathbf{F}_N^T.$$

2.2.1 Jacobi Polynomials and Special Cases

In this section we define Jacobi Polynomials, our main object of interest, and point out a few special cases. We note that families of named orthogonal polynomials $\{p_i(X)\}$ are sometimes defined through different means, hence are normalized differently up to constants. The corresponding discrete orthogonal polynomial transform (e.g. Discrete Legendre Transform) frequently refers to multiplication by $\mathbf{P}$ instead of $\mathbf{F}$. In these cases, the transform satisfies $\mathbf{P}_N^T \mathbf{D_w} \mathbf{P}_N = \mathbf{D}$ for a diagonal matrix $\mathbf{D}$ corresponding to the normalization. The transform $\mathbf{F}_N = \mathbf{D}_{\mathbf{w}}^{\frac{1}{2}} \mathbf{P} \mathbf{D}^{-\frac{1}{2}}$ we consider (note that this matrix is indeed orthogonal) is thus equivalent up to diagonal multiplication.

Jacobi polynomials

Jacobi polynomials are indexed by two parameters $\alpha, \beta > -1$ and these are polynomials $\left\{P_j^{(\alpha,\beta)}\right\}_{j\geq 0}$ that are orthogonal with respect to the measure

$$w^{(\alpha,\beta)}(X) = (1-X)^{\alpha} \cdot (1+X)^{\beta}$$

in the range $[-1,1]$. This definition is not normalized, in the sense that we have $\mathbf{P}_N^T \mathbf{D}_{\mathbf{w}} \mathbf{P}_N = \mathbf{D}$, where

$$\mathbf{D}[j,j] = \frac{2^{\alpha+\beta+1}}{2j+\alpha+\beta+1} \cdot \frac{\Gamma(j+\alpha+1)\Gamma(j+\beta+1)}{\Gamma(j+1)\Gamma(j+\alpha+\beta+1)}$$

(see [35, Pg. 68, (4.3.3)]).

We record three well-known special cases: Chebyshev polynomials (of the first kind) are special case of $\alpha = \beta = -\frac{1}{2}$ and Legendre polynomials are the special case of $\alpha = \beta = 0$ (up to potentially a multiplicative factor that could depend on the degree j). Another notable special case of Jacobi polynomials are the *Gegenbauer* or *ultraspherical polynomials* ($\alpha = \beta$). Our results hold for all Jacobi polynomials with $\alpha, \beta \geq -\frac{1}{2}$, which include essentially all named special cases of Jacobi polynomials used in practice.

Chebyshev polynomials of the 1st kind

The Chebyshev polynomials of the 1st kind are orthogonal with respect to the weight measure $w(X) = (1-X^2)^{-\frac{1}{2}}$.

The normalized transform $\mathbf{F}_N$ has the closed form

$$\mathbf{F}_N[i,j] = \begin{cases} \sqrt{\frac{1}{N}} & j = 0 \\ \sqrt{\frac{2}{N}} \cdot \cos\left[\frac{\pi}{N} j \left(i + \frac{1}{2}\right)\right] & j = 1, \ldots, N-1. \end{cases}$$

This is a variant of the Discrete Cosine Transform (DCT-III, or the inverse DCT). It is well-known that the DCT-III can be “embedded” into a DFT of twice the dimension, and we work out some of the details of how to use the sparse FFT to compute a sparse DCT in the full version of the paper.

Legendre polynomials

Legendre polynomials are orthogonal with respect to the uniform measure i.e. $w(X) = 1$ and play a critical role in multipole expansions of potential functions (whether electrical or gravitational) in spherical coordinates. They are also important for solving Laplace’s equation in spherical coordinates.

2.2.2 Roots of Orthogonal Polynomials

Since $\lambda_i \in [-1,1]$ for all i, there is a unique $\theta_i \in [0,\pi]$ such that $\lambda_i = \cos\theta_i$. Our reduction holds for orthogonal polynomials that have roots that are “well-separated” in this angle space:

▶ **Definition 2.2.** *Let $0 < C_0 < C_1$. A family of orthogonal polynomials $p_0(X), p_1(X), \dots$ is* (C_0, C_1, γ_0)-dense *if for all large enough d, the following holds.*

Let $\lambda_0, \dots, \lambda_{d-1}$ be the roots of p_d, and $\theta_i = \arccos \lambda_i$. Then for any $i \in [d]$, for any $\gamma \geq \gamma_0/d$:

$$C_0 \gamma d \leq \left| \{\theta_0, \dots, \theta_{d-1}\} \cap \left[\theta_i - \frac{\gamma}{2}, \theta_i + \frac{\gamma}{2}\right] \right| \leq C_1 \gamma d.$$

It turns out that any family of Jacobi polynomials has the required property: their roots are spaced out such that θ_ℓ is close to $\ell\pi/N$.

2.3 Sparse Recovery Problem

We will consider approximately k-sparse vectors $\hat{\mathbf{v}} = \hat{\mathbf{x}} + \hat{\mathbf{w}}$, where $\hat{\mathbf{x}}$ is k-sparse and $\|\hat{\mathbf{w}}\|_2$ is sufficiently small. We will require that $\hat{\mathbf{x}}$ has a "spread out" support, defined as follows.

▶ **Definition 2.3.** *Let $k \in [N]$ and $0 \leq \sigma < 1$. We say that a vector $\mathbf{x} \in \mathbb{R}^N$ is* (k, σ)-sparsely separated *if there are k non-zero locations in $\mathbf{x}$ and any two non-zero locations are more than σN indices apart.*

It is not hard to see that a vector $\mathbf{x}$ with random support of size k is, with constant probability, $\left(k, \Omega\left(\frac{1}{k^2}\right)\right)$-sparsely separated.

In our reduction, we will reduce the k-sparse recovery problem to the special case of $k = 1$. Next, we define some notation for the 1-sparse case.

▶ **Definition 2.4.** *We say that the matrix $\mathbf{F}_N$ has an $(N, \epsilon, \delta, \mu)$ one-sparse recovery algorithm with query complexity $Q(N, \epsilon, \delta, \mu)$ and time complexity $T(N, \epsilon, \delta, \mu)$ if there exists an algorithm $\mathcal{A}$ with the properties below:*

For all $\mathbf{y}$ so that $\hat{\mathbf{y}} = \mathbf{F}_N \mathbf{y}$ can be decomposed as

$$\hat{\mathbf{y}} = \tilde{\mathbf{y}} + \mathbf{w},$$

where $\tilde{\mathbf{y}} = v \cdot \mathbf{e}_h$ is 1-sparse and

$$\|\mathbf{w}\|_2 \leq \epsilon \, |v| \, ,$$

we have:

1. *$\mathcal{A}$ makes at most $Q(N, \epsilon, \delta, \mu)$ queries into $\mathbf{y} = \mathbf{F}_N^{-1}(v \cdot \mathbf{e}_h + \mathbf{w})$.*
2. *With probability at least $1 - \mu$, $\mathcal{A}$ outputs $\tilde{v} \cdot \mathbf{e}_h$ with $|v - \tilde{v}| \leq \delta \, |v|$ in time $T(N, \epsilon, \delta, \mu)$.*

Pre-processing time

Our algorithm requires some pre-processing of $\mathbf{F}_N$. Our pre-processing step involves computing the roots $\lambda_1, \dots, \lambda_N$ of p_N and storing them in an appropriate data structure, and additionally forming and storing some matrices that we will use in our algorithm. Finding the roots and creating the data structure can be done in time poly(N), and the rest of the pre-processing step also takes time poly(N). We note that this is an up-front cost that needs to be only paid once.

Precision

We note that we need to make certain assumptions on size of the entries in $\hat{\mathbf{v}}$ since otherwise we would not even be able to read coefficients that are either too large or too small and need $\Omega(\log N)$ bits to represent. Towards this end we will make the standard assumption that $\|\hat{\mathbf{v}}\|_2 = 1$. In particular, this allows us to ignore any coefficients that are smaller than say $\frac{1}{N}$

since their contribution to $\|\hat{\mathbf{v}}\|_2$ is at most $\frac{1}{\sqrt{N}}$, which will be too small for our purposes.[7] In particular, this implies that we only have to deal with numbers that need $O(\log N)$ bits and as is standard in the RAM model, basic arithmetic operations on such numbers can be done in $O(1)$ time. We will implicitly assume this for the rest of the paper (except in the proof of one lemma, where we will explicitly make use of this assumption).

3 Results

In this section we state our main results. These results follow from more detailed versions which are stated in the full version of the paper.

We start off with our main result for Jacobi polynomials. We state an informal version here, and refer the reader to the full version of the paper for the formal result.

▶ **Theorem 3.1** (General Sparse Recovery for Jacobi Polynomial Transform, Informal). *Fix arbitrary parameters $\alpha, \beta \geq -\frac{1}{2}$ for Jacobi polynomials and let $\mathbf{J}_N^{(\alpha,\beta)}$ be the $N \times N$ orthogonal matrix that arises from it as in Definition 2.1. Then there is an algorithm* Recover *that does the following. Let $\mathbf{v} = \mathbf{x} + \mathbf{w}$ where $\hat{\mathbf{x}} = \mathbf{J}_N^{(\alpha,\beta)}\mathbf{x}$ is $(k, C_1/k^2)$-sparsely separated, and suppose that $\|\hat{\mathbf{w}}\|_2 \lesssim \delta \min_{h \in \mathrm{supp}(\hat{\mathbf{x}})} |\hat{\mathbf{x}}[h]|$. Then with probability at least 0.99,* Recover *outputs $\hat{\mathbf{z}}$ such that*

$$\|\hat{\mathbf{x}} - \hat{\mathbf{z}}\|_2 \lesssim \delta \|\hat{\mathbf{x}}\|_2,$$

with $\mathrm{poly}\left(\frac{k \log N}{\delta}\right)$ *queries and running time* $\mathrm{poly}\left(\frac{k \log N}{\delta}\right)$.

▶ Remark 3.2. The requirement on the noise term might be bad if one entry of $\hat{\mathbf{x}}$ is extremely small compared to the rest. However in this case we can decrease k and add the very small entries of $\hat{\mathbf{x}}$ to the noise term $\hat{\mathbf{w}}$ resulting in a potentially better guarantee. We note that our algorithm iteratively finds the large components of $\hat{\mathbf{x}}$ and in fact has a mechanism for stopping early when all of the "large-enough" entries have been found.

▶ Remark 3.3. The $(k, O(1/k^2))$-sparsely separated requirement is chosen to reflect the separation of a random k-sparse vector (c.f. comment below Definition 2.3). Smaller amounts of sparse separation are acceptable, which translate accordingly into the query and time complexity. The full dependence is in the complete result in the full version of the paper.

To prove the above result, we first reduce the k-sparse recovery problem to 1-sparse recovery problem, in the presence of a small amount of noise. Next, we present an informal statement of our reduction.

▶ **Theorem 3.4** (Main Reduction, Informal). *Let $p_1, \ldots, p_N$ be a (C_0, C_1, γ_0)-dense orthogonal polynomial family, and let $\mathbf{F}_N$ be the $N \times N$ orthogonal matrix that arises from it as in Definition 2.1. Suppose that $|\mathbf{F}_N^{-1}[i,j]| \lesssim 1/\sqrt{N}$ for all $i, j \in [N]$. Suppose that for some sufficiently small $\delta > 0$, $\mathbf{F}_N$ has a $\left(N, O(\delta), \delta, O(C_0/k^2)\right)$ one-sparse recovery algorithm with query complexity Q and running time T.*

Then there is an algorithm Recover *that does the following. Let $\mathbf{v} = \mathbf{x} + \mathbf{w}$ where $\hat{\mathbf{x}} = \mathbf{F}_N\mathbf{x}$ is $(k, C_1/k^2)$-sparsely separated, and suppose that $\|\hat{\mathbf{w}}\|_2 \lesssim \delta \min_{h \in \mathrm{supp}(\hat{\mathbf{x}})} |\hat{\mathbf{x}}[h]|$. Then with probability at least 0.99,* Recover *outputs $\hat{\mathbf{z}}$ so that*

$$\|\hat{\mathbf{x}} - \hat{\mathbf{z}}\|_2 \lesssim \delta \|\hat{\mathbf{x}}\|_2,$$

with $\mathrm{poly}(k/\delta C_0)Q$ *queries and running time* $\mathrm{poly}(k/\delta C_0)T$.

[7] More generally, we can ignore smaller coefficients as long as they are polynomial sized in N.

The final algorithmic piece missing from the result above is the algorithm for 1-sparse recovery. We provide this missing piece for Jacobi polynomials:

▶ **Theorem 3.5** (1-Sparse Recovery for Jacobi Transform, Informal)**.** *There exists a universal constant C such that the following holds. Consider the Jacobi transform for any fixed parameters $\alpha, \beta \geq -\frac{1}{2}$. There exists an $(N, \epsilon, C \cdot \epsilon, \gamma)$ 1-sparse recovery algorithm for the Jacobi transform that makes* $\text{poly}\left(\log\left(\frac{N}{\gamma}\right) \cdot \frac{1}{\epsilon}\right)$ *queries and takes time* $\text{poly}\left(\log\left(\frac{N}{\gamma}\right) \cdot \frac{1}{\epsilon}\right)$.

4 Open Questions

To conclude, we list a few questions left open by our work.

1. First, it is natural to try and improve our k-sparse recovery algorithm to work for arbitrary k-sparse support, rather than "well-separated" supports. One natural way to do this is to address the fourth (open) challenge in Section 1.3 for a general class of OPs.
2. Second, we could hope to handle a more general class of noise $\hat{\mathbf{w}}$ than we currently do. One could hope to handle *any* vector $\mathbf{v}$, with an error guarantee that degrades smoothly with the ℓ_2 norm of the "tail" of $\mathbf{v}$. There is a long list of work on "de-noising" the contribution of the "head" to the "tail" in the sFFT literature that could potentially be useful here [23–27].
3. Third, we would like to extend our results to hold for OPs defined over infinite intervals (e.g. Hermite and Laguerre polynomials).
4. Fourth, we would like to solve the sparse recovery for $\mathbf{F}^T$ (where $\mathbf{F}$ is as in Definition 2.1): i.e. given query access to $\mathbf{x}$ figure out a good k-sparse approximation to $\mathbf{F}^T\mathbf{x}$ (recall that $\mathbf{F}^{-1} = \mathbf{F}^T$). (Note that this problem can be equivalently stated as follows: given query access to $\mathbf{F}\mathbf{y}$, compute a good k-sparse approximation to $\mathbf{y}$.) Currently our results do not solve this problem since we cannot show that the existence of a filter $\mathbf{b}$ such that $\mathbf{F}\mathbf{D}_\mathbf{b}\mathbf{F}^T$ is row-sparse. Note that this is not an issue for DFT since it is symmetric.
5. Finally, we would like to reduce the exponent on k in our final runtime. In particular, for the case of random k-sparse support, the dependence on k in the runtime for Jacobi transform is k^8. We note that we have not tried too hard to optimize the constants though we believe even getting a quadratic dependence on k with our framework would be challenging. We would like to stress that the majority of the work in the sFFT literature has been to make the dependence on k be linear and for such results, it seems very unlikely that a generic reduction from k-sparse recovery to 1-sparse recovery would work. In other words, using the knowledge about the 1-sparse recovery algorithm for DFT seems necessary to get a overall k-sparse FFT with running time $k\text{poly}(\log n)$.

References

1 `https://en.wikipedia.org/wiki/Approximation_theory#Chebyshev_approximation`.

2 `https://en.wikipedia.org/wiki/Jacobi_polynomials#Differential_equation`.

3 `http://www.chebfun.org`.

4 Jaroslaw Blasiok, Patrick Lopatto, Kyle Luh, Jake Marcinek, and Shravas Rao. An improved lower bound for sparse reconstruction from subsampled hadamard matrices. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 1564–1567. IEEE Computer Society, 2019. `doi:10.1109/FOCS.2019.00091`.

5 Jean Bourgain. *An Improved Estimate in the Restricted Isometry Problem*, pages 65–70. Springer International Publishing, Cham, 2014. `doi:10.1007/978-3-319-09477-9_5`.

6 James Bremer, Qiyuan Pang, and Haizhao Yang. Fast Algorithms for the Multi-dimensional Jacobi Polynomial Transform. *arXiv e-prints*, January 2019. `arXiv:1901.07275`.

7 James Bremer and Haizhao Yang. Fast algorithms for Jacobi expansions via nonoscillatory phase functions. *arXiv e-prints*, March 2018. `arXiv:1803.03889`.

8 Volkan Cevher, Michael Kapralov, Jonathan Scarlett, and Amir Zandieh. An adaptive sublinear-time block sparse fourier transform. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 702–715. ACM, 2017. `doi:10.1145/3055399.3055462`.

9 Xue Chen, Daniel M. Kane, Eric Price, and Zhao Song. Fourier-sparse interpolation without a frequency gap. In Irit Dinur, editor, *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 741–750. IEEE Computer Society, 2016. `doi:10.1109/FOCS.2016.84`.

10 Bosu Choi, Mark Iwen, and Felix Krahmer. Sparse Harmonic Transforms: A New Class of Sublinear-time Algorithms for Learning Functions of Many Variables. *arXiv 1808.04932*, 2018. `arXiv:1808.04932`.

11 Bosu Choi, Mark Iwen, and Toni Volkmer. Sparse harmonic transforms ii: Best s-term approximation guarantees for bounded orthonormal product bases in sublinear-time. *arXiv preprint*, 2019. `arXiv:1909.09564`.

12 John D. Cook. Orthogonal polynomials and gaussian quadrature. URL: `https://www.johndcook.com/OrthogonalPolynomials.pdf`.

13 Tri Dao, Christopher M De Sa, and Christopher Ré. Gaussian quadrature for kernel features. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 6107–6117. Curran Associates, Inc., 2017. URL: `http://papers.nips.cc/paper/7191-gaussian-quadrature-for-kernel-features.pdf`.

14 Christopher De Sa, Albert Cu, Rohan Puttagunta, Christopher Ré, and Atri Rudra. A two-pronged progress in structured dense matrix vector multiplication. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1060–1079. SIAM, 2018.

15 J. R. Driscoll, D. M. Healy, Jr., and D. N. Rockmore. Fast discrete polynomial transforms with applications to data analysis for distance transitive graphs. *SIAM J. Comput.*, 26(4):1066–1099, August 1997. `doi:10.1137/S0097539792240121`.

16 Simon Foucart and Holger Rauhut. *A Mathematical Introduction to Compressive Sensing*. Springer Science & Business Media, August 2013.

17 A. C. Gilbert, S. Guha, P. Indyk, S. Muthukrishnan, and M. Strauss. Near-optimal sparse fourier representations via sampling. In *Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing*, STOC '02, pages 152–161, New York, NY, USA, 2002. ACM. `doi:10.1145/509907.509933`.

18 Anna C Gilbert, Piotr Indyk, Mark Iwen, and Ludwig Schmidt. Recent Developments in the Sparse Fourier Transform. *IEEE Signal Processing Magazine*, 2014. `doi:10.1109/MSP.2014.2329131`.

19 Haitham Hassanieh, Piotr Indyk, Dina Katabi, and Eric Price. Nearly optimal sparse fourier transform. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 563–578. ACM, 2012.

20 Ishay Haviv and Oded Regev. The restricted isometry property of subsampled fourier matrices. In *Geometric aspects of functional analysis*, pages 163–179. Springer, 2017.

21 Xianfeng Hu, Mark Iwen, and Hyejin Kim. Rapidly computing sparse legendre expansions via sparse fourier transforms. *Numerical Algorithms*, 74(4), 2017.

22 Y. Hua and T. K. Sarkar. Matrix pencil method for estimating parameters of exponentially damped/undamped sinusoids in noise. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 38(5):814–824, May 1990. `doi:10.1109/29.56027`.

23 Piotr Indyk and Michael Kapralov. Sample-optimal fourier sampling in any constant dimension. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 514–523. IEEE, 2014.

24 Piotr Indyk, Michael Kapralov, and Eric Price. (nearly) sample-optimal sparse fourier transform. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 480–499. Society for Industrial and Applied Mathematics, 2014.

25 Michael Kapralov. Sparse fourier transform in any constant dimension with nearly-optimal sample complexity in sublinear time. In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 264–277. ACM, 2016. doi:10.1145/2897518.2897650.

26 Michael Kapralov. Sample efficient estimation and recovery in sparse FFT via isolation on average. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 651–662. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.66.

27 Michael Kapralov, Ameya Velingker, and Amir Zandieh. Dimension-independent sparse fourier transform. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 2709–2728, 2019. doi:10.1137/1.9781611975482.168.

28 Cameron Musco. Chebyshev polynomials in tcs and algorithm design. URL: http://www.cameronmusco.com/personal_site/pdfs/retreatTalk.pdf.

29 Thomas Peter and Gerlind Plonka. A generalized prony method for reconstruction of sparse sums of eigenfunctions of linear operators. *Inverse Problems*, 29(2):025001, January 2013. doi:10.1088/0266-5611/29/2/025001.

30 Daniel Potts and Manfred Tasche. Parameter estimation for exponential sums by approximate prony method. *Signal Processing*, 90(5):1631–1642, 2010. Special Section on Statistical Signal and Array Processing. doi:10.1016/j.sigpro.2009.11.012.

31 Daniel Potts and Manfred Tasche. Reconstruction of sparse legendre and gegenbauer expansions. *BIT Numerical Mathematics*, 56(3):1019–1043, 2016.

32 Holger Rauhut and Rachel Ward. Sparse legendre expansions via ℓ_1-minimization. *J. Approx. Theory*, 164(5):517–533, May 2012. doi:10.1016/j.jat.2012.01.008.

33 T.J. Rivlin. *An Introduction to the Approximation of Functions.* Blaisdell book in numerical analysis and computer science. Dover Publications, 1981. URL: https://books.google.com/books?id=wtS2xm8ggA4C.

34 Mark Rudelson and Roman Vershynin. On sparse reconstruction from fourier and gaussian measurements. *Communications on Pure and Applied Mathematics*, 61(8):1025–1045, 2008. doi:10.1002/cpa.20227.

35 G. Szegö. *Orthogonal Polynomials.* Number v. 23 in American Mathematical Society colloquium publications. American Mathematical Society, 1975. URL: https://books.google.com/books?id=3hcW8HBh7gsC.

36 William J. Tango. The circle polynomials of zernike and their application in optics. *Applied physics*, 13(4):327–332, August 1977. doi:10.1007/BF00882606.

37 Anna Thomas, Albert Gu, Tri Dao, Atri Rudra, and Christopher Ré. Learning compressed transforms with low displacement rank. In *Advances in neural information processing systems*, pages 9052–9060, 2018.

38 Aaron Voelker, Ivana Kajić, and Chris Eliasmith. Legendre memory units: Continuous-time representation in recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 15544–15553, 2019.

39 Kun-Hsing Yu, Ce Zhang, Gerald J Berry, Russ B Altman, Christopher Ré, Daniel L Rubin, and Michael Snyder. Predicting non-small cell lung cancer prognosis by fully automated microscopic pathology image features. *Nature Communications*, 7, 2016.

Hitting Long Directed Cycles Is Fixed-Parameter Tractable

Alexander Göke
Technische Universität Hamburg, Germany
alexander.goeke@tuhh.de

Dániel Marx
Max Planck Institut für Informatik, Saarland Informatics Campus, Saarbrücken, Germany
dmarx@mpi-inf.mpg.de

Matthias Mnich
Technische Universität Hamburg, Germany
matthias.mnich@tuhh.de

Abstract

In the Directed Long Cycle Hitting Set problem we are given a directed graph G, and the task is to find a set S of at most k vertices/arcs such that $G - S$ has no cycle of length longer than ℓ. We show that the problem can be solved in time $2^{\mathcal{O}(\ell^6+\ell k^3 \log k+k^5 \log k \log \ell)} \cdot n^{\mathcal{O}(1)}$, that is, it is fixed-parameter tractable (FPT) parameterized by k and ℓ. This algorithm can be seen as a far-reaching generalization of the fixed-parameter tractability of Mixed Graph Feedback Vertex Set [Bonsma and Lokshtanov WADS 2011], which is already a common generalization of the fixed-parameter tractability of (undirected) Feedback Vertex Set and the Directed Feedback Vertex Set problems, two classic results in parameterized algorithms. The algorithm requires significant insights into the structure of graphs without directed cycles of length longer than ℓ and can be seen as an exact version of the approximation algorithm following from the Erdős-Pósa property for long cycles in directed graphs proved by Kreutzer and Kawarabayashi [STOC 2015].

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Fixed parameter tractability; Mathematics of computing → Graph algorithms

Keywords and phrases Directed graphs, directed feedback vertex set, circumference

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.59

Category Track A: Algorithms, Complexity and Games

Related Version A full version of the paper is available at `https://arxiv.org/abs/2003.05267`.

Funding *Alexander Göke*: Supported by DFG grant MN 59/1-1.
Dániel Marx: Supported by ERC Consolidator Grant SYSTEMATICGRAPH (No. 725978).
Matthias Mnich: Supported by DFG grant MN 59/4-1.

1 Introduction

Feedback Vertex Set (FVS) and its directed variant Directed FVS (DFVS) are among the most classical problems in algorithmic graph theory: given a (directed) graph G the task is to find a minimum-size set $S \subseteq V(G)$ of vertices such that $G - S$ contains no (directed) cycles. Interestingly, the directed version is *not* a generalization of the undirected one. There is no obvious reduction from FVS to DFVS (replacing each undirected edge with two arcs of opposite directions does not work, as this would create directed cycles of length 2).

Both problems received significant amount of attention from the perspective of parameterized complexity. The main parameter of interest there is the optimal solution size $k = |S|$. Both problems can easily be solved in time $n^{\mathcal{O}(k)}$ by enumerating all size-k vertex subsets $S \subseteq V(G)$ and then checking whether $G - S$ is acyclic. The interesting question is thus whether the problems are *fixed-parameter tractable* with respect to k, i.e. whether there is an algorithm with run time $f(k) \cdot n^{\mathcal{O}(1)}$ for some computable function f depending only on k. FVS is one of the most studied problems in parameterized complexity: starting in the early 1990's, a long series of improved fixed-parameter algorithms [5, 6, 10, 13, 18, 25] lead to the currently fastest (randomized) algorithm from 2020 with run time $2.7^k \cdot n^{\mathcal{O}(1)}$ [20]. The DFVS problem has also received a significant amount of attention from the perspective of parameterized complexity. It was a long-standing open problem whether DFVS admits such an algorithm; the question was finally resolved by Chen et al. who gave a $4^k k! k^4 \cdot \mathcal{O}(nm)$-time algorithm for graphs with n vertices and m edges. Recently, an algorithm for DFVS with run time $4^k k! k^5 \cdot \mathcal{O}(n+m)$ was given by Lokshtanov et al. [22]. A fruitful research direction is trying to extend the algorithm to more general problems than DFVS. On the one hand, Chitnis et al. [8] generalized the result by giving a fixed-parameter algorithm for Directed Subset FVS: here we are given a subset U of arcs and only require the k-vertex set S to hit every cycle that contains an arc of U. On the other hand, Lokshtanov et al. [21] showed that the Directed Odd Cycle Transversal problem, where only the directed cycles of odd length needed to be hit, is W[1]-hard parameterized by solution size.

It is worth noting that very different algorithmic tools form the basis of the fixed-parameter tractability of FVS and DFVS: the undirected version behaves more like a hitting set-type problem, whereas the directed version has a more cut-like flavor. These differences motivated Bonsma and Lokshtanov [4] to consider Mixed FVS, the common generalization of FVS and DFVS where the input graph contains both directed and undirected edges. In such *mixed graphs*, cycles can contain directed arcs and undirected edges, but in particular the walk visiting an undirected edge twice is not a cycle. They obtained an algorithm for Mixed FVS with run time $2^{\mathcal{O}(k \log k)} \cdot n^{\mathcal{O}(1)}$ for k the size of the smallest feedback vertex set.

In this paper we study the following generalization of DFVS: We want to find a minimum size vertex set S such that all cycles of $G - S$ to have length at most ℓ. For $\ell = 1$ this is DFVS in loopless graphs. For $\ell = 2$ this is Mixed FVS in mixed graphs. The length of a longest cycle in a (directed) graph is also known as (directed) circumference of a graph. The parameterized version of our problem thus reads:

Directed Long Cycle Hitting Set	*Parameter:* $k + \ell$.
Input: A directed multigraph G and integers $k, \ell \in \mathbb{N}$.	
Task: Find a set S of at most k vertices such that $G - S$ has circumference at most ℓ.	

Note that Directed Long Cycle Hitting Set for $\ell = 2$ generalizes Mixed FVS (and hence both FVS and DFVS): to see this, subdivide anti-parallel arcs to make all cycles have length at least three and then replace undirected edges by anti-parallel arcs.

In contrast to FVS and DFVS, even checking feasibility of a given solution is a non-trivial task. It amounts to checking, for a digraph G and integer ℓ, whether G contains a cycle of length more than ℓ. This is also known as the Long Directed Cycle problem, which is obviously NP-hard since it contains the Directed Hamiltonian Cycle problem for $\ell = |V(G)| - 1$. However, Long Directed Cycle is fixed-parameter tractable parameterized by ℓ [29], hence *verifying* the solution of Directed Long Cycle Hitting Set is fixed-parameter tractable in ℓ.

Our contributions. Our main result is a fixed-parameter algorithm for Directed Long Cycle Hitting Set.

▶ **Theorem 1.** *There is an algorithm that solves* Directed Long Cycle Hitting Set *in time* $2^{\mathcal{O}(\ell^6 + \ell k^3 \log k + k^5 \log k \log \ell)} \cdot n^{\mathcal{O}(1)}$ *for n-vertex digraphs G and parameters $k, \ell \in \mathbb{N}$.*

The result also extends to the arc deletion variant of the problem, as we show both of them to be equivalent in a parameter preserving way.

The run time in Theorem 1 depends on two parameters, k and ℓ. This is necessary for the following reason. For $\ell = 1$, Directed Long Cycle Hitting Set corresponds to the DFVS problem, which is NP-hard. Moreover, the problem is also NP-hard for $k = 0$, as it contains the Directed Hamiltonian Cycle problem. This also shows that the run time cannot be polynomial in k or ℓ (unless P = NP). Assuming ETH, it is even necessary that the run time depends *exponentially* on both k and ℓ. Our algorithm achieves a run time that is single-exponential in both parameters k and ℓ. It is, in this sense, optimal.

Our algorithm is based on an elaborate combination algorithmic techniques, some of them used previously, some of them new.

- We use the standard opening step of iterative compression, which allows to assume that each directed cycle of length more than ℓ goes through a small number of exceptional vertices.
- We do not want to deal with the situation when there are two exceptional vertices x and y that are in the same strong component of the solution $G - S$. If we guess that this happens in the solution, then a way to avoid this problem is to guess a directed cycle C containing both x and y, and to contract this cycle. In order to guess this cycle, we essentially need a **representative set** of $x \to y$ paths, that is, a collection of paths such that if an (unknown) set S of at most k vertices does not disconnect y from x, then there is at least one $x \to y$ path disjoint from S in our collection. As an interesting self-contained result, we construct such a collection of size $\ell^{\mathcal{O}(k^2 \log k)} \cdot \log n$ on directed graphs without cycles of length greater than ℓ.
- If we can assume that the exceptional vertices are in different strong components of the solution, then this defines a separation problem on the exceptional vertices and makes the **directed shadow removal** technique of Chitnis et al. [8] relevant to simplify the structure of the instance. In particular, a major structural goal that we want to achieve is to ensure that every arc of the input graph lies in a directed cycle of length at most ℓ.
- Removing the exceptional vertices breaks the graph into some number of strong components with no cycle of length longer than ℓ in any of them. We call **portal vertices** the endpoints of the arcs connecting these strong components with each other and with the exceptional vertices. We show that the portal vertices can be partitioned into **clusters**: portals in each cluster are close to each other, while the distance between any two clusters is large. Furthermore, every solution has to separate the clusters from each other, defining another directed multiway cut problem.
- In the final step of the algorithm, we would like to use the technique of **important separators** to solve the directed multiway cut problem defined above: these are separators that are maximally "pushed" towards the target of the separations. However, the exact notion of importance is difficult to define due to the additional constraints of the problem being solved. Ergo, we perform a detailed analysis of the instance structure to identify **outlet vertices** that allows us to represent the additional constraints as separation, and to formally reduce the problem to branching on the choice of an important separator.

Related work. The structure of long cycles in directed graphs has been of interest for long time. For instance, Lewin [19] analyzed the density of such graphs, and Kintali [16] analyzes the directed treewidth of such directed graphs. Algorithmically, though, it was only recently shown by Kawarabayashi and Kreutzer [15] that the vertex version of the Erdős-Posa property holds for long directed cycles: namely, they show that any digraph G

either contains a set of $k+1$ vertex-disjoint directed cycles of length at least ℓ or some set S of at most $f(k,\ell)$ vertices that intersects all directed cycles of G with length at least ℓ. The corresponding questions for directed cycles without length restrictions have also been well-investigated [2, 26].

Note that an algorithmic proof of the Erdős-Pósa property can be a useful opening step for a fixed-parameter algorithm: we either find a set of $k+1$ arc- or vertex-disjoint cycles of length at least ℓ (and thus can reject the instance (G,k,ℓ) as "no"-instance) or obtain a set S which can serve as a feasible approximate solution. Such an opening step was also discussed in the well-known fixed-parameter algorithm for DFVS by Chen et al. [7, Remark 5.3], where the function $f(k,1)$ is known to be near-linear. In our case though, the function $f(k,\ell)$ from the Kawarabayashi-Kreutzer result is too large for us to obtain an algorithm for Directed Long Cycle Hitting Set with run time $2^{\mathrm{poly}(k,\ell)} \cdot n^{\mathcal{O}(1)}$.

Further, directed circumference can be seen as an intermediate step towards a general algorithmic framework for graph optimization problems related to *directed treewidth*. In undirected graphs, treewidth as a graph measure has enjoyed unprecedented success as a tool towards efficient approximation algorithms and fixed-parameter algorithms. For instance, as part of their Graph Minors series, Robertson and Seymour [28] showed that the k-linkage problem is fixed-parameter tractable, heavily relying on the reduction of the problem to graphs of bounded treewidth. In directed graphs, the situation is again much more complicated: Johnson et al. [14] introduced the notion of directed treewidth for digraphs. Yet, for digraphs the k-linkage problem is NP-hard already for $k=2$, and no fixed-parameter algorithm is known which recognizes digraphs of nearly-bounded directed treewidth. On the positive side, though, digraphs of bounded directed circumference are nicely squeezed between acyclic digraphs and digraphs of bounded directed treewidth [16]. Also, the arc version of the k-linkage problem is fixed-parameter tractable on digraphs of directed circumference 2 [3]; the question remains open for digraphs of larger directed circumference.

Returning to the original motivation of studying generalizations of DFVS, Neogi et al. [24] gave a fixed-parameter algorithm for the problem of finding a set S of size at most k in a given digraph G such that every strong component of $G-S$ excludes graphs in a fixed finite family $\mathcal{H}$ as (not necessarily induced) subgraphs, when $\mathcal{H}$ contains only rooted graphs, or contains at least one directed path. Göke et al. [12] considered the problem of finding a set S of size at most k in a given digraph G such that every strong component of $G-S$ has size at most s; they gave a fixed-parameter algorithm for parameter $k+s$.

2 Definitions and Notations

In this paper, we mainly consider finite loop-less directed graphs (or digraphs) G with vertex set $V(G)$ and arc set $A(G)$. We allow multiple arcs and arcs in both directions between the same pairs of vertices. A *walk* is a sequence of vertices $(v_1,\ldots,v_\ell)$ with corresponding arcs (v_i,v_{i+1}) for $i=1,\ldots,\ell-1$ which forms a subgraph of G; the *length* of a walk is its number of arcs. A walk is *closed* if $v_1=v_\ell$; otherwise, it is *open*. A *path* in G is an open walk where all vertices are visited at most once. A *cycle* in G is a closed walk in which every vertex is visited at most once, except for $x_1=x_\ell$ which is visited twice. (Throughout this entire paper, by "cycle" we always mean directed cycle.) We call G *acyclic* if G does not contain any cycle. For two vertices x_i,x_j of a walk W with $i\leq j$ we denote by $W[x_i,x_j]$ the subwalk of W starting at x_i and ending in x_j. For a walk W ending in a vertex x and a second walk R starting in x, $W\circ R$ is the walk resulting when concatenating W and R.

For each vertex $v \in V(G)$, its *out-degree* in G is the number $d^+_G(v)$ of arcs of the form (v, w) for some $w \in V(G) \setminus \{v\}$, and its *in-degree* in G is the number $d^-_G(v)$ of arcs of the form (w, v) for some $w \in V(G) \setminus \{v\}$. For each subset $V' \subseteq V(G)$, the subgraph induced by V' is the graph $G[V']$ with vertex set V' and arc set $\{(u, v) \in A(G) \mid u, v \in V'\}$. For a set X of vertices or arcs, let $G - X$ denote the subgraph of G obtained by deleting the elements in X from G. For a subgraph G' and an integer d we denote by $R^+_{G'}(X)$ the set of vertices that are reachable from X in G'.

A digraph G is *strong* if either G consists of a single vertex (then G is called *trivial*), or for any distinct $u, v \in V(G)$ there is a (directed) path from u to v. A *strong component* of G is an inclusion-wise maximal induced subgraph of G that is strong. The *(directed) circumference* of a digraph G is the length $\mathsf{cf}(G)$ of a longest cycle of G; if G is acyclic, then define $\mathsf{cf}(G) = 0$.

3 Directed Long Cycle Hitting Set Algorithm

The goal of this section is to devise an algorithm for Directed Long Cycle Hitting Set and thereby proof Theorem 1. We will only consider the vertex deletion variant. This will suffice, as the arc deletion version can be reduced to the vertex deletion version in a parameter-preserving way, as we show in the full version.

The algorithm performs a sequence of reductions to special cases of the original Bounded Cycle Length Vertex Deletion. All these sections are modular and just need the problem formulation and theorem at the end of the previous section.

Compression. Recall that in the Directed Long Cycle Hitting Set problem we are given a digraph G and integers k and ℓ. The task then is to find a set of at most k vertices such that $G - S$ contains no cycles of length more than ℓ.

As already stated in the introduction, checking a solution for correctness is an non-trivial task. For this we use a fixed-parameter algorithm by Zehavi [29].

▶ **Theorem 2.** *There is an algorithm that decides in time $2^{\mathcal{O}(\ell)} \cdot n^{\mathcal{O}(1)}$ whether an n-vertex digraph G contains a cycle of length more than ℓ.*

This already solves the case for $k = 0$. We now want to design an algorithm for general k.

The goal of this subsection is to get an existing solution T for which we have to find a disjoint solution S of size less than $|T|$. For this we use the standard techniques of iterative compression and disjoint solution.

We start our algorithm by applying the iterative compression technique introduced by Reed, Smith and Vetta [27]. This technique was also used by Chen et al. [7] to show the fixed parameter tractability of DFVS. We choose an arbitrary enumeration $v_1, \ldots, v_n$ of the vertices of G. By G_i we denote the digraph $G[v_1, \ldots, v_i]$. We want to iteratively construct solutions S_{i+1} to (G_{i+1}, k, ℓ) by using the solution S_i of (G_i, k, ℓ). We start with the empty digraph G_0 and the empty solution $S_0 = \emptyset$. This solution is feasible for every choice of G, k and ℓ as the empty digraph contains no cycles.

Now, if S_i hits all cycles of length more than ℓ in G_i, then $T_{i+1} = S_{i+1} \cup \{v_{i+1}\}$ does the same for G_{i+1} (as $G_i - S_i = G_{i+1} - T_{i-1}$). The only problem now is that T_{i+1} may be to large. Therefore, we consider the compression version of our problem: given an instance (G_i, k, ℓ) with a solution T_i of size $k + 1$, find a solution S_i of size at most k. If we can solve this problem, by above procedure we get a solution S_n for the digraph $G_n = G$ and hence have solved the original problem. This adds a factor of n to the run-time, preserving fixed-parameter tractability.

The compression problem can be modified further in two useful ways. The first modification is to get disjointness of solutions T_i and S_i. This can be achieved by guessing the intersection $U_i = T_i \cap S_i$ by taking all possible subsets of T_i. For every choice we can then solve the disjoint compression problem $(G_i - U_i, k - |U_i|, \ell)$ with starting solution $T_i \setminus U_i$. If the non-disjoint instance has a solution, then the instance where we guessed the intersection correctly has a solution. Otherwise, none of the disjoint instances has a solution. This adds a factor of $2^{|T_i|} = 2^{k+1}$ to the run-time, also preserving fixed-parameter tractability.

The other useful modification is about not solving the problem exactly but instead to guess a set $\mathcal{S}$ of bounded size intersecting S_i in at least one vertex. If we have a routine that for an instance (G', k', ℓ', T') of the disjoint compression problem returns us a set $\mathcal{S}$ that is guaranteed to intersect some solution (if a non-empty solution exists), we can branch as follows. First we check whether the empty solution already solves the problem by Theorem 2. This solves the the problem for $k = 0$. Otherwise we call our routine on the instance obtaining a set $\mathcal{S}$. For every $v \in \mathcal{S}$ we recurse on the instance $(G' - v, k' - 1, \ell, T')$. This adds a factor of $f(k', \ell', n')^{k'}$ to the run-time where $f(k', \ell', n')$ is an upper bound on the size of $\mathcal{S}$ on an instance (G', k', ℓ', T') with $n' = |G'|$. This preserves fixed-parameter tractability if we can write $f(k', \ell', n')^{k'}$ as $g(k', \ell') \cdot \mathrm{poly}(n')$ for some appropiate function g. Note that this is the case even if $f(k', \ell', n') = h_1(k', \ell') \cdot \log^{h_2(k', \ell')}(n')$.

After all these reductions we are left with the following problem:

INTERSECTING DIRECTED LONG CYCLE HITTING SET *Parameter:* $k + \ell$.
Input: A directed multigraph G, $T \subseteq V(G)$ and integers $k, \ell \in \mathbb{N}$.
Properties: $\mathsf{cf}(G - T) \leq \ell$
Task: Find a set $\mathcal{S} \subseteq V(G) \setminus T$ that intersects a set $S \subseteq V(G) \setminus T$ of size at most k with $\mathsf{cf}(G - S) \leq \ell$ if such a set exists.

The set $\mathcal{S}$ will often help us to argue about solutions S disjoint from it. This is often described as branching steps throughout the algorithm but we decided to collect all the branching here to be more precise about the assumptions needed in the theorems.

Contraction. In the last section we reduced the original DIRECTED LONG CYCLE HITTING SET problem to a variant where we are already given a solution T and now want to find a set $\mathcal{S}$ intersecting every solution S disjoint from T. Except for the intersection step, such reductions have been used by Chen et al. [7] in their algorithm for DFVS. The next key observation in their algorithm for DFVS is that every vertex of T must lie in their own strong component of $G - S$. The reason is that for every directed feedback vertex set S of G, each strong component of $G - S$ is a single vertex. For DIRECTED LONG CYCLE HITTING SET, the situation is way more complicated, as strong components of $G - S$ contain cycles of length up to ℓ. Moreover, those cycles can concatenate to arbitrarily large strong components. So it is possible that $G - S$ contains strong components with more than one vertex of T. In this section, we want to contract (parts of) such components to a single vertex such that eventually, after contraction, every strong component of $G - S$ contains at most one vertex of T. A structural result allowing the contraction is the following lemma:

▶ **Lemma 3.** *Let G be a digraph and let $X \subseteq V(G)$ be such that $G[X]$ is strong and $\mathsf{cf}(G[X]) \leq \ell$. Suppose that the following two properties hold:*

1. *Every cycle of G has length at most ℓ or length at least ℓ^2.*
2. *For any $a, b \in X$ there cannot be both*
   a. *an $a \to b$-path P_{ab} of length at least ℓ in $G[X]$*
   b. *a $b \to a$-path P_{ba} of length at most ℓ in $G - (X \setminus \{a, b\})$*

Let G/X be the digraph obtained by contracting X to a single vertex x.

- *If $\mathsf{cf}(G-S) \leq \ell$ for some $S \subseteq V(G) \setminus X$, then $\mathsf{cf}(G/X - S) \leq \ell$.*
- *If $\mathsf{cf}(G/X - S') \leq \ell$ for some $S' \subseteq V(G/X) \setminus \{x\}$, then $\mathsf{cf}(G - S') \leq \ell$.*

To algorithmically use this result we have to make sure its requirements are fulfilled. This is easy if G has a cycle C with $\ell < |C| < \ell^2$. We can can detect these cycles in time $2^{\mathcal{O}(\ell^2)} \cdot n^{\mathcal{O}(1)}$ by using color coding (see Alon et al. [1]). If there exists such an cycle, any long cycle hitting set has to intersect it. Also it's length is bounded, so we can return it as set $\mathcal{S}$.

To find a set X fulfilling the remaining properties is more difficult. For this we build on a tool called "k-representative set of paths".

▶ **Definition 4.** *Let G be a digraph, $x, y \in V(G)$ and $k \in \mathbb{Z}_{\geq 0}$. A set $\mathcal{P}$ of $x \to y$-paths is a* k-representative set of $x \to y$-paths, *if for every set $S \subseteq V(G)$ of size at most k it holds: If there is an $x \to y$-path in $G - S$ there is an $x \to y$-path $P \in \mathcal{P}$ that is disjoint from S.*

In our case such a k-representative set of paths will be useful for constructing a closed walk visting several vertices of T to use as our set X. Later on the reversed property of k-representative sets of paths will also be handy: if you hit all paths of $\mathcal{P}$ with a set S of size at most k, then there exist no $x \to y$-paths in $G - S$.

Alas, we were not able to find k-representative sets of paths of small size in general graphs. In the case of strong digraphs of bounded circumference, however, we obtain the following:

▶ **Theorem 5.** *Let G be a strong digraph, $x, y \in V(G)$ and $k \in \mathbb{Z}_{\geq 0}$. Then a k-representative set of $x \to y$-paths having size $\mathsf{cf}(G)^{\mathcal{O}(k^2 \log k)} \cdot \log n$ can be found in time $\mathsf{cf}(G)^{\mathcal{O}(k^2 \log k)} \cdot n^{\mathcal{O}(1)}$.*

We need to generalize this tool a bit further as our graph G neither has bounded circumference nor does it need to be strong. However, we have a set T of small size such that $\mathsf{cf}(G - T) \leq \ell$ and we are only interested in the strong components of the graph. This leads us to the following specialized lemma:

▶ **Lemma 6.** *Let G be a strong digraph, $T \subseteq V(G)$ and $k, \ell \in \mathbb{Z}_{\geq 0}$ with $\mathsf{cf}(G - T) \leq \ell$. Then in time $2^{\mathcal{O}(k\ell + k^2 \log k \log \ell)} n^{\mathcal{O}(1)}$, we can find a set $\mathcal{Q}$ of $|T|^2 2^{\mathcal{O}(k\ell + k^2 \log k \log \ell)} \log^2 n$ closed walks with the following property: If there is a set $S \subseteq V(G)$ of size at most k with $\mathsf{cf}(G - S) \leq \ell$ and there are two vertices of T in the same strong component of $G - S$ then there is*

- *a closed walk in $Q \in \mathcal{Q}$ containing two vertices of T that is disjoint from S or*
- *a simple cycle of length at most ℓ containing two vertices of T.*

The lemma allows for a branching procedure that creates several instances. For each instance we assume that there is a solution S such that no two vertices of T lie in the same strong component of $G - S$. We call such a solution *isolating* long cycle hitting set. The instances are created in the following way: We keep our original instance just in case it has an isolating long cycle hitting set. Then we search for a simple cycle C of length at most ℓ in G visiting at least two vertices of T. Such a cycle can be found by color coding in time $2^{\mathcal{O}(\ell)} \cdot n^{\mathcal{O}(1)}$. If such a cycle exists, we branch into two instances: In the first instance, we assume that S intersects C and we can just return $\mathcal{S} = V(C)$ as solution to our intersection problem. In the second instance, C is disjoint from S and we have a candidate X for contraction with Lemma 3. If no such cycle exists we get our candidate applying Lemma 6 by branching on which closed walk of $\mathcal{Q}$ is disjoint from S and take that as a candidate X.

We now have to check whether our candidate X fulfills the assumptions of Lemma 3. If $\mathsf{cf}(G[X]) > \ell$ then X cannot be disjoint from S in contradiction to our branching assumption and we give up on this branch. Then we check for every pair $a, b \in X$, if paths P_{ab} and P_{ba}

as in Lemma 3 exist. If they do we cannot contract X, but $P_{ab} \circ P_{ba}$ forms a cycle of length more than ℓ, so it has to be intersected by S. By branching assumption $V(P_{ab}) \subseteq X$ is disjoint from S and therefore P_{ba} has to intersect S. Luckily, $|V(P_{ba})| \leq \ell + 1$ and we can return $\mathcal{S} = V(P_{ba})$ as set intersecting S. If these paths do not exist we can contract X to a single vertex to obtain a new instance (G', k, ℓ, T'). We continue our branching procedure with this new instance until we guess that our instance has an isolating long cycle hitting separator. As the cardinality of T is decreased in each branching step this is the case at the latest when $|T| = 1$. Note that this may lead to the strange case that we are indeed searching for a solution S that is larger than our set T.

The remaining problem can be stated as:

Isolating Long Cycle Hitting Set Intersection	*Parameter:* $k + \ell + |T|$.
Input:	A directed multigraph G, integers $k, \ell \in \mathbb{N}$ and a set $T \subseteq V(G)$
Properties:	$\mathsf{cf}(G - T) \leq \ell$
Task:	Find a set $\mathcal{S}$ intersecting some isolating long cycle hitting set S of size at most k with respect to T if such a set exists.

The above procedure (and the result of this section) can be summarized as follows.

▶ **Theorem 7.** *Instances (G, k, ℓ) of* Directed Long Cycle Hitting Set *can be solved in time $2^{\mathcal{O}(\ell^2+\ell k^3 \log k)} \cdot (f_{ii}(k, \ell))^k \cdot n^{\mathcal{O}(1)}$ by at most $2^{\mathcal{O}(\ell k^3 \log k)} \cdot (f_{ii}(k, \ell))^k \cdot n^2 \log^{2k+2}(n)$ calls to an algorithm A_{ii} solving the* Isolating Long Cycle Hitting Set Intersection *problem, where $f_{ii}(k, \ell)$ is a size bound on the set produced by A_{ii}.*

Reducing to Important Hitting Separator. In the previous section we reduced the Directed Long Cycle Hitting Set problem to the Isolating Long Cycle Hitting Set Intersection problem, a variant where we are already given a solution T and search for a solution S disjoint from T of size at most k. Additionally, we know that T has at most one vertex in each strong component of $G - S$. For the remainder of this subsection, assume that there is such a solution S of size at most k.

Intuitively, we did the reduction to the isolating variant in order to apply something like Skewed Multiway Cut. This was done by Chen et al. [7] in their algorithm for Directed FVS. They guessed a topological ordering of the vertices of T in $G - S$ and used Skewed Multiway Cut to cut away the backward paths. This also implied that each strong component consisted of a single vertex. In our case, though, we still have cycles left, and a direct construction to Skewed Multiway Cut gives us no control over their length.

We instead guess only a last vertex $t \in T$ in some topological ordering of the strong components of $G - S$. From here our approach differs significantly from that of Chen et al. [7] for Directed FVS. Instead of finding all cuts at once, we focus on the $t \to T \setminus \{t\}$-cuts while still hitting long cycles. This is we want to find the strong component of t in $G - S$.

For this it is useful to see, that two types of arcs may not lie in the strong component of t in $G - S$. The first kind of arcs are simply the arcs having their endpoint in $T \setminus \{t\}$, i.e. arcs in $\delta^-(T \setminus \{t\})$. This is because S is isolating. The other kind of arcs are the arcs that lie only on long cycles, i.e. arcs $a = (v, w) \in A(G)$ with $\mathrm{dist}_{G-a}(w, v) \geq \ell$. This follows from the fact that S hits all long cycles. Therefore, the strong component of t in $G - S$ must be a subset of the connected component of t in G after above arcs are removed. We call the later component $C_t^\star$. We want to focus our search onto that component.

However, there may also be other arcs and vertices which do not lie in the strong component of t in $G - S$ which are still in $C_t^\star$. To make it easier to argue about these, we use the shadow-covering technique introduced by Chitnis et al. [9] to solve the Directed Multiway Cut problem. For this we need to define what the shadow of our solution S with respect to the set T is:

▶ **Definition 8** (shadow). *Let G be a digraph and let $T, S \subseteq V(G)$. A vertex $v \in V(G)$ is* in the forward shadow $f_{G,T}(S)$ of S (with respect to T) *if S is a $T \to v$-separator in G, and v is* in the reverse shadow $r_{G,T}(S)$ of S (with respect to T) *if S is a $v \to T$-separator in G. All vertices of G which are either in the forward shadow or in the reverse shadow of S (with respect to T) are said to be* in the shadow of S (with respect to T)*.*

Note that S itself is not in its own shadow. The most useful property of the shadow for our purposes is that every vertex that is not in the shadow of S with respect to T is reachable from a vertex of T and can reach some (maybe different) vertex of T. In particular, if a vertex v is not in the shadow of S and reachable from t (our last vertex) in $G - S$ then it has to lie in the strong component of t in $G - S$. This is because v can also reach a vertex of T but there is no $T \to T \setminus \{t\}$-walk in $G - S$.

We use the deterministic algorithm for shadow covering by Chitnis et al. [8]. The following corollary follows from their paper:

▶ **Corollary 9.** *Let (G, k, ℓ, T) be an instance of* Isolating Long Cycle Hitting Set Intersection*. One can construct, in time $2^{\mathcal{O}(k^2)} \cdot n^{\mathcal{O}(1)}$, sets $Z_1, \dots, Z_p$ with $p \leq 2^{\mathcal{O}(k^2)} \log^2 n$ such that if there is an isolating long cycle hitting set of size at most k, there is isolating long cycle hitting set S of size at most k and an $i \in \{1, \dots, p\}$ such that $Z_i \cap (S \cup T) = \emptyset$ and Z_i includes the shadow of S with respect to T.*

By branching on the possible choices of Z_i we can assume that we have a set Z that covers (read: includes) the shadow of S with respect to T and is disjoint from S and T. Now every vertex outside of Z is reachable from a vertex of T and can also reach a vertex of T.

Consider now the set $V_{\mathsf{out}} \subseteq V(G) \setminus Z$ defined as follows. For every $v \in V_{\mathsf{out}}$ there is a $v \to w$-path P with $w \in V(G) \setminus Z$ whose inner vertices are inside Z and one of the following properties holds. The endpoint w is contained in $T \setminus \{t\}$ or P contains an arc $a = (x, y) \in A(G)$ that only lies on long cycles (i.e. $\mathrm{dist}_{G-a}(y, x) \geq \ell$). The set of these vertices may not be reached from t in $G - S$ if the other endpoint w is not in S. Because then w reaches a vertex of T and that gives us either a $t \to T \setminus \{t\}$ path in $G - S$ or a closed walk in $G - S$ containing an arc that only lies on long cycles (which therefore contains itself a long cycle).

To get rid of the condition that the other endpoint of the path P we apply the tool of critical vertices also used by Chitnis et al. [8]. For this we use an auxiliary graph (called torso) which is created by taking all vertices of $V(G) \setminus Z$ and shrinking paths with interior points in Z to arcs. By remembering the paths which contained arcs only on long cycle we get a set U_t^{long} of dangerous arcs that must not be traversable from t. In case they are not traversable only by the endpoint w lying in S these vertices are called k-critical with respect to U_t^{long}. Now, we can use the following theorem by Chitnis et al. [8].

▶ **Proposition 10** ([8]). *Given a digraph G, a subset U of its arcs, and some $t \in V(G)$, in time $2^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$ we can find a set F_{critical} of size $2^{\mathcal{O}(k)}$ that contains all k-critical vertices.*

We apply Proposition 10 to $G, U_t^{\mathsf{long}}, t$ and k, and add the resulting set F_{critical} to our solution $\mathcal{S}$. If our solution S is still disjoint from $\mathcal{S}$, it cannot contain a k-critical vertex. This implies that t cannot reach a vertex of V_{out} in $G - S$. So S is a $t \to V_{\mathsf{out}}$-separator in the strong component $C_t^\star$.

But $S \cap C_t^\star$ may also have other properties like hitting long cycles that intersect or lie within $C_t^\star$. To cover these we introduce the notation of important hitting $t \to V_{\mathsf{out}}$-separators. These are $t \to V_{\mathsf{out}}$-separators U that hit all long cycles in a digraph and are backward-range minimal in the sense that there is no other $t \to V_{\mathsf{out}}$-separators U' hitting all cycles with $|U'| \leq |U|$ and $R^-_{G-U'}(V_{\mathsf{out}}) \subsetneq R^-_{G-U}(V_{\mathsf{out}})$.

The main result of this section is that it is enough to find a set $\mathcal{S}$ intersecting one important hitting $t \to V_{\mathsf{out}}$-separator U for every backward range $R^-_{G-U}(V_{\mathsf{out}})$ in a strong digraph (read $G[C_t^\star]$). We call the remaining problem Important Hitting Separator in Strong Digraphs which reads as follows:

Important Hitting Separator in Strong Digraphs *Parameter:* $k + \ell$.

Input: A strong digraph G, integers $k, \ell \in \mathbb{N}$, $t \in V(G)$ and $V_{\mathsf{out}} \subseteq V(G)$.

Properties: $\mathsf{cf}(G - t) \leq \ell$, every arc of G lies on a cycle of length at most ℓ.

Task: Find a set $\mathcal{S}_{\mathsf{hs}}$ intersecting a important hitting $t \to V_{\mathsf{out}}$-separator of size at most k in every range equivalence class.

Portals and Clusters. In the previous section we reduced Isolating Long Cycle Hitting Set Intersection to Important Hitting Separator in Strong Digraphs. In an intuitive way we replaced the constraint on S of hitting all cycles going through $T \setminus \{t\}$ by a constraint that S needs to be a backward-range minimal $t \to V_{\mathsf{out}}$-separator. Also we simplified our graph to be strongly connected and that every arc lies on a short cycle. We now want to also break the long cycles going only through t into cut constraints. For this we consider the strong components of $G - t$. Let $\mathcal{C}$ the set of all such components. Our main interest are now portal vertices.

▶ **Definition 11.** *Let G be a digraph and let $C \subset V(G)$. A vertex $v \in C$ is a* portal vertex *of C, if $\Delta_G(v) > \Delta_{G[C]}(v)$, where $\Delta_H(v)$ is the number of incident arcs (both in-coming and out-going) of v in a graph H. We denote by X_C the set of all portal vertices of C.*

As all arcs of G lie on a short cycles the arcs going between clusters must cycle back to t.

▶ **Lemma 12.** *Let $C \in \mathcal{C}$ and $v \in X_C$. There is a cycle O_v with $v, t \in V(O_v)$ and $|O_v| \leq \ell$.*

These cycles allow us to transform paths between portal vertices of the same strong component into closed walks. Like in the first part of our algorithm we can eliminate cycles of length between $\ell + 1$ and $2\ell^6$ by detecting them and returning them as set $\mathcal{S}_{\mathsf{hs}}$ as any hitting separator has to intersect these. This gap between small and large cycles however allows us to get a similar gap for the distance between portal vertices.

▶ **Lemma 13.** *For any $v_1, v_2 \in X_C$, either $\mathsf{dist}_{G[C]}(v_1, v_2) \leq 2\ell^2$ or $\mathsf{dist}_{G[C]}(v_1, v_2) \geq 2\ell^6 - 2\ell$.*

This in turn allows us to cluster the portal vertices of every component. For $\ell_{\max} = 2\ell^2$ we put all portal vertices at distance at most $\ell_{\max}$ from a portal vertex v into the set X_v. This defines a partition into clusters:

▶ **Lemma 14.** *For any $C \in \mathcal{C}$ and $v_1, v_2 \in X_C$, sets X_{v_1} and X_{v_2} are either disjoint or equal.*

Consider now a long cycle O in G. Assume it contains for each strong component $C \in \mathcal{C}$ it visits only portal vertices of one of the clusters of C. In strong digraphs of circumference at most ℓ (like $G[C]$) the length of a path can at most be ℓ^2 times the distance between its endpoints. Therefore the length of O inside a single component C can be at most $2\ell^4$. We already made sure that long cycles in G have length more than $2\ell^6$. But we can shortcut O after the visit of C to a cycle with length between $\ell+1$ and $2\ell^6$ – a contradiction. Therefore every long cycle has a path between different clusters of some strong component of $G-t$.

By carefully analyzing the structure of the strong components and guessing vertices of S (by including them into $\mathcal{S}_{\mathsf{hs}}$) we make sure there are neither too many strong components with more than one cluster nor do these have to many clusters themselves. Moreover, we were able to restrict $t \to V_{\mathsf{out}}$-paths to a single strong component.

This means that our sought after solution S forms in each strong component C a multiway cut between the different cluster X_i of C and additionally cuts all $X_i \to V_{\mathsf{out}}$-paths in backward-range minimal way. This structure of S we call important cluster separators:

▶ **Definition 15.** *Let G be a digraph and let $X_1, \ldots, X_t, Y \subseteq V(G)$ be pairwise disjoint vertex sets. We call a vertex set $U \subseteq V \setminus (X_1 \cup \ldots \cup X_t \cup Y)$ a* cluster separator *if $G-U$ contains*

- *no path from X_i to X_j for $i \neq j$ and*
- *no path from X_i to Y for $i = 1, \ldots, t$.*

A cluster separator U is important *if there is no cluster separator U' with $|U'| \leq |U|$ and $R^-_{G-U'}(Y) \subsetneq R^-_{G-U}(Y)$.*

If $V_{\mathsf{out}} = \emptyset$ this boils down to the Directed Multiway Cut problem, which we solve by the algorithm of Chitnis et al. [9]. It thus remains to solve the case where $V_{\mathsf{out}} \neq \emptyset$.

Important Cluster Separator in Strong Digraphs of Bounded Circumference — *Parameter:* $k+\ell$.

Input: A strong digraph G, integers $k, \ell \in \mathbb{N}$ and sets $X_1, \ldots, X_p, V_{\mathsf{out}} \subseteq V(G)$.

Properties: $\mathsf{cf}(G) \leq \ell$, $X_i, V_{\mathsf{out}} \neq \emptyset$, $2 \leq p \leq k(k+1)+1$,
$\mathrm{dist}(v,w) \leq 2\ell^2 \quad \forall v, w \in X_i, i \in \{1, \ldots, p\}$.

Task: Find a vertex set $\mathcal{S}_{\mathsf{cluster}}$ intersecting any important cluster separator with respect to $X_1, \ldots, X_\ell, V_{\mathsf{out}}$ of size at most k.

Finding Important Cluster Separators. In this section we want to solve the Important Cluster Separator in Strong Digraphs of Bounded Circumference problem. This problem is strongly related to the Directed Multiway Cut Problem. By adding an additional vertex $v^\star$ that has an incoming arc from ever vertex of V_{out} we see that cluster separator are indeed multiway cuts in this modified graph. The difficulty lies in the notion of importance we introduced (and needed). That is, we want to intersect all cluster separators S where $R^-_{G-S}(V_{\mathsf{out}})$ is minimal for their size.

We introduce two definitions to handle this. The first is the concept of the *frontier* F of a cluster separator S. These are the vertices that define the backward range from V_{out}, i.e. the vertices of S that can reach V_{out} without going through another vertex of S. The other concept is that of an *outlet*. Given an $X_i \to X_j$-path P and two integers $\alpha, \beta \in \mathbb{Z}_{\geq 0}$, an (α, β)-outlet of P is a vertex ω of P with the following property: there is a $\omega \to V_{\mathsf{out}}$-path R_ω such that every vertex on P except for the α-many preceding and following vertices of ω on P have distance at least β from R_ω. These outlets are in some sense key positions where $X_i \to X_j$-paths start to significantly differ from $X_i \to V_{\mathsf{out}}$-paths.

For outlets we differentiate between "open" and "closed" outlets. *Open* outlets are outlets that lie in $R^-_{G-S}(V_{\mathsf{out}})$, i.e. behind the frontier; the other outlets *closed.* The frontier F is therefore separates $\mathcal{X} = \bigcup_{i=1}^t X_i$ from the open outlets. The rest of our efforts now focuses on finding a set V_Ω such that the frontier F is an important $\mathcal{X} \to (V_{\mathsf{out}} \cup V_\Omega)$-separator. This set V_Ω contains (a subset of) the open outlets for some $X_i \to X_j$-paths. As set of paths tho search on we take for every ordered pair of distinct X_i, X_j an arbitrary $X_i \to X_j$-path $P_{i,j}$.

The main property we use for finding the set V_Ω is some kind of locality argument. As our graph G is strong and has bounded circumference, the length of any path cannot differ too much from the distance of its endpoints. So if we know that a path is hit by S not to far from an outlet, we can guess these vertices. This is done with the help of k-representative sets of paths: if we know that an $\omega \to V_{\mathsf{out}}$-path is hit near ω, we construct a k-representative set of $\omega \to V_{\mathsf{out}}$-paths. We can argue that also one of the paths in this set has to be hit near ω. So we can guess all the vertices near ω on the paths of the k-representative set for their intersection with S.

By carefully guessing such potential intersections, and choosing α, β properly, we obtain:

- If a path $P_{i,j}$ has more than $\gamma = \mathrm{poly}(k, \ell)$ outlets, one of them is open.
- If there is a $X_i \to X_j$-path in $G - F$ then $P_{i,j}$ has an open outlet.

The last step is now to get rid of the $X_i \to X_j$-paths in $G - F$. We achieve this by guessing that a so called *landing strip* in front of an open outlet of $P_{i,j}$ (which exists by the second property) is disjoint from S. This landing strip has the task that that if there is a $X_i \to X_j$-path in $G - F$ than also the open outlet would be reachable. This again works by the locality of our strong graphs of bounded circumference.

After all these guessing we obtain (for the right guess) a set V_Ω such that F is an important $\mathcal{X} \to (V_{\mathsf{out}} \cup V_\Omega)$-separator. These we can enumerate by a result of Chitnis et al. [9].

▶ **Proposition 16** ([9]). *Let G be a digraph and let $X, Y \subseteq V(G)$ be disjoint non-empty vertex sets. For every $p \geq 0$ there are at most 4^p important $X - Y$-separators of size at most p, and all these separators can be enumerated in time $\mathcal{O}(4^p \cdot p(n + m))$.*

Putting Everything Together. Finally, we are able to prove our main result. By combining the reductions of the previous sections, we get an overall algorithm solving Directed Long Cycle Hitting Set:

▶ **Theorem 1.** *There is an algorithm that solves* Directed Long Cycle Hitting Set *in time $2^{\mathcal{O}(\ell^6 + \ell k^3 \log k + k^5 \log k \log \ell)} \cdot n^{\mathcal{O}(1)}$ for n-vertex digraphs G and parameters $k, \ell \in \mathbb{N}$.*

4 k-Representative Sets of Paths

In this section, we show how to obtain a k-representative set of paths of small size in strong digraphs of bounded circumference. Let us briefly recall the definition.

▶ **Definition 4.** *Let G be a digraph, $x, y \in V(G)$ and $k \in \mathbb{Z}_{\geq 0}$. A set $\mathcal{P}$ of $x \to y$-paths is a* k-representative set of $x \to y$-paths, *if for every set $S \subseteq V(G)$ of size at most k it holds: If there is an $x \to y$-path in $G - S$ there is an $x \to y$-path $P \in \mathcal{P}$ that is disjoint from S.*

Our goal is to prove the following:

▶ **Theorem 5.** *Let G be a strong digraph, $x, y \in V(G)$ and $k \in \mathbb{Z}_{\geq 0}$. Then a k-representative set of $x \to y$-paths having size $\mathsf{cf}(G)^{\mathcal{O}(k^2 \log k)} \cdot \log n$ can be found in time $\mathsf{cf}(G)^{\mathcal{O}(k^2 \log k)} \cdot n^{\mathcal{O}(1)}$.*

If all $x \to y$-paths are short we can use the following result of Monien [23]:

▶ **Proposition 17** ([23]). *Let G be a digraph, let $x, y \in V(G)$ and let $k \in \mathbb{N}$. If every $x \to y$-path in G has length at most ℓ, then a k-representative set containing at most ℓ^k many $x \to y$-paths can be found in time $\ell^{\mathcal{O}(k)} \cdot n^{\mathcal{O}(1)}$.*

Recently, Fomin et al. [11] improved the computation of representative sets of paths, both in terms of the size of the set and the run time, but Proposition 17 will be sufficient for our purposes.

A motivational example. Before we give our construction for k-representative sets of paths in strong digraphs of bounded circumference, we want to consider graphs of treewidth two and a special example of treewidth three. Strong digraphs of treewidth two are trees with bidirected arcs. In this case we have that for every pair $x, y \in V(G)$ there is an unique $x \to y$-path P. Thus, $\{P\}$ is a feasible k-representative set of paths.

The situation is significantly different even for $\mathsf{cf}(G) = 3$. Consider the strong digraph in Figure 1.

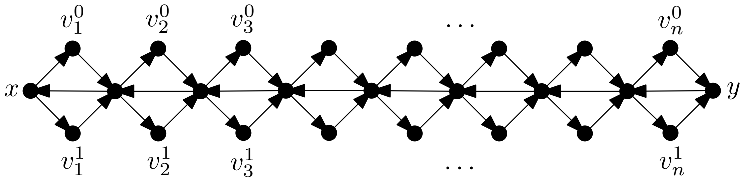


Figure 1 A digraph G with $\mathsf{cf}(G) = 3$ where every k-representative set of $x \to y$-paths has size $2^{\Omega(k)} \log n$.

There are exactly 2^n different $x \to y$-paths in G; each such path corresponds to a $0-1$ vector of length n. Thus, if we remove a vertex v_i^0 or v_i^1, then only those paths survive that have 1 or 0 at the i-th coordinate, respectively. Therefore, a collection of paths in this graph is k-representative only if no matter how we fix the values of k arbitrary coordinates, there is a vector in the collection satisfying these constraints. Such collections of vectors are also known as binary covering arrays. Kleitman and Spencer [17] proved that every collection of vectors of length n satisfying this property has size $2^{\Omega(k)} \cdot \log n$ (more precisely, they gave a lower bound on the dual question of k-independent families, but it can be easily rephrased into this lower bound).

We will now construct a k-representative set of paths for this graph G by using so called k-perfect families of hash functions.

▶ **Definition 18.** *Let $\mathcal{F}$ be a family of functions $f : U \to \{1, \ldots, k\}$ on the universe U. We say that $\mathcal{F}$ is a* k-perfect family of hash functions *if for every $X \subseteq U$ of size at most k, there is an $f \in \mathcal{F}$ that is injective on X, i.e. $f(x) \neq f(x')$ for any two distinct $x, x' \in X$.*

We use the following result by Alon et al. [1] for our construction.

▶ **Proposition 19** ([1]). *Let U be a universe and $k \in \mathbb{N}$. Then there exists a k-perfect family $\mathcal{F}$ of size $2^{\mathcal{O}(k)} \log |U|$ that can be constructed in time $2^{\mathcal{O}(k)}|U|^{\mathcal{O}(1)}$.*

Before considering arbitrary strong digraphs of bounded circumference, let us explain how k-perfect families of hash functions can be used for the construction in the case of the digraph G of Figure 1. Let $\mathcal{F}$ be a k-perfect family of hash functions over the universe $U = \{1, \ldots, n\}$ as in Proposition 19. Moreover, let $\mathcal{H}$ be the set of all functions $h : \{1, \ldots, k\} \to \{0, 1\}$. For $(f, h) \in \mathcal{F} \times \mathcal{H}$ denote by $P_{f,h}$ the $x \to y$-path in G that uses the vertices $v_i^{h(f(i))}$ for $i = 1, \ldots, n$. Then we add for every pair $(f, h) \in \mathcal{F} \times \mathcal{H}$ the path $P_{f,h}$ to our set $\mathcal{P}$.

Now consider a deletion set $S \subseteq V(G)$ of size at most k such that there is still an $x \to y$-path in G. Then S contains only vertices of type v_i^j and at most one of v_i^0 and v_i^1 for every $i \in \{1, \ldots, n\}$. In other words, for some $X \subseteq U$ of size at most k and function $g : X \to \{0, 1\}$, the vertices $v_i^{g(i)}$ form the set S. As $\mathcal{F}$ is a k-perfect family of hash functions, there is an $f \in \mathcal{F}$ that is injective on X. Now consider the function h defined as follows.

For every $i \in X$, let $h(f(i)) = 1 - g(i)$; as f is injective on X, this is well-defined and gives a function $h : f(X) \to \{0,1\}$. Complete, h to a function $h : \{1,\dots,k\} \to \{0,1\}$ by choosing the remaining values arbitrarily. We claim that the path $P_{f,h}$ introduced for this choice of f and h is disjoint from S. For $i \notin X$, it does not matter if $P_{f,h}$ uses v_i^0 or v_i^1. For $i \in X$, set S contains $v_i^{g(i)}$. By our definition of h, we have $h(f(i)) = 1 - g(i)$, hence $P_{f,h}$ uses $v_i^{1-g(i)}$, avoiding S. Thus $P_{f,h}$ is indeed disjoint from S.

Our proof of Theorem 5 generalizes this construction to arbitrary strong digraphs of bounded circumference: we construct the path by concatenating a series of fairly independent "short jumps." For each of these short jumps, we construct a k-representative set of paths by Proposition 17. The choice of which short path to select is determined by a k-perfect family of hash functions, similarly to the argument in the previous paragraph.

Strong digraphs with bounded circumference. Before we formally start proving Theorem 5, we establish some structural properties of strong digraphs with bounded circumference.

▶ **Lemma 20.** *Let G be a digraph and let $x, y \in V(G)$. If P_1 is an $x \to y$-path and P_2 is a $y \to x$-path, then $|P_1| \le (\mathsf{cf}(G) - 1)|P_2|$ holds. Consequently, we have $\mathsf{dist}_G(x,y) \le (\mathsf{cf}(G) - 1)\mathsf{dist}_G(y,x)$.*

Proof. Suppose, for sake of contradiction, that $|P_1| > (\mathsf{cf}(G) - 1)|P_2|$. By $x, y \in V(P_1) \cap V(P_2)$, we can split P_1 into $|P_2|$ pairwise disjoint subpaths whose internal vertices are disjoint from $V(P_2)$. Note that there are $|V(P_1)| - |V(P_2)| = (|P_1| + 1) - (|P_2| + 1) = |P_1| - |P_2|$ of these internal vertices. By pigeonhole principle, at least one of the subpaths has at least $\left\lceil \frac{|P_1|-|P_2|}{|P_2|} \right\rceil = \left\lceil \frac{|P_1|}{|P_2|} \right\rceil - 1$ internal vertices. By our assumption, these are at least $\mathsf{cf}(G) - 1$ many. But than the whole subpath has at least $\mathsf{cf}(G) + 1$ vertices. Since $P_1 \circ P_2$ is a closed walk, our segment is contained in a closed walk. Moreover, P_1 is acyclic and our segment is internally disjoint from P_2. Thus, the segment is even contained in a cycle. But this cycle then has length at least $\mathsf{cf}(G) + 1$, contradicting the definition of circumference. ◀

By using that there is always a backward path in strong digraphs, applying above result twice yields:

▶ **Lemma 21.** *Let G be a strong digraph and $x, y \in V(G)$. Then $|P| \le (\mathsf{cf}(G) - 1)^2 \mathsf{dist}_G(x,y)$ for every $x \to y$-path P.*

Proof. Let W be a shortest $x \to y$-path in G. As G is strong, there is also a $y \to x$-path Q in G. By Lemma 20 we then have

$$|P| \le (\mathsf{cf}(G) - 1)|Q| \le (\mathsf{cf}(G) - 1)^2|W| = (\mathsf{cf}(G) - 1)^2 \mathsf{dist}_G(x,y). \quad ◀$$

Similarly to the length, we can also argue about the distance between two paths.

▶ **Lemma 22.** *Let G be a strong digraph, $x, y \in V(G)$ be two vertices, and P_1, P_2 be two $x \to y$-paths. For every vertex v of P_1, we have $\mathsf{dist}_G(P_2, v) \le 2(\mathsf{cf}(G) - 2)$ and $\mathsf{dist}_G(v, P_2) \le 2(\mathsf{cf}(G) - 2)$.*

Proof. As G is strong, there is a $y \to x$-path Q.

▷ Claim 23. For $i \in \{1,2\}$, we have that

(i) $\mathsf{dist}_G(Q, v) \le \mathsf{cf}(G) - 2$ and $\mathsf{dist}_G(v, Q) \le \mathsf{cf}(G) - 2$ for every $v \in P_i$, and

(ii) $\mathsf{dist}_G(P_i, v) \le \mathsf{cf}(G) - 2$ and $\mathsf{dist}_G(v, P_i) \le \mathsf{cf}(G) - 2$ for every $v \in Q$.

Proof. Let $v \in P_i$. As P_i is acyclic (it is a path), but $P_i \circ Q$ is a closed walk in G, v has to lie on a cycle O in $P_i \circ Q$. This cycle has length at most $\mathsf{cf}(G)$. Furthermore, O has at least two vertices in $V(Q)$, as it contains an arc of Q. So there is a path in O from v to a vertex of Q of length at most $|O| - 2 \leq \mathsf{cf}(G) - 2$, showing $\mathsf{dist}_G(v, Q) \leq \mathsf{cf}(G) - 2$. On the other hand there is also a $V(Q) \to v$-path (from another vertex of $V(Q) \cap V(O)$) in O of length at most $|O| - 2 \leq \mathsf{cf}(G) - 2$, showing $\mathsf{dist}_G(Q, v) \leq \mathsf{cf}(G) - 2$. This shows Statement (i). Statement (ii) can be seen analogously by switching the roles of P_i and Q. ◁

Now fix a $v \in V(P_1)$. By Claim 23, we have that there is a $w \in V(Q)$ such that $\mathsf{dist}_G(w, v) = \mathsf{dist}_G(Q, v) \leq \mathsf{cf}(G) - 2$. Applying Claim 23 another time and using triangle inequality we get

$$\mathsf{dist}_G(P_2, v) \leq \mathsf{dist}_G(P_2, w) + \mathsf{dist}_G(w, v) \leq 2(\mathsf{cf}(G) - 2).$$

Similarly, we get from Claim 23 that there is an $u \in V(Q)$ with $\mathsf{dist}_G(v, u) = \mathsf{dist}_G(v, Q) \leq \mathsf{cf}(G) - 2$. Another application of Claim 23 and the triangle inequality yields

$$\mathsf{dist}_G(v, P_2) \leq \mathsf{dist}_G(v, u) + \mathsf{dist}_G(u, P_2) \leq 2(\mathsf{cf}(G) - 2),$$

concluding the proof. ◀

We are now ready to prove Theorem 5.

▶ **Theorem 5.** *Let G be a strong digraph, $x, y \in V(G)$ and $k \in \mathbb{Z}_{\geq 0}$. Then a k-representative set of $x \to y$-paths having size $\mathsf{cf}(G)^{\mathcal{O}(k^2 \log k)} \cdot \log n$ can be found in time $\mathsf{cf}(G)^{\mathcal{O}(k^2 \log k)} \cdot n^{\mathcal{O}(1)}$.*

Proof. Let us fix an arbitrary $x \to y$-path R (which exists as G is strong) to guide our construction. Denote by r the length of R and by $v_0 = x, v_1, \ldots, v_{r-1}, v_r = y$ its vertices. We only consider a subset of vertices z_i at distance $d = 2\mathsf{cf}(G)^4$ from each other or more formally $z_i = v_{i \cdot d}$. These z_i will be the anchor vertices for our short jumps. We divide then z_i further into $k+1$ subsets Z^o by taking every $(k+1)$st vertex starting at offset o. Formally we define $z_i^o = z_{i(k+1)+o}$ and $Z^o = \{z_i^o\}$. These subsets have the advantage that one of these is far away from a deletion set S of size at most k. For this we fix a set S of size at most k such that an $x \to y$-path in $G - S$ exists.

▷ Claim 24. There is some $o_S \in \{0, \ldots, k\}$ such that

- $\mathsf{dist}_G(Z^{o_S}, S) > 2(\mathsf{cf}(G) - 2)$ and
- $\mathsf{dist}_G(S, Z^{o_S}) > 2(\mathsf{cf}(G) - 2)$.

Proof. We claim that for every $s \in S$ there is at most one value $o \in \{0, \ldots, k\}$ such that $\mathsf{dist}_G(Z^o, s) \leq 2\mathsf{cf}(G)^2$. Suppose that $\mathsf{dist}_G(w_1, s), \mathsf{dist}(w_2, s) \leq 2\mathsf{cf}(G)^2$ for some $w_1 \in Z^{o_1}$ and $w_2 \in Z^{o_2}$ with $o_1 \neq o_2$. Assume, without loss of generality, that w_1 is before w_2 on R; then $R[w_1, w_2]$ has length at least d (as different z_i have distance at least d). By Lemma 20, we have $\mathsf{dist}_G(s, w_1) \leq (\mathsf{cf}(G) - 1)\mathsf{dist}_G(w_1, s) \leq (\mathsf{cf}(G) - 1) \cdot 2\mathsf{cf}(G)^2$, thus $\mathsf{dist}_G(w_2, w_1) \leq \mathsf{dist}_G(w_2, s) + \mathsf{dist}_G(s, w_1) \leq 2\mathsf{cf}(G)^3$. Again by Lemma 20, we have $d \leq |R[w_1, w_2]| \leq (\mathsf{cf}(G) - 1)\mathsf{dist}_G(w_2, w_1) < 2\mathsf{cf}(G)^4$, a contradiction. Thus, we have proven that for each of the k vertices $s \in S$ there is at most one value $o \in \{0, \ldots, k\}$ such that s is at distance at most $2\mathsf{cf}(G)^2$ from Z^o. Therefore, by the pigeon-hole principle there is an $o_S \in \{0, \ldots, k\}$ such that $\mathsf{dist}_G(Z^{o_S}, S) > 2\mathsf{cf}(G)^2$. By Lemma 20 this also implies $\mathsf{dist}_G(S, Z^{o_S}) > 2\mathsf{cf}(G)^2 / (\mathsf{cf}(G) - 1) > 2(\mathsf{cf}(G) - 2)$. This completes the proof of Claim 24. ◁

Thus we know that a small surrounding of one of the Z^o's will be disjoint from S. Furthermore, Lemma 21 gives a bound on the length of a path P between two consecutive vertices z_i^o and z_{i+1}^o of Z^o, by $|P| \leq (\mathsf{cf}(G)-1)^2|R[z_i^o, z_{i+1}^o]| = \mathcal{O}(\mathsf{cf}(G)^7 k)$. This allows us to introduce sets $\mathcal{P}_i^o$ of k-representative $z_i^o \to z_{i+1}^o$-paths using the algorithm of Proposition 17 and have their size bounded by some $B = \mathcal{O}(\mathsf{cf}(G)^7 k)^k = \mathsf{cf}(G)^{\mathcal{O}(k \log k)}$ (using $k = 2^{\log k}$ and $\mathsf{cf}(G) \geq 2$). By duplicating paths as necessary we can assume that every P_i^o has size exactly B.

To make sure that our path collections with offset are connected to x and y, we construct additional sets $\mathcal{P}_x^o$ and $\mathcal{P}_y^o$ as follows: Let z_x^o be the first vertex in Z^o *after* x and z_y^o the last vertex *before* y. Then compute, using the algorithm of Proposition 17, $\mathcal{P}_x^o$ as a k-representative set of $x \to z_x^o$-paths and $\mathcal{P}_y^o$ as a k-representative set of $z_y^o \to y$-paths. As the distances between these pairs of vertices are bounded by the distance of neighboring vertices in Z^o we can analogously get a size bound of B for $\mathcal{P}_x^o$ and $\mathcal{P}_y^o$. Note that for some offsets o either $\mathcal{P}_x^o$ or $\mathcal{P}_y^o$ may align with some $\mathcal{P}_i^o$; then we leave out this $\mathcal{P}_i^o$ as we do not need it anymore. For each o, let $\mathcal{P}^o := \{\mathcal{P}_x^o, \mathcal{P}_y^o\} \cup \{\mathcal{P}_i^o\}_i$ be the set of these relevant sets.

▷ Claim 25. Every $\mathcal{P}_T^{o_S} \in \mathcal{P}^{o_S}$ contains a path disjoint from S.

Proof. Consider a set $\mathcal{P}_T^{o_S}$ with $T \in \{x, y, i\}$ such that the paths in $\mathcal{P}_T^{o_S}$ are $x_T \to y_T$-paths. As above sets are k-representative sets of paths, we must only show that there is any $x_T \to y_T$-path in $G - S$. By assumption there is a $x \to y$ path Q in $G - S$. By Lemma 22 we can find a $q_x \in V(Q)$ such that $\mathsf{dist}(x_T, q_x) \leq 2(\mathsf{cf}(G) - 2)$ and a $x_T \to q_x$-path Q_x in G achieving this distance. By Claim 24 we know that Q_x is disjoint from S and therefore, $Q_x \circ Q[q_x, y]$ is a $q_x \to y$ walk disjoint from S. Let $\hat{Q}_x$ be a $q_x \to y$-path contained in this walk. Another application of Lemma 22 yields a vertex $q_y \in V(\hat{Q})$ with $\mathsf{dist}(q_y, y_T) \leq 2(\mathsf{cf}(G) - 2)$ and a $q_y \to y_T$-path Q_y in G achieving this distance. Again, by Claim 24, Q_y is disjoint from S. Then $\hat{Q}_x[x_T, q_y] \circ Q_y$ contains a $x_T \to y_T$-path as proposed. This completes the proof of Claim 25. ◁

Of course, enumerating all possible tuples of paths would construct too many candidates, as the size of $\mathcal{P}^{o_S}$ can be $\Omega(m)$. Therefore, we want to use a $f(k)$-perfect family of hash functions. This is possible if we can bound the number of intersections with the sets $\mathcal{P}^{o_S}$ by $f(k)$.

▷ Claim 26. The set S intersects at most $2k$ sets of $\mathcal{P}^{o_S}$.

Proof. We show that $s \in S$ can intersect for at most two sets that share an endpoint, thus achieving the claimed size bound. Suppose for contradiction that s intersects two paths Q_1 and Q_2 out of sets in $\mathcal{P}^{o_S}$ that do not share an endpoint. Let each Q_i be a $x_i \to y_i$-path, then we can assume without loss on generality that the order the endpoints appear on R is x_1, y_1, x_2, y_2 and $|R[y_1, x_2] \geq 2\mathsf{cf}(G)^5$ (by the distance of the z_i. On the other hand $R[x_i, y_i]$ and Q_i connect the same endpoints, hence Lemma 22 implies that there is a $t_1 \in V(R[x_1, y_1])$ with $\mathsf{dist}(t_1, s) \leq 2(\mathsf{cf}(G) - 2)$ and a $t_2 \in V(R[x_2, y_2])$ with $\mathsf{dist}(s, t_2) \leq 2(\mathsf{cf}(G) - 2)$ as $s \in Q_1 \cap Q_2$. This implies that $\mathsf{dist}(t_1, t_2) \leq \mathsf{dist}(t_1, s) + \mathsf{dist}(s, t_2) \leq 4(\mathsf{cf}(G) - 2)$. If we now consider $R[t_1, t_2]$, we get $|R[t_1, t_2]| \geq |R[y_1, x_2] \geq 2\mathsf{cf}(G)^5 > (\mathsf{cf}(G) - 1)^2 \cdot \mathsf{dist}(t_1, t_2)$ in contradiction to Lemma 21. This completes the proof of Claim 26. ◁

We can now construct a $2k$-perfect family Ψ^o of hash functions over the universe $\mathcal{P}^o$ for each o. For o_S this family contains an element ψ which gives every set of $\mathcal{P}^{o_S}$ that is intersected by S a different number in $\{1, \ldots, 2k\}$ (by Claim 26). Further, there is a map π_{free} that maps the numbers of $\{1, \ldots, 2k\}$ to a number of $\{1, \ldots, B\}$, such that for every $\mathcal{P} \in \mathcal{P}^{o_S}$ which has a path intersected by S, we have that the $\psi \circ \pi_{\mathsf{free}}(\mathcal{P})$th path of $\mathcal{P}$ is not intersected by S. There is such a path by Claim 25. Denote by $Q_{\psi,\pi_{\mathsf{free}}}(\mathcal{P})$ this path. As we cannot know π_{free} in advance we create a set Π of all possible functions from $\{1,\dots,2k\}$ to $\{1,\dots,B\}$.

We know that for the specific choices of o_S, ψ and π_{free} we get a that the union of paths in $\{Q_{\psi,\pi_{\mathsf{free}}}(\mathcal{P}) | \mathcal{P} \in \mathcal{P}^{o_S}\}$ forms a $x \to y$ walk W in $G - S$. Every $x \to y$-path within W is also disjoint from S. Therefore, the set $\mathcal{P}_{x,y,k}$ created as follows contains a path disjoint from S: For every $o \in \{1,\dots,k+1\}$, every $\psi \in \Psi$ and every $\pi \in \Pi$ consider the $x \to y$-walk $\bigcup_{\mathcal{P} \in \mathcal{P}^o} Q_{\psi,\pi}(\mathcal{P})$ and introduce an arbitrary $x \to y$-path contained in it into $\mathcal{P}_{x,y,k}$.

The size bound on $\mathcal{P}_{x,y,k}$ is proven by multiplying the possibilities for each choice:

$$\underbrace{(k+1)}_{\text{choice of } o} \cdot \underbrace{2^{\mathcal{O}(k)} \log m}_{|\Psi|} \cdot \underbrace{B^{2k}}_{|\Pi|} = \mathsf{cf}(G)^{\mathcal{O}(k^2 \log k)} \log n.$$

The run time follows similarly. ◀

We think that k-representative sets of paths could prove a useful tool in many vertex deletion problems. Together with iterative compression, it allows one to argue about connectivity structure of the old solution. In our case it led to contraction argument strengthening the solution structure (Lemma 3) and a focused guessing of solution vertices (subsection about finding important cluster separators).

There may be other use cases as well. Therefore, we decided to present our result on k-representative sets of paths in a self-contained way. We hope that this helps other researchers and improves our understanding of vertex-deletion problems in directed graphs.

References

1 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995.

2 Saeed Akhoondian Amiri, Ken-Ichi Kawarabayashi, Stephan Kreutzer, and Paul Wollan. The Erdős-Pósa property for directed graphs, 2016. `arXiv:1603.02504`.

3 Jørgen Bang-Jensen and Tilde My Larsen. DAG-width and circumference of digraphs. *J. Graph Theory*, 82(2):194–206, 2016.

4 Paul Bonsma and Daniel Lokshtanov. Feedback vertex set in mixed graphs. In *Proc. WADS 2011*, volume 6844 of *Lecture Notes Comput. Sci.*, pages 122–133. Springer, 2011.

5 Yixin Cao, Jianer Chen, and Yang Liu. On feedback vertex set: new measure and new structures. *Algorithmica*, 73(1):63–86, 2015.

6 Jianer Chen, Fedor V. Fomin, Yang Liu, Songjian Lu, and Yngve Villanger. Improved algorithms for feedback vertex set problems. *J. Comput. System Sci.*, 74(7):1188–1198, 2008.

7 Jianer Chen, Yang Liu, Songjian Lu, Barry O'Sullivan, and Igor Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. *J. ACM*, 55(5):Art. 21, 19, 2008.

8 Rajesh Chitnis, Marek Cygan, Mohammataghi Hajiaghayi, and Dániel Marx. Directed subset feedback vertex set is fixed-parameter tractable. *ACM Trans. Algorithms*, 11(4):Art. 28, 28, 2015.

9 Rajesh Chitnis, MohammadTaghi Hajiaghayi, and Dániel Marx. Fixed-parameter tractability of directed multiway cut parameterized by the size of the cutset. In *Proc. SODA 2012*, pages 1713–1725, 2012.

10 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michał Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time (extended abstract). In *Proc. FOCS 2011*, pages 150–159, 2011.

11 Fedor V. Fomin, Daniel Lokshtanov, and Saket Saurabh. Efficient computation of representative sets with applications in parameterized and exact algorithms. In *Proc. SODA 2014*, pages 142–151, 2014.

12 Alexander Göke, Dániel Marx, and Matthias Mnich. Parameterized algorithms for generalizations of directed feedback vertex set. In *Proc. CIAC 2019*, volume 11485 of *Lecture Notes Comput. Sci.*, pages 249–261, 2019.

13 Jiong Guo, Jens Gramm, Falk Hüffner, Rolf Niedermeier, and Sebastian Wernicke. Compression-based fixed-parameter algorithms for feedback vertex set and edge bipartization. *J. Comput. System Sci.*, 72(8):1386–1396, 2006.

14 Thor Johnson, Neil Robertson, P.D. Seymour, and Robin Thomas. Directed tree-width. *J. Combinat. Theory, Series B*, 82(1):138–154, 2001.

15 Ken-ichi Kawarabayashi and Stephan Kreutzer. The directed grid theorem. In *Proc. STOC 2015*, pages 655–664, 2015.

16 Shiva Kintali. Directed width parameters and circumference of digraphs. *Theoret. Comput. Sci.*, 659:83–87, 2017.

17 Daniel J. Kleitman and Joel Spencer. Families of k-independent sets. *Discrete Math.*, 6:255–262, 1973.

18 Tomasz Kociumaka and Marcin Pilipczuk. Faster deterministic feedback vertex set. *Inform. Process. Lett.*, 114(10):556–560, 2014.

19 Mordechai Lewin. On maximal circuits in directed graphs. *J. Combinatorial Theory, Ser. B*, 18(2):175–179, 1975.

20 Jason Li and Jesper Nederlof. Detecting feedback vertex sets of size k in $O^*(2.7^k)$ time. In *Proc. SODA 2020*, pages 971–989, 2018.

21 Daniel Lokshtanov, M. S. Ramanujan, Saket Saurab, and Meirav Zehavi. Parameterized complexity and approximability of directed odd cycle transversal. In *Proc. SODA 2020*, pages 2181–2200, 2020.

22 Daniel Lokshtanov, M. S. Ramanujan, and Saket Saurabh. When recursion is better than iteration: A linear-time algorithm for acyclicity with few error vertices. In *Proc. SODA 2018*, pages 1916–1933, 2018.

23 B. Monien. How to find long paths efficiently. In *Analysis and design of algorithms for combinatorial problems (Udine, 1982)*, volume 109 of *North-Holland Math. Stud.*, pages 239–254. North-Holland, Amsterdam, 1985.

24 Rian Neogi, M. S. Ramanujan, Saket Saurabh, and Roohani Sharma. On the parameterized complexity of deletion to $\mathcal{H}$-free strong components, 2020. `arXiv:2005.01359`.

25 Venkatesh Raman, Saket Saurabh, and C. R. Subramanian. Faster fixed parameter tractable algorithms for undirected feedback vertex set. In *Proc. ISAAC 2002*, pages 241–248, 2002.

26 Bruce Reed, Neil Robertson, Paul Seymour, and Robin Thomas. Packing directed circuits. *Combinatorica*, 16(4):535–554, 1996.

27 Bruce Reed, Kaleigh Smith, and Adrian Vetta. Finding odd cycle transversals. *Operations Research Letters*, 32(4):299–301, 2004.

28 Neil Robertson and Paul D. Seymour. Graph minors XIII. The disjoint paths problem. *J. Combinat. Theory, Series B*, 63(1):65–110, 1995.

29 Meirav Zehavi. A randomized algorithm for long directed cycle. *Inform. Process. Lett.*, 116(6):419–422, 2016.

On the Central Levels Problem

Petr Gregor

Department of Theoretical Computer Science and Mathematical Logic,
Charles University, Prague, Czech Republic
gregor@ktiml.mff.cuni.cz

Ondřej Mička

Department of Theoretical Computer Science and Mathematical Logic,
Charles University, Prague, Czech Republic
micka@ktiml.mff.cuni.cz

Torsten Mütze

Department of Computer Science, University of Warwick, Coventry, UK
Department of Theoretical Computer Science and Mathematical Logic,
Charles University, Prague, Czech Republic
torsten.mutze@warwick.ac.uk

Abstract

The *central levels problem* asserts that the subgraph of the $(2m+1)$-dimensional hypercube induced by all bitstrings with at least $m+1-\ell$ many 1s and at most $m+\ell$ many 1s, i.e., the vertices in the middle 2ℓ levels, has a Hamilton cycle for any $m \geq 1$ and $1 \leq \ell \leq m+1$. This problem was raised independently by Savage, by Gregor and Škrekovski, and by Shen and Williams, and it is a common generalization of the well-known *middle levels problem*, namely the case $\ell = 1$, and classical binary Gray codes, namely the case $\ell = m+1$. In this paper we present a general constructive solution of the central levels problem. Our results also imply the existence of optimal cycles through any sequence of ℓ consecutive levels in the n-dimensional hypercube for any $n \geq 1$ and $1 \leq \ell \leq n+1$. Moreover, extending an earlier construction by Streib and Trotter, we construct a Hamilton cycle through the n-dimensional hypercube, $n \geq 2$, that contains the symmetric chain decomposition constructed by Greene and Kleitman in the 1970s, and we provide a loopless algorithm for computing the corresponding Gray code.

2012 ACM Subject Classification Mathematics of computing → Combinatorial algorithms; Mathematics of computing → Matchings and factors

Keywords and phrases Gray code, Hamilton cycle, hypercube, middle levels, symmetric chain decomposition

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.60

Category Track A: Algorithms, Complexity and Games

Related Version A full version of the paper is available at `https://arxiv.org/abs/1912.01566`.

Funding This work was supported by the Czech Science Foundation grant GA19-08554S.
Torsten Mütze: was also supported by DFG grant 413902284.

Acknowledgements We thank Jiří Fink for several valuable discussions about symmetric chain decompositions, and for feedback on an earlier draft of this paper.

1 Introduction

The *n-dimensional hypercube*, or *n-cube* for short, is the graph Q_n formed by all $\{0,1\}$-strings of length n, with an edge between any two bitstrings that differ in exactly one bit. This family of graphs has numerous applications in computer science and discrete mathematics, many of which are tied to famous problems and conjectures, such as the sensitivity conjecture of Nisan and Szegedy [29], recently proved by Huang [23]; Erdős and Guys' crossing number problem [8] (see [9]); Füredi's conjecture [13] on equal-size chain partitions (see [41]); Shearer and Kleitman's conjecture [37] on orthogonal symmetric chain decompositions (see [39]); the Ruskey-Savage problem [32] on matching extendability (see [11, 12]), and the conjectures of Norine, and Feder and Subi on edge-antipodal colorings [10, 30], to name just a few.

The focus of this paper are Hamilton cycles in the n-cube and its subgraphs. A Hamilton cycle in a graph is a cycle that visits every vertex exactly once, and in the context of the n-cube, such a cycle is often referred to as a *Gray code.* Gray codes have found applications in signal processing, circuit testing, hashing, data compression, experimental design, and in solving puzzles like the Towers of Hanoi or the Chinese rings; see Savage's survey [35]. They are also fundamental for efficient exhaustive generation algorithms, a topic that is covered in depth in the most recent volume of Knuth's *'The Art of Computer Programming'* [25].

To start with, it is an easy exercise to show that the n-cube has a Hamilton cycle for any $n \geq 2$. One such cycle is given by the classical *binary reflected Gray code* Γ_n [14], defined inductively by $\Gamma_1 := 0, 1$ and $\Gamma_{n+1} := 0\Gamma_n, 1\Gamma_n^R$, where Γ^R denotes the reversal of the sequence Γ, and 0Γ or 1Γ means prefixing all strings in the sequence Γ by 0 or 1, respectively. For instance, this construction gives $\Gamma_2 = 00, 01, 11, 10$ and $\Gamma_3 = 000, 001, 011, 010, 110, 111, 101, 100$. The problem of finding a Hamilton cycle becomes considerably harder when we restrict our attention to subgraphs of the cube induced by a sequence of consecutive levels, where the *k-th level* of Q_n, $0 \leq k \leq n$, is the set of all bitstrings with exactly k many 1s in them. One such instance is the famous *middle levels problem*, raised in the 1980s by Havel [22] and independently by Buck and Wiedemann [4], which asks for a Hamilton cycle in the subgraph of the $(2m + 1)$-cube induced by levels m and $m + 1$. This problem received considerable attention in the literature, and a construction of such a cycle for all $m \geq 1$ was provided only recently by Mütze [27]. A much simpler construction was described subsequently by Gregor, Mütze, and Nummenpalo [19].

1.1 Our results

In this paper we consider the *central levels problem*, a broad generalization of the middle levels problem: Does the subgraph of the $(2m + 1)$-cube induced by the middle 2ℓ levels, i.e., by levels $m + 1 - \ell, \ldots, m + \ell$, have a Hamilton cycle for any $m \geq 1$ and $1 \leq \ell \leq m + 1$? This problem was raised independently by Savage [34], Gregor and Škrekovski [20], and by Shen and Williams [38]. Clearly, the case $\ell = 1$ of the central levels problem is the aforementioned middle levels problem (solved in [27]). Moreover, the case $\ell = 2$ was solved affirmatively in a paper by Gregor, Jäger, Mütze, Sawada, and Wille [16] presented at ICALP 2018. Also, the case $\ell = m + 1$ is established by the binary reflected Gray code Γ_{2m+1}. Furthermore, the case $\ell = m$ was solved by El-Hashash and Hassan [7], and in a more general setting by Locke and Stong [26], and the case $\ell = m - 1$ was settled in [20].

The main contribution of this paper is to solve the central levels problem affirmatively in full generality; see Figure 1 (a)–(d).

▶ **Theorem 1.** *For any $m \geq 1$ and $1 \leq \ell \leq m + 1$, the subgraph of the $(2m + 1)$-cube induced by the middle 2ℓ levels has a Hamilton cycle.*

The most general question in this context is to ask for a Hamilton cycle in Q_n that visits all vertices in *any* sequence of ℓ consecutive levels, i.e., the levels need not be symmetric around the middle, and the dimension n needs not be odd. These graphs are all bipartite, and to circumvent the imbalances that prevent the existence of a Hamilton cycle for general n and ℓ, we have to slightly generalize the notion of Hamilton cycles: Firstly, a *saturating cycle* in a bipartite graph is a cycle that visits all vertices in the smaller partition class (if it has size 1, then a single edge is considered to be a cycle). Secondly, a *tight enumeration* in a (bipartite) subgraph of the cube is a cyclic listing of all its vertices where the total number of bits flipped is exactly the number of vertices plus the difference in size between the two partition classes. Clearly, if both partition classes have the same size, these two notions are equal to a Hamilton cycle. In fact, all cases of this more general problem, except the central levels problem, were solved already in [18], some of them conditional on a "yes" answer to the central levels problem. Combining Theorem 1 with these previous results, we now also obtain an unconditional result for this more general question.

▶ **Corollary 2.** *For any $n \geq 1$ and $1 \leq \ell \leq n+1$, the subgraph of the n-cube induced by any sequence of ℓ consecutive levels has both a saturating cycle and a tight enumeration.*

An essential tool in our proof of Theorem 1 are symmetric chain decompositions. This is a well-known concept from the theory of posets, which we now define specifically for the n-cube using graph-theoretic language. A *symmetric chain* in Q_n is a path $(x_k, x_{k+1}, \ldots, x_{n-k})$ in the n-cube where x_i is from level i for all $k \leq i \leq n-k$, and a *symmetric chain decomposition*, or SCD for short, is a partition of the vertices of Q_n into symmetric chains. It is well-known that the n-cube has an SCD for all $n \geq 1$, and the simplest explicit construction was given by Greene and Kleitman [15] (see Section 2.2 below). Streib and Trotter [40] first investigated the interplay between SCDs and Hamilton cycles in the n-cube, and they described an SCD in Q_n that can be extended to a Hamilton cycle; see Figure 1 (e). Their SCD, however, is different from the aforementioned Greene-Kleitman SCD. In this paper, we extend Streib and Trotter's result as follows; see Figure 1 (f).

▶ **Theorem 3.** *For any $n \geq 2$, the Greene-Kleitman SCD can be extended to a Hamilton cycle in Q_n.*

The Greene-Kleitman SCD has found a large number of applications in the literature, e.g., to construct symmetric Venn diagrams [21, 33], to solve the Littlewood-Offord problem [3, Chap. 4], or to learn monotone Boolean functions [25, Sec. 7.2.1.6] (see also [1, 6, 31, 37, 42]). Knowing that this SCD extends to a Hamilton cycle and that it is a crucial ingredient for solving the general central levels problem adds to this list of interesting properties and applications. Observe also that a Hamilton cycle that extends an SCD has the intriguing property that it minimizes the number of changes of direction from moving up to moving down, or vice versa, between consecutive levels in the cube. For comparison, the monotone paths constructed by Savage and Winkler [36] maximize these changes.

Motivated by these results and by the aforementioned conjecture of Ruskey and Savage [32] that every matching in Q_n extends to a Hamilton cycle, we raise the following conjecture:

▶ **Conjecture 4.** *Every SCD can be extended to a Hamilton cycle in Q_n.*

Although every SCD of Q_n is the union of two matchings, there are matchings in Q_n that do not extend to an SCD; take for example the two edges obtained by starting at the vertices 0^n and 1^n and flipping the same bit. Consequently, an affirmative answer to Conjecture 4 would cover only some cases of the Ruskey-Savage conjecture.

1.2 Efficient algorithms

Our proof of Theorem 1 is constructive and translates directly into an algorithm for computing the Hamilton cycle in time and space that are polynomial in the size of the graph (the middle 2ℓ levels of Q_n, $n := 2m+1$), which is exponential in n. Often, it is desirable to have

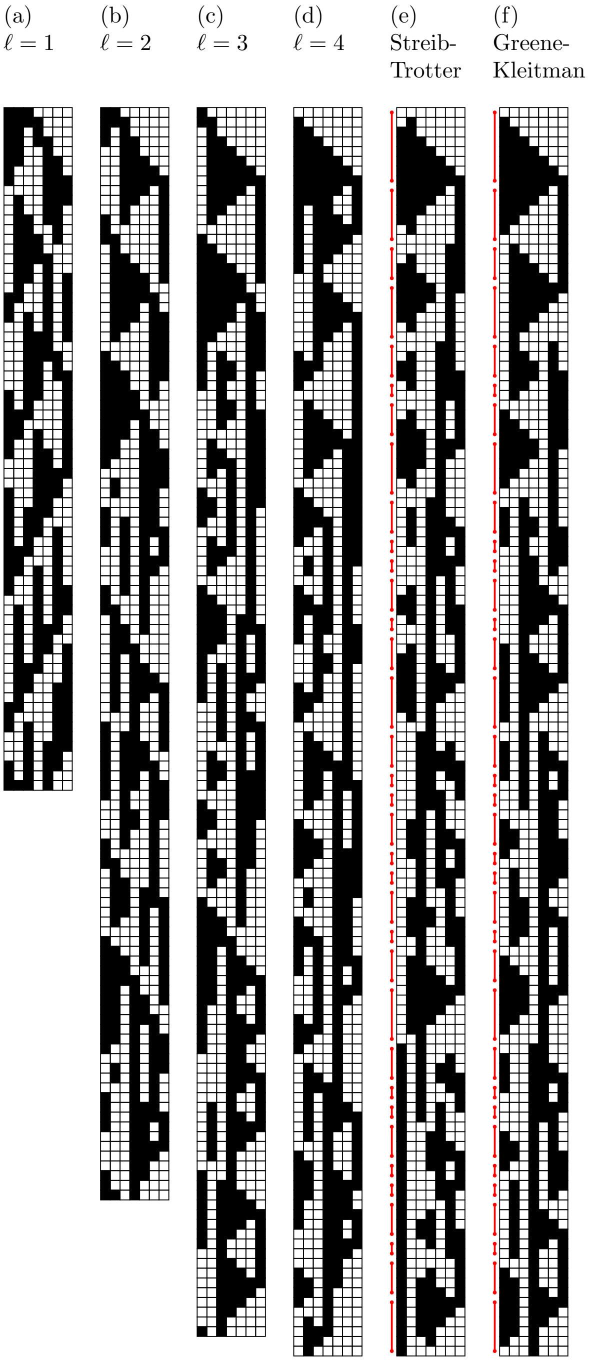


Figure 1 (a)–(d) The Hamilton cycles in $Q_{7,\ell}$ for $\ell = 1, 2, 3, 4$ constructed as in our proof of Theorem 1. (e) The Hamilton cycle in Q_7 containing an SCD obtained from the Streib-Trotter construction, with symmetric chains highlighted on the side. (f) The Hamilton cycle in Q_7 containing the Greene-Kleitman SCD obtained from our proof of Theorem 3. In this figure, 1-bits are drawn as black squares, 0-bits as white squares.

a "local" algorithm that uses only time and space that are polynomial in n. Ideally, one might hope for $\mathcal{O}(n)$ space to store the current bitstring and some additional data structures, and $\mathcal{O}(1)$ time to compute the next bitstring on the cycle. Such algorithms are known for the binary reflected Gray code Γ_n [2], and for the middle levels problem [28], i.e., for the extreme cases $\ell = m+1$ and $\ell = 1$ of the central levels problem. There are fundamental obstacles that prevent us to obtain such a local algorithm from our proof, and it remains a challenging open problem to find such an algorithm. Our Theorem 3 on the other hand, can be translated into a simple algorithm that uses only $\mathcal{O}(n)$ space and $\mathcal{O}(1)$ time in every iteration to compute the next bitstring along the Hamilton cycle. A pseudocode description of this algorithm is available in [17]. We also implemented it in C++, available for download and for demonstration on the Combinatorial Object Server [5].

1.3 Proof ideas

We first describe the ideas for proving Theorem 1. For any $m \geq 1$ we define $n := 2m+1$, and for $1 \leq \ell \leq m+1$ we let $Q_{n,\ell}$ denote the subgraph of Q_n induced by the middle 2ℓ levels. To prove that $Q_{n,\ell}$ has a Hamilton cycle for general m and ℓ, we combine and generalize the tools and techniques developed for the cases $\ell = 1$ and $\ell = 2$ in [19] and [16], respectively. Our proof proceeds in two steps: In a first step, we construct a *cycle factor* in $Q_{n,\ell}$, i.e., a collection of disjoint cycles which together visit all vertices of $Q_{n,\ell}$. In a second step, we use local modifications to join the cycles in the factor to a single Hamilton cycle. Essentially, this technique reduces the Hamiltonicity problem in $Q_{n,\ell}$ to proving that a suitably defined auxiliary graph is connected, which is much easier.

In fact, the predecessor paper [16] already proved the existence of a cycle factor in $Q_{n,\ell}$, but this construction does not seem to yield a factor that would be amenable to analysis. In this paper, we therefore construct another cycle factor in $Q_{n,\ell}$, based on modifying the aforementioned Greene-Kleitman SCD of Q_n by the lexical matchings introduced by Kierstead and Trotter [24]. The resulting cycle factor in $Q_{n,\ell}$ has a rich structure, in particular the number of cycles and their lengths can be described combinatorially.

The simplest way to join two cycles C and C' from this factor to a single cycle is to consider a 4-cycle F that shares exactly one edge with each of the cycles C and C' (the other two edges of F must then go between C and C'), and to take the symmetric difference of the edge sets of $C \cup C'$ and of F, yielding a single cycle $(C \cup C') \bigtriangleup F$ on the same vertex set as $C \cup C'$. We refer to such a cycle F as a *flipping* 4*-cycle*. For example, if we interpret the binary reflected Gray code Γ_n as a cycle in Q_n, we see that $\Gamma_{n+1} = (0\Gamma_n \cup 1\Gamma_n^R) \bigtriangleup F$ where F is the 4-cycle $F = 0^{n+1}, 010^{n-1}, 110^{n-1}, 10^n$. In addition to flipping 4-cycles, we also use flipping 6-cycles, which intersect with the two cycles to be joined in a slightly more complicated way, albeit with the same effect of joining them to a single cycle. The most technical aspect of this part of the proof is to ensure that all flipping cycles used are edge-disjoint, so that the joining operations do not interfere with each other.

To prove Theorem 3, we proceed by induction from dimension n to $n+2$, treating the cases of even and odd n separately. We first specify a particular ordering of all chains of the Greene-Kleitman SCD, and then show that this ordering admits a matching that alternatingly joins the bottom or top vertices of any two consecutive chains in our ordering. In fact, there is a close relation between our proofs of Theorem 1 and 3: The aforementioned construction of a cycle factor in $Q_{n,\ell}$ is particularly nice for $\ell = m+1$, i.e., for the case where we consider the entire cube. Specifically, in this case our cycle factor contains *all* chains from the Greene-Kleitman SCD. These cycles can be joined to a single Hamilton cycle in such a way, so as to give exactly the aforementioned Hamilton cycle constructed for proving Theorem 3.

1.4 Outline of this paper

In Section 2 we discuss the Greene-Kleitman SCD and lexical matchings, and collect some other preliminaries. In Section 3 we describe our construction of a cycle factor in $Q_{n,\ell}$. Due to space constraints, in this extended abstract we are unable to provide the full details of the analysis of this cycle factor, and how to join its cycles to a Hamilton cycle. We rather give an informal high-level sketch of these steps in Section 4. In Section 5 we present our proof of Theorem 3. The omitted proof details, together with the pseudocode description of the corresponding loopless algorithm can be found in [17].

2 Preliminaries

For the reader's convenience, important notations that are introduced in the following and used repeatedly in the paper are summarized in Table 1 at the end of this paper.

2.1 Bitstrings and lattice paths

For any string x and any integer $k \geq 0$, we let x^k denote the concatenation of k copies of x. We often interpret a bitstring x as a path in the integer lattice $\mathbb{Z}^2$ starting at the origin $(0,0)$, where every 0-bit is interpreted as a $\diagdown$-step that changes the current coordinate by $(+1,-1)$ and every 1-bit is interpreted as an $\diagup$-step that changes the current coordinate by $(+1,+1)$; see Figure 2.

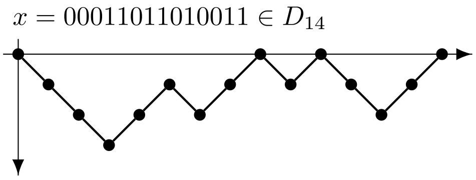


Figure 2 The correspondence between bitstrings (top) and lattice paths (bottom).

Let D_{2k} denote the set of bitstrings with exactly k many 1s and k many 0s, such that in every prefix, the number of 0s is at least as large as the number of 1s. We also define $D := \bigcup_{k\geq 0} D_{2k}$. Note that $D_0 = \{\varepsilon\}$, where ε denotes the empty bitstring. In terms of lattice paths, D corresponds to so-called *Dyck paths* that never move above the line $y = 0$ and end on this line. If a lattice path x contains a substring $u \in D$, then we refer to this substring u as a *valley* in x.

2.2 The Greene-Kleitman SCD

We now describe Greene and Kleitman's [15] construction of an SCD in the n-cube; see Figure 3. For any vertex x of the n-cube, we interpret the 0s in x as opening brackets and the 1s as closing brackets. By matching closest pairs of opening and closing brackets in the natural way, the chain containing x is obtained by flipping the leftmost unmatched 0 to ascend the chain, or the rightmost unmatched 1 to descend the chain, until no more unmatched bits can be flipped. It is easy to see that this indeed yields an SCD of the n-cube for any $n \geq 1$. We always work with this SCD due to Greene and Kleitman, and whenever referring to a chain, we mean a chain from this decomposition.

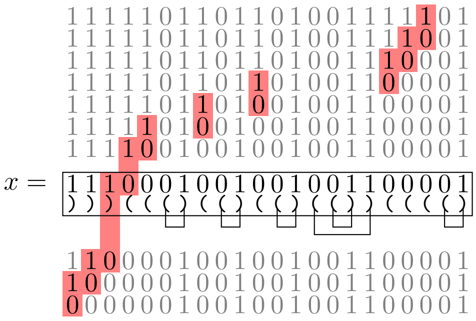


Figure 3 Construction of the Greene-Kleitman SCD containing a bitstring x via parenthesis matching. The highlighted bits are the leftmost unmatched 0 and the rightmost unmatched 1 in each bitstring.

Each chain C of length h in Q_n can be encoded compactly as a string of length n over the alphabet $\{0, 1, *\}$ in the form

$$C = u_0 * u_1 * \cdots * u_{h-1} * u_h, \tag{1}$$

where $u_0, \ldots, u_h \in D$. The symbols $*$ represent unmatched positions, and the vertices along the chain are obtained by replacing the $*$s by 1s followed by 0s in all possible ways; see (2). For example, the chain shown in Figure 3 is $C = *****01*01*010011***01$, so we have $u_0 = u_1 = u_2 = u_3 = u_4 = u_8 = u_9 = \varepsilon$, $u_5 = u_6 = u_{10} = 01$, and $u_7 = 010011$.

We distinguish four types of chains depending on whether u_0 and u_h, i.e., the first and last valleys in (1), are empty or not. These chain types are denoted by $[--]$, $[+-]$, $[++]$, and $[-+]$, where the first symbol is $-$ if $u_0 = \varepsilon$ and $+$ otherwise, and the second symbol is $-$ if $u_h = \varepsilon$ and $+$ otherwise. For example, the chain in Figure 3 is a $[-+]$-chain. We also use the symbol ? in these type specifications if we do not know whether a valley is empty or not. Note that there is no $[--]$-chain in Q_n of length $h = 1$ unless $n = 1$.

Given a chain C of length h as in (1), the ith vertex of C from the bottom is

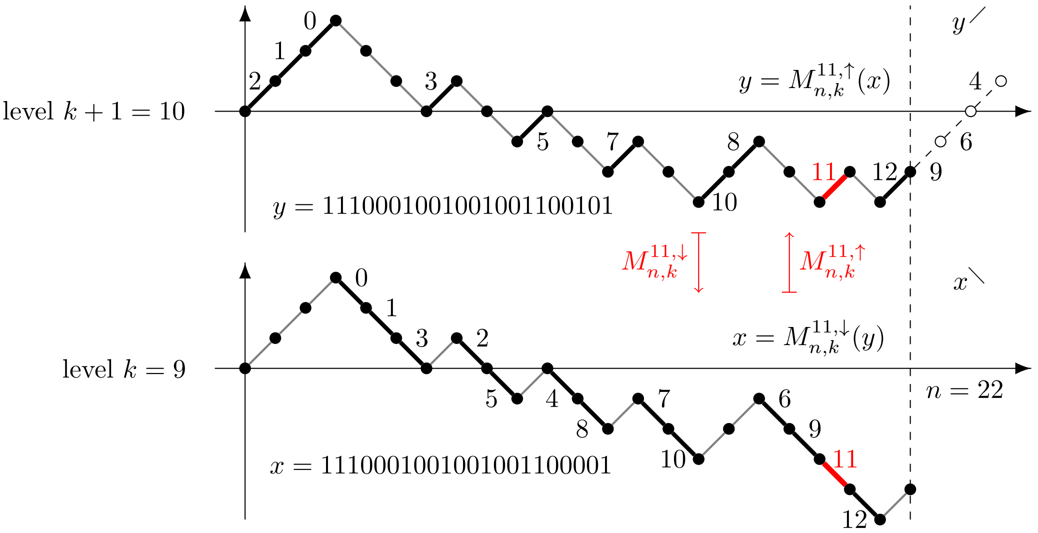
$$x = u_0\, 1 \cdots u_{i-1}\, 1\, u_i\, 0\, u_{i+1} \cdots 0\, u_h \tag{2}$$

where $i = 0, \ldots, h$, and this vertex belongs to level $k = \frac{n-h}{2} + i$. Note that every vertex x of Q_n can be written uniquely in the form (2), and we refer to this as the *chain factorization of* x. For the following arguments, it will be crucial to consider the lattice path representation of x, with the valleys $u_0, \ldots, u_h$ that are separated by i many $\nearrow$-steps, followed by $h - i$ many $\searrow$-steps, i.e., the valley u_i is the highest one on the lattice path.

We use $C_{h,i}$, $0 \leq i \leq h$, to denote the set of the ith vertices in all chains of length h. Moreover, we partition $C_{h,i}$ into two sets $C^-_{h,i}$ and $C^+_{h,i}$, depending on whether the valley u_i in (2) is empty or nonempty, respectively. Clearly, $C^+_{h,h}$ are exactly the top vertices of [?+]-chains of length h and $C^+_{h,0}$ are exactly the bottom vertices of [+?]-chains of length h, and similarly with $-$ instead of $+$. Note that the sets $C_{h,i}$ are empty if n is odd and h is even, or vice versa.

2.3 Lexical matchings

Lexical matchings in Q_n were introduced by Kierstead and Trotter [24], and they are parametrized by some integer $p \in \{0, 1, \ldots, n-1\}$. These matchings are defined as follows; see Figure 4. We interpret a bitstring x as a lattice path, and we let $x^{\searrow}$ denote the lattice



$y^{\diagup}$
$y = M_{n,k}^{11,\uparrow}(x)$
level $k+1 = 10$
0
1
2
3
4
5
6
7
8
9
10
11
12
$y = 1110001001001001100101$
$M_{n,k}^{11,\downarrow}$
$M_{n,k}^{11,\uparrow}$
$x^{\diagdown}$
$x = M_{n,k}^{11,\downarrow}(y)$
level $k = 9$
$n = 22$
$x = 1110001001001001100001$

Figure 4 Definition of p-lexical matchings between levels 9 and 10 of Q_{22}, where steps flipped along the p-lexical edge are marked with p. Between those two levels, the vertex x is incident with p-lexical edges for each $p \in \{0, 1, \ldots, 12\}$, and the vertex y is incident with p-lexical edges for each $p \in \{0, 1, \ldots, 12\} \setminus \{4, 6, 9\}$.

path that is obtained by appending $\diagdown$-steps to x until the resulting path ends at height -1. If x ends at a height less than -1, then $x^{\diagdown} := x$. Similarly, we let $x^{\diagup}$ denote the lattice path obtained by appending $\diagup$-steps to x until the resulting path ends at height $+1$. If x ends at a height more than $+1$, then $x^{\diagup} := x$. We let $L_{n,k}$ denote the set of all vertices on level k of Q_n, and we define a matching by two partial mappings $M_{n,k}^{p,\uparrow}\colon L_{n,k} \to L_{n,k+1}$ and $M_{n,k}^{p,\downarrow}\colon L_{n,k+1} \to L_{n,k}$ defined as follows: For any $x \in L_{n,k}$ we consider the lattice path $x^{\diagdown}$ and scan it row-wise from top to bottom, and from right to left in each row. The partial mapping $M_{n,k}^{p,\uparrow}(x)$ is obtained by flipping the pth $\diagdown$-step encountered in this fashion, where counting starts with $0, 1, \ldots$, if this $\diagdown$-step is part of the subpath x of $x^{\diagdown}$; otherwise x is left unmatched. Similarly, for any $x \in L_{n,k+1}$ we consider the lattice path $x^{\diagup}$ and scan it row-wise from top to bottom, and from left to right in each row. The partial mapping $M_{n,k}^{p,\downarrow}(x)$ is obtained by flipping the pth $\diagup$-step encountered in this fashion if this $\diagup$-step is part of the subpath x of $x^{\diagup}$; otherwise x is left unmatched. It is straightforward to verify that these two partial mappings are inverse to each other, so they indeed define a matching between levels k and $k+1$ of Q_n, called the *p-lexical matching*, which we denote by $M_{n,k}^{p}$. We also define $M_n^p := \bigcup_{0 \le k < n} M_{n,k}^p$, where we omit the index n whenever it is clear from the context. In the following, we will only ever use p-lexical edges for $p = 0, 1, 2$. For instance, it is

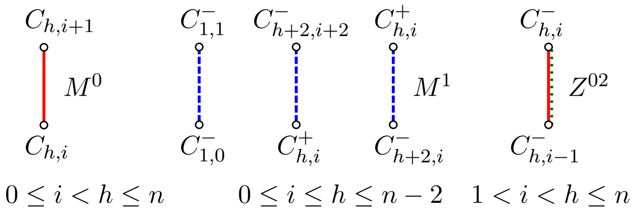


Figure 5 Perfect matchings described by Lemma 5. The $\{0, 1, 2\}$-lexical edges are drawn solid, dashed, and dotted, respectively.

well-known that taking the union of all 0-lexical edges, i.e., the set M^0, yields exactly the Greene-Kleitman SCD [24]. This property is captured by the following lemma, together with several other explicit perfect matchings, consisting of $\{0,1,2\}$-lexical edges between certain sets of vertices from our SCD; see Figure 5.

To state the lemma, for a set M of edges of Q_n and disjoint sets X, Y of vertices, we let $M[X,Y]$ denote the set of all edges of M between X and Y. Moreover, for any vertex $x \in C^-_{h,i}$, $1 < i < h \leq n$, we consider the chain factorization $x = u_0\,1 \cdots u_{i-2}\,1\,u_{i-1}\,1\,0\,u_{i+1} \cdots 0\,u_h$ with $u_0, \ldots, u_h \in D$, and we define a neighbor $z(x)$ on the level below by

$$z(x) := \begin{cases} u_0\,1 \cdots u_{i-2}\,1\,0\,0\,u_{i+1} \cdots 0\,u_h & \text{if } u_{i-1} = \varepsilon, \\ u_0\,1 \cdots u_{i-2}\,1\,0\,v\,0\,w\,1\,0\,u_{i+1} \cdots 0\,u_h & \text{if } u_{i-1} = 0\,v\,1\,w \text{ with } v, w \in D. \end{cases} \tag{3}$$

Note that $(x, z(x))$ is a 0-lexical or 2-lexical edge in the first or second case, respectively.

▶ **Lemma 5.** *For every $n \geq 3$, the following sets of edges $M[X,Y]$ are perfect matchings in Q_n between the vertex sets X and Y.*

(i) $M^0[C_{h,i}, C_{h,i+1}]$ *for every* $0 \leq i < h \leq n$*;*

(ii) $M^1[C^-_{1,0}, C^-_{1,1}]$*,* $M^1[C^+_{h,i}, C^-_{h+2,i+2}]$*, and* $M^1[C^+_{h,i}, C^-_{h+2,i}]$ *for every* $0 \leq i \leq h \leq n-2$*;*

(iii) $Z^{02}[C^-_{h,i-1}, C^-_{h,i}]$ *for every* $1 < i < h \leq n$*, where* $Z^{02} := \{(x, z(x)) \mid x \in C^-_{h,i}\}$*.*

The proof of Lemma 5 can be found in [17].

3 Cycle factor construction

We now construct a cycle factor $\mathcal{C}_{n,\ell}$ in the graph $Q_{n,\ell}$, $n = 2m+1$, i.e., in the subgraph of the n-cube induced by the middle 2ℓ levels. Throughout this section we consider fixed $m \geq 1$ and $2 \leq \ell \leq m+1$. We construct the cycle factor incrementally, starting with chains from the Greene-Kleitman SCD and adding $\{0,1,2\}$-lexical edges between certain sets of vertices, see Figure 6. In the following, when referring to a subgraph given by a set of edges, we mean the subgraph of $Q_{n,\ell}$ induced by those edges. Moreover, we say that a chain is *short* if its length is at most $2\ell-3$, i.e., if it does not span all levels of $Q_{n,\ell}$.

Our construction starts by taking all those short chains, formally

$$X^0 := \bigcup_{0 \leq i < h \leq 2\ell-3} M^0[C_{h,i}, C_{h,i+1}]; \tag{4a}$$

recall Lemma 5 (i). From Lemma 5 (ii) we know that 1-lexical edges perfectly match all bottom vertices of $[-+]$-chains of length 1 with all top vertices of $[+-]$-chains of length 1 along the edges

$$X^1_{\mathtt{m}} := M^1[C^-_{1,0}, C^-_{1,1}]. \tag{4b}$$

Furthermore, for $1 \leq h \leq 2\ell-5$, 1-lexical edges perfectly match all top vertices of $[?+]$-chains of length h with all top vertices of $[?-]$-chains of length $h+2$, and all bottom vertices of $[+?]$-chains of length h with all bottom vertices of $[-?]$-chains of length $h+2$ along the edges

$$X^1_{\mathtt{t}} := \bigcup_{1 \leq h \leq 2\ell-5} M^1[C^+_{h,h}, C^-_{h+2,h+2}], \quad X^1_{\mathtt{b}} := \bigcup_{1 \leq h \leq 2\ell-5} M^1[C^+_{h,0}, C^-_{h+2,0}], \tag{4c}$$

respectively. Note that the only vertices of short chains that have degree 1 in the set

$$X := X^0 \cup X^1_{\mathtt{m}} \cup X^1_{\mathtt{t}} \cup X^1_{\mathtt{b}} \tag{4d}$$

are exactly the vertices of $C^+_{2\ell-3,2\ell-3}$ and $C^+_{2\ell-3,0}$; that is, the top vertices of $[?+]$-chains of length $2\ell-3$ and the bottom vertices of $[+?]$-chains of length $2\ell-3$.

Next, between every pair of consecutive levels of $Q_{n,\ell}$ we take all 0-lexical and 1-lexical edges that are not incident to a degree-2 vertex in X. Specifically, between these pairs of levels we take all 0-lexical edges from chains that are not short and all 1-lexical edges between chains that are not short. In addition, between the top two levels we take all 1-lexical edges between top vertices of [?+]-chains of length $2\ell-3$ and top vertices of [?−]-chains of length $2\ell-1$, and symmetrically, between the bottom two levels we take all 1-lexical edges between bottom vertices of [+?]-chains of length $2\ell-3$ and bottom vertices of [−?]-chains of length $2\ell-1$. Formally, these sets of edges are

$$Y_1 := Y_1' \cup M^1[C^+_{2\ell-3,0}, C^-_{2\ell-1,0}], \quad Y_\ell := Y_\ell' \cup M^1[C^+_{2\ell-3,2\ell-3}, C^-_{2\ell-1,2\ell-1}], \quad Y_k := Y_k' \tag{5a}$$

for $1 < k < \ell$ where

$$Y_k' := \bigcup_{\substack{h\geq 2\ell-1\\ i:=(h-(2\ell-1))/2+2(k-1)}} M^0[C_{h,i}, C_{h,i+1}] \cup M^1[C^+_{h,i}, C^-_{h+2,i+2}] \cup M^1[C^+_{h,i+1}, C^-_{h+2,i+1}] \tag{5b}$$

for $1 \leq k \leq \ell$. Note that Y_1 and Y_ℓ contain all $\{0,1\}$-lexical edges between the bottom two levels or the top two levels of $Q_{n,\ell}$, respectively. We also define

$$Y := \bigcup_{1\leq k\leq \ell} Y_k. \tag{5c}$$

As a consequence of these definitions and Lemma 5 (i) and (ii), the only vertices of $Q_{n,\ell}$ that have degree 1 in the set $X \cup Y$ are exactly the vertices of $C^-_{2\ell-1,i}$ for $1 \leq i \leq 2\ell-2$. We thus add the edges

$$Z := \bigcup_{i=1,3,5,\ldots,2\ell-3} Z^{02}[C^-_{2\ell-1,i}, C^-_{2\ell-1,i+1}] \tag{6}$$

defined in part (iii) of Lemma 5, which makes

$$\mathcal{C}_{n,\ell} := X \cup Y \cup Z \tag{7}$$

a cycle factor in the graph $Q_{n,\ell}$.

3.1 Comparison with previous constructions

Our cycle factor construction generalizes the construction for $\ell = 1$ presented in [19, 27], which simply consisted in taking the union of all 0-lexical and 1-lexical edges between the middle two levels. It also generalizes the construction for $\ell = 2$ presented in [16], which also only used $\{0,1,2\}$-lexical matchings. In fact, all these earlier papers actually used $\{m, m-1, m-2\}$-lexical matching edges, but these are isomorphic to $\{0,1,2\}$-lexical edges by reversing bitstrings. The earlier construction for $\ell = 2$ seemed rather arbitrary at the time, but now nicely fits into the general picture shown in Figure 6[1].

[1] As the picture of this construction resembles a rocket, with the tip on the left and the boosters on the right, one might be tempted to consider this rocket science.

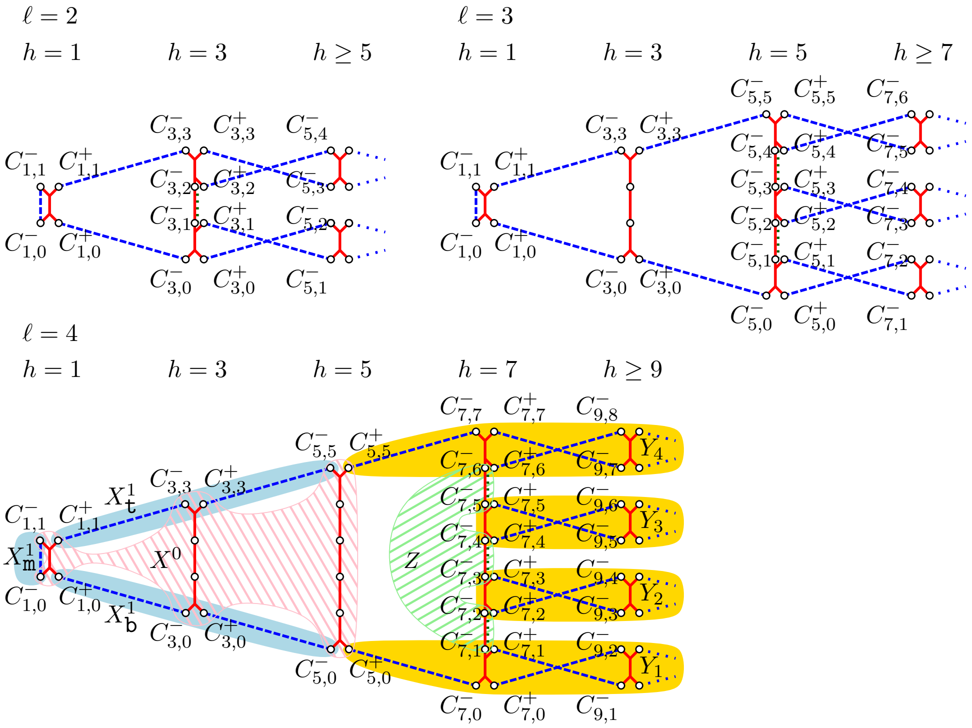

$\ell = 2$
$h = 1$
$h = 3$
$h \geq 5$
$\ell = 3$
$h = 5$
$h \geq 7$
$\ell = 4$
$h = 7$
$h \geq 9$
$C^-_{1,1}$ $C^+_{1,1}$ $C^-_{1,0}$ $C^+_{1,0}$
$C^-_{3,3}$ $C^+_{3,3}$ $C^-_{3,0}$ $C^+_{3,0}$
$C^-_{5,5}$ $C^+_{5,5}$ $C^-_{5,0}$ $C^+_{5,0}$
$C^-_{7,7}$ $C^+_{7,7}$ $C^-_{7,0}$ $C^+_{7,0}$
$C^-_{9,8}$ $C^-_{9,1}$
$X^1_{\mathtt{t}}$ $X^1_{\mathtt{m}}$ $X^1_{\mathtt{b}}$ X^0
Z
Y_1 Y_2 Y_3 Y_4

Figure 6 Illustration of the cycle factor $\mathcal{C}_{n,\ell}$ for $\ell = 2, 3, 4$. Each bullet represents an entire set of vertices, as specified in the figure, lines between them specify perfect matchings between these sets. The $\{0, 1, 2\}$-lexical edges are drawn with solid, dashed, and dotted lines, respectively. In the bottom part, various sets of matching edges are highlighted.

4 Sketch of the remaining proof steps

It turns out that each cycle from the factor $\mathcal{C}_{n,\ell}$ defined in (7) visits vertices from an interval of $2r$ levels, where $2 \leq r \leq \ell$, around the middle. We refer to the number $2r$ as the *range* of the cycle, and we say the cycle is *short* if $2 \leq r < \ell$, and *long* if $r = \ell$. One can show that any short cycle with range r has length $8(r-1)$, contains exactly one $[--]$-chain of length $2r-1$, one $[-+]$- and one $[+-]$-chain of length $2r-3$ each, and one $[++]$-chain of length $2r-5$ (the latter only if $r \geq 3$), i.e., short cycles are in bijection with short $[--]$-chains. For long cycles, on the other hand, we are lacking such a detailed understanding of their structure. However, we are able to identify certain vertices on them, and to describe the operation of moving along one cycle from one such special vertex to the next one in terms of certain *rotation operations* on ordered rooted trees. Consequently, long cycles are obtained as equivalence classes of ordered rooted trees under such rotations.

As outlined in Section 1.3, to join the cycles in our factor to a Hamilton cycle, we explicitly construct flipping 4-cycles and 6-cycles. The 4-cycles are used to join short cycles among each other and to long cycles, in such a way that every short cycle is joined to some long cycle, possibly via other short cycles. For this we exploit the fact that certain pairs of short chains from the Greene-Kleitman SCD are connected by many 4-cycles. Specifically, consider any short chain C of length $h \geq 3$, and any chain C' of length $h-2$ obtained from C by replacing two consecutive *s at positions a and b by 0 and 1, respectively. Using the definition of Greene-Kleitman chains, it is easy to check that C and C' are connected by $h-2$ many 4-cycles, each using a distinct edge of C and C', except the two consecutive edges of C that flip the coordinates a and b. The Greene-Kleitman SCD has an abundance of such pairs of heavily connected pairs of chains, and as our cycle factor contains all these short chains, we can exploit this to join short cycles to each other and to some long cycle in a tree-like fashion, by considering the short cycles by increasing range. Particular care must be taken to ensure that all selected flipping 4-cycles are edge-disjoint from each other, so that they do not interfere with each other in the joining process.

The remaining task is to join long cycles to each other, and for this we use flipping 6-cycles between the topmost two levels of $Q_{n,\ell}$, ensuring that they are edge-disjoint from any flipping 4-cycles, which all live in the levels below. Such a flipping 6-cycle can be used to connect two long cycles with each other, and this operation can again be interpreted in terms of an operation on ordered rooted trees, which we call a *pull operation*. These 6-cycles have been described and used heavily already in the predecessor papers [16, 19], where it was shown that they are all edge-disjoint. To complete the proof of Theorem 1, we show that all long cycles can be joined to each other by flipping 6-cycles, by showing that all equivalence classes of ordered rooted trees under the aforementioned rotations (which correspond to long cycles) can be transformed into each other by pull operations (which correspond to flipping 6-cycles). This step of the proof reduces the Hamiltonicity problem in $Q_{n,\ell}$ to proving that a suitably defined auxiliary graph is connected, which turns out to be much easier.

5 Proof of Theorem 3

In this section, we prove Theorem 3. All lemmas stated below follow from straightforward calculations; see [17] for details. For any chain C, we let $|C|$ denotes its length, i.e., the number of $*$s in C. For any chain C with $|C| \geq 2$, we let $f(C)$ and $\ell(C)$, respectively, denote the chains obtained by replacing the first two $*$s or the last two $*$s in C by 0 and 1. Note that if $|C| \geq 2$, then we have $f(\ell(*C*)) = \ell(f(*C*))$.

Our goal is to order the chains of the Greene-Kleitman SCD in Q_n, $n \geq 2$, so that any consecutive pair of chains is joined at their top end vertices or bottom end vertices alternatingly, with the exception of any two consecutive chains of length 1 that are connected from the bottom end of one of them to the top end of the other, so as to form a Hamilton cycle. We call such an ordering of chains a *cycle ordering*. The following simple but powerful lemma, valid for arbitrary SCDs, shows that the direction in which each chain is traversed along the Hamilton cycle (upwards or downwards) is determined only by the chain length.

▶ **Lemma 6.** *Let Λ_n be a cycle ordering of chains of an SCD in Q_n, $n \geq 2$. In this Hamilton cycle, any two chains C and C' with $|C| \equiv |C'| \pmod 4$ are traversed in the same direction.*

We now define a cycle ordering Λ_n, $n \geq 2$, for the Greene-Kleitman SCD. The corresponding Hamilton cycle is oriented so that it traverses the longest chain $*^n$, which will be the first in the ordering Λ_n, from bottom to top. Our construction works inductively, and the induction step goes from n to $n+2$, with separate rules for even and odd n. The base cases are $n = 0$ and $n = 1$, for which the entire cube consists only of a single vertex and a single edge, respectively, so for these cases the notion of a cycle ordering is not defined.

For even n, we define $\Lambda_0 := \varepsilon$, and for $n \geq 0$ and given $\Lambda_n =: C_1, \ldots, C_N$ we define $\Lambda_{n+2} := \rho(\Lambda_n) = \rho(C_1), \ldots, \rho(C_N)$ with

$$\rho(C) := \begin{cases} \lambda(C) & \text{if } |C| \equiv n \pmod 4, \\ \lambda(C)^R & \text{if } |C| \not\equiv n \pmod 4, \end{cases} \tag{8}$$

and

$$\lambda(C) := \begin{cases} *C*, f(*C*), f(\ell(*C*)), \ell(*C*) & \text{if } |C| \geq 2, \\ *C*, 0C1 & \text{if } |C| = 0. \end{cases} \tag{9}$$

We call the chains of $\lambda(C)$ arising from C the *descendants of* C. This rule replaces each chain C in Λ_n by its descendants $\lambda(C)$, where the order of descendants can be reversed, indicated by the superscript R, depending on the length of C modulo 4.

For odd n, we define $\Lambda_1 := *$, and for $n \geq 1$ and given Λ_n we define $\Lambda_{n+2} := \rho(\Lambda_n)$, where ρ is as before and

$$\lambda(C) := \begin{cases} *C*, \ell(*C*), \ell(f(*C*)), f(*C*) & \text{if } |C| \geq 3, \\ *C*, \ell(*C*), f(*C*) & \text{if } |C| = 1. \end{cases} \tag{10}$$

▶ **Lemma 7.** Λ_n *contains every chain of the Greene-Kleitman SCD exactly once.*

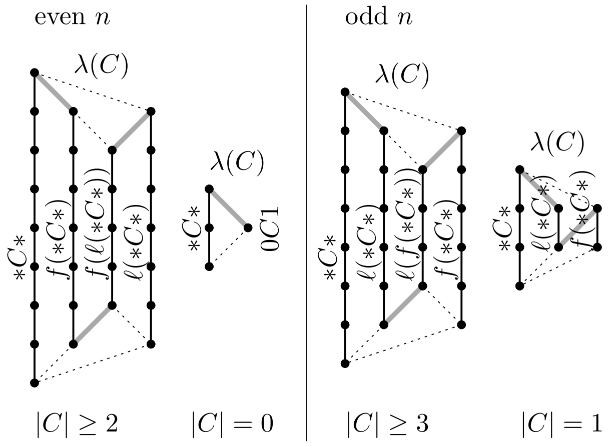


Figure 7 Connections between top and bottom ends of the descendants $\lambda(C)$ of a chain C, as guaranteed by Lemma 8. Bold gray edges are used along the Hamilton cycle. Dotted edges are present but not used.

To complete the proof of Theorem 3, it remains to show that any two consecutive chains in Λ_n can be joined by an edge between their top ends or bottom ends alternatingly. For this we need the following simple lemmas that guarantee these connecting edges.

▶ **Lemma 8.** *For any* $n \geq 2$ *and any chain* C *with* $|C| \geq 2$*, the chains* C *and* $f(C)$*, and the chains* C *and* $\ell(C)$ *are connected both at their top and bottom ends in* Q_n.

All connecting edges between top and bottom ends among the descendants of a chain guaranteed by Lemma 8 are shown in Figure 7. The next two lemmas are illustrated in Figure 8.

▶ **Lemma 9.** *For any* $n \geq 2$ *and any two chains* C, C' *connected at their bottom ends in* Q_n*, we have that* $*C*$ *and* $*C'*$ *are connected at their bottom ends in* Q_{n+2}.

▶ **Lemma 10.** *For any* $n \geq 2$ *and any chain* C *with* $|C| \geq 2$ *in* Q_n*, we have that* $\ell(*C*)$ *and* $\ell(*f(C)*)$ *are connected at their bottom ends in* Q_{n+2}.

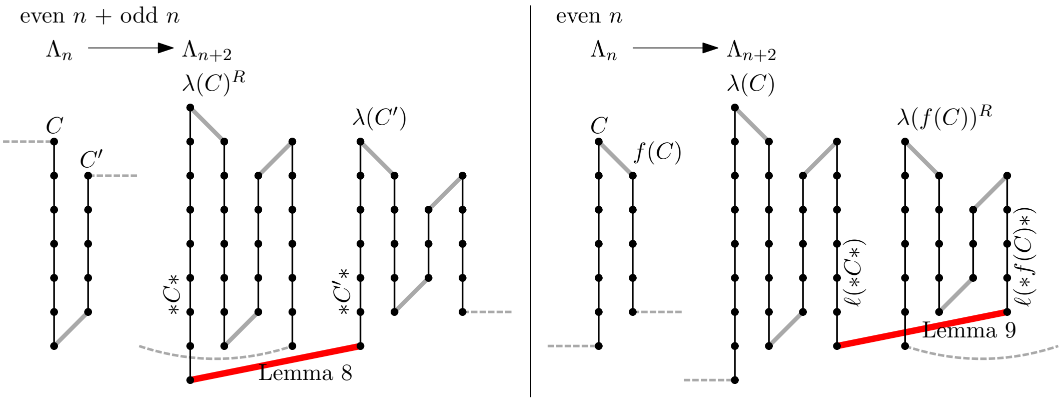


Figure 8 Joining of the descendants of two consecutive chain from Λ_n in the induction step $n \to n+2$, via the thick edges guaranteed by Lemmas 9 and 10. The dashed edges are connections to preceding and subsequent chains on the Hamilton cycle.

Proof of Theorem 3 (even n). We show that Λ_n, $n \geq 2$ even, defined in (9) is a cycle ordering of the Greene-Kleitman chains, by proving that any consecutive pair of chains is connected at their top or bottom ends alternatingly, starting with the first chain $*^n$ of length n that is traversed from bottom to top. We will also establish the following additional property P: For any two consecutive chains C and C' connected at their top ends, we either have $C = f(C')$ or $f(C) = C'$. These invariants can easily be checked for the induction base case $n = 2$, which is given by $\Lambda_2 = **, 01$.

For the induction step consider $n \geq 2$ to be even, and assume that Λ_n is a cycle ordering satisfying property P. By Lemma 8, the descendants $\lambda(C)$ for any chain C from Λ_n can be joined as shown on the left hand side of Figure 7, so we only need to check the connections between the first and last chains among consecutive groups of descendants. Indeed, if C and C' are consecutive in Λ_n and joined at their bottom ends, then C is traversed from top to bottom and C' from bottom to top in the Hamilton cycle; see the left part of Figure 8. Consequently, by Lemma 6, we have $|C| \not\equiv |*^n| = n \pmod 4$ and $|C'| \equiv n \pmod 4$, i.e., by (8) the sequence Λ_{n+2} contains $\lambda(C)^R$ and $\lambda(C')$, and indeed, the bottom vertex of the last chain of $\lambda(C)^R$, namely $*C*$, is connected to the bottom vertex of the first chain of $\lambda(C')$, namely $*C'*$, by Lemma 9. Similarly, if C and C' are consecutive in Λ_n and joined at their top ends, then C is traversed from bottom to top and C' from top to bottom in the Hamilton cycle; see the right part of Figure 8. Consequently, by Lemma 6, we have $|C| \equiv n \pmod 4$ and $|C'| \not\equiv n \pmod 4$, i.e., by (8) the sequence Λ_{n+2} contains $\lambda(C)$ and $\lambda(C')^R$, and indeed, the bottom vertex of the last chain of $\lambda(C)$, namely $\ell(*C*)$, is connected to the bottom vertex of the first chain of $\lambda(C')^R$, namely $\ell(*C'*)$, using that by property P we have either $C = f(C')$ or $f(C) = C'$, so we can invoke Lemma 10. Moreover, property P still holds for Λ_{n+2} by the definition (9) (note that if $|C| = 0$, then we have $0C1 = f(*C*)$). ◀

The proof of Theorem 3 for odd n is very similar. In [17] we provide all details and a loopless algorithm for computing this Gray code. An implementation of this algorithm in C++ is available for download and for demonstration [5].

Table 1 A glossary for notation used in the paper.

Q_n	$n \geq 1$	the n-dimensional hypercube
$Q_{n,\ell}$	$1 \leq \ell \leq m+1$ $n = 2m+1$, $m \geq 1$	the subgraph of Q_n induced by the middle 2ℓ levels
D_{2k}	$k \geq 0$	the set of all Dyck paths (bitstrings) of length $2k$
D		the set of all Dyck paths
C		a chain $C = u_0 * u_1 * \cdots * u_{h-1} * u_h$ of length $h \geq 0$ in the Greene-Kleitman decomposition, $u_i \in D$ for every i
$C_{h,i}$	$0 \leq i \leq h$	the set of the ith vertices in all chains of length h
$C^-_{h,i}$	$0 \leq i \leq h$	as above but only in chains with $u_i = \varepsilon$
$C^+_{h,i}$	$0 \leq i \leq h$	as above but only in chains with $u_i \neq \varepsilon$
$L_{n,k}$	$0 \leq k \leq n$	the set of vertices on level k in Q_n
$M^p_{n,k}$	$0 \leq k < n$, $0 \leq p < n$	the p-lexical matching between $L_{n,k}$ and $L_{n,k+1}$
M^p_n, M^p	$0 \leq p < n$	the set of all p-lexical edges in Q_n
$\|C\|$		the length of a chain C, i.e., $\|C\| = h$ for C as above
$f(C)$	$\|C\| \geq 2$	the chain $f(C) = u_0\,0\,u_1\,1\,u_2 * \cdots * u_h$ for C as above
$l(C)$	$\|C\| \geq 2$	the chain $l(C) = u_0 * \cdots * u_{h-2}\,0\,u_{h-1}\,1\,u_h$ for C as above
$\lambda(C)$		a sequence of descendant chains for a chain C, see (9), (10)

References

1 M. Aigner. Lexicographic matching in Boolean algebras. *J. Combin. Theory Ser. B*, 14:187–194, 1973.

2 J. Bitner, G. Ehrlich, and E. Reingold. Efficient generation of the binary reflected Gray code and its applications. *Comm. ACM*, 19(9):517–521, 1976.

3 B. Bollobás. *Combinatorics: set systems, hypergraphs, families of vectors and combinatorial probability.* Cambridge University Press, Cambridge, 1986.

4 M. Buck and D. Wiedemann. Gray codes with restricted density. *Discrete Math.*, 48(2-3):163–171, 1984. doi:10.1016/0012-365X(84)90179-1.

5 The Combinatorial Object Server: http://www.combos.org/chains.

6 N. de Bruijn, C. van Ebbenhorst Tengbergen, and D. Kruyswijk. On the set of divisors of a number. *Nieuw Arch. Wiskunde (2)*, 23:191–193, 1951.

7 M. El-Hashash and A. Hassan. On the Hamiltonicity of two subgraphs of the hypercube. In *Proceedings of the Thirty-second Southeastern International Conference on Combinatorics, Graph Theory and Computing (Baton Rouge, LA, 2001)*, volume 148, pages 7–32, 2001.

8 P. Erdős and R. K. Guy. Crossing number problems. *Amer. Math. Monthly*, 80:52–58, 1973. doi:10.2307/2319261.

9 L. Faria, C. M. H. de Figueiredo, O. Sýkora, and I. Vrt'o. An improved upper bound on the crossing number of the hypercube. *J. Graph Theory*, 59(2):145–161, 2008. doi:10.1002/jgt.20330.

10 T. Feder and C. Subi. On hypercube labellings and antipodal monochromatic paths. *Discrete Appl. Math.*, 161(10-11):1421–1426, 2013. doi:10.1016/j.dam.2012.12.025.

11 J. Fink. Perfect matchings extend to Hamilton cycles in hypercubes. *J. Combin. Theory Ser. B*, 97(6):1074–1076, 2007. doi:10.1016/j.jctb.2007.02.007.

12 J. Fink. Matchings extend into 2-factors in hypercubes. *Combinatorica*, 39(1):77–84, 2019. doi:10.1007/s00493-017-3731-8.

13 Z. Füredi. Problem session. In *Kombinatorik geordneter Mengen*, Oberwolfach, BRD, 1985.

14 F. Gray. Pulse code communication, 1953. March 17, 1953 (filed Nov. 1947). U.S. Patent 2,632,058.

15 C. Greene and D. J. Kleitman. Strong versions of Sperner's theorem. *J. Combin. Theory Ser. A*, 20(1):80–88, 1976.

16 P. Gregor, S. Jäger, T. Mütze, J. Sawada, and K. Wille. Gray codes and symmetric chains. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, pages 66:1–66:14, 2018. doi:10.4230/LIPIcs.ICALP.2018.66.

17 P. Gregor, O. Mička, and T. Mütze. On the central levels problem. arXiv:1912.01566. Full preprint version of the present article, 2019.

18 P. Gregor and T. Mütze. Trimming and gluing Gray codes. *Theoret. Comput. Sci.*, 714:74–95, 2018. doi:10.1016/j.tcs.2017.12.003.

19 P. Gregor, T. Mütze, and J. Nummenpalo. A short proof of the middle levels theorem. *Discrete Analysis*, 2018:8:12 pp., 2018.

20 P. Gregor and R. Škrekovski. On generalized middle-level problem. *Inform. Sci.*, 180(12):2448–2457, 2010. doi:10.1016/j.ins.2010.02.009.

21 J. Griggs, C. E. Killian, and C. D. Savage. Venn diagrams and symmetric chain decompositions in the Boolean lattice. *Electron. J. Combin.*, 11(1):Paper 2, 30 pp., 2004. URL: http://www.combinatorics.org/Volume_11/Abstracts/v11i1r2.html.

22 I. Havel. Semipaths in directed cubes. In *Graphs and other combinatorial topics (Prague, 1982)*, volume 59 of *Teubner-Texte Math.*, pages 101–108. Teubner, Leipzig, 1983.

23 H. Huang. Induced subgraphs of hypercubes and a proof of the sensitivity conjecture. *Ann. of Math. (2)*, 190(3):949–955, 2019. doi:10.4007/annals.2019.190.3.6.

24 H. A. Kierstead and W. T. Trotter. Explicit matchings in the middle levels of the Boolean lattice. *Order*, 5(2):163–171, 1988. doi:10.1007/BF00337621.

25 D. E. Knuth. *The Art of Computer Programming. Vol. 4A. Combinatorial Algorithms. Part 1.* Addison-Wesley, Upper Saddle River, NJ, 2011.

26 S. Locke and R. Stong. Problem 10892: Spanning cycles in hypercubes. *Amer. Math. Monthly*, 110:440–441, 2003.

27 T. Mütze. Proof of the middle levels conjecture. *Proc. Lond. Math. Soc.*, 112(4):677–713, 2016. doi:10.1112/plms/pdw004.

28 T. Mütze and J. Nummenpalo. A constant-time algorithm for middle levels Gray codes. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2238–2253. SIAM, Philadelphia, PA, 2017. doi:10.1137/1.9781611974782.147.

29 N. Nisan and M. Szegedy. On the degree of Boolean functions as real polynomials. *Comput. Complexity*, 4(4):301–313, 1994. Special issue on circuit complexity (Barbados, 1992). doi:10.1007/BF01263419.

30 S. Norine. Edge-antipodal colorings of cubes. The Open Problem Garden. Available at http://www.openproblemgarden.org/op/edge_antipodal_colorings_of_cubes, 2008.

31 O. Pikhurko. On edge decompositions of posets. *Order*, 16(3):231–244 (2000), 1999. doi:10.1023/A:1006419611661.

32 F. Ruskey and C. Savage. Hamilton cycles that extend transposition matchings in Cayley graphs of S_n. *SIAM J. Discrete Math.*, 6(1):152–166, 1993. doi:10.1137/0406012.

33 F. Ruskey, C. D. Savage, and S. Wagon. The search for simple symmetric Venn diagrams. *Notices Amer. Math. Soc.*, 53(11):1304–1312, 2006.

34 C. D. Savage. Long cycles in the middle two levels of the Boolean lattice. *Ars Combin.*, 35(A):97–108, 1993.

35 C. D. Savage. A survey of combinatorial Gray codes. *SIAM Rev.*, 39(4):605–629, 1997. doi:10.1137/S0036144595295272.

36 C. D. Savage and P. Winkler. Monotone Gray codes and the middle levels problem. *J. Combin. Theory Ser. A*, 70(2):230–248, 1995. doi:10.1016/0097-3165(95)90091-8.

37 J. Shearer and D. J. Kleitman. Probabilities of independent choices being ordered. *Stud. Appl. Math.*, 60(3):271–276, 1979. doi:10.1002/sapm1979603271.

38 X. S. Shen and A. Williams. A middle levels conjecture for multiset permutations with uniform-frequency. Williams College Technical Report CSTR-201901. Available at `http://tmuetze.de/papers/multiset.pdf`, 2019.

39 H. Spink. Orthogonal symmetric chain decompositions of hypercubes. *SIAM J. Discrete Math.*, 33(2):910–932, 2019. `doi:10.1137/18M1187179`.

40 N. Streib and W. T. Trotter. Hamiltonian cycles and symmetric chains in Boolean lattices. *Graphs Combin.*, 30(6):1565–1586, 2014. `doi:10.1007/s00373-013-1350-8`.

41 I. Tomon. On a conjecture of Füredi. *European J. Combin.*, 49:1–12, 2015. `doi:10.1016/j.ejc.2015.02.026`.

42 D. E. White and S. G. Williamson. Recursive matching algorithms and linear orders on the subset lattice. *J. Combin. Theory Ser. A*, 23(2):117–127, 1977.

Linearly Representable Submodular Functions: An Algebraic Algorithm for Minimization

Rohit Gurjar
Indian Institute of Technology Bombay, India
http://www.cse.iitb.ac.in/~rgurjar
rohitgurjar0@gmail.com

Rajat Rathi
Indian Institute of Technology Bombay, India
rajatrathidgr81@gmail.com

Abstract

A set function $f\colon 2^E \to \mathbb{R}$ on the subsets of a set E is called submodular if it satisfies a natural diminishing returns property: for any $S \subseteq E$ and $x, y \notin S$, we have $f(S \cup \{x, y\}) - f(S \cup \{y\}) \leq f(S \cup \{x\}) - f(S)$. Submodular minimization problem asks for finding the minimum value a given submodular function takes. We give an algebraic algorithm for this problem for a special class of submodular functions that are "linearly representable". It is known that every submodular function f can be decomposed into a sum of two monotone submodular functions, i.e., there exist two non-decreasing submodular functions f_1, f_2 such that $f(S) = f_1(S) + f_2(E \setminus S)$ for each $S \subseteq E$. Our class consists of those submodular functions f, for which each of f_1 and f_2 is a sum of k rank functions on families of subspaces of $\mathbb{F}^n$, for some field $\mathbb{F}$.

Our algebraic algorithm for this class of functions can be parallelized, and thus, puts the problem of finding the minimizing set in the complexity class randomized NC. Further, we derandomize our algorithm so that it needs only $O(\log^2(kn|E|))$ many random bits.

We also give reductions from two combinatorial optimization problems to linearly representable submodular minimization, and thus, get such parallel algorithms for these problems. These problems are (i) covering a directed graph by k a-arborescences and (ii) packing k branchings with given root sets in a directed graph.

2012 ACM Subject Classification Mathematics of computing → Combinatorial optimization; Theory of computation → Parallel algorithms; Theory of computation → Pseudorandomness and derandomization

Keywords and phrases Submodular Minimization, Parallel Algorithms, Derandomization, Algebraic Algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.61

Category Track A: Algorithms, Complexity and Games

Acknowledgements We would like to thank Siddharth Barman for helpful comments.

1 Introduction

Submodular functions have been studied in a wide variety of contexts like combinatorics, electrical networks, game theory, and machine learning. For a set E, a submodular function is a set function $f\colon 2^E \to \mathbb{R}$ that satisfies a natural diminishing returns property: for any $T \subseteq S \subseteq E$ and $x \in E \setminus S$, we have

$$f(S \cup \{x\}) - f(S) \leq f(T \cup \{x\}) - f(T).$$

That is, the marginal value of an element with respect to a set decreases as the set grows. Another equivalent way to describe submodularity is: for any $S, T \subseteq E$, we have $f(S)+f(T) \geq f(S \cup T) + f(S \cap T)$. Submodular functions appear in a diverse set of areas. To give a few

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 61; pp. 61:1–61:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

examples, a linear function, maximum number in the given subset, the rank function on a set of vectors or subspaces, the cut function on the set of vertices of a graph, entropy on a set of random variables, the coverage function on a collection of subsets, are all submodular.

There are various natural optimization problems involving submodular functions: *Submodular Minimization* asks for the set that minimizes a given submodular function among all subsets of the ground set. Similarly, *Submodular Maximization* asks for the maximizing set. Note that the these two questions are interesting only for non-monotone submodular functions like graph cut. There are also constrained versions of minimization and maximization, for example, optimizing a submodular function over subsets of a given size. The submodular function might be given by an explicit representation, for example, a given graph can represent the corresponding cut function. However, not all submodular functions are succinctly representable, as their number grows as doubly-exponential in the ground set size [27]. In the most general framework, the function is given via a value oracle, i.e., given any subset $S \subseteq E$, the oracle will provide the function value on S.

Submodular minimization is in some sense a discrete version of convex minimization [20], and thus, admits polynomial time algorithms (even with just the value oracle). The initial algorithms for it were based on the ellipsoid algorithm [11, 12], but later on combinatorial algorithms were also obtained [6, 15, 28]. Submodular maximization, on the other hand, is known to be hard: Max-cut [16] and maximum facility location [5] are instances of submodular maximization which are NP-hard. Moreover, in the oracle model, there is an exponential lower bound known on the number of queries required [9].

When we put cardinality constraints, then in fact, both minimization and maximization problems become hard even for monotone submodular functions. Examples of such maximization problems that are NP-hard include max-k-cover (set-cover, which is NP-hard [16], reduces to it) and sparse approximation [7] (for a set E of vectors and fixed vector v, the function $f_v(S) = \|proj_{span(S)}(v)\|_2$ for $S \subseteq E$ is submodular). Similarly, min-k-vertex-cover is an example of an NP-hard minimization question (see [13]). Moreover, in the oracle model, cardinality constrained submodular minimization has a sub-exponential lower bound on the number of queries (follows from [31]).

Parallel complexity of submodular minimization. In this paper, we investigate the question of parallel complexity of unconstrained submodular minimization. In the oracle model, the parallel complexity question can be phrased as follows: if one is allowed to simultaneously make polynomially many function value queries in one round, how many rounds are required to find the minimum value (and the minimizing set). The number of rounds required is also known the *adaptivity* (see [1]). To the best of our knowledge, the best upper bound on the adaptivity of submodular minimization is $O(n \log(nM))$ [19], where M is the largest absolute value the function takes (they use a separation oracle that can be implemented with one round of n parallel queries to the value oracle). While on the lower bound side, there is a known impossibility result for one round [2], and $\tilde{\Omega}(n^2/k^5)$ query lower bound for k rounds [1]. Very recently, it was shown that there are no adaptive algorithms that run in $o(\frac{\log n}{\log \log n})$ rounds with poly(n) queries per round [3]. In particular, it is not clear whether the adaptivity of submodular minimization can be sublinear.

On the other hand, if we consider explicitly given submodular functions, there are instances for which the minimization problem admits parallel algorithms. Such special cases of submodular minimization include (s, t)-min-cut (small capacities), maximum bipartite matching[1], and its generalization linear matroid intersection. These problems have algebraic algorithms, which just involve randomized reductions to matrix rank computation and thus, fall into the class randomized NC (RNC) [17, 21, 23, 24]. In recent years, these algorithms have also been partially derandomized [10, 14], i.e., they can work with only $O(\log^2 n)$ random bits. A natural question arises: what is the most general class of submodular functions for which such algebraic algorithms can work. One would expect such algebraic algorithms for submodular functions that are linear algebraic in some sense. Towards this, we define a class of *linearly representable* submodular functions.

Linearly representable (LR) submodular functions. Suppose we have a family of subspaces $\mathcal{V} = \{V_e \subseteq \mathbb{F}^n\}_{e\in E}$, for some field F. Recall that the rank function of the family $\mathcal{V}$ given by $r(S) = \dim(\sum_{e\in S} V_e)$ for $S \subseteq E$ is submodular. The rank function is non-decreasing, and hence, the minimization question for it is not interesting. One can try to consider the difference of two rank functions, but that is not submodular. Interestingly, there is a way to construct a non-increasing submodular function from a non-decreasing submodular function: if a function $f(S)$ is submodular then so is $g(S) := f(E\setminus S)$. So, if we have two non-decreasing submodular functions $f_1(S)$ and $f_2(S)$, we can get a non-monotone submodular function by considering $f_1(S)+f_2(E\setminus S)$ (since the sum of two submodular functions is also submodular). In fact, using this way one can arrive at any submodular function. It is known (see [6]) that every submodular function f can be decomposed into two non-decreasing submodular functions f_1 and f_2 such that for any $S \subseteq E$, $f(S) = f_1(S) + f_2(E \setminus S)$.

Our contributions

The above facts motivate us to define the following natural class of linear algebraic submodular functions that are not necessarily monotone. For a ground set E, let $\bar{S} := E \setminus S$.

▶ **Definition 1** (Linearly representable (LR) submodular functions). *We call a submodular function $f: 2^E \to \mathbb{Z}$ linearly representable (LR) by k families of subspaces $\mathcal{V}_j = \{V_{j,e} \subseteq \mathbb{F}^n\}_{e\in E}$ for $1 \le j \le k$ and a number $\ell \le k$ if*

$$f(S) = \sum_{j=1}^{\ell} r_j(S) + \sum_{j=\ell+1}^{k} r_j(\bar{S}),$$

where $r_j: 2^E \to \mathbb{Z}$ is the rank function for family $\mathcal{V}_j$.

This class includes many interesting submodular functions like directed graph cut, hypergraph cut, coverage function, integral linear function (up to additive normalization), and more interestingly, any combination of them in the above form. Our main results are a randomized algebraic algorithm for minimizing LR submodular functions that puts the minimization problem in RNC, and an almost complete derandomization of the algorithm; see Section 3.

▶ **Theorem 2** (Linearly representable submodular minimization). *Given an LR submodular function $f: 2^E \to \mathbb{Z}$ via families of subspaces $\mathcal{V}_j = \{V_{j,e} \subseteq \mathbb{F}^n\}_{e\in E}$ for $1 \le j \le k$ and a number $\ell \le k$ (Definition 1), we can find a set minimizing $f(S)$ in RNC. Further, the randomized algorithm can be almost completely derandomized so that it uses only $O(\log^2(kn|E|))$ random bits.*

[1] For a bipartite graph $G(V_1 \cup V_2, E)$, the maximum matching size is equal to $|V_1| + \min_{S\subseteq V_1}(|N(S)| - |S|)$ (Hall's theorem), where $N(S) \subseteq V_2$ is the set of neighbor of S. The function $|N(S)| - |S|$ is submodular.

Another way to put the derandomization result is that minimizing an LR submodular function is in quasi-NC (see [10, 14] for the details of class quasi-NC). Our results also imply a randomized parallel algorithm and its almost deterministic version for a problem called linear polymatroid intersection, generalizing the corresponding results for linear matroid intersection [24, 14].

In the *linear polymatroid intersection* problem, we are given two families of subspaces, $\mathcal{V}_j = \{V_{j,e} \subseteq \mathbb{F}^n\}_{e \in E}$ for $j = 1, 2$, with their rank functions r_1, r_2, respectively. And the goal is to find

$$\max\{\sum_{e \in E} x_e \mid x_e \geq 0 \; \forall e \in E, \text{ and } \sum_{e \in S} x_e \leq r_1(S), \sum_{e \in S} x_e \leq r_2(S) \text{ for each } S \subseteq E\}.$$

Min-max relation. It is known that this maximum value is equal to $\min_{S \subseteq E} r_1(S) + r_2(\bar{S})$ (see [29, Corollary 46.1c]). Thus, the maximization problem is captured by LR submodular minimization.

▶ **Corollary 3.** *Linear polymatroid intersection has a randomized* NC *algorithm that uses only $O(\log^2(n|E|))$ random bits.*

Linear matroid intersection is the special case of linear polymatroid intersection when each of the above subspaces $V_{j,e}$ is of dimension 1. Thus, the above min-max relation with a LR submodular function also holds for linear matroid intersection. Our parallel algorithm has a crucial difference from the known parallel algorithms [24, 14] for linear matroid intersection. They give the minimum value of the corresponding LR submodular function, but they do not lead to a minimizing set, while our algorithm also finds a minimizing set.

Further applications

As mentioned above, LR submodular minimization captures linear matroid intersection and thus, several other combinatorial optimization problems that reduce to linear matroid intersection, like bipartite matching, packing spanning trees, finding arborescences (see [29]), packing a-arborescences [29, Theorem 53.10], and hypergraph min-cut [18]. Since linear matroid intersection already has parallel algorithms [24, 14], so do these problems.

However, there are also combinatorial problems that reduce to submodular minimization but are not captured by linear matroid intersection. We show that two such problems, in fact, reduce to LR submodular minimization (see Section 4.2 for definitions and reductions).

- **Covering by a-arborescences.** For a given directed graph and a number k, decide whether the edge set is covered by k a-arborescences.
- **Packing of branchings.** For a given directed graph and given subsets $R_1, R_2, \ldots, R_k$ of vertices, decide whether there exist k edge-disjoint branchings that are rooted at $R_1, R_2, \ldots, R_k$, respectively.

To the best of our knowledge, there is no straightforward reduction known from these problems to linear matroid intersection. Using Theorem 2, we get the following.

▶ **Theorem 4.** *Covering by a-arborescences and packing of branchings can be solved in* RNC *using only $O(\log^2 n)$ random bits, n being the size of the input graph.*

Variants

Furthermore, we list out two problems, one of which is an extension of LR submodular minimization and the other one is equivalent to it (see Section 4.1).

1. **Minimization with containment constraints**: There is a variant of submodular minimization that appears frequently in combinatorial optimization. Given two subsets $S_0 \subseteq S_1 \subseteq E$, the goal is to minimize the given submodular function $f(S)$ subject to $S_0 \subseteq S \subseteq S_1$. We can extend our algorithm to LR submodular minimization with this kind of constraints. This is a generalization of the minimum $(S_0, \bar{S}_1)$ cut problem in a graph $G(V,E)$ with two given disjoint subsets $S_0, \bar{S}_1 \subseteq V$.
2. **Separation oracle for a linear polymatroid**: Given a family $\mathcal{V} = \{V_e \subseteq \mathbb{F}^n\}_{e \in E}$ of subspaces with its rank function $r \colon 2^E \to \mathbb{Z}$, the corresponding polymatroid is a polytope $P_r \subseteq \mathbb{R}^E$ defined as

$$P_r = \{x \in \mathbb{R}^E \mid x \geq 0, \sum_{e \in S} x_e \leq r(S) \ \forall S \subseteq E\}.$$

Given a rational point $\beta \in \mathbb{R}^E$, one needs to decide if β lies in P_r, and if not then find a violating constraint from the above set. We reduce this problem to LR submodular minimization assuming the coordinates in β are rational numbers with a polynomially bounded common denominator.

2 Preliminaries

Complexity Class NC. NC represents the class of problems that can be solved by polynomially many parallel processors in poly-logartihmic time. RNC, i.e., randomized NC, represents problems that can be solved with the same resources, but with the use of randomness.

2.1 Submodular functions

For a set E, a *submodular* function is a set function $f \colon 2^E \to \mathbb{R}$ for any $S, T \subseteq E$, we have $f(S) + f(T) \geq f(S \cup T) + f(S \cap T)$. If f satisfies this with equality then it is called *modular*. There are various properties of submodular functions that are useful for us and are easy to verify. For $S \subseteq E$, $\bar{S}$ will denote $E \setminus S$.

- If f_1 and f_2 are submodular then so is $f_1 + f_2$.
- If f is submodular then so is $g(S) = f(\bar{S})$.

For a set E and a family of subspaces $\mathcal{V} = \{V_e \subseteq \mathbb{F}^n\}_{e \in E}$, its corresponding rank function $r \colon 2^E \to \mathbb{Z}$ is defined to be as $r(S) = \dim(\sum_{e \in S} V_e)$ for $S \subseteq E$. It is not hard to verify that the rank function is submodular for any family of subspaces.

2.2 Polynomial identity testing

To design our randomized algorithm, we will need a fundamental result about zeros of polynomials, which says that if a polynomial is nonzero then at a random point, its evaluation is nonzero with high probability (see, for example, [25, 30, 33, 8]).

▶ **Lemma 5.** *Let there be an n-variate degree-d nonzero polynomial $P(z_1, z_2, \ldots, z_n)$. If for each $1 \leq i \leq n$, the variable x_i is substituted with a random number R_i chosen uniformly and independently from a set of size D then*

$$\Pr\{P(R_1, R_2, \ldots, R_n) = 0\} \leq d/D.$$

Note that D should be at least as large as the degree. Throughout the paper, we will assume that the underlying field is large enough. This lemma gives a simple algorithm to test if a given polynomial is nonzero: just evaluate it at a random point and output nonzero if and only if the evaluation is nonzero. To derandomize our submodular minimization algorithm, we will need to derandomize this test of nonzeroness of a polynomial. For general polynomials, there is no non-trivial derandomization known. However, we will need the derandomization result only for polynomials that have certain special structure.

Let U be a square matrix whose entries are all linear polynomials (degree-1). The polynomials of interest in our setting will be determinants of such symbolic matrices, where any particular variable appears in at most one column of the matrix. For this class of polynomials, an almost complete derandomization of nonzero testing is known. A way to do deterministic nonzero testing is to obtain a small hitting-set – a set of points such that any nonzero polynomial in the class of our interest gives a nonzero evaluation on at least one of the points.

The work of [14] gave a quasi-polynomial size hitting-set for polynomials of the following form which subsume the above case: $P(\mathbf{z}) = \det(\sum_{i=1}^{n} A_i z_i)$, where each A_i is a rank-1 matrix. The result of [14] can be easily modified to generate a slightly stronger notion of a hitting-set, though it is not explicitly stated there.

▶ **Lemma 6** ([14]). *There is an NC-computable hitting-set generator, that is, a function $h\colon \{0,1\}^t \to \mathbb{F}^n$, $t = O(\log^2 mn)$, with the following property: for any nonzero polynomial $P(\mathbf{z}) = \det(\sum_{i=1}^{n} A_i z_i)$ where each $A_i \in \mathbb{F}^{m\times m}$ is a rank-1 matrix*

$$\Pr_{R\in\{0,1\}^t}\{P(h(R)) = 0\} \le 1/poly(mn).$$

3 Parallel algorithm for linearly representable submodular minimization

Our first step towards the parallel algorithm is to consider one of the special cases of LR submodular minimization. We give an algebraic algorithm for this special case. The algorithm is essentially a reduction to basic linear algebraic operations like computing rank and inverse of a matrix, which are doable in NC. The reduction is randomized and thus, puts the special case in RNC. Finally, we reduce the LR submodular minimization problem to this special case. We start with describing the special case and a solution for it.

3.1 LR submodular minimization for a special case

The special case we first consider is when the submodular function is the difference of a rank function and a positive linear function. Let $\mathcal{V} = \{V_e \subseteq \mathbb{F}^n\}_{e\in E}$ be a family of subspaces for a ground set E and a field $\mathbb{F}$, and $r\colon 2^E \to \mathbb{Z}$ be the corresponding rank function. Let $w \in \mathbb{Z}_+^E$ be a positive integer vector and define a modular function $w(S) := \sum_{e\in S} w_e$ for any $S \subseteq E$. Consider the function defined as $f(S) = r(S) - w(S)$ for $S \subseteq E$. Note that since w is modular, so is $-w$, and hence, f is submodular because both r and $-w$ are. We show that there is a randomized algebraic algorithm to find a minimizing set for $f(S)$ over $S \subseteq E$.

▶ **Lemma 7.** *Given a family of subspaces $\mathcal{V} = \{V_e \subseteq \mathbb{F}^n\}_{e\in E}$ with rank function r and a vector $w \in \mathbb{Z}_+^E$, there is an RNC algorithm to find the minimizing set $S^* \subseteq E$ for $f(S) = r(S) - w(S)$ that uses only $O(\log^2(nw(E)))$ random bits.*

To prove the Lemma 7, we work with a random/generic vectors that belong to any subspace V_e. Let us first build some terminology towards that. Let $B_e \subseteq \mathbb{F}^n$ be a basis for V_e for $e \in E$. For any set $S \subseteq E$, let us define $S_w = \{(e,i) \mid e \in S,\ 1 \le i \le w_e\}$. Clearly, $|S_w| = w(S)$. We will construct a matrix U whose columns will consist of w_e many generic vectors from the subspace V_e for $e \in E$. Formally, consider a tuple of indeterminates $\boldsymbol{\alpha} = (\alpha_{e,i,v} \mid (e,i) \in E_w,\ v \in B_e)$. Now, construct an $n \times E_w$ matrix U over $\mathbb{F}[\boldsymbol{\alpha}]$ whose columns are as follows:

$$u_{(e,i)} = \sum_{v \in B_e} \alpha_{e,i,v} v \text{ for } (e,i) \in E_w. \tag{1}$$

All notions of rank and linear independence for columns of U will be over the field of fractions $\mathbb{F}(\boldsymbol{\alpha})$. For any set $T \subseteq E_w$, let U_T denote the set of columns of U indexed by elements in T. Our first step is to show a lower bound on $\min_S f(S)$ in terms of rank of U.

▷ Claim 8. For any set $S \subseteq E$,

$$f(S) = r(S) - w(S) \geq \text{rank}(U) - |E_w| = \text{rank}(U) - w(E). \tag{2}$$

Proof. Consider the sets $S_w \subseteq E_w$ and $\bar{S}_w := E_w \setminus S_w$. One can write

$$\text{rank}(U) \leq \text{rank}(U_{S_w}) + \text{rank}(U_{\bar{S}_w}) \leq r(S) + |\bar{S}_w|.$$

The first inequality is from basic linear algebra. The second inequality holds because every column in S_w is in the space $\sum_{e \in S} V_e$ and rank of $U_{\bar{S}_w}$ can be at most its cardinality. Writing $|\bar{S}_w| = |E_w| - |S_w| = w(E) - w(S)$ and rearranging the above inequality will give us the claim. ◁

Once we obtain this lower bound, a natural approach to find $\min_S f(S)$ is to find a set $S^* \subseteq E$ which satisfies (2) with equality. We describe a construction of such a set. First let us define $T^* \subseteq E_w$ to be the set of elements (e,i) such that the column $u_{(e,i)}$ participates non-trivially in some linear dependency among the columns of U. Equivalently,

$$T^* := \{(e,i) \in E_w \mid \text{rank}(U) = \text{rank}(U_{E_w \setminus (e,i)})\}.$$

Then define

$$S^* = \{e \in E \mid (e,i) \in T^* \text{ for some } i\}.$$

▶ **Lemma 9.** *S^* is a set minimizing $f(S)$ over all subsets $S \subseteq E$.*

Proof. As mentioned above, the strategy is to show that S^* satisfies (2) with equality. Towards this we will first prove that

$$\text{rank}(U_{T^*}) = r(S^*). \tag{3}$$

Recall that the columns U_{T^*} are contained in $\sum_{e \in S^*} V_e$. What we need to show for (3) is that $\sum_{e \in S^*} V_e$ is in the linear span of U_{T^*}. We show this for each $V_{\tilde{e}}$, $\tilde{e} \in S^*$.

▷ Claim 10. For each $\tilde{e} \in S^*$, the subspace $V_{\tilde{e}}$ is in the linear span of columns in U_{T^*}.

Proof. By definition of S^*, there must be some $1 \leq \tilde{i} \leq w_{\tilde{e}}$ so that $(\tilde{e}, \tilde{i}) \in T^*$. By definition of T^*, the column $u_{(\tilde{e},\tilde{i})}$ non-trivially participates in some linear dependency among the columns of U_{T^*}. So, there exists a set $J \subseteq T^* \setminus \{(\tilde{e},\tilde{i})\}$ and a vector $\Gamma \in \mathbb{F}[\boldsymbol{\alpha}]^J$ such that

$$u_{(\tilde{e},\tilde{i})} = U_J \Gamma. \tag{4}$$

Now, recall that $u_{(\tilde{e},\tilde{i})}$ is a generic vector in $V_{\tilde{e}}$, and thus, can be used to express any vector in $V_{\tilde{e}}$. Hence, Equation (4) implies that every vector in $V_{\tilde{e}}$ is in the linear span of the columns in U_J. Below, we argue this point more formally.

Recall (1) that gives $u_{(\tilde{e},\tilde{i})}$ as $\sum_{v \in B_{\tilde{e}}} \alpha_{\tilde{e},\tilde{i},v} v$. One can invert (4) to write the vector Γ as a function of indeterminates $\{\alpha_{\tilde{e},\tilde{i},v}\}_v$ as follows. By basic linear algebra, there is an invertible submatrix $\hat{U}_J$ of U_J and a truncation $\hat{u}_{(\tilde{e},\tilde{i})}$ of $u_{(\tilde{e},\tilde{i})}$ such that

$$\Gamma = \hat{U}_J^{-1} \hat{u}_{(\tilde{e},\tilde{i})}. \tag{5}$$

Consider any vector $v' \in V_{\tilde{e}}$. Suppose v' can be expressed using the basis $B_{\tilde{e}}$ as $\sum_{v \in B_{\tilde{e}}} \delta_v v$. We can substitute $(\alpha_{\tilde{e},\tilde{i},v})_v = (\delta_v)_v$ in the right-hand side of (5) to obtain a vector Γ'. Note that since the matrix U_J is free of indeterminates $\{\alpha_{\tilde{e},\tilde{i},v}\}_v$, this substitution in (5) does not create any issues like division by zero. It will follow that

$$v' = U_J \Gamma'.$$

To conclude, every vector $v' \in V_{\tilde{e}}$ is in the linear span of columns in U_{T^*}. ◁

Claim 10 proves Equation (3). Now, we come back to proving that S^* satisfies (2) with equality. From (3), we have

$$f(S^*) = r(S^*) - w(S^*) = \text{rank}(U_{T^*}) - w(S^*) \leq \text{rank}(U_{T^*}) - |T^*|.$$

The inequality holds because $w(S^*) = |S^*_w| \geq |T^*|$ by the definition of S^*. By construction of T^*, the columns in $U_{E_w \setminus T^*}$ do not participate in any column dependency. Thus, we have $\text{rank}(U) - \text{rank}(U_{T^*}) = |E_w \setminus T^*| = |E_w| - |T^*|$. Putting this in the above inequality, we get

$$f(S^*) = r(S^*) - w(S^*) \leq \text{rank}(U) - |E_w|.$$

This together with (2) implies that S^* is a set that minimizes $f(S)$ over $S \subseteq E$. ◀

Proof of Lemma 7: the parallel algorithm

Let us review the construction of the minimizing set S^* from the previous subsection.

- Construct a matrix U, whose columns are generic vectors from the given subspaces. To be precise U has exactly w_e generic vectors from V_e for each $e \in E$.
- Construct the set $T^* := \{(e,i) \in E_w \mid \text{rank}(U) = \text{rank}(U_{E_w \setminus (e,i)})\}$.
- Construct the set $S^* \subseteq E$ that contains all those elements e such that T^* contains (e,i) for some $1 \leq i \leq w_e$.

The rank computations in the second step can all be done in parallel. Importantly, rank computation for any matrix over the base field $\mathbb{F}$ can be done in NC. However, this computation is not efficient for U (or its submatrices) as it is a matrix with indeterminates $\boldsymbol{\alpha}$. To overcome this, we plan to substitute all the indeterminates with field constants. Observe that as long as our substitution preserves the ranks of all column subsets of U, one can safely run the above procedure on the substituted matrix and expect to get the correct answer.

How do we find the right substitution? We argue that a random substitution from a large enough set of field elements does the job. It is known that the rank of a subset of columns remains the same with high probability if each indeterminate is replaced with a field element randomly chosen from a set of size poly(size(U)) = poly($n \times w(E)$). One can see this by applying Lemma 5 on the largest nonzero minor. But, note that we need one substitution that preserves ranks for U and each submatrix $U_{E_w \setminus (e,i)}$ simultaneously. One can use union bound to argue that with high probability, all the desired submatrices preserve their rank. Note that this algorithm needs to use polynomially many random bits. Next we show how to reduce this number of random bits.

Derandomization

To reduce the number of random bits, we use results from deterministic polynomial identity testing. Recall Lemma 6 that gives a pseudorandom substitution that preserves nonzeroness of polynomials of the form $\det(\sum_{i=1}^{n} A_i z_i)$ for $\operatorname{rank}(A_i) = 1$, with high probability. Note that any minor of U is also a polynomial of this form because any variable $\alpha_{e,i,u}$ appears in exactly one column of U. Again, one can use union bound to argue that with high probability, the substitution preserves nonzeroness for all the desired minors of U and each $U_{E_w \setminus (e,i)}$ simultaneously.

To conclude, the pseudorandom substitution from Lemma 6 uses $O(\log^2(nw(E)))$ random bits and with that substitution our algorithm will give the correct minimizing set S^* with high probability.

3.2 Reduction to the special case

Recall Definition 1 which defines LR submodular functions to be those which can be written as a sum of a collection of rank functions together with another collection of rank functions applied on the complement set. We will first argue that the same class of functions is also captured by just taking sum of a rank function and another rank function applied on the complement.

▶ **Observation 11.** *Let $f\colon 2^E \to \mathbb{Z}$ be an LR submodular function given as*

$$f(S) = \sum_{j=1}^{\ell} r_j(S) + \sum_{j=\ell+1}^{k} r_j(\bar{S}),$$

for some $\ell \leq k$, where r_j is the rank function for a family of subspaces $\mathcal{V}_j = \{V_{j,e} \subseteq \mathbb{F}^n\}_{e\in E}$ for $1 \leq j \leq k$. Then f can also be written as

$$f(S) = r'_1(S) + r'_2(\bar{S}),$$

where r'_1 and r'_2 are the rank functions of the families $\mathcal{V}'_1 = \{\oplus_{j=1}^{\ell} V_{j,e} \subseteq \mathbb{F}^{\ell n}\}_{e\in E}$ and $\mathcal{V}'_2 = \{\oplus_{j=\ell+1}^{k} V_{j,e} \subseteq \mathbb{F}^{(k-\ell)n}\}_{e\in E}$, respectively.

Next, we show that we can, in fact, take one of the rank functions to be modular. That is, any LR submodular function can be written as a sum of a rank function and a modular function. In context of general submodular functions, this is a known fact and was used in the first pseudo-polynomial time submodular minimization [6, Lemma 2.1]. Here, we show a more specific result for LR submodular functions that says that the new rank function is also linearly representable and the corresponding family of subspaces can be constructed efficiently.

▶ **Lemma 12.** *Given an LR submodular function $f(S)$ by k families of subspaces of $\mathbb{F}^n$ as in Definition 1, one can compute in NC a family of subspaces $\mathcal{V} = \{V_e \subseteq \mathbb{F}^{kn|E|}\}_{e\in E}$ with rank function r, a vector $w \in \{1, 2, \ldots, kn\}^E$, and a constant C such that for each $S \subseteq E$,*

$$f(S) = r(S) - w(S) + C.$$

To prove Lemma 12, the first step is to use Observation 11 to get the LR submodular function in the form $r'_1(S) + r'_2(\bar{S})$. Then the next step is to write $r'_2(\bar{S})$ as a sum of a rank function on S and a modular function, which is what the following lemma does. Final step is to combine the new rank function with $r'_1(S)$ to get a single rank function, again using Observation 11.

▶ **Lemma 13.** *Let $\mathcal{V} = \{V_e \subseteq \mathbb{F}^n\}_{e\in E}$ be a family of subspaces with its rank function $r\colon 2^E \to \mathbb{Z}$. Then one can construct in* NC *another family of subspaces $\mathcal{V}^* = \{V_e^* \subseteq \mathbb{F}^{n'}\}_{e\in E}$ with rank function r^* for some $n' \leq n|E|$, and a vector $b \in \{1, 2, \dots, n\}^E$ such that for each $S \subseteq E$,*

$$r(\bar{S}) = r^*(S) - b(S) + r(E).$$

Proof. For each $e \in E$, let b_e be the dimension of the subspace V_e and $B_e = \{u_{e,i} \mid 1 \leq i \leq b_e\}$ be a basis for it. Let $E' = \{(e, i) \mid e \in E,\ 1 \leq i \leq b_e\}$ be a new ground set. Note that since $b_e \leq n$, we have $|E'| \leq n|E|$. Consider an $n \times |E'|$ matrix M whose set of columns is $\{u_{e,i}\}_{e,i}$. Without loss of generality, we can assume that M has full row-rank, i.e., $n = r(E)$ (otherwise we could drop some rows). Let M^* be a $(|E'| - n) \times |E'|$ matrix whose row-space is the orthogonal complement of the row-space of M (for a construction in NC, see [4, 22]). The following claim is well known in matroid theory and is used for representation of a dual matroid (see [26, 2.1.9 and 2.2.8]). For any $T \subseteq E'$ and matrix M, let M_T stand for the set of columns of M corresponding to the set of indices T.

▷ Claim 14. For any set $T \subseteq E'$,

$$\mathrm{rank}(M_{E'\setminus T}) = \mathrm{rank}(M^*_T) - |T| + n.$$

Let $\{u^*_{e,i}\}_{e,i}$ be the set of columns of M^*. Consider the family of subspaces $\mathcal{V}^* = \{V_e^* \subseteq \mathbb{F}^{|E'|-n}\}_{e\in E}$, where $V_e^* = \mathrm{span}\{u^*_{e,i} \mid 1 \leq i \leq b_e\}$. Let $r^*\colon 2^E \to \mathbb{Z}$ be the rank function of $\mathcal{V}^*$. Then for any $S \subseteq E$, take $T = \{(e, i) \mid e \in S,\ 1 \leq i \leq b_e\}$ in Claim 14, and we get

$$r(E \setminus S) = r^*(S) - b(S) + n. \qquad ◀$$

Proof of Theorem 2. Lemma 12 gives an NC-reduction from LR submodular minimization to minimizing functions of the form $f(S) = r(S) - w(S) + C$. To minimize $f(S)$, it is sufficient to minimize $r(S) - w(S)$, which is what Lemma 7 does. This concludes the RNC algorithm for LR submodular functions as claimed in Theorem 2. ◀

4 Variants and Applications

In this section, we first consider two variants of submodular minimization which can be reduced to submodular minimization. Here we basically show that this kind of reductions can also made to work in the setting of LR submodular functions. Later on, we also show reductions from two combinatorial problems to LR submodular minimization.

4.1 Variants

Submodular minimization with containment constraints. We first consider an extension of submodular minimization that asks for a minimizing set with containment constraints. Given a submodular function $f\colon 2^E \to \mathbb{R}$ and two sets $S_0 \subseteq S_1 \subseteq E$, suppose the goal is to minimize $f(S)$ subject to $S_0 \subseteq S \subseteq S_1$. It is known that there is a submodular function g on the ground set $S_1 \setminus S_0$ such that for any $S_0 \subseteq S \subseteq S_1$, $f(S) = g(S \setminus S_0) + C'$ for some constant C'. If f is linearly representable then we would like to come up with such a function g that is also linearly representable. Towards this, we will need the following claim.

▷ Claim 15. Let $\mathcal{V} = \{V_e \subseteq \mathbb{F}^n\}_{e\in E}$ be a family of subspaces and r be the corresponding rank function. Let there be two sets $S_0 \subseteq S_1 \subseteq E$. Then we can construct a family of subspaces $\mathcal{V}' = \{V'_e\}_{e\in S_1\setminus S_0}$ with rank function r' such that for each $S_0 \subseteq S \subseteq S_1$

$$r(S) - r(S_0) = r'(S \setminus S_0).$$

Proof. Let V_{S_0} be the subspace $\sum_{e \in S_0} V_e$. For each $e \in S_1 \setminus S_0$, we define V'_e to be the quotient space $V'_e = V_e / V_{S_0}$. Now, for any $S_0 \subseteq S \subseteq S_1$, we have

$$r'(S \setminus S_0) = \dim(\sum_{e \in S \setminus S_0} V'_e) = \dim \left((\sum_{e \in S \setminus S_0} V_e)/V_{S_0} \right) = \dim(V_S/V_{S_0}) = r(S) - r(S_0).$$

◁

From Lemma 12, any linearly representable submodular function can be given as $f(S) = r(S) - w(S) + C$, for a rank functions $r(S)$ on a family of subspaces $\mathcal{V}$, a modular function $w(S)$ and a constant C. For given two sets $S_0 \subseteq S_1 \subseteq E$, let r' be the function constructed in Claim 15 from r. For any $T \subseteq S_1 \setminus S_0$ define $g(T) = r'(T) - w(T) + C$. Using Claim 15, one can verify that for any $S_0 \subseteq S \subseteq S_1$,

$$g(S \setminus S_0) = r'(S \setminus S_0) - w(S \setminus S_0) + C = r(S) - r(S_0) - w(S) + w(S_0) + C = f(S) - f(S_0) + C.$$

Choose $C' = C - f(S_0)$ and we get the desired relation. Now, to minimize f under containment constraints, one can just minimize g on the smaller ground set.

Submodular minimization over non-empty sets. In applications, one often needs to minimize a submodular function over non-empty sets, for example, min-cut in an undirected graph. To do this, one can go over all elements $e \in E$ and minimize $f(S)$ with the containment constraint $\{e\} \subseteq S$ (as discussed above). This will give us a minimum value for each choice of e. The minimum among these values will be the minimum over non-empty subsets.

Separation oracle for a linear polymatroid. Given a family of subspaces $\mathcal{V} = \{V_e \subseteq \mathbb{F}^n\}_{e \in E}$ with its rank function $r \colon 2^E \to \mathbb{Z}$, the corresponding polymatroid is a polytope $P_r \subseteq \mathbb{R}^E$ defined as

$$P_r = \{x \in \mathbb{R}^E \mid x \geq 0, \sum_{e \in S} x_e \leq r(S) \ \forall S \subseteq E.\}$$

Given a rational point $\beta \in \mathbb{R}^E$, one needs to decide if β lies in P_r, and if not then find a violating constriant. The non-negativity constraints are easy to check. The other rank constraints are equivalent to

$$\min_{S \subseteq E} (r(S) - x(S)) \geq 0.$$

Thus, to check if β satisfies the rank constraints, it suffices to minimize the function $f(S) = r(S) - \beta(S)$. Moreover, if there is a violating constraint, then the set S^* minimizing $f(S)$ will give a violating constraint. If β is an integer vector, we have already seen how to find S^* in Lemma 7. When β has rational coordinates, then one can assume them to have a common denominator q, i.e., $\beta_e = p_e/q$ for integers p_e, q. Now, the minimization function becomes $q \times r(S) - p(S)$. Now, $p(S)$ is an integral function. To get the multiplicative factor q in the rank, for each subspace V_e in the family, one can take the direct sum with its copies as $\oplus_{j=1}^{q} V_e$. Note that this is efficient as long as the number q is polynomially bounded.

4.2 Applications

In this section, we show that the two combinatorial problems mentioned in Section 1 reduce to LR submodular minimization. To the best of our knowledge, these problems do not have any known reduction to linear matroid intersection. We start with defining the necessary terminology. Branchings and arborescences are directed analogues of forests and spanning trees.

▶ **Definition 16** (Branching and Arborescence). *For a directed graph $G(V, E)$, a subset $B \subseteq E$ of edges is a branching if it contains no undirected cycles (i.e, there are no cycles induced by B if edges in B are considered to be undirected) and for every vertex v, there is at most one edge in B that is incoming to v. A vertex is a root of B if it has no incoming edges in B. An arborescence is a branching with exactly one root, that is, it is a rooted tree. If the root vertex of an arborescence is a, then we call it an a-arborescence.*

Covering by a-arborescences

A directed graph $G(V, E)$ is said to be covered by k a-arborescences if there exists subsets $B_1, B_2, \ldots, B_k \subseteq E$, each of which is an a-arborescence and $E = B_1 \cup B_2 \cup \cdots \cup B_k$. Vidyasankar [32] gave the following characterization for the graph to covered by k a-arborescences. For any set of vertices $S \subseteq V$, let $H[S] := \{u \in S \mid \exists v \in V \setminus S,\ (v, u) \in E\}$ and let $\delta^{in}(S) := \{(v, u) \in E \mid \exists v \in V \setminus S,\ u \in S\}$ be the set of incoming edges in S. Let $\deg^{\text{in}}(v)$ be the number of incoming edges to v.

▶ **Theorem 17** ([32]). *For a vertex $a \in V$ and a positive number k, the directed graph $G(V, E)$ is be covered by k a-arborescences if and only if*

- $\deg^{\text{in}}(a) = 0$ *and* $\deg^{\text{in}}(v) \le k$ *for each* $v \in V$ *and*
- $\sum_{v \in H[S]}(k - \deg^{\text{in}}(v)) \ge k - |\delta^{in}(S)|$, *for each non-empty subset S of $V \setminus \{a\}$.*

Reduction to submodular minimization. We show that testing the conditions required in Theorem 17 can be reduced to LR submodular minimization. Testing the first condition is trivial. We come to the second one. Let us define a function $f \colon 2^{V \setminus \{a\}} \to \mathbb{Z}$ as follows:

$$f(S) = \sum_{v \in H[S]} (k - \deg^{\text{in}}(v)) + |\delta^{in}(S)|.$$

We will just show that $f(S)$ is a linearly representable submodular function. Clearly, one can check the required condition by finding $\min_{S \subseteq V \setminus \{a\}} f(S)$ and verifying that it is at least k.

For any vertex $u \in V$, let $\chi_u \in \{0, 1\}^V$ be the characteristic vector of v. Let us define two families of subspaces $\mathcal{L}_1$ and $\mathcal{L}_2$ with rank functions r_1 and r_2.

- $\mathcal{L}_1 = \{L_{1,u} \subseteq \mathbb{R}^{k \times |V|}\}_{u \in V}$, where $L_{1,u} = \sum\limits_{\substack{v=u \text{ or} \\ (u,v) \in E}} \oplus_{j=1}^{k - \deg^{\text{in}}(v)} \operatorname{span}(\chi_v)$ and
- $\mathcal{L}_2 = \{L_{2,u} \subseteq \mathbb{R}^{k \times |V|}\}_{u \in V}$, where $L_{2,u} = \oplus_{j=1}^{k - \deg^{\text{in}}(u)} \operatorname{span}(\chi_u)$

One can observe that for any set $S \subseteq V$, $r_1(S)$ is just the sum of the quantity $k - \deg^{\text{in}}(v)$ over all vertices v that are either in S or out-neighbors of S, and $r_2(S)$ is sum of the same quantity over all the vertices of S. Thus, we can write

$$r_1(S) - r_2(S) = \sum_{v \in H[\bar{S}]} (k - \deg^{\text{in}}(v)), \tag{6}$$

where $\bar{S} = V \setminus S$. Note that $-r_2(S)$ is same as $r_2(\bar{S}) - r_2(V)$. Thus,

$$\sum_{v \in H[S]} (k - \deg^{\text{in}}(v)) = r_1(\bar{S}) + r_2(S) - r_2(V). \tag{7}$$

Now, we will express the second part of $f(S)$, that is $|\delta^{in}(S)|$, as a LR submodular function. For any edge $e \in E$, let $\chi_e \in \{0, 1\}^E$ be the characteristic vector of e. Let us define three families of subspaces $\mathcal{L}_3$, $\mathcal{L}_4$ and $\mathcal{L}_5$ with rank functions r_3, r_4 and r_5, respectively.

- $\mathcal{L}_3 = \{L_{3,u} \subseteq \mathbb{R}^E\}_{u \in V}$, where $L_{3,u} = \text{span}\{\chi_e \mid e = (u,v) \text{ for some } v \in V\}$.
- $\mathcal{L}_4 = \{L_{4,u} \subseteq \mathbb{R}^E\}_{u \in V}$, where $L_{4,u} = \text{span}\{\chi_e \mid e = (v,u) \text{ for some } v \in V\}$.
- $\mathcal{L}_5 = \{L_{5,u} \subseteq \mathbb{R}^E\}_{u \in V}$, where $L_{5,u} = \text{span}\{\chi_e \mid e = (u,v) \text{ or } e = (v,u) \text{ for some } v \in V\}$.

One can observe that for any set $S \subseteq V$, $r_3(S)$ is the total number of edges which are outgoing from some vertex in S, $r_4(S)$ is the total number of edges which are incoming to some vertex in S, and $r_5(S)$ is the total number of edges that are incident (outgoing or incoming) to some vertex in S. Thus, one can write

$$2|\delta^{in}(S)| = r_3(\bar{S}) + r_4(S) + r_5(S) + r_5(\bar{S}) - 2|E|. \tag{8}$$

Together with (7) and (8), we can write,

$$2f(S) = 2 \times (r_1(\bar{S}) + r_2(S) - r_2(V)) + r_3(\bar{S}) + r_4(S) + r_5(S) + r_5(\bar{S}) - 2|E|.$$

The terms $r_2(V)$ and $2|E|$ are constants here. The other terms give us a linearly representable submodular function.

Recall that we have to minimize the function $f(S)$ over subsets S that do not contain the vertex a. We had reduced such a constrained minimization to general minimization in the previous subsection.

Packing of branchings

For a given directed graph and given subsets $R_1, R_2, \ldots, R_k$ of vertices, we need to decide if there exist k edge-disjoint branchings that are rooted at $R_1, R_2, \ldots, R_k$, respectively. Edmonds (see [29, Theorem 53.1]) gave the following characterization.

▶ **Theorem 18.** *Let $G = (V, E)$ be a directed graph with $R_1, R_2, \ldots, R_k$ being subsets of V. Then there exist disjoint branchings $B_1, B_2, \ldots, B_k$ such that B_i has root set R_i for $1 \leq i \leq k$ if and only if $|\delta^{in}(S)| \geq |i : R_i \cap S = \emptyset|$ for each non-empty subset S of V.*

Let $f(S) = |\delta^{in}(S)| - |i : R_i \cap S = \emptyset|$. To check the condition in the theorem, it is sufficient to minimize $f(S)$ over non-empty subsets of V. We have already expressed $|\delta^{in}(S)|$ as a linearly representable submodular function in (8). We need to now express the other part of the function. Let us define

$$g_i(S) := \begin{cases} 1 & \text{if } S \cap R_i \neq \emptyset \\ 0 & \text{otherwise.} \end{cases}$$

One can verify that g_i is the rank function of the following family of subspaces: $\mathcal{L}_i = \{L_{i,u} \subseteq \mathbb{R}\}_{u \in V}$, where $L_{i,u} = \{1\}$ if $u \in R_i$ and $L_{i,u} = \{0\}$ otherwise. Now, one can express the desired function in terms of g_i's.

$$-|i : R_i \cap S = \emptyset| = \sum_{i=1}^{k} g_i(S) - k.$$

Together with (8), this gives us a linear representation for $f(S)$ (up to additive constants).

5 Discussion

We have given a parallel algorithm for submodular minimization in the special case of linearly representable submodular functions. It would be interesting to know if there are other classes of submodular functions that admit parallel algorithms. More generally, it is not clear if there can be an efficient parallel algorithm for submodular minimization in the oracle model.

We have given two examples of combinatorial problems that are captured by LR submodular minimization, but are not known to be reducible to linear matroid intersection. One needs to investigate what are other examples of such problems.

References

1 Sepehr Assadi, Yu Chen, and Sanjeev Khanna. Polynomial pass lower bounds for graph streaming algorithms. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2019, page 265–276, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3313276.3316361.

2 Eric Balkanski and Yaron Singer. Minimizing a submodular function from samples. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 814–822, Red Hook, NY, USA, 2017. Curran Associates Inc.

3 Eric Balkanski and Yaron Singer. A lower bound for parallel submodular minimization. In *Proceedings of the 52nd Annual ACM Symposium on Theory of Computing (STOC) June 22–26, 2020 in Chicago, IL*, 2020. URL: http://people.seas.harvard.edu/~yaron/papers/lower_bound_submodular_minimization.pdf.

4 Allan Borodin, Joachim von zur Gathen, and John Hopcroft. Fast parallel matrix and GCD computations. *Information and Control*, 52(3):241–256, 1982.

5 G. Cornuéjols, G.L. Nemhauser, and L.A. Wolsey. The uncapacitated facility location problem. In *In: Discrete Location Theory (P. Mirchandani, R. Francis, eds.)*, pages 119–171. Wiley, New York, 1990.

6 William H. Cunningham. On submodular function minimization. *Combinatorica*, 5(3):185–192, 1985. doi:10.1007/BF02579361.

7 G. Davis, S. Mallat, and M. Avellaneda. Adaptive greedy approximations. *Constructive Approximation*, 13(1):57–98, 1997. doi:10.1007/BF02678430.

8 Richard A. Demillo and Richard J. Lipton. A probabilistic remark on algebraic program testing. *Information Processing Letters*, 7(4):193–195, 1978.

9 Uriel Feige, Vahab S. Mirrokni, and Jan Vondrák. Maximizing non-monotone submodular functions. *SIAM J. Comput.*, 40(4):1133–1153, 2011. doi:10.1137/090779346.

10 Stephen A. Fenner, Rohit Gurjar, and Thomas Thierauf. Bipartite perfect matching is in quasi-nc. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA*, pages 754–763, 2016.

11 Martin Grötschel, László Lovász, and Alexander Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981. doi:10.1007/BF02579273.

12 Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2 of *Algorithms and Combinatorics*. Springer, 1988. doi:10.1007/978-3-642-97881-4.

13 Jiong Guo, Rolf Niedermeier, and Sebastian Wernicke. Parameterized complexity of vertex cover variants. *Theory Comput. Syst.*, 41:501–520, October 2007. doi:10.1007/s00224-007-1309-3.

14 Rohit Gurjar and Thomas Thierauf. Linear matroid intersection is in quasi-NC. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 821–830, 2017. doi:10.1145/3055399.3055440.

15 Satoru Iwata, Lisa Fleischer, and Satoru Fujishige. A combinatorial strongly polynomial algorithm for minimizing submodular functions. *J. ACM*, 48(4):761–777, 2001. doi:10.1145/502090.502096.

16 Richard M. Karp. Reducibility among combinatorial problems. In *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, pages 85–103, 1972. doi:10.1007/978-1-4684-2001-2_9.

17 Richard M. Karp, Eli Upfal, and Avi Wigderson. Constructing a perfect matching is in random NC. *Combinatorica*, 6(1):35–48, 1986.

18 E. L. Lawler. Cutsets and partitions of hypergraphs. *Networks*, 3(3):275–285, 1973. doi:10.1002/net.3230030306.

19 Yin Tat Lee, Aaron Sidford, and Sam Chiu-Wai Wong. A faster cutting plane method and its implications for combinatorial and convex optimization. In *Proceedings of the 2015 IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS)*, FOCS '15, page 1049–1065, USA, 2015. IEEE Computer Society. doi:10.1109/FOCS.2015.68.

20 L. Lovász. Submodular functions and convexity. In Achim Bachem, Bernhard Korte, and Martin Grötschel, editors, *Mathematical Programming The State of the Art: Bonn 1982*, pages 235–257. Springer Berlin Heidelberg, Berlin, Heidelberg, 1983. doi:10.1007/978-3-642-68874-4_10.

21 László Lovász. On determinants, matchings, and random algorithms. In *Fundamentals of Computation Theory*, pages 565–574, 1979.

22 Ketan Mulmuley. A fast parallel algorithm to compute the rank of a matrix over an arbitrary field. *Combinatorica*, 7(1):101–104, 1987.

23 Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7:105–113, 1987. doi:10.1007/BF02579206.

24 H. Narayanan, Huzur Saran, and Vijay V. Vazirani. Randomized parallel algorithms for matroid union and intersection, with applications to arboresences and edge-disjoint spanning trees. *SIAM J. Comput.*, 23(2):387–397, 1994. doi:10.1137/S0097539791195245.

25 Øystein Ore. Über höhere Kongruenzen. *Norsk Mat. Forenings Skrifter Ser. I*, 7(15):27, 1922.

26 James G. Oxley. *Matroid Theory (Oxford Graduate Texts in Mathematics)*. Oxford University Press, Inc., New York, NY, USA, 2006.

27 M. J. Piff and D. J. A. Welsh. On the number of combinatorial geometries. *Bulletin of the London Mathematical Society*, 3(1):55–56, 1971. doi:10.1112/blms/3.1.55.

28 Alexander Schrijver. A combinatorial algorithm minimizing submodular functions in strongly polynomial time. *J. Comb. Theory, Ser. B*, 80(2):346–355, 2000. doi:10.1006/jctb.2000.1989.

29 Alexander Schrijver. *Combinatorial optimization : polyhedra and efficiency. Vol. B. , Matroids, trees, stable sets. chapters 39-69.* Algorithms and combinatorics. Springer-Verlag, Berlin, Heidelberg, New York, N.Y., et al., 2003. URL: http://opac.inria.fr/record=b1124843.

30 Jacob T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM*, 27(4):701–717, October 1980.

31 Zoya Svitkina and Lisa Fleischer. Submodular approximation: Sampling-based algorithms and lower bounds. *SIAM J. Comput.*, 40(6):1715–1737, December 2011. doi:10.1137/100783352.

32 K. Vidyasankar. Covering the edge set of a directed graph with trees. *Discrete Mathematics*, 24(1):79–85, 1978. doi:10.1016/0012-365X(78)90174-7.

33 Richard Zippel. Probabilistic algorithms for sparse polynomials. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation (EUROSAM)*, pages 216–226. Springer-Verlag, 1979.

d-To-1 Hardness of Coloring 3-Colorable Graphs with $O(1)$ Colors

Venkatesan Guruswami
Carnegie Mellon University, Pittsburgh, PA, USA
guruswami@cmu.edu

Sai Sandeep
Carnegie Mellon University, Pittsburgh, PA, USA
spallerl@andrew.cmu.edu

Abstract

The d-to-1 conjecture of Khot asserts that it is NP-hard to satisfy an ϵ fraction of constraints of a satisfiable d-to-1 Label Cover instance, for arbitrarily small $\epsilon > 0$. We prove that the d-to-1 conjecture for any fixed d implies the hardness of coloring a 3-colorable graph with C colors for arbitrarily large integers C.

Earlier, the hardness of $O(1)$-coloring a 4-colorable graphs is known under the 2-to-1 conjecture, which is the strongest in the family of d-to-1 conjectures, and the hardness for 3-colorable graphs is known under a certain "fish-shaped" variant of the 2-to-1 conjecture.

2012 ACM Subject Classification Theory of computation → Computational complexity and cryptography

Keywords and phrases graph coloring, hardness of approximation

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.62

Category Track A: Algorithms, Complexity and Games

Related Version A full version of the paper is available at `https://eccc.weizmann.ac.il/report/2019/116/`.

Funding Research supported in part by NSF grants CCF-1422045 and CCF-1908125.

1 Introduction

Determining if a graph is 3-colorable is one of the classic NP-complete problems. Thus, given a 3-colorable graph it is NP-hard to color it with 3 colors. The best known polynomial time algorithms for coloring 3-colorable graphs use about $n^{0.2}$ colors, where n is the number of vertices in the graph [9]. On the other hand, on the hardness front, we only know that 5-coloring 3-colorable graphs is NP-hard [3].

This embarrassingly large gap between the hardness and algorithmic results has prompted the quest for conditional hardness results for approximate graph coloring. The canonical starting point for most strong inapproximability results is the Label Cover problem. Label Cover refers to constraint satisfaction problems of arity two over a large (but fixed) domain whose constraint relations are *functions*. Label Cover is known to be very hard to approximate even on satisfiable instances.

The Unique Games Conjecture of Khot [10], which asserts strong inapproximability of Label Cover when the constraint maps are bijections, has formed the basis of numerous tight hardness results for problems which have defied NP-hardness proofs. However, the imperfect completeness inherent in the Unique Games Conjecture makes it unsuitable as the basis for hardness results for graph coloring, where we want *all* edges to be properly colored under the coloring.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 62; pp. 62:1–62:12
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In [10], along with the Unique Games Conjecture, Khot introduced the d-to-1 conjecture. The d-to-1 conjecture says that given a Label Cover instance whose constraint relations are d-to-1 functions, it is NP-hard to decide if there exists a labelling that satisfies all the constraints or no labelling can satisfy even an ϵ fraction of constraints, for arbitrarily small $\epsilon > 0$. (The key is that d can be held fixed and achieve soundness $\epsilon \to 0$.) Constraints similar to 2-to-1 also played an implicit role in the beautiful work of Dinur and Safra on inapproximability of vertex cover [8].

Based on the 2-to-1 conjecture, Dinur, Mossel and Regev [7], extending the invariance principle based techniques of [11,15], proved the hardness of coloring graphs that are promised to be 4-colorable with any constant number of colors. Furthermore, they prove the same for 3-colorable graphs under a certain "fish shaped" variant of the 2-to-1 conjecture. In this paper, we prove that the same result can be proved under the weaker assumption of d-to-1 conjecture[1], for some (arbitrarily large) constant d.

▶ **Theorem 1.** *Assume that d-to-1 conjecture is true for some constant d. Then, for every positive integer $t \geq 3$, it is NP-hard to color a 3-colorable graph G with t colors.*

We stress that the d-to-1 conjecture insists on perfect completeness (i.e., hardness on satisfiable instances), and this important feature seems necessary for its implications for coloring problems, where we seek to properly color all edges. The variant of the 2-to-1 conjecture where one settles for near-perfect completeness was recently established in a striking sequence of works [5,6,12,13].

The result of [7] in fact shows hardness of finding an independent set of density ϵ in a 3-colorable graph for arbitrary $\epsilon > 0$ (which immediately implies the hardness of finding a coloring with $1/\epsilon$ colors). Our result in Theorem 1 above does *not* get this stronger hardness for finding independent sets. But it is conditioned on the d-to-1 conjecture for arbitrary d rather than the specific 2-to-1 conjecture. We note that proving the d-to-1 conjecture for some large d could be significantly easier than the 2-to-1 conjecture, so Theorem 1 perhaps provides an avenue for resolving a longstanding challenge concerning the complexity of approximate graph coloring.

Our proof of Theorem 1 is a simple combination of two results. First, following the methodology of [7], we prove that the d-to-1 conjecture implies that coloring a $2d$-colorable graph with $O(1)$ colors is NP-hard. The result of [7] is the $d = 2$ case of this claim. In fact, they state in a future work section that the d-to-1 conjecture should imply hardness of $O(1)$-coloring q-colorable graphs for some large enough $q = q(d)$. However, they did not specify the details of the reduction or an explicit value of q, and mention determining the dependence of q on d as an interesting question. Here we show the conditional hardness based on d-to-1 conjecture holds for $q = 2d$ (achieving $q < 2d$ seems unlikely with the general reduction approach of [7]).

The key technical ingredient necessary for such a reduction is a symmetric Markov chain on $[q]^d$ where transitions are allowed only between disjoint tuples and which has spectral radius bounded away from 1. We show the existence of such a symmetric Markov chain for $q = 2d$. We do so via a connection to matrix scaling, which enables us to deduce the necessary chain at a conceptual level without messy calculations. Specifically, we use the result [4], which follows from the Sinkhorn-Knopp iterative matrix scaling algorithm [19],

[1] For d-to-1 Label Cover, there are two definitions possible, one where the constraint maps are *at most* d-to-1 with each element in the range having at most d pre-images, and one where the constraint maps are *exactly* d-to-1. In this paper, we stick with the exact variant.

that if a non-negative symmetric matrix A has *total support* then there is a symmetric doubly stochastic matrix supported on the non-zero entries of A. When A is the adjacency matrix of a graph G, the total support condition is equivalent to every edge of G belonging to a cycle cover. We describe a graph on $[q]^d$ whose edges connect disjoint tuples and where every edge belongs to a cycle cover.

Our second ingredient is a remarkable yet simple reduction due to Krokhin, Opršal, Wrochna and Živný [14], which exploits the relation between the arc-chromatic number and chromatic number of a digraph [17]. Let $b : \mathbb{N} \to \mathbb{N}$ be defined by $b(n) := \binom{n}{\lfloor n/2 \rfloor}$. Their result then is that $b(t)$-coloring $b(c)$-colorable graphs is polynomial time (in fact logspace) reducible to t-coloring c-colorable graphs. Since $b(n)$ is increasing and $b(n) > n$ for all $n \geq 4$, it follows that a NP-hardness result for $O(1)$-coloring q-colorable graphs also implies NP-hardness of $O(1)$-coloring 4-colorable graphs. Furthermore, the NP hardness of $O(1)$-coloring of 3-colorable graphs follows from the above by applying the arc graph reduction twice to K_4.

Overview

In Section 2, we define the Label Cover problem, and state the d-to-1 conjecture formally. We also introduce low degree influences that we need later. In Section 3, we first prove the existence of the Markov chain with required properties, and then describe the reduction from Label Cover to Approximate Coloring. We note that the reduction is in fact exactly the same one used in [7], the difference being in using a different Markov Chain. We present the reduction and the preliminaries required in this paper for the sake of completeness.

2 Preliminaries

We first formally define the Label Cover problem and then state the hardness conjectures.

2.1 Label Cover

▶ **Definition 2** (Label Cover). *In the Label Cover instance, we are given a tuple $G = ((V, E), R, \Psi)$ where*

1. *(V, E) is a graph on vertex set V with edge set E.*
2. *Each vertex in V has to be assigned a label from the set $\Sigma = [R] = \{1, 2, \dots, R\}$.*
3. *For every edge $e = (u, v) \in E$, there is an associated relation $\Psi_e \subseteq \Sigma \times \Sigma$. This corresponds to a constraint between u and v.*

A labeling $\sigma : V \to \Sigma$ satisfies a constraint associated with the edge $e = (u, v)$ if and only if $(\sigma(u), \sigma(v)) \in \Psi_e$. Given such an instance, the goal is to distinguish if there is a labeling that can satisfy all the constraints or no labeling can satisfy a significant fraction of constraints.

We now state the d-to-1 conjecture. As is the case with [7], we will state and use the *exact* d-to-1 variant where the constraint maps have exactly d pre-images for each element in the range. Khot's original formulation only required that there are at most d pre-images for each element in the range. The d-to-1 conjecture becomes stronger for smaller d (so that the 2-to-1 is the strongest form of the conjecture) – this is obvious for the variant where the maps are at most d-to-1. For the exact variant, if we allow the Label cover graph to have multiple edges, we can reduce d-to-1 conjecture to $(d + 1)$-to-1 conjecture using a simple argument. We present this reduction in Section 4. On that note, we remark without details that our reduction indeed works with the multigraph variant of d-to-1 conjecture.

▶ **Conjecture 3** ((Exact) d-to-1 Conjecture). *For every $\epsilon > 0$, given a bipartite Label Cover instance $G = ((V = X \cup Y, E), (dR, R), \Psi)$ satisfying the following constraints:*

(i) *We refer to X as the vertices on the left, and Y as the set of vertices on the right. The vertices belonging to X are to be assigned labels from $[dR]$ while the vertices in Y are to be assigned labels from $[R]$.*

(ii) *The constraints are d-to-1 i.e. for every $b \in [R]$, there are precisely d values $a \in [dR]$ such that $(a, b) \in \Psi_e$ for every relation Ψ_e in the instance.*

It is NP-hard to distinguish between the following cases:

1. *There is a labeling that satisfies all the constraints in G.*
2. *No labeling can satisfy more than ϵ fraction of constraints in G.*

Similar to the d-to-1 constraints, one can consider d-to-d constraints in the Label Cover. In order to do so, we define the relation $d \leftrightarrow d$ on $[dR] \times [dR]$:

$$d \leftrightarrow d = \{(di - p + 1, di - q + 1) \ \mid \ 1 \le i \le R, \ \ 1 \le p, q \le d\} \ .$$

A constraint $\psi \subseteq [dR] \times [dR]$ is said to be d-to-d if there exist permutations π_1 and π_2 on $[dR]$ such that $(a, b) \in \psi$ iff $(\pi_1^{-1}(a), \pi_2^{-1}(b)) \in d \leftrightarrow d$.

In [7], it is proved that Conjecture 3 implies the following conjecture.

▶ **Conjecture 4** (d-to-d conjecture). *For every $\epsilon > 0$ and every $t \in \mathbb{N}$, there exists $R \in \mathbb{N}$ such that given a Label Cover instance $G = ((V, E), dR, \Psi)$ where all the constraints are d-to-d, it is NP-hard to distinguish between the following cases:*

(i) $sat(G) = 1$*, or*

(ii) $isat_t(G) < \epsilon$

Here, $sat(G)$ denotes the maximum fraction of constraints satisfied by any labeling. Similarly, $isat(G)$ denotes the size of the largest set $S \subseteq V$ such that there exists a labeling that satisfies all the constraints induced on S. The value $isat_t(G)$ denotes the size of largest set $S \subseteq V$ such that there exists a labeling that assigns at most t labels to each vertex that satisfies all the constraints induced on S. A constraint between u, v is said to be satisfied by labeling assigning multiple labels to u and v if and only if there exists at least one pair of labels to u and v among the multiple labels that satisfy the constraint.

2.2 Low degree influences

Next, we define the low degree influences that we need later. We refer the reader to [7] for a comprehensive treatment of the same.

Let $\alpha_0 = \mathbf{1}, \alpha_1, \dots, \alpha_{q-1}$ be an orthonormal basis of $\mathbb{R}^q$. We can define the set of functions $\alpha_x : [q]^n \to \mathbb{R}, x \in [q]^n$ as $\alpha_x(y) = (\alpha_{x_1}(y_1), \alpha_{x_2}(y_2), \dots, \alpha_{x_n}(y_n))$. Observe that these functions form a basis for the functions from $[q]^n$ to $\mathbb{R}$. Let $\hat{f}(\alpha_x) = \langle f, \alpha_x \rangle$, where we define the inner product between functions $f, g : [q]^n \to \mathbb{R}$ as $\langle f, g \rangle = q^{-n} \sum_{x \in [q]^n} f(x) g(x)$. We define the low degree influence of f as follows:

▶ **Definition 5.** *For a function $f : [q]^n \to \mathbb{R}$, the degree k influence of the coordinate i is defined as follows:*

$$I_i^{\le k}(f) = \sum_{x : x_i \neq 0, |x| \le k} \hat{f}^2(\alpha_x)$$

Note that the above definition is independent of the basis $\alpha_0, \alpha_1, \ldots, \alpha_{q-1}$ that we start with, as long as $\alpha_0 = \mathbf{1}$. From the above definition, we can infer that for functions $f : [q]^n \to [0,1]$, the sum of low degree influences is bounded by

$$\sum_i I_i^{\le k}(f) \le k$$

For a vector $x \in [q]^{dR}$, let $\overline{x} \in [q^d]^R$ be the corresponding element in $[q^d]^R$ i.e.

$$\overline{x} = ((x_1, x_2, \ldots, x_d), (x_{d+1}, x_{d+2}, \ldots, x_{2d}), \ldots, (x_{dR-d+1}, x_{dR-d+2}, \ldots, x_{dR}))$$

Similarly, for $y \in [q^d]^R$, let $\underline{y}$ denote the inverse of above operation. We can extend this notion to functions as well: For a function $f : [q]^{dR} \to \mathbb{R}$, let the function $\overline{f} : [q^d]^R \to \mathbb{R}$ be defined naturally by

$$\overline{f}(y) = f(\underline{y})$$

Similarly, for a function $f : [q^d]^R \to \mathbb{R}$, let $\underline{f} : [q]^{dR} \to \mathbb{R}$ be defined as $\underline{f}(x) = f(\overline{x})$.

We need the following lemma:

▶ **Lemma 6.** *For any function $f : [q]^{dR} \to \mathbb{R}$ and any $k \in \mathbb{N}$ and $i \in [R]$,*

$$I_i^{\le k}(\overline{f}) \le \sum_{j=1}^{d} I_{di-d+j}^{\le dk}(f)$$

Proof. Fix a basis α_x of functions from $[q]^{dR} \to \mathbb{R}$ as above. The functions $\alpha_{\overline{x}}$ form a basis for functions from $[q^d]^R \to \mathbb{R}$, where $\alpha_{\overline{x}}(\overline{y}) = \alpha_x(y)$. Note that $\hat{\overline{f}}(\alpha_{\overline{x}}) = \hat{f}(\alpha_x)$. Thus we get

$$\begin{aligned}
\sum_i I_i^{\le k}(\overline{f}) &= \sum_{\overline{x}:\overline{x}_i \neq (0,0,\ldots,0), |\overline{x}| \le k} \hat{\overline{f}}^2(\alpha_{\overline{x}}) = \sum_{\overline{x}:\overline{x}_i \neq (0,0,\ldots,0), |\overline{x}| \le k} \hat{f}^2(\alpha_x) \\
&\le \sum_{x:\overline{x}_i \neq (0,0,\ldots,0), |x| \le dk} \hat{f}^2(\alpha_x) \\
&\le \sum_{j=1}^{d} \sum_{x:x_{di-d+j} \neq 0, |x| \le dk} \hat{f}^2(\alpha_x) \\
&= \sum_{j=1}^{d} I_{di-d+j}^{\le dk}(f)
\end{aligned}$$

◀

Using the invariance principle and Borell's inequality, [7] prove the following:

▶ **Theorem 7.** *Let q be a fixed integer, and T be a symmetric Markov chain on $[q]$ with $r(T) < 1$. Then for every $\epsilon > 0$, there exists a $\delta > 0$ and a positive integer k such that the following holds: For every $f, g : [q]^n \to [0,1]$ if $\mathbb{E}[f] > \epsilon, \mathbb{E}[g] > \epsilon$ and $\langle f, Tg \rangle = 0$, then*

$$\exists i \in [n] : I_i^{\le k}(f) \ge \delta, I_i^{\le k}(g) \ge \delta$$

where $r(T)$ denotes the second largest eigenvalue (in absolute value) of T.

3 *d*-to-1 hardness for 3-colorable graphs

In this section, we will prove Theorem 1.

3.1 Reducing chromatic number to 3

The following lemma is present in [14] based on a beautiful result concerning the arc-chromatic numbers of digraphs from [17].

▶ **Lemma 8** (Theorem 1.8 of [14]). *Suppose there exists $q \in \mathbb{N}$ such that $O(1)$ coloring q-colorable graphs is NP-hard. Then, $O(1)$ coloring 3-colorable graphs is NP hard.*

Let Graph-Coloring(t, c) denote the promise problem of distinguishing if a graph can be colored with c colors, or cannot even be colored with t colors. The statement is proved by presenting a reduction from Graph-Coloring$(b(t), b(c))$ to Graph-Coloring(t, c) in polynomial time, for the function $b(n) := \binom{n}{\lfloor n/2 \rfloor}$. The reduction works by constructing the arc-graph of the underlying graphs, and using the property of arc graphs that the chromatic number of the arc graph can be bounded precisely using the chromatic number of the original graph. Since b is an increasing function and $b(n) > n$ for all $n \geq 4$, setting $c = 4$ and t large enough proves the statement claimed in the lemma. The reduction from 4-colorable graphs to 3-colorable graphs is achieved by applying the arc graph construction twice recursively.

Thanks to Lemma 8, we can restrict ourselves to the weaker goal of proving that $O(1)$ coloring q-colorable graphs is NP-hard for some fixed constant q assuming Conjecture 3. In fact, following [7], we prove a stronger statement showing hardness of finding independent sets of ϵ fraction of vertices for any $\epsilon > 0$. Combined with Lemma 8, this immediately gives us Theorem 1.

▶ **Theorem 9.** *Suppose that Conjecture 4 is true for a constant d. Then, there exists a constant $q = q(d)$ such that for every $\epsilon > 0$, given a graph G, it is NP-hard to distinguish the following cases:*

1. *G can be colored with q colors.*
2. *G does not have any independent set of relative size ϵ.*

In fact, we can take $q = 2d$.

In the remainder of the section, we will prove Theorem 9. We next develop the main technical ingredient that we will plug into the reduction framework of [7] to establish Theorem 9.

3.2 A symmetric Markov chain supported on disjoint tuples

A Markov chain T defined on a state space Ω is said to be symmetric if the transition matrix of T is symmetric, namely for all pairs of states $x, y \in \Omega$, the probability of transition from x to y is equal to the probability of transition from y to x. Symmetry of the Markov chain ensures that the uniform distribution is stationary which is essential when we compose the Label Cover-Long Code reduction with the Markov chain. We define the spectral radius $r(T)$ of a symmetric Markov chain as the second largest eigenvalue in absolute value of its transition probability matrix, i.e., if $1 = \lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_q$ are the eigenvalues, then $r(T) = \max(|\lambda_2|, |\lambda_q|)$.

We now show the existence of a symmetric Markov Chain T on $[q]^d$ with $r(T) < 1$ if $d \geq 2, q \geq 2d$. Furthermore, there is a nonzero transition probability between two elements $x, y \in [q]^d$ only if the support of x and y are disjoint. In [7], such a Markov Chain is shown to exist for the values $(q, d) = (3, 1), (4, 2)$.

▶ **Lemma 10.** *Suppose that $q, d \in \mathbb{N}, q \geq 2d, d \geq 2$. There exists a symmetric Markov chain T on $[q]^d$ such that $r(T) < 1$. Furthermore, if the transition $\{x_1, x_2, \ldots, x_d\} \leftrightarrow \{y_1, y_2, \ldots, y_d\}$ has positive probability in T, then $\{x_1, x_2, \ldots, x_d\} \cap \{y_1, y_2, \ldots, y_d\} = \phi$.*

Proof. We first construct an undirected graph G on $[q]^d$ such that there is an edge between $x, y \in [q]^d$ only if the support of x and y are disjoint. We then use a matrix scaling algorithm to obtain a symmetric Markov chain T from the adjacency matrix of G. For the resulting Markov chain to have $r(T) < 1$, we need that the underlying graph G is connected, and is not bipartite. Furthermore, for the scaling algorithm to produce a valid Markov chain, we need that every edge of G is present in a cycle cover, where a cycle cover of a graph is a disjoint union of cycles that covers every vertex in the graph. Note that we allow trivial 2-cycles in a cycle cover, where we just take an edge twice.

We say that two multisets $x = (x_1, x_2, \ldots, x_d), y = (y_1, y_2, \ldots, y_d) \in [q]^d$ are of the same *type* if the following condition holds: for all pairs of indices $i, j \in [d]$, $x_i = x_j$ if and only if $y_i = y_j$ and $(x_i - x_j)(y_i - y_j) \geq 0$. Note that this is an equivalence relation, and thus each element $x \in [q]^d$ uniquely determines its type.

Consider the graph $G = (V, E)$ where the vertex set is $V = [q]^d$. We add two kinds of edges in this graph. We add an edge between every pair of $x, y \in [q]^d$ that are of the same type, and have disjoint support. Let the subset of $[q]^d$ of elements that are supported on single element be denoted by S, i.e.,

$$S = \{(1, 1, \ldots, 1), (2, 2, \ldots, 2), \ldots, (q, q, \ldots, q)\} .$$

We also add edges between x and y if their support is disjoint, and at least one of x and y belongs to S.

First, we claim that G is connected. This follows from the fact that the set of nodes in S are connected to each other, and every vertex in V is adjacent to at least one vertex in S. As $q \geq 4$, the graph is not bipartite (indeed S induces a q-clique). We will now prove that every edge in this graph is part of a cycle cover. Given an undirected graph on vertex set V, a cycle cover of it is a function $\sigma : V \to V$ that is bijective, and $\sigma(u) = v$ only when u and v are adjacent in the underlying graph.

Towards this, we first prove that for every edge in G between multisets of the same type, there is a cycle cover that uses that edge. For each type, consider the graph obtained by taking the vertices as multisets of that type, and with edges between two multisets of the same type if they are disjoint. Note that for every type, this graph is isomorphic to a Kneser graph $KG(q, k)$ (for some $k \leq d$), whose vertex set corresponds to k-element subsets of $[q]$ and there is an edge between two subsets if they are disjoint.

By symmetry across the subsets, we can infer that the Kneser graphs are regular. Note that every regular graph contains a cycle cover: For a regular graph H, consider a bipartite graph H' which contains a copy of H on both the left side L, and right side R. There is an edge between $x \in L, y \in R$ of H' if and only if x, y are adjacent in H. As H is a regular graph, H' is a regular bipartite graph, and thus, contains a perfect matching. This perfect matching in H' directly gives a cycle cover of H. Furthermore, as Kneser graphs are also vertex-transitive, every edge in these graphs is part of a cycle cover.

Next, we consider edges of G that are between multisets of different types i.e. edges between multisets x, y where exactly one of x and y is in S. Consider an edge between $s \in S$ and $x \in V \setminus S$. As $q \geq 2d$, every multiset in G is adjacent to at least one multiset of the same type. Let y be a multiset that is adjacent to x in G and is of the same type as x. Let $s' \in S$ be chosen such that it is adjacent to y in G. As S is a complete subgraph of G, s and s' are adjacent in G. From the previous argument about edges between multisets of the same type, we can infer that there is a cycle cover of G where y is mapped to x, and s is mapped to s'. We can modify this cycle cover by transforming it as follows - $(s \to x)$ can be made part of cycle cover by transforming $(s \to s'), (y \to x)$ to $(s \to x), (y \to s')$ and keeping rest of the cycle cover intact. Thus, we have proved that every edge of G is part of a cycle cover.

Let A denote the adjacency matrix of the above graph G. Using the Sinkhorn Knopp iterative algorithm, it is proved in [4] that if a non-negative symmetric matrix A has total support, then there exists a diagonal matrix D such that DAD is a doubly stochastic matrix. A square matrix $A = (a_{ij})$ of order n is said to have total support if $A \neq 0$, and for every nonzero entry a_{ij} of A, there exists a permutation σ of $[n]$ such that $\sigma(i) = j$ and for all $e \in [n], a_{e,\sigma(e)} \neq 0$. When the matrix A is an adjacency matrix of a graph G, the total support condition translates to the requirement that every edge in G is part of a cycle cover, a property we have already shown to hold for the graph G.

Thus, we can apply the above scaling result, and view the resulting matrix $B = DAD$ as the transition matrix of a Markov chain T. As A and D are symmetric, B is symmetric, i.e., T is symmetric. As A is connected and no principal diagonal element of D is zero, T is connected as well. Note that every nonzero element of A stays nonzero in T, and A is not bipartite. The above two facts combined ensure that the spectral radius $r(T)$ of T is strictly less than 1. We conclude that there exists a symmetric Markov chain T on state space $[q]^d$ that has both the properties: (i) $r(T) < 1$, and (ii) there is nonzero probability of transition between two multisets only when their support is disjoint. ◀

3.3 Proof of Theorem 9

Let d be the constant for which Conjecture 3 is true. Thus, Conjecture 4 is true for the same value d as well. Choose q, T from Lemma 10 such that T is a symmetric Markov chain on $[q]^d$ such that $r(T) < 1$.

We now reduce the given d-to-d Label Cover instance to the problem of finding independent sets in q-colorable graphs. To be precise, given a Label Cover instance $G = ((V, E), dR, \Psi)$, we output a graph $G' = (V', E')$ such that

1. Completeness: If G is satisfiable, G' can be colored with q colors.
2. Soundness: If $isat_t(G) < \epsilon'$, then G' does not have any independent set of size ϵ.

The parameters t and ϵ' will be set later.

Reduction

Our reduction follows the standard Label Cover Long Code paradigm, and in particular closely mirrors [7]. We replace each vertex $w \in V$ of the Label Cover with a set f_w of $[q]^{dR}$ nodes, each corresponding to a vertex in G'. Consider an edge $e = (u, v)$ where Ψ_e is an associated constraint with permutations π_1, π_2 on $[dR]$ such that $(a, b) \in \Psi_e$ if and only if $(\pi_1^{-1}(a), \pi_2^{-1}(b)) \in d \leftrightarrow d$.

We add an edge between $(x_1, x_2, \ldots, x_{dR}) \in f_u$ and $(y_1, y_2, \ldots, y_{dR}) \in f_v$ to E' if and only if

$$\forall i \in [R], T((x_{\pi_1(di-d+1)}, x_{\pi_1(di-d+2)}, \ldots, x_{\pi_1(di)}) \leftrightarrow (y_{\pi_2(di-d+1)}, y_{\pi_2(di-d+2)}, \ldots, y_{\pi_2(di)})) > 0.$$

Completeness

Suppose $\sigma : V \to [dR]$ be a labeling satisfying all the constraints of the Label Cover instance G. We color the node $(x_1, x_2, \ldots, x_{dR}) \in f_w$ with $x_{\sigma(w)} \in [q]$. We claim that this is a legit q-coloring of G'. Suppose that we added an edge between $x \in f_u$ and $y \in f_v$. Let x be colored with x_a and y be colored with y_b. As $(a, b) \in \Psi_{(u,v)}$, we have $(\pi_1^{-1}(a), \pi_2^{-1}(b)) \in d \leftrightarrow d$. Thus, there exist $i \in [R], 1 \le p, q \le d$ such that $a = \pi_1(di - d + p)$ and $b = \pi_2(di - d + q)$.

As we have added an edge between $x \in f_u$ and $y \in f_v$, $x_a \neq y_b$ as the Markov chain T has nonzero probability only between two elements of $[q]^d$ with disjoint support. Thus, there exists a q-coloring of G' when G is satisfiable.

Soundness

We prove the contrapositive that if G' has an independent set of relative size ϵ, then there exists a labeling of G with $isat_t(G) \geq \epsilon'$. Let $S \subseteq V'$ be the largest independent set of G'. We know that $|S| \geq \epsilon|V'|$. This implies that in at least $\epsilon' = \frac{\epsilon}{2}$ fraction of the long code blocks, at least $\frac{\epsilon}{2}$ fraction of nodes belong to S. Let this subset of V be denoted by Z. Our goal is to show that there exists a small set of labels $\tau : Z \to 2^{[dR]}$ to which we can decode the vertices in Z such that all the constraints induced in Z are satisfied by τ.

For every vertex $w \in Z$, we define functions $g_w : [q]^{dR} \to \{0,1\}$ to be the indicator functions of set S inside the long code blocks corresponding to w i.e. $g_w(x) = 1$ if and only if $x \in S$. Consider an edge $e = (u,v)$ corresponding to the constraint Ψ_e induced in Z. Let the functions $f : [q]^{dR} \to \{0,1\}$ and $g : [q]^{dR} \to \{0,1\}$ be defined such that $f(x^{\pi_1}) = g_u(x)$ and $g(y^{\pi_2}) = g_v(y)$, where π_1 and π_2 are the permutations underlying the relation Ψ_e i.e. $(a,b) \in \Psi_e$ if and only if $(\pi_1^{-1}(a), \pi_2^{-1}(b)) \in d \leftrightarrow d$.

We note that $\langle f, Tg \rangle$ is equal to zero. In other words, suppose that $x, y \in [q]^{dR}, x \in f_u, y \in f_v$ are such that

$$\forall i \in [R], T((x_{di-d+1}, x_{di-d+2}, \ldots, x_{di}) \leftrightarrow (y_{di-d+1}, y_{di-d+2}, \ldots, y_{di})) > 0. \tag{1}$$

Then, $f(x)g(y) = 0$. Suppose for contradiction that there exist $x, y \in [q]^{dR}$ satisfying the above condition, and $f(x) = g(y) = 1$. Let $x' \in f_u, y' \in f_v$ be such that $(x')^{\pi_1} = x, (y')^{\pi_2} = y$. We have $g_u(x') = g_v(y') = 1$. That is, both $x' \in f_u, y' \in f_v$ are in the independent set S. However, Equation (1) can be rewritten as the following:

$$\forall i \in [R], T((x'_{\pi_1(di-d+1)}), (x'_{\pi_1(di-d+2)}), \ldots, x'_{\pi_1(di)}) \leftrightarrow (y'_{\pi_2(di-d+1)}, y'_{\pi_2(di-d+2)}, \ldots, y'_{\pi_2(di)})) > 0. \tag{2}$$

Note that this is precisely the condition for adding edges in G'. Thus, Equation (2) implies that $x' \in f_u$ and $y' \in f_v$ are adjacent in E', and thus cannot both be part of the independent set S. This completes the proof that $\langle f, Tg \rangle = 0$.

Thus, $\langle \overline{f}, T\overline{g} \rangle$ is also equal to zero, where $\overline{f} : [q^d]^R \to \{0,1\}$ and $\overline{g} : [q^d]^R \to \{0,1\}$ are the corresponding functions in $[q^d]^R$ of f, g. From the definition of Z, $\mathbb{E}(\overline{f}) \geq \frac{\epsilon}{2}$ and $\mathbb{E}(\overline{g}) \geq \frac{\epsilon}{2}$. We apply Theorem 7 to $\overline{f}$ and $\overline{g}$ to deduce that there exists $i \in [R]$, a positive integer $k = k(\epsilon)$ and $\delta = \delta(\epsilon)$ such that $I_i^{\leq k}(\overline{f}) \geq \delta$ and $I_i^{\leq k}(\overline{g}) \geq \delta$. This motivates us to define the label set of vertex $w \in Z$, $L(w)$ as the following -

$$L(w) := \{i \in [dR] : I_i^{\leq dk}(g_w) \geq \frac{\delta}{d}\}$$

As the sum of k degree influences of all variables is at most k, the size of $L(w)$ is upper bounded by $\frac{kd}{\delta}$ for every v. Thus, we set the parameter t to be $\frac{kd}{\delta}$.

The final step is to prove that the labeling L is indeed a valid labeling inside edges induced in Z. Consider an edge $e = (u,v)$ induced in Z with the constraint relation being Ψ_e such that $(a,b) \in \Psi_e$ if and only if $(\pi_1(a), \pi_2(b)) \in d \leftrightarrow d$. Our goal is to show that there exist indices $\sigma_1, \sigma_2 \in [dR]$ such that $\sigma_1 \in L(u), \sigma_2 \in L(v)$ and $(\sigma_1, \sigma_2) \in \Psi_e$. Using Theorem 7, we can deduce that there exists $i \in [R]$ such that $I_i^{\leq k}(\overline{f}) \geq \delta$ and $I_i^{\geq k}(\overline{g}) \geq \delta$. Using Lemma 6, we can conclude that there exist $i_1, i_2 \in [dR]$ such that $I_{i_1}^{\leq dk}(f) \geq \frac{\delta}{d}$ and $I_{i_2}^{\leq dk}(g) \geq \frac{\delta}{d}$ such that $(i_1, i_2) \in d \leftrightarrow d$. Let $\sigma_1, \sigma_2 \in [dR]$ be such that $i_1 = \pi_1(\sigma_1), i_2 \in \sigma_2$. As $f(x^{\pi_1}) = g_u(x)$, $I^{\leq dk}_{\pi_1^{-1}(i_1)}(g_u) \geq \frac{\delta}{d}$. And thus, $\sigma_1 \in L(u)$, and similarly $\sigma_2 \in L(v)$. As $(i_1, i_2) \in d \leftrightarrow d$, $(\sigma_1, \sigma_2) \in \Psi_e$, which completes the proof.

4 Reducing multigraph (exact) d-to-1 to $(d + 1)$-to-1 conjecture

For the version of d-to-1 conjecture where we only require the constraint maps to be at most d-to-1, the d-to-1 conjecture trivially implies the $(d + 1)$-to-1 conjecture. O'Donnell and Wu [16] remark that no such reduction appears to be known for the exact d-to-1 conjecture. Here we prove that the exact d-to-1 conjecture implies the exact $(d+1)$-to-1 conjecture when the underlying Label Cover instances are allowed to have parallel edges. We remark that multigraph version of exact d-to-1 conjecture, which is implied by the simple graph version, also suffices for our reduction to graph coloring (and indeed all known reductions from d-to-1 Label Cover).

Let $G = ((V = X \cup Y, E), (dR, R), \Psi)$ be a Label Cover instance such that every constraint is of d-to-1 structure. We reduce it to $G' = ((V = X \cup Y, E'), ((d + 1)R, R), \Psi')$ such that

1. If G is satisfiable, G' is satisfiable as well.
2. If every labeling violates at least ϵ fraction of constraints in G, then every labeling violates at least $\epsilon' = 2\epsilon$ fraction of constraints in G'.

Reduction

We first change the label set of X from $[dR]$ to $[(d + 1)R]$. For every constraint ψ in G between nodes $u \in X$ and $v \in Y$, we replace it with R constraints $\psi_1, \psi_2, \ldots, \psi_R$ between u and v in the following way: the relation between old labels is the same as ψ i.e. when $x \leq dR$, $(x, y) \in \psi_j$ for $j = 1, 2, \ldots, R$ if and only if $(x, y) \in \psi$. When $x > dR$, $(x, y) \in \psi_j$ if and only if R divides $(x + j - y)$. This ensures that each new label is mapped to a different label in each of the R new constraints. The constraints are clearly of $(d + 1) - to - 1$ form.

Completeness

If there is a labeling satisfying all the constraints of G, the same labeling satisfies all the constraints in G' as well.

Soundness

Suppose that there is no labeling satisfying at least ϵ fraction of constraints in G. Note that this implies that R is at least $\frac{1}{\epsilon}$ as there is always a labeling satisfying at least $\frac{1}{R}$ fraction of constraints: fix a labeling to the vertices on the left, and assign a label to the vertices in R uniformly at random from $[R]$. We claim that there is no labeling satisfying more than 2ϵ fraction of constraints in G'. Consider an arbitrary labeling of G, $\sigma : V \to [(d + 1)R]$. We can divide the set of edges E' of G' into two parts: the edges (u, v) such that $\sigma(u) \leq dR$ and the edges (u, v) such that $\sigma(u) > dR$. Let the set of first type of edges where the left vertex is assigned the new label be denoted by E_1, and the set of second type of edges be denoted by E_2. In E_1, the fraction of constraints that can be satisfied by σ is at most $\frac{1}{R} \leq \epsilon$. Note that we can get a labeling σ' of G by replacing labels of vertices in X with label greater than dR with an arbitrary label in $[dR]$, and keeping rest of the labels intact. For the edges in E_2, the labelings σ and σ' coincide. As σ' can satisfy at most ϵ fraction of constraints of G, σ can only satisfy at most ϵ fraction of overall edges in E'. Thus, overall σ satisfies at most $\epsilon + \frac{1}{R} \leq 2\epsilon$ fraction of constraints in E', which proves the required soundness claim.

5 Conclusion

In this paper, we prove that the d-to-1 conjecture, for arbitrarily large d, implies the NP-hardness of the longstanding and elusive problem of coloring 3-colorable graphs with constantly many colors. Note that the d-to-1 conjecture requires the soundness parameter to be arbitrarily small, independent of d. Currently, the best NP-hardness of d-to-1 Label Cover achieves a soundness of $d^{-\Omega(1)}$. This follows from the PCP Theorem [1, 2] combined with Raz's parallel repetition [18]. However, this does not yield any explicit constant in the exponent, obtaining which is an interesting open question. One can also investigate whether improving the soundness of d-to-1 Label Cover to something quantitatively much stronger, say inverse exponential in d, would have some implications for inapproximability of graph coloring.

References

1 Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM*, 45(3):501–555, 1998. doi:10.1145/278298.278306.

2 Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: A new characterization of NP. *Journal of the ACM*, 45(1):70–122, 1998. doi:10.1145/273865.273901.

3 Jakub Bulín, Andrei Krokhin, and Jakub Opršal. Algebraic approach to promise constraint satisfaction. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2019, pages 602–613, 2019.

4 J Csima and B.N Datta. The DAD theorem for symmetric non-negative matrices. *Journal of Combinatorial Theory, Series A*, 12(1):147–152, 1972.

5 Irit Dinur, Subhash Khot, Guy Kindler, Dor Minzer, and Muli Safra. On non-optimally expanding sets in grassmann graphs. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 940–951, 2018.

6 Irit Dinur, Subhash Khot, Guy Kindler, Dor Minzer, and Muli Safra. Towards a proof of the 2-to-1 games conjecture? In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 376–389, 2018.

7 Irit Dinur, Elchanan Mossel, and Oded Regev. Conditional hardness for approximate coloring. *SIAM J. Comput.*, 39(3):843–873, 2009.

8 Irit Dinur and Samuel Safra. On the hardness of approximating minimum vertex cover. *Annals of Mathematics*, 162(1):439–485, 2005.

9 Ken-Ichi Kawarabayashi and Mikkel Thorup. Coloring 3-colorable graphs with less than $n^{1/5}$ colors. *J. ACM*, 64(1), March 2017.

10 Subhash Khot. On the power of unique 2-prover 1-round games. In *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, pages 767–775, 2002. doi:10.1145/509907.510017.

11 Subhash Khot, Guy Kindler, Elchanan Mossel, and Ryan O'Donnell. Optimal inapproximability results for MAX-CUT and other 2-variable CSPs? *SIAM J. Comput.*, 37(1):319–357, 2007. doi:10.1137/S0097539705447372.

12 Subhash Khot, Dor Minzer, and Muli Safra. On independent sets, 2-to-2 games, and Grassmann graphs. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 576–589, 2017.

13 Subhash Khot, Dor Minzer, and Muli Safra. Pseudorandom sets in Grassmann graph have near-perfect expansion. In *59th IEEE Annual Symposium on Foundations of Computer Science*, pages 592–601, 2018.

14 Andrei Krokhin, Jakub Opršal, Marcin Wrochna, and Stanislav Živný. Topology and adjunction in promise constraint satisfaction, 2020. arXiv:2003.11351.

15 Elchanan Mossel, Ryan O'Donnell, and Krzysztof Oleszkiewicz. Noise stability of functions with low influences: Invariance and optimality. *Annals of Mathematics*, 171(1):295–341, 2010. doi:10.4007/annals.2010.171.295.

16 Ryan O'Donnell and Yi Wu. Conditional hardness for satisfiable 3-CSPs. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing*, pages 493–502, 2009.

17 S. Poljak and V. Rödl. On the arc-chromatic number of a digraph. *Journal of Combinatorial Theory, Series B*, 31(2):190–198, 1981. doi:10.1016/S0095-8956(81)80024-X.

18 Ran Raz. A parallel repetition theorem. *SIAM J. Comput.*, 27(3):763–803, 1998.

19 Richard Sinkhorn and Paul Knopp. Concerning nonnegative matrices and doubly stochastic matrices. *Pacific Journal of Mathematics*, 21(2):343–348, 1967.

Feasible Interpolation for Polynomial Calculus and Sums-Of-Squares

Tuomas Hakoniemi
Universitat Politècnica de Catalunya, Barcelona, Spain

Abstract

We prove that both Polynomial Calculus and Sums-of-Squares proof systems admit a strong form of feasible interpolation property for sets of polynomial equality constraints. Precisely, given two sets $P(x, z)$ and $Q(y, z)$ of equality constraints, a refutation Π of $P(x, z) \cup Q(y, z)$, and any assignment a to the variables z, one can find a refutation of $P(x, a)$ or a refutation of $Q(y, a)$ in time polynomial in the length of the bit-string encoding the refutation Π. For Sums-of-Squares we rely on the use of Boolean axioms, but for Polynomial Calculus we do not assume their presence.

2012 ACM Subject Classification Theory of computation → Proof complexity

Keywords and phrases Proof Complexity, Feasible Interpolation, Sums-of-Squares, Polynomial Calculus

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.63

Category Track A: Algorithms, Complexity and Games

Funding Partially funded by European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme, grant agreement ERC-2014-CoG 648276 (AUTAR) and MICCIN grant TIN2016-76573-C2-1P (TASSAT3).

Acknowledgements I want to thank Albert Atserias for thorough comments on a preliminary versions of this work. I would also want to thank the anonymous reviewers who helped vastly to improve the presentation of this paper.

1 Introduction

In this paper we consider the proof systems Polynomial Calculus (PC) and Sums-of-Squares (SOS). PC is a proof system that is used to derive polynomial equalities from a set of polynomial equality constraints in a step-by-step fashion similar to traditional logical proof systems. A PC proof is a compact certificate that the proved polynomial is in the ideal generated by the constraints. PC was introduced by Clegg et al. [4].

Sums-of-Squares proof system on the other hand is a proof system used to derive polynomial inequalities from a set of polynomial constraints. As a proof system Sums-of-Squares was first investigated by Grigoriev and Vorobjov in [6], but it has its roots in semialgebraic geometry and combinatorial optimization. We refer the reader to [10] for a thorough presentation of these connections.

Feasible interpolation was introduced by Krajíček in [9] as a framework to prove lengths-of-proofs lower bounds for propositional proof system from lower bounds on Boolean circuits or other computational models. The feasible interpolation has been applied to prove lower bounds for example for Resolution [9] and Cutting Planes [12] from lower bounds on monotone Boolean and real circuits, respectively. On the negative side Krajíček and Pudlák showed in [8] that Extended Frege does not admit feasible interpolation with respect to Boolean circuits unless RSA is not secure against P/poly. This was later extended to Frege in [3] and to bounded depth Frege in [2] under other cryptographic assumptions.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 63; pp. 63:1–63:14
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

We will prove here feasible interpolation for both PC and SOS for equality constrains $P(x, y)$ and $Q(y, z)$ in disjoint sequences x, y and z of variables. We show that for both proof systems given a refutation Π of $P(x, z) \cup Q(y, z)$ and an assignment a to the variables z, one can in polynomial time in the bit-complexity of Π find a refutation of $P(x, a)$ or a refutation of $Q(y, a)$. Previously, a form of feasible interpolation for PC was proven for degree bounded PC-refutations by Pudlák and Sgall [13]. We know of no previous results on feasible interpolation for Sums-of-Squares proofs.

From [4] we know that PC is degree-automatable: any degree d proof can be found in time $n^{O(d)}$. The same is not true in general for SOS, since the coefficients in a small degree proof might be exceedingly large. This was first noted by O'Donnell in [11]. O'Donnell demonstrated a simple system of polynomial constraints that admit a degree 2 proofs of non-negativity, so that every degree 2 proof necessarily has coefficients of exponential bit-complexity. Later the example of [11] was strengthened by Raghavendra and Weitz in [14] by giving a system of constraints over the Boolean cube that have proofs of non-negativity of degree 2, but any proof of degree less than $O(\sqrt{n})$ must have exponential bit-complexity.

In view of these issues on the bit-complexity of SOS, the question arises whether doubly exponential coefficients can pose a problem for feasible interpolation. However we show that we can use the given refutation of $P(x, z) \cup Q(y, z)$ to bound the coefficients appearing in a refutation of $P(x, a)$ or a refutation of $Q(y, a)$.

Our proofs rely on a 'semantic' characterizations of refutations with bounded resources. A standard way to prove lower bounds in proof complexity is to exhibit a 'semantic' object whose existence is in contradiction with the existence of refutations with bounded resources. These include the reduction operators first used in [15] against low-degree PC-refutations, the d-designs first used in [1] against low-degree Nullstellensatz refutations and pseudoexpectations first used in [6] against low-degree Sums-of-Squares refutations. In many cases these objects actually characterize the associated classes of refutations, and thus they, from a logical point of view, give soundness and completeness theorems for resource bounded refutations. If soundness of these characterizations can be used to prove lower bounds, the completeness properties are useful in establishing upper bounds on proofs as exemplified by the proofs of Theorems 3 and 9 below.

Main results. Let $P(x, z)$ and $Q(y, z)$ be sets of polynomial equations, where x, y and z are disjoint sequences of variables. Our main results are as follows:

- For any finite field $\mathbb{F}$ there is a polynomial time algorithm that given a PC-refutation of $P(x, z) \cup Q(y, z)$, and an assignment a to the variables z, outputs a PC-refutation of $P(x, a)$ or a PC-refutation of $Q(y, a)$. (Theorem 5)
- There is a polynomial time algorithm that given an SOS-refutation of $P(x, z) \cup Q(y, z)$ over the Boolean hypercube and a Boolean assignment a to the variables z outputs an SOS-refutation of $P(x, a)$ over the Boolean hypercube or an SOS-refutation of $Q(y, a)$ over the Boolean hypercube. (Theorem 14)

Proof methods. We study the two systems in two separate parts. Each part follows the same outline. First we define a suitable class of proofs and its semantic counterpart. We define proofs over some fixed set of monomials. The idea is to shift the focus from trying to obtain size-of-proof upper bounds directly to proving the existence of proofs that use only monomials from some small set S. The corresponding semantic operators are then defined on the vector space of all polynomials which are linear combinations of elements of S.

Secondly we prove a feasible disjunction property for the system using the obtained semantic characterizations. Given a refutation of $P(x) \cup Q(y)$, where x and y are disjoint sequences of variables, we can define sets S_x and S_y whose size are polynomial in the size of the given refutation, such that either $P(x)$ has a refutation over S_x or $Q(y)$ has a refutation over S_y.

Finally we argue that the refutations whose existence is guaranteed by the feasible disjunction property can be found in time polynomial in the size of the underlying set of monomials. For PC we give a simple proof search algorithm, and for SOS we use the ellipsoid algorithm to search for a proof after meeting sufficient conditions for polynomial run-time.

2 Preliminaries

2.1 Polynomials and the Boolean Ideal

A monomial is a product of variables. A term over a field $\mathbb{F}$ is a product of a non-zero element of $\mathbb{F}$, called the coefficient of the term, and a monomial. A polynomial is a finite sum of terms, i.e. a finite linear combination of monomials. We write $\mathbb{F}[x]$ for the set of all polynomials over a field $\mathbb{F}$. In particular $\mathbb{R}[x]$ denotes the set of all monomials with real coefficients. For any set S of monomials we denote by $\mathbb{F}[S]$ the set of all linear combinations of monomials from S. For any $p \in \mathbb{R}[x]$ we denote by $\|p\|$ the largest absolute value of a coefficient that appears in p.

For SOS we consider polynomials over n pairs of twin variables $x_1, \dots, x_n, \bar{x}_1, \dots, \bar{x}_n$. The intended meaning is that the variables range over Boolean values $\{0,1\}$ and that a pair of twin variables assumes opposite values. Accordingly we define the Boolean ideal I_n to be the ideal generated by the Boolean axioms $\{x_i^2 - x_i, x_i + \bar{x}_i - 1 : i \in [n]\}$. We write $p \equiv q \mod I_n$ if $p - q \in I_n$.

The Boolean axioms form a Gröbner basis for the Boolean ideal. This is readily seen using the Buchberger's criterion. The important consequence of this for our purposes is that the multivariate division algorithm with respect to the Boolean axioms leaves a unique remainder, and in particular the remainder is 0 if and only if $p \in I_n$. For more information on multivariate division and Gröbner bases, we refer the reader to [5].

2.2 Polynomial Calculus and Sums-of-Squares proofs

Let Q be a set of polynomials over an arbitrary field $\mathbb{F}$. We think of elements of Q as equality constraints $q = 0$. Let p be another polynomial. A PC-proof of p from Q is a sequence $p_1, \dots, p_\ell$ of polynomials such that $p_\ell = p$ and for each $i \in [\ell]$ one of the following hold:

(i) $p_i \in Q$;
(ii) there are $j, k < i$ and $a, b \in \mathbb{F}$ such that $p_i = ap_j + bp_k$;
(iii) there is $j < i$ and a variable x such that $p_i = xp_j$.

A PC-proof of p from Q is a certificate that p is in the ideal generated by Q. A PC-refutation of Q is a PC-proof of 1 from Q.

Let now Q be a set of real polynomials over n pairs of Boolean variables. A Sums-of-Squares proof of non-negativity of p from Q over the Boolean hypercube is a polynomial equality of the form

$$p = \sum_{i \in [k]} r_i^2 + \sum_{q \in Q} t_q q + \sum_{i \in [n]} \left(u_i \left(x_i^2 - x_i \right) + v_i \left(x_i + \bar{x}_i - 1 \right) \right), \tag{1}$$

where r_i, t_q, u_i and v_i are arbitrary real polynomials. An SOS refutation of Q over the Boolean hypercube is a proof of non-negativity of -1. Usually we will simply write the SOS proof (1) as

$$p \equiv \sum_{i\in[k]} r_i^2 + \sum_{q\in Q} t_q q \mod I_n$$

and omit the explicit lifts of the Boolean axioms.

3 Feasible interpolation for Polynomial Calculus

3.1 PC proofs over a set of monomials

Let Q be a set of polynomials over field $\mathbb{F}$ and let S be a set of monomials containing all the monomials in Q and the empty monomial 1. Let $\hat{S} = S \cup xS$, where $xS = \{xm : m \in S \text{ and } x \text{ is a variable}\}$. A PC-proof of p from Q over S is a PC-proof of p from Q, where only monomials from the set $\hat{S}$ appear, and the inference rule p/xp is only applied when $p \in \mathbb{F}[S]$. Denote by $\mathrm{PC}_S(Q)$ the set of all p such that there exists a PC-proof of p from Q over S.

Let now $<$ be a total order on $\hat{S}$ satisfying the following two conditions:

(i) $1 \leq m$ for any $m \in \hat{S}$;

(ii) if $m \in S$ and $m' \in \hat{S} \setminus S$, then $m < m'$.

The leading monomial of a polynomial $p \in \mathbb{F}[\hat{S}]$, denoted $\mathrm{LM}(p)$, is the largest monomial with respect to $<$ that appears in p with a non-zero coefficient. The leading term of a polynomial $p \in \mathbb{F}[\hat{S}]$, denoted $\mathrm{LT}(p)$ is the term, whose underlying monomial is the leading monomial of p.

We say that a term $t \in \mathbb{F}[\hat{S}]$ is S-reducible modulo Q if there is $p \in \mathrm{PC}_S(Q)$ such that $t = \mathrm{LT}(p)$. Otherwise the term is S-irreducible modulo Q. The following lemma shows that any polynomial in $\mathbb{F}[\hat{S}]$ can be uniquely factorized into a provable and an S-irreducible component.

▶ **Lemma 1.** *For any polynomial $p \in \mathbb{F}[\hat{S}]$ there are unique $q \in \mathbb{F}[\hat{S}]$ and $r \in \mathbb{F}[\hat{S}]$ such that*

- *$p = q + r$;*
- *$q \in \mathrm{PC}_S(Q)$;*
- *r is a sum of S-irreducible terms modulo Q.*

Moreover $\mathrm{LT}(p) \geq t$ for each term t in r.

Proof. To prove the existence of such q and r, we construct sequences p_i, q_i, r_i such that

- $p = p_i + q_i + r_i$;
- $q_i \in \mathrm{PC}_S(Q)$;
- r_i is a sum of S-irreducible terms.
- $p_m = 0$ for some m.

Let $p_1 = p$ and $q_1 = r_1 = 0$. For step i, let $\mathrm{LT}(p_i) = t_i$. If t_i is S-reducible as witnessed by $q \in \mathrm{PC}_S(Q)$ let $p_{i+1} = p_i - q$, $q_{i+1} = q_i + q$ and $r_{i+1} = r_i$. On the other hand, if t_i is S-irreducible, let $p_{i+1} = p_i - t_i$, $q_{i+1} = q_i$ and $r_{i+1} = r_i + t_i$.

Now $p_m = 0$ for some m, since the rank of the leading term of p_i decreases at each step. By construction, q_m and r_m satisfy the conditions of the lemma.

To prove the uniqueness of q and r, suppose $p = q + r$ and $p = q' + r'$, i.e. $q - q' = r' - r$. Now $q - q' \in \mathrm{PC}_S(Q)$ and so $r' - r \in \mathrm{PC}_S(Q)$, Hence $\mathrm{LT}(r' - r)$ is not S-irreducible. However, since both r and r' are sums of S-irreducible terms, it follows that $\mathrm{LT}(r' - r) = 0$ and so $r = r'$. Hence also $q = q'$. ◀

Consider now the mapping $R^Q_S \colon \mathbb{F}[\hat{S}] \to \mathbb{F}[\hat{S}]$ that maps each p to the unique sum r of S-irreducible terms modulo Q such that $p - r \in \mathrm{PC}_S(Q)$. The following lemma gathers four basic properties of the mapping.

▶ **Lemma 2.** *The following hold.*

(i) *If there is no refutation of Q over S, then $R^Q_S(1) = 1$;*
(ii) *R^Q_S is a linear function;*
(iii) *$R^Q_S(R^Q_S(p)) = R^Q_S(p)$ for any polynomial $p \in \mathbb{F}[\hat{S}]$;*
(iv) *$R^Q_S(xm) = R^Q_S(xR^Q_S(m))$ for any $m \in S$ and any variable x.*

Proof. (i) If there is no refutation of Q over S, then, by part (i) of the definition of $<$, the constant polynomial 1 is S-irreducible modulo Q. On the other hand $0 \in \mathrm{PC}_S(Q)$ and so, by the uniqueness of the factorization, $R^Q_S(1) = 1$.

(ii) Firstly, we have that $p - R^Q_S(p), q - R^Q_S(q) \in \mathrm{PC}_S(Q)$, and so $p + q - (R^Q_S(p) + R^Q_S(q)) \in \mathrm{PC}_S(Q)$. Now $R^Q_S(p) + R^Q_S(q)$ is a sum of S-irreducible terms modulo Q and so, by the uniqueness of the factorization, $R^Q_S(p+q) = R^Q_S(p) + R^Q_S(q)$. Similarly $ap - aR^Q_S(p) \in \mathrm{PC}_S(Q)$ and so $R^Q_S(ap) = aR^Q_S(p)$.

(iii) We have that $p - R^Q_S(p), R^Q_S(p) - R^Q_S(R^Q_S(p)) \in \mathrm{PC}_S(Q)$ and so also $p - R^Q_S(R^Q_S(p)) \in \mathrm{PC}_S(Q)$, where $R^Q_S(R^Q_S(p))$ is a sum of S-irreducible terms modulo Q. Hence, again by the uniqueness of the factorization, $R^Q_S(p) = R^Q_S(R^Q_S(p))$.

(iv) Again, we have that $m - R^Q_S(m) \in \mathrm{PC}_S(Q)$. Now, by Lemma 1, each term t in $R^Q_S(m)$ satisfies $t \leq m$. Hence, by part (ii) of the definition of $<$, each t in $R^Q_S(m)$ is in S, and so $R^Q_S(m) \in \mathbb{F}[S]$. Hence also $xm - xR^Q_S(m) \in \mathrm{PC}_S(Q)$. It follows that $R^Q_S(xm) = R^Q_S(xR^Q_S(m))$. ◀

3.2 Feasible disjunction for PC

In this section we prove a feasible disjunction property for Polynomial Calculus using the machinery developed in the previous section. Below $P(x)$ and $Q(y)$ are set of polynomials in disjoint sequences x and y of variables.

For a set of monomials S, and a sequence x of variables, we denote by S_x the projection of S onto the variables x, i.e. $m \in S_x$, if only variables from x appear in m, there is some m', where no variables from x appear and $mm' \in S$.

▶ **Theorem 3.** *Let Π be a PC-refutation of $P(x) \cup Q(y)$, and let S be the set of all monomials appearing in the refutation Π. Then there is a PC-refutation of $P(x)$ over S_x or a PC-refutation of $Q(y)$ over S_y.*

Proof. Suppose towards a contradiction that the conclusion does not hold, and consider the reduction operators $R^{P(x)}_{S_x}$ and $R^{Q(y)}_{S_y}$. Let $S' := \{m_x m_y : m_x \in S_x \text{ and } m_y \in S_y\}$, and define a linear function $R \colon \mathbb{F}[S'] \to \mathbb{F}[S']$ with

$$R(m_x m_y) = R^{P(x)}_{S_x}(m_x) R^{Q(y)}_{S_y}(m_y)$$

for any $m_x m_y \in S'$ and extend linearly.

We claim now that R has the following properties:

(i) $R(1) = 1$;
(ii) $R(p(x,a)) = 0$ for any $p(x,a) \in P(x,a)$;
(iii) $R(q(y,a)) = 0$ for any $q(y,a) \in Q(y,a)$;
(iv) $R(x_i m) = R(x_i R(m))$ if $m \in S$;
(v) $R(y_i m) = R(y_i R(m))$ if $m \in S$.

The item (i) holds, since by Lemma 2(i), $R_{S_x}^{P(x)}(1) = R_{S_y}^{Q(y)}(1) = 1$. It is clear that both (ii) and (iii) hold.

Finally (iv) holds, by Lemma 2, since

$$\begin{aligned} R(x_i m) &= R_{S_x}(x_i m_x) R_{S_y}(m_y) \\ &= R_{S_x}(x_i R_{S_x}(m_x)) R_{S_y}(R_{S_y}(m_y)) \\ &= R(x_i R_{S_x}(m_x) R_{S_y}(m_y)) \\ &= R(x_i R(m)) \end{aligned}$$

The case (v) is proved similarly.

Now the existence of such R is in contradiction with the assumption that in Π there appears only monomials from S. Firstly R is defined for all the polynomial appearing in Π. Secondly, by (ii) and (iii), R maps each axiom in $P(x) \cup Q(y)$ to zero, and, by linearity and (iv) and (v), respects the inference rules in the sense that R maps the consequent of a rule to zero whenever it maps the premises to zero. Hence, by induction on the structure of the refutation, $R(1) = 0$, against (i). ◀

3.3 Proof search over S

In this section we show how to find proofs over a given set S of monomials in time polynomial in $|S|$. We make this claim only for proofs over a finite field $\mathbb{F}$. In order to avoid pathological counterexamples we tacitly assume that the size of S is at least the number of distinct variables in S.

We begin by constructing a basis B for $\mathrm{PC}_S(Q)$. The construction is given by the following algorithm, which is a modification of an algorithm from [4].

Algorithm 1 Proof search over S.

```
Initially $A = Q$ and $B = \emptyset$;
while $A \neq \emptyset$ do
    Pick $p \in A$ and remove it from $A$;
    while $\mathrm{LM}(p) \in \mathrm{LM}(B)$ do
        Let $q \in B$ be such that $\mathrm{LM}(q) = \mathrm{LM}(p)$;
        Let $p \leftarrow p - aq$, where $a$ is such that $\mathrm{LT}(p) = a\mathrm{LT}(q)$;
    end
    If $p \neq 0$, add $p$ to $B$;
    If $p \in \mathbb{F}[S]$, add $xp$ to $A$ for every variable $x$;
end
Output $B$;
```

Now B is a linearly independent set of polynomials, since all elements of B have distinct leading monomials. As all elements of B have distinct leading monomials there is never more than $|S|^3$ elements in A and thus the algorithm halts after polynomially many steps in $|S|$. Hence for any finite field the above algorithm will halt in time polynomial in $|S|$. In the following we prove that B is actually a basis for $\mathrm{PC}_S(Q)$.

▶ **Lemma 4.** *At the end of the above algorithm* $\mathrm{span}(B) = \mathrm{PC}_S(Q)$.

Proof. Clearly each $q \in B$ has a proof from Q over S, and so $\mathrm{span}(B) \subseteq \mathrm{PC}_S(Q)$.

Now suppose $p \in \mathrm{PC}_S(Q)$ and let $p_1, \ldots, p_\ell$ be a PCR proof of p from Q over S. We show by induction on the structure of the proof that $p_i \in \mathrm{span}(B)$ for any $i \in [\ell]$. To see that each axiom is in B, note that $\mathrm{span}(A \cup B)$ can only increase at each stage of the algorithm. Hence, as the algorithm halts with $A = \emptyset$, at the end each axiom is in $\mathrm{span}(B)$. If $p_i = ap_j + bp_k$ for some $j, k < i$, and $p_j, p_k \in \mathrm{span}(B)$, then clearly $p_i \in \mathrm{span}(B)$.

Finally, suppose that $p_i = xp_j$ for some $j < i$ and some variable x. Now $p_j \in \mathbb{F}[S]$, and by induction assumption, $p_j \in \mathrm{span}(B)$. Write $p_j = \sum a_k q_k$ for some $a_k \in \mathbb{F}$ and $q_k \in B$. We claim that $q_k \in \mathbb{F}[S]$ for each k with non-zero a_k. To see this, let m be the maximal monomial that appears in any q_k with a non-zero coefficient. Now m appears in only one of the q_k's, since they all have distinct leading monomials, and so the monomial m has a non-zero coefficient in p_j. Hence $m \in S$, and so $q_k \in \mathbb{F}[S]$ for every k. Now for any k, q_k was added to B and xq_k was added to A at some stage of the algorithm. Now, at that stage $xq_k \in \mathrm{span}(A \cup B)$. However, since the span only increases during the execution of the algorithm, $xq_k \in \mathrm{span}(B)$ at the end of the algorithm. Hence $xp_j \in \mathrm{span}(B)$ at the end of the algorithm. ◀

Now to check whether there is a PC proof of p from Q over S one simply needs to reduce the polynomial p with respect to the basis B. This is easy to do, since all the elements of B have distinct leading monomials. In order to construct the proof, one needs proofs for the basis elements. The construction of these proofs is easily incorporable into the algorithm above.

3.4 Feasible interpolation

Finally as a consequence of Theorem 3 and Section 3.3 we obtain the feasible interpolation property for PC over any finite field. Below $P(x, z)$ and $Q(y, z)$ are two sets of polynomials, where x, y and z are disjoint sequences of variables.

▶ **Theorem 5.** *For any finite field $\mathbb{F}$, there is a polynomial time algorithm that given a PC-refutation of $P(x, z) \cup Q(y, z)$, and an assignment a to the variables z, outputs a PC-refutation of $P(x, a)$ or a PC-refutation of $Q(y, a)$.*

4 Feasible interpolation for Sums-of-Squares

4.1 Bounded SOS proofs over a set of monomials

Let Q be a set of polynomials and let S be a set of monomials that includes all the monomials appearing in Q and the empty monomial 1.

Denote by S^2 the set of all monomials m such that $m = m_1 m_2$, where $m_1, m_2 \in S$. An SOS proof of non-negativity of some $p \in \mathbb{R}[S^2]$ from Q over S is a polynomial equality of the form

$$p \equiv \sum_{i \in [k]} r_i^2 + \sum_{q \in Q} t_q q \mod I_n$$

where $r_i, t_q \in \mathbb{R}[S]$. We write $Q \vdash_S p \geq q$ if there is a proof of non-negativity of $p - q$ from Q over S. The proof is R-bounded if $\|t_q\| \leq R$ for each $q \in Q$. We need to consider explicitly bounded proofs in order to later be able to give a polynomial time proof search algorithm.

We prove first the important fact that every polynomial in $\mathbb{R}[S^2]$ has provable upper bounds over S modulo the Boolean ideal.

▶ **Lemma 6.** *For any $p \in \mathbb{R}[S^2]$ there is $r \in \mathbb{R}_+$ such that*

$$\emptyset \vdash_S r \geq p.$$

Proof. Let first $m \in S$, and let $a \in \mathbb{R}$. We want to show that there is some $b \in \mathbb{R}_+$ such that $Q \vdash_S b \geq am$. If $a < 0$, then $-am \equiv (\sqrt{-a}m)^2 \mod I_n$ and so $Q \vdash_S 0 \geq am$. On the other hand if $a > 0$, then $a - am \equiv (\sqrt{a} - \sqrt{a}m)^2 \mod I_n$, and so $Q \vdash_S a \geq am$.

Let then $m_1, m_2 \in S$ and $a \in \mathbb{R}$. We show again that there is some $b \in \mathbb{R}_+$ such that $Q \vdash_S b \geq am_1m_2$. If $a < 0$, then $-am_1 - 2am_1m_2 - am_2 \equiv (\sqrt{-a}m_1 + \sqrt{-a}m_2)^2 \mod I_n$. On the other hand, by the above paragraph, there are $b_1, b_2 \in \mathbb{R}_+$ such that $Q \vdash_S b_1 \geq -am_1$ and $Q \vdash_S b_2 \geq -am_2$. Hence $Q \vdash_S (b_1 + b_2)/2 \geq am_1m_2$. If $a > 0$, then $am_1 - 2am_1m_2 + am_2 \equiv (\sqrt{a}m_1 - \sqrt{a}m_2)^2 \mod I_n$. Again there are $b_1, b_2 \in \mathbb{R}_+$ such that $Q \vdash_S b_1 \geq am_1$ and $Q \vdash_S b_2 \geq am_2$, and so $Q \vdash_S (b_1 + b_2)/2 \geq am_1m_2$. ◀

Now we define the objects that we consider to be the semantic counterparts of bounded refutations over a set of monomials. Let $\varepsilon > 0$. A linear functional $E \colon \mathbb{R}[S^2] \to \mathbb{R}$ is an ε-pseudoexpectation for Q over S if the following properties hold:

(i) $E(1) = 1$;
(ii) $E(p) = E(q)$ if $p \equiv q \mod I_n$;
(iii) $E(p^2) \geq 0$ for any $p \in \mathbb{R}[S]$;
(iv) $|E(mq)| \leq \varepsilon$ for any $m \in S$ and any $q \in Q$.

The following two lemmas show connections between ε-pseudoexpectations and proofs with bounded coefficients.

▶ **Lemma 7.** *If there is an ε-pseudoexpectation for Q over S, then there is no R-bounded refutation of Q over S for R less than $1/\varepsilon|S||Q|$.*

Proof. Let E be an ε-pseudoexpectation for Q over S, and suppose that

$$-1 \equiv \sum_{i \in [k]} r_i^2 + \sum_{q \in Q} t_q q \mod I_n$$

is a refutation over S with $\|t_q\| < 1/\varepsilon|S||Q|$ for any $q \in Q$. Now $|E(amq)| \leq |a|\varepsilon$ for each $m \in S$, $q \in Q$ and $a \in \mathbb{R}$. Hence $|E(t_q q)| < 1/|Q|$ for each $q \in Q$, and so $|E(\sum_{q \in Q} t_q q)| < 1$. Now applying E to both sides of the refutation we obtain that $-1 \geq \sum_{q \in Q} E(t_q q) > -1$. ◀

▶ **Lemma 8.** *If there is no R-bounded refutation of Q over S, then there is a $(1/R)$-pseudoexpectation for Q over S.*

Proof. Suppose there is no R-bounded refutation of Q over S, and consider the following two sets

$$A := \{p \in \mathbb{R}[S^2] : \emptyset \vdash_S p \geq 0\}$$

and

$$B := \{-1 + \sum_{q \in Q} t_q q : t_q \in \mathbb{R}[S] \text{ and } \|t_q\| \leq R \text{ for every } q \in Q\}.$$

Now, by assumption, A and B are disjoint, A is a convex cone and B is a convex set. Hence, by the hyperplane separation theorem, there is a non-trivial linear functional $L \colon \mathbb{R}[S^2] \to \mathbb{R}$ such that $L(p) \geq 0$ for every $p \in A$, and $L(p') \leq 0$ for every $p' \in B$.

We want to first argue that $L(1) \neq 0$. So suppose towards a contradiction that $L(1) = 0$. By Lemma 6, for any $p \in \mathbb{R}[S^2]$ there is some $R \in \mathbb{R}_+$ such that $\emptyset \vdash_S R \geq p \geq -R$. It follows that $L(R) \geq L(p) \geq L(-R)$, and so $L(p) = 0$ for every $p \in \mathbb{R}[S^2]$ against the non-triviality of L.

Now define $E(p) = L(p)/L(1)$ for any $p \in \mathbb{R}[S^2]$. We claim that E has the desired properties. We prove the last case. By definition, $-1 \pm Rmq \in B$, and so $E(-1 \pm Rmq) \leq 0$ for any $m \in S$ and $q \in Q$. Hence $|E(mq)| \leq 1/R$. ◀

4.2 Feasible disjunction for SOS

In this section we prove a feasible disjunction property for SOS. For a refutation

$$-1 = \sum_{i \in [k]} r_i^2 + \sum_{q \in Q} t_q q + \sum_{i \in [n]} \left(u_i\left(x_i^2 - x_i\right) + v_i\left(x_i + \bar{x}_i - 1\right)\right)$$

the explicit monomials of the refutation are all the monomials appearing in the polynomials r_i, t_q, q, u_i, v_i, $x_i^2 - x_i$ and $x_i + \bar{x}_i - 1$, i.e. the explicit monomials are the monomials that appear in an explicit representation of the refutation.

▶ **Theorem 9.** *Let*

$$\begin{aligned} -1 = \sum_{i \in [k]} r_i^2 + \sum_{p(x) \in P(x)} t_p p(x) + \sum_{q(y) \in Q(y)} t_q q(y) + \\ \sum_{i \in [n]} \left(u_i\left(x_i^2 - x_i\right) + v_i\left(x_i + \bar{x}_i - 1\right)\right) \sum_{i \in [n']} \left(u_i'\left(y_i^2 - y_i\right) + v_i'\left(y_i + \bar{y}_i - 1\right)\right) \end{aligned}$$

be an SOS refutation of $P(x) \cup Q(y)$ with $\|t_p\|, \|t_q\| \leq R$ for every $p(x) \in P(x)$ and $q(y) \in Q(y)$, let S be the set of explicit monomials appearing in the refutation, and let $R' = 2R|P(x) \cup Q(y)||S|$. Then there is a R'-bounded refutation of $P(x)$ over S_x or a R'-bounded refutation of $Q(y)$ over S_y.

Proof. Suppose towards a contradiction that the conclusion does not hold. Then, by Lemma 8, there are $1/R'$-pseudoexpectations for $P(x)$ over S_x and $Q(y)$ over S_y. Now define a linear functional $E\colon \mathbb{R}[S^2] \to \mathbb{R}$ with

$$E(m) = E_x(m_x)E_y(m_y),$$

for $m \in S^2$ and extend linearly. Here m_x and m_y are the projections of the monomial m to variables x and y, respectively. We claim that E has the following properties.

(i) $E(1) = 1$;
(ii) $E(m(x_i^2 - x_i)) = 0$ for any $m \in S$ and any variable x_i;
(iii) $E(m(x_i + \bar{x}_i - 1)) = 0$ for any $m \in S$ and any variable x;
(iv) $E(m(y_i^2 - y_i)) = 0$ for any $m \in S$ and any variable y_i;
(v) $E(m(y_i + \bar{y}_i - 1)) = 0$ for any $m \in S$ and any variable y_i;
(vi) $E(p^2) \geq 0$ for any $p \in \mathbb{R}[S]$;
(vii) $|E(m(p(x))| \leq 1/R'$ for any $m \in S$ and any $p(x) \in P(x)$;
(viii) $|E(m(q(y))| \leq 1/R'$ for any $m \in S$ and any $q(y) \in Q(y)$.

The cases (i)-(v) are easy to see. For (vi), write $p = \sum_{m \in S} a_m m$. Now the matrix $(E_y(m_y m'_y))_{m,m' \in S}$ is positive semidefinite and so there are vectors u su ch that $E_y(m_y m'_y) = \sum_u u_m u_{m'}$. Now we have

$$\begin{aligned} E(p^2) &= \sum_{m,m'} a_m a_{m'} E_x(m_x m'_x) E_y(m_y m'_y) \\ &= \sum_{m,m'} \sum_u a_m u_m a_{m'} u_{m'} E_x(m_x m'_x) \\ &= E_x((\sum_m \sum_u a_m u_m m_x)^2) \geq 0 \end{aligned}$$

Finally (vii) holds, since

$$|E(m(p(x))| = |E_x(m_x(p(x))||E_y(m_y)| \leq 1/R',$$

where the last inequality holds since E_x is an $1/R'$-pseudoexpectation for $P(x)$ over S_x and $|E_y(m_y)| \leq 1$ for all $m \in S$. Case (viii) is proved similarly.

Now the existence of such E is in contradiction with the assumptions about the given refutation of $P(x) \cup Q(y)$. Although the mapping E does not necessarily fulfill the condition (ii) of an ε-pseudoexpectation, as E is defined for all the summands in the given refutation, we reach a contradiction by a similar argument as in Lemma 7. ◀

4.3 Proof search over S with bounded coefficients

In this section we show how to find the bounded refutation, whose existence is guaranteed by Theorem 9, in time polynomial in the size of S and $\log R$. Again we tacitly assume that the size of S is at least the number of distinct variables appearing in S. For proof search we use the ellipsoid algorithm. Before we can apply the algorithm we need to show that we can bound the other coefficients appearing in the proof using the bound on the t_q polynomials.

As a first step we show that we can bound the coefficients appearing in the sum of squares part of a given refutation. The next lemma is a simple special case of the main theorem of [14].

▶ **Lemma 10.** *Let $p \in \mathbb{R}[S^2]$. If there is a proof of non-negativity of p from $\emptyset$ over S, then there are $r_i \in \mathbb{R}[S]$ such that*

$$p \equiv \sum_{i \in [k]} r_i^2 \mod I_n$$

and $\|r_i\|$ is at most polynomial in $2^{\text{poly}(|S|)}$ and $\|p\|$ for any $i \in [k]$.

Proof. The proof is practically the same as the proof of the main theorem of [14] with only small differences. We'll sketch the proof for completeness.

Let $\mathbf{v}_S$ be a vector of all the monomials in S, and let C be a PSD matrix such that $p \equiv \langle C, \mathbf{v}_S \mathbf{v}_S^T \rangle \mod I_n$. Now denote by M_S the averaged matrix $\mathbb{E}_{\alpha \in \{0,1\}^n} \mathbf{v}_S(\alpha) \mathbf{v}_S^T(\alpha)$ over all Boolean assignments. Now, by Lemma 6 of [14], the smallest non-zero eigenvalue δ of M_S is at least $1/2^{\text{poly}(|S|)}$.

Let now $P = \sum uu^T$ be a projection to the zero eigenspace of M_S. Now, for each u, $u^T \mathbf{v}_S$ is zero on all Boolean assignments, and thus $u^T \mathbf{v}_S \equiv 0 \mod I_n$. Hence

$$\begin{aligned} \langle C, \mathbf{v}_S \mathbf{v}_S^T \rangle &\equiv \langle C, (P + P^\perp) \mathbf{v}_S \mathbf{v}_S^T (P + P^\perp) \rangle \mod I_n \\ &\equiv \langle C, P^\perp \mathbf{v}_S \mathbf{v}_S^T P^\perp \rangle \mod I_n \\ &\equiv \langle P^\perp C P^\perp, \mathbf{v}_S \mathbf{v}_S^T \rangle \mod I_n \end{aligned}$$

Let $C' = P^\perp C P^\perp$, so that $p \equiv \langle C', \mathbf{v}_S \mathbf{v}_S^T \rangle$. Now, by taking averages on both sides, we obtain that

$$\mathbb{E}_{\alpha \in \{0,1\}^n}[p(\alpha)] = \langle C', M_S \rangle.$$

Now the left hand side is at most polynomial in $\|p\|$ and $|S|$. On the other hand the right hand side is at least $\delta \mathrm{Tr}(C')$, since every non-zero eigenvalue of M_S is at least δ and the zero eigenspace of C' is included in the zero-eigenspace of M_S. Since the Frobenius norm of C' is bounded by $\mathrm{Tr}(C')$ we have that each entry of C' is at most polynomial in $2^{\mathrm{poly}(|S|)}$ and $\|p\|$. Now let $r_i, i \in [k]$ be such that $\sum_{i \in [k]} r_i^2 = \langle C', \mathbf{v}_S \mathbf{v}_S^T \rangle$. Now each coefficient of r_i is bounded by a polynomial in $2^{\mathrm{poly}(|S|)}$ and $\|p\|$. ◀

▶ **Corollary 11.** *Let $p \in \mathbb{R}[S^2]$. If there is an R-bounded proof of p from Q over S, then there are $r_i \in \mathbb{R}[S]$ such that*

$$p = \sum_{i \in [k']} r_i^2 + \sum_{q \in Q} t_q q \mod I_n,$$

and the absolute value of all the coefficients appearing in each r_i is at most polynomial in $2^{\mathrm{poly}(|S|)}$, R and $\|p\|$.

Proof. If $p \equiv \sum_{i \in [k]} r_i^2 + \sum_{q \in Q} t_q q \mod I_n$, then $p - \sum_{q \in Q} t_q q \equiv \sum_{i \in [k]} r_i^2 \mod I_n$, and the result follows from the previous lemma. ◀

Secondly we need to restrict the search space for the lifts of the Boolean axioms. In our definition of a proof over a set of monomials, we worked over the Boolean ideal, and thus did not restrict the lifts of the Boolean axioms in any way. However since the Boolean axioms form a Gröbner basis for the Boolean ideal, we can show that there is a well-behaved set $\bar{S}$ of monomials computable from S in time polynomial in $|S|$ such that for any $p \in \mathbb{R}[S^2]$ with $p \equiv 0 \mod I_n$ there are $u_i, v_i \in \mathbb{R}[\bar{S}]$ such that

$$p = u_i(x_i^2 - x_i) + v_i(x_i + \bar{x}_i - 1)$$

To see this consider any monomial ordering $<$ such that B_n forms a Gröbner basis for I_n with respect to $<$, and define the set S_m for any monomial m with the following algorithm.

Algorithm 2 Construction of the set S_m.

```
Initially $I = \{m\}$ and $S_m = \emptyset$;
while leading monomial of some Boolean axiom divides $\mathrm{LM}(I)$ do
    Let $p$ be the first Boolean axiom such that $\mathrm{LM}(p)$ divides $\mathrm{LM}(I)$;
    Let $m'$ be such that $\mathrm{LM}(I) = m'\mathrm{LM}(p)$;
    Let $p' = m - m'p$;
    Add $m'$ to $S_m$;
    Add all the monomials in $p'$ to $I$;
end
Output $S_m$;
```

The runtime of the above algorithm is polynomial in the degree of m. Now define $\bar{S} = \bigcup_{m \in S^2} S_m$. Now, if S is a set of multilinear monomials, set $\bar{S}$ can be computed in time polynomial in $|S|$.

▶ **Lemma 12.** *For each $p \in \mathbb{R}[S^2]$ such that $p \equiv 0 \mod I_n$ there are $u_i, v_i \in \mathbb{R}[\bar{S}]$ such that*

$$p = \sum_{i \in [n]} (u_i(x_i^2 - x_i) + v_i(x_i + \bar{x}_i - 1)).$$

Moreover the absolute value of the coefficients in u_i and v_i is bounded by a polynomial in $\|p\|$, $|S|$ and the degree of p.

Proof. The proof follows since, as the Boolean axioms form a Gröbner basis for I_n with respect to $<$, we have that $p \equiv 0 \mod I_n$ if and only if p reduces to 0 with the multivariate division algorithm with respect to the monomial ordering $<$. The multivariate division algorithm will construct u_i and v_i that are linear combinations of monomials from $\bar{S}$. The last part follows from the fact that the algorithm halts after polynomially many steps in the degree of p and $|S|$. ◀

Now as a corollary to Corollary 11 and Lemma 12 we have the following

▶ **Corollary 13.** *Let $p \in \mathbb{R}[S^2]$. If there is an R-bounded proof of p from Q over S, then there are $r_i \in \mathbb{R}[S]$ and $u_i, v_i \in \mathbb{R}[\bar{S}]$ such that*

$$p = \sum_{i \in [k']} r_i^2 + \sum_{q \in Q} t_q q + \sum_{i \in [n]} \left(u_i \left(x_i^2 - x_i \right) + v_i \left(x_i + \bar{x}_i - 1 \right) \right),$$

and the absolute value of all the coefficients appearing in each s_i, u_i and v_i is at most polynomial in $2^{\mathrm{poly}(|S|)}$, R and $\|p\|$.

Now the existence of a proof given by Corollary 13 can be expressed as feasibility of a set of linear and semidefinite constraints over explicitly bounded variables and so we can find an approximate solution to the set of constraints in polynomial time using the ellipsoid algorithm. We will sketch some details below. For details on ellipsoid algorithm see [7].

For each $q \in Q$, let $\bar{q} \in \mathbb{R}^S$ such that $\bar{q}^T \mathbf{v}_S = q$, and introduce a vector x_q of variables. Similarly for each $m, m' \in S$, introduce a variable $x_{m,m'}$. In addition, introduce variables y_{m,x_i^2}, y_{m,x_i} for each i and $m \in \bar{S}$ and variables $z_{m,x_i}, z_{m,\bar{x}_i}, z_{m,i}$ for each $i \in [n]$ and $m \in \bar{S}$ Now let $p = \sum_{k \in S^2} a_k k$ and introduce for every $k \in S^2$ a constraint $C_k = a_k$, where

$$C_k = \sum_{\substack{m,m' \in S \\ mm' = k}} (x_{m,m'} + \sum_{q \in Q} \bar{q}_m x_{q,m'}) \\ + \sum_{i \in [n]} (\sum_{\substack{m, \in \bar{S} \\ mx_i^2 = k}} y_{m,x_i^2} + \sum_{\substack{m \in \bar{S} \\ mx_i = k}} (z_{m,x_i} - y_{m,x_i}) + \sum_{\substack{m \in \bar{S} \\ m\bar{x}_i = k}} z_{m,\bar{x}_i} - \sum_{\substack{m \in \bar{S} \\ m = k}} z_{m,i}).$$

Let X be the matrix $X_{m,m'} = x_{m,m'}$, and add the constraint $X \succeq 0$. Finally add the bounding constraints $-R' \leq x \leq R'$ for each variable for R' of magnitude polynomial in $2^{\mathrm{poly}(|S|)}$, R and $\|p\|$. Now any feasible solution gives a proof of p from Q with all coefficients bounded by R' and vice versa.

For $\varepsilon > 0$, an ε-relaxation of the above constraints is the set of constraints $|C_k - a_k| \leq \varepsilon$, $X \succeq 0$ and $-R' - \varepsilon \leq x \leq R' + \varepsilon$ for every variable x. Now if there is a feasible solution for the original set of constraints, the set of solutions of the ε-relaxation has volume at least $1/2^{\mathrm{poly}(\log(1/\varepsilon),|S|)}$.

Choose now $\varepsilon = 1/2^{\mathrm{poly}(|S|)}$. Now the ellipsoid method can find a feasible solution to the ε-relaxation in time polynomial in $|S|$, $\log R$ and $\log \|p\|$. Any such solution translates into a polynomial $p + q$, where $\|q\| \leq \varepsilon$. Now for each am that appears in q, define q_m as follows: if $a > 0$ let $q_m = a(1-m)^2$, and if $a < 0$ let $q_m = -a(m)^2$. Now adding all q_m to $p + q$ gives Sums-of-Squares proof of $p - \varepsilon'$ for some $\varepsilon' = 1/2^{\mathrm{poly}(|S|)}$.

4.4 Feasible interpolation

Finally we obtain the feasible interpolation property for SOS as a corollary to Theorem 9 and section 4.3. For the theorem below $P(x,z)$ and $Q(y,z)$ are sets of multilinear polynomials over Boolean variables, where x, y and z are disjoint sequences of variables.

▶ **Theorem 14.** *Let $P(x,z)$ and $Q(y,z)$ be sets of multilinear polynomials. There is a polynomial time algorithm that given an SOS-refutation of $P(x,z) \cup Q(y,z)$ and an assignment a to the variables z outputs an SOS-refutation of $P(x,a)$ or an SOS-refutation of $Q(y,a)$.*

5 Concluding remarks

We have seen that both Polynomial Calculus and Sums-of-Squares admit a strong form of feasible interpolation. Using similar methods we can also prove that Sherali-Adams proof system admits equally strong feasible interpolation property. The proof can be obtained by a simple modification of the proof for Sums-of-Squares. The proof is actually considerably simpler since the problem of too large coefficients does not appear with Sherali-Adams proofs.

Sums-of-Squares proofs cannot admit monotone feasible interpolation, since the Clique-Coloring formulas have small Sums-of-Squares refutations. Pudlák and Sgall prove in [13] that degree bounded Polynomial Calculus admits monotone feasible interpolation with respect to monotone polynomial programs. An interesting open question is whether one can prove monotone feasible interpolation for Polynomial Calculus with respect to monotone circuits.

We only prove feasible interpolation for SOS for sets of equality constraints. If there are inequality constraints, we can only prove a feasible disjunction property with respect to monomial size: if there is a refutation of $P(x,z) \cup Q(y,z)$ of monomial size s, then for any a there is a refutation of $P(x,a)$ or a refutation of $Q(y,a)$ of monomial size $O(s)$. The problem is that we don't have nice counterparts of the ε-pseudoexpectations when we add inequality constraints.

Finally we want to emphasize that although we proved the feasible interpolation for Sums-of-Squares only over the $\{0,1\}$-values, importantly the argument works also for Boolean values over the ± 1 basis.

References

1 Paul Beame, Stephen A. Cook, Jeff Edmonds, Russell Impagliazzo, and Toniann Pitassi. The relative complexity of NP search problems. In Frank Thomson Leighton and Allan Borodin, editors, *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing, 29 May-1 June 1995, Las Vegas, Nevada, USA*, pages 303–314. ACM, 1995. doi:10.1145/225058.225147.

2 Maria Luisa Bonet, Carlos Domingo, Ricard Gavaldà, Alexis Maciel, and Toniann Pitassi. Non-automatizability of bounded-depth frege proofs. *Computational Complexity*, 13(1-2):47–68, 2004. doi:10.1007/s00037-004-0183-5.

3 Maria Luisa Bonet, Toniann Pitassi, and Ran Raz. On interpolation and automatization for frege systems. *SIAM J. Comput.*, 29(6):1939–1967, 2000. doi:10.1137/S0097539798353230.

4 Matthew Clegg, Jeff Edmonds, and Russell Impagliazzo. Using the groebner basis algorithm to find proofs of unsatisfiability. In Gary L. Miller, editor, *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pages 174–183. ACM, 1996. doi:10.1145/237814.237860.

5 David A. Cox, John Little, and Donal O'Shea. *Ideals, Varieties, and Algorithms*. Undergraduate Texts in Mathematics. Springer, fourth edition, 2015. doi:10.1007/978-3-319-16721-3.

6 Dima Grigoriev. Linear lower bound on degrees of positivstellensatz calculus proofs for the parity. *Theor. Comput. Sci.*, 259(1-2):613–622, 2001. doi:10.1016/S0304-3975(00)00157-2.

7 Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2 of *Algorithms and Combinatorics*. Springer, 1988. doi:10.1007/978-3-642-97881-4.

8 Jan Krajíček and Pavel Pudlák. Some consequences of cryptographical conjectures for $s^1{}_2$ and EF. *Inf. Comput.*, 140(1):82–94, 1998. doi:10.1006/inco.1997.2674.

9 Jan Krajíček. Interpolation theorems, lower bounds for proof systems, and independence results for bounded arithmetic. *The Journal of Symbolic Logic*, 62(2):457–486, 1997. URL: http://www.jstor.org/stable/2275541.

10 Monique Laurent. Sums of squares, moment matrices and optimization over polynomials. In Mihai Putinar and Seth Sullivant, editors, *Emerging Applications of Algebraic Geometry*, pages 157–270. Springer New York, New York, NY, 2009. doi:10.1007/978-0-387-09686-5_7.

11 Ryan O'Donnell. SOS Is Not Obviously Automatizable, Even Approximately. In Christos H. Papadimitriou, editor, *8th Innovations in Theoretical Computer Science Conference (ITCS 2017)*, volume 67 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 59:1–59:10, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ITCS.2017.59.

12 Pavel Pudlák. Lower bounds for resolution and cutting plane proofs and monotone computations. *J. Symb. Log.*, 62(3):981–998, 1997. doi:10.2307/2275583.

13 Pavel Pudlák and Jirí Sgall. Algebraic models of computation and interpolation for algebraic proof systems. In Paul Beame and Samuel R. Buss, editors, *Proof Complexity and Feasible Arithmetics, Proceedings of a DIMACS Workshop, New Brunswick, New Jersey, USA, April 21-24, 1996*, volume 39 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 279–295. DIMACS/AMS, 1996. doi:10.1090/dimacs/039/15.

14 Prasad Raghavendra and Benjamin Weitz. On the Bit Complexity of Sum-of-Squares Proofs. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, volume 80 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 80:1–80:13, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ICALP.2017.80.

15 Alexander A. Razborov. Lower bounds for the polynomial calculus. *Computational Complexity*, 7(4):291–324, 1998. doi:10.1007/s000370050013.

Active Learning a Convex Body in Low Dimensions

Sariel Har-Peled

Department of Computer Science, University of Illinois at Urbana-Champaign, IL, USA
sariel@illinois.edu

Mitchell Jones

Department of Computer Science, University of Illinois at Urbana-Champaign, IL, USA
mfjones2@illinois.edu

Saladi Rahul

Dept. of Computer Science and Automation, Indian Institute of Science, Bangalore, India
saladi@iisc.ac.in

Abstract

Consider a set $P \subseteq \mathbb{R}^d$ of n points, and a convex body C provided via a separation oracle. The task at hand is to decide for each point of P if it is in C using the fewest number of oracle queries. We show that one can solve this problem in two and three dimensions using $O(\hexagon_P \log n)$ queries, where $\hexagon_P$ is the largest subset of points of P in convex position. In 2D, we provide an algorithm which efficiently generates these adaptive queries.

Furthermore, we show that in two dimensions one can solve this problem using $O(\circledcirc(P, C) \log^2 n)$ oracle queries, where $\circledcirc(P, C)$ is a lower bound on the minimum number of queries that any algorithm for this specific instance requires. Finally, we consider other variations on the problem, such as using the fewest number of queries to decide if C contains all points of P.

As an application of the above, we show that the discrete geometric median of a point set P in $\mathbb{R}^2$ can be computed in $O(n \log^2 n (\log n \log\log n + \hexagon_P))$ expected time.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Approximation algorithms, computational geometry, separation oracles, active learning

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.64

Category Track A: Algorithms, Complexity and Games

Related Version The full version of this paper is available at `https://arxiv.org/abs/1903.03693`.

Funding *Sariel Har-Peled*: Supported in part by NSF AF award CCF-1907400.
Mitchell Jones: Supported in part by NSF AF award CCF-1907400.

1 Introduction

1.1 Background

Active learning

Active learning is a subfield of machine learning, in which at any time, the learning algorithm is able to query an oracle for the label of a particular data point. One model for active learning is the membership query synthesis model [2]. Here, the learner wants to minimize the number of oracle queries, as such queries are expensive – they usually correspond to either consulting with a specialist, or performing an expensive computation. In this setting, the learning algorithm is allowed to query the oracle for the label of any data point in the instance space. See [21] for a more in-depth survey on the various active learning models.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 64; pp. 64:1–64:17
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

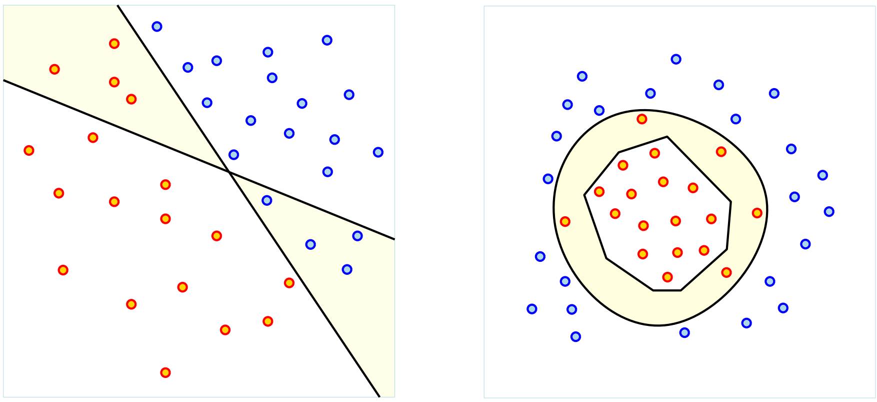


Figure 1.1 The shaded region shows the symmetric difference between the hypothesis and true classifier. (I) Learning halfspaces. (II) Learning arbitrary convex regions.

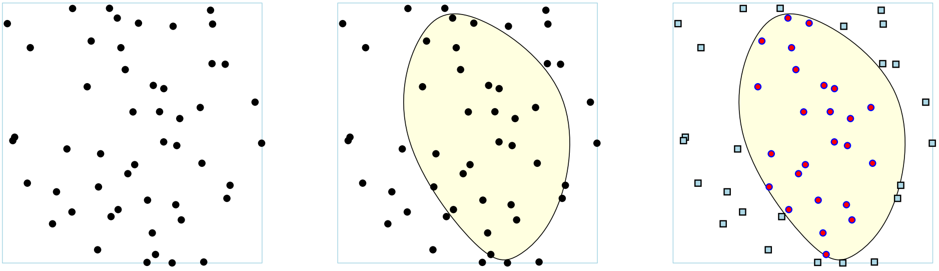
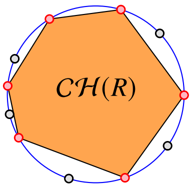


Figure 1.2 (I) A set of points P. (II) The unknown convex body C. (III) Classifying all points of P as either inside or outside C.

PAC learning

A classical approach for learning is using random sampling, where one gets labels for the samples (i.e., in the above setting, the oracle is asked for the labels of all items in the random sample). PAC learning studies the size of the sample needed. For example, consider the problem of learning a halfplane for n points $P \subset \mathbb{R}^2$, given a parameter $\varepsilon \in (0, 1)$. The first stage is to take a labeled random sample $R \subseteq P$. The algorithm computes any halfplane that classifies the sample correctly (i.e., the hypothesis). The misclassified points lie in the symmetric difference between the learned halfplane, and the (unknown) true halfplane, see Figure 1.1. In this case, the error region is a double wedge, and it is well known that its VC-dimension [22] is a constant (at most eight). As such, by the ε-net Theorem [13], a sample of size $O(\varepsilon^{-1} \log \varepsilon^{-1})$ is an ε-net for double wedges, which implies that this random sampling algorithm has at most εn error.

A classical example of a hypothesis class that cannot be learned is the set of convex regions (even in the plane). Indeed, given a set of points P in the plane, any sample $R \subseteq P$ cannot distinguish between the true region being $\mathcal{CH}(R)$ or $\mathcal{CH}(P)$. Intuitively, this is because the hypothesis space in this case grows exponentially in the size of the sample (instead of polynomially).



We stress that the above argument does not necessarily imply these types of hypothesis classes are unlearnable in practice. In general, there are other ways for learning algorithms to handle hypothesis classes with high (or even infinite) VC-dimension (for example, using regularization or assuming there is a large margin around the decision boundary).

Weak ε-nets

Because ε-nets for convex ranges do not exist, an interesting direction to overcome this problem is to define weak ε-nets [13]. A set of points R in the plane, not necessarily a subset of P, is a *weak ε-net* for P if for any convex body C containing at least εn points of P, it also contains a point of R. Matoušek and Wagner [16] gave a weak ε-net construction of size $O(\varepsilon^{-d}(\log \varepsilon^{-1})^{O(d^2 \log d)})$, which is doubly exponential in the dimension. The state of the art is the recent result of Rubin [20], that shows a weak ε-net construction in the plane of size (roughly) $O(1/\varepsilon^{3/2})$. However, these weak ε-nets cannot be used for learning such concepts. Indeed, the analysis above required an ε-net for the symmetric difference of two convex bodies of finite complexity, see Figure 1.1.

PAC learning with additional parameters

If one assumes the input instance obeys some additional structural properties, then random sampling can be used. For example, suppose that the point set P has at most k points in convex position. For an arbitrary convex body C, the convex hull $\mathcal{CH}(P \cap C)$ has complexity at most k. Let $R \subseteq P$ be a random sample, and C' be the learned classifier for R. The region of error is the symmetric difference between C and C'. In particular, since k-vertex polytopes in $\mathbb{R}^d$ have VC-dimension bounded by $O(d^2 k \log k)$ [15], this implies that the error region also has VC-dimension at most $O(d^2 k \log k)$. Hence if R is a random sample of size $O(d^2 k \log k \varepsilon^{-1} \log \varepsilon^{-1})$, the ε-net Theorem [13] implies that this sampling algorithm has error at most εn. However, even for a set of n points chosen uniformly at random from the unit square $[0,1]^2$, the expected number of points in convex position is $O(n^{1/3})$ [1]. Since we want $|R| < n$, this random sampling technique is only useful when ε is larger than $\log^2 n / n^{2/3}$ (ignoring constants).

To summarize the above discussions, random sampling on its own does not seem powerful enough to learn arbitrary convex bodies, even if one allows some error to be made. In this paper we focus on developing algorithms for learning convex bodies in low dimensions, where the algorithms are deterministic and do not make any errors.

1.2 Problem and motivation

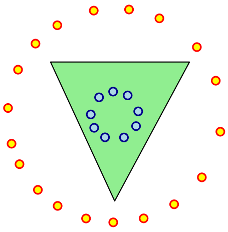
The problem

In this paper, we consider a variation on the active learning problem, in the membership query synthesis model. Suppose that the learner is trying to learn an unknown convex body C in $\mathbb{R}^d$. Specifically, the learner is provided with a set P of n unlabelled points in $\mathbb{R}^d$, and the task is to label each point as either inside or outside C, see Figure 1.2. For a query $q \in \mathbb{R}^d$, the oracle either reports that $q \in C$, or returns a hyperplane separating q and C (as a proof that $q \notin C$). Note that if the query is outside the body, the oracle answer is significantly more informative than just the label of the point. The problem is to minimize the overall number of queries performed.

Hard and easy instances

Note that in the worst case, an algorithm may have to query the oracle for all input points – such a scenario happens when the input points are in convex position, and any possible subset of the points can be the points in the (appropriate) convex body. As such, the purpose here is to develop algorithms that are *instance sensitive* – if the given instance is easy, they work well. If the given instance is hard, they might deteriorate to the naive algorithm that queries all points.

Natural inputs where one can hope to do better, are when relatively few points are in convex position. Such inputs are grid points, or random point sets, among others. However, there are natural instances of the problem that are easy, despite the input having many points in convex position. For example, consider when the convex body is a triangle, with the input point set being $n/2$ points spread uniformly on a tiny circle centered at the origin, while the remaining $n/2$ points are outside the convex body, spread uniformly on a circle of radius 10 centered at the origin. Clearly, such a point set can be fully classified using a sequence of a constant number of oracle queries. See Figure 3.1 for some related examples.



1.3 Additional motivation & previous work

Separation oracles

The use of separation oracles is a common tool in optimization (e.g., solving exponentially large linear programs) and operations research. It is natural to ask what other problems can be solved efficiently when given access to this specific type of oracle. For example, Bárány and Füredi [3] study the problem of computing the volume of a convex body in $\mathbb{R}^d$ given access to a separation oracle.

Other types of oracles

Various models of computation utilizing oracles have been previously studied within the community. Examples of other models include nearest-neighbor oracles (i.e., black-box access to nearest neighbor queries over a point set P) [11], proximity probes (which given a convex polygon C and a query q, returns the distance from q to C) [18], and linear queries. Recently, Ezra and Sharir [7] gave an improved algorithm for the problem of point location in an arrangement of hyperplanes. Here, a *linear query* consists of a point x and a hyperplane h, and outputs either that x lies on h, or else which side of h contains x. Alternatively, their problem can be interpreted as querying whether or not a given point lies in a halfspace h^+. Here, we study the more general problem as the convex body can be the intersection of many halfspaces.

Furthermore, other types of active learning models (in addition membership query model) have also been studied within the learning community, see, for example, [2].

Active learning

As discussed, the problem at hand can be interpreted as active learning a convex body in relation to a set of points P that need to be classified (as either inside or outside the body), where the queries are via a separation oracle. We are unaware of any work directly on this problem in the theory community, while there is some work in the machine learning community that studies related active learning classification problems [6, 9, 21, 14].

For example, Kane et al. [14] study the problem of actively learning halfspaces with access to *comparison queries*. Given a halfspace h^+ to learn, the model has two types of queries: (i) label queries (given $x \in \mathbb{R}^d$, is $x \in h^+$?), and (ii) comparison queries (given $x_1, x_2 \in \mathbb{R}^d$, is x_1 closer to the boundary of h^+ than x_2?). For example, they show that in the plane, one can classify all points using $O(\log n)$ comparison/label queries in expectation.

1.4 Our results

Due to space constraints, not all of the results listed below are included in this version. We refer the reader to the full version of the paper [12] for proofs of missing results.

(A) We develop a greedy algorithm, for points in the plane, which solves the problem using $O(\hexagon_P \log n)$ oracle queries, where $\hexagon_P$ is the largest subset of points of P in convex position. See Theorem 8. It is known that for a random set of n points in the unit square, $\mathbf{E}[\hexagon_P] = \Theta(n^{1/3})$ [1], which readily implies that classifying these points can be solved using $O(n^{1/3} \log n)$ oracle queries. A similar bound holds for the $\sqrt{n} \times \sqrt{n}$ grid. An animation of this algorithm is on YouTube [10]. We also show that this algorithm can be implemented efficiently, using dynamic segment trees, see Lemma 9.
We remark that Kane et al. [14] develop a framework and randomized algorithm for learning a concept C, where the expected number of queries depends near-linearly on a parameter they define as the *inference dimension* [14, Definition III.1] of the concept class. For our problem, one can show that the inference dimension is $O(\hexagon_P)$. As a corollary of their framework, one can obtain a randomized algorithm which solves our problem where the expected number of queries is $O(\hexagon_P \log \hexagon_P \log n)$. Our algorithm shaves a logarithmic factor in the number of queries and is deterministic.

(B) The above algorithm naturally extends to three dimensions, also using $O(\hexagon_P \log n)$ oracle queries. While the proof idea is similar to that of the algorithm in 2D, we believe the analysis in three dimensions is also technically interesting. See Theorem 10.

(C) For a given point set P and convex body C, we define the separation price $\circledcirc(P, C)$ of an instance (P, C), and show that any algorithm classifying the points of P in relation to C must make at least $\circledcirc(P, C)$ oracle queries (Lemma 11).
As an aside, we show in [12] that when P is a set of n points chosen uniformly at random from the unit square and C is a (fixed) smooth convex body, $\mathbf{E}[\circledcirc(P, C)] = O(n^{1/3})$, and this bound is tight when C is a disk (our result also generalizes to higher dimensions). For randomly chosen points, the separation price is related to the expected size of the convex hull of $P \cap C$, which is also known to be $\Theta(n^{1/3})$ [23]. We believe this result may be of independent interest/

(D) In Section 3 we present an improved algorithm for the 2D case, and show that the number of queries made is $O(\circledcirc(P, C) \log^2 n)$. This result is $O(\log^2 n)$ approximation to the optimal solution, see Theorem 12.

(E) We consider the extreme scenarios of the problem: Verifying that all points are either inside or outside of C. For each problem we present a $O(\log n)$ approximation algorithm to the optimal strategy. The results are presented in the full version of the paper [12].

(F) Section 4 presents an application of the above results, we consider the problem of minimizing a *convex* function $f : \mathbb{R}^2 \to \mathbb{R}$ over a point set P. Specifically, the goal is to compute $\arg\min_{p \in P} f(p)$. If f and its derivative can be efficiently evaluated at a given query point, then f can be minimized over P using $O(\varhexagon_P \log^2 n)$ queries to f (or its derivative) in expectation. We refer the reader to Lemma 17.

Given a set of n points P in $\mathbb{R}^d$, the discrete geometric median of P is a point $p \in P$ minimizing the function $\sum_{q \in P} \|p - q\|_2$. As a corollary of Lemma 17, we obtain an algorithm for computing the discrete geometric median for n points in the plane. The algorithm runs in $O(n \log^2 n \cdot (\log n \log\log n + \varhexagon_P))$ expected time. See Lemma 18. In particular, if P is a set of n points chosen uniformly at random from the unit square, it is known $\mathbf{E}[\varhexagon_P] = \Theta(n^{1/3})$ [1] and hence the discrete geometric median can be computed in $O(n^{4/3} \log^2 n)$ expected time.

While there has been ample work on approximating the geometric median (recently, Cohen et al. [5] gave a $(1+\varepsilon)$-approximation algorithm to the geometric median in $O(dn \log^3(1/\varepsilon))$ time), we are unaware of any *exact* sub-quadratic algorithm for the discrete case even in the plane.

▶ Remark. Throughout this paper, the model of computation we have assumed is unit-cost real RAM.

2 The greedy algorithm in two and three dimensions

2.1 Preliminaries

For a set of points $P \subseteq \mathbb{R}^2$, let $\mathcal{CH}(P)$ denote the convex hull of P. Given a convex body $C \subseteq \mathbb{R}^d$, two points $p, x \in \mathbb{R}^d \setminus \mathrm{int}(C)$ are **mutually visible**, if the segment px does not intersect $\mathrm{int}(C)$, where $\mathrm{int}(C)$ is the interior of C. We also use the notation $P \cap C = \{p \in P \mid p \in C\}$.

For a point set $P \subseteq \mathbb{R}^d$, a **centerpoint** of P is a point $¢ \in \mathbb{R}^d$, such that for any closed halfspace h^+ containing $¢$, we have $|h^+ \cap P| \geq |P|/(d+1)$. A centerpoint always exists, and it can be computed exactly in $O(n^{d-1} + n \log n)$ time [4].

Let C be a convex body in $\mathbb{R}^d$ and $q \in \mathbb{R}^d$ be a point such that q lies outside C. A hyperplane h **separates** q from C if q lies in the *closed* halfspace h^+ bounded by h, and C is contained in the *open* halfspace h^- bounded by h. This definition allows the separating hyperplane to contain the point q, and will simplify the descriptions of the algorithms.

2.2 The greedy algorithm in 2D

2.2.1 Operations

Initially, the algorithm copies P into a set U of unclassified points. The algorithm is going to maintain an inner approximation $B \subseteq C$. There are two types of updates (Figure 2.1 illustrates the two operations):

(A) `expand`(p): Given a point $p \in C \setminus B$, the algorithm is going to:
  - **(i)** Update the inner approximation: $B \leftarrow \mathcal{CH}(B \cup \{p\})$.
  - **(ii)** Remove (and mark) newly covered points: $U \leftarrow U \setminus B$.

(B) `remove`(ℓ): Given a closed halfplane ℓ^+ such that $\mathrm{int}(C) \cap \ell^+ = \emptyset$, the algorithm marks all the points of $U_\ell = U \cap \mathrm{int}(\ell^+)$ as being outside C, and sets $U \leftarrow U \setminus U_\ell$.

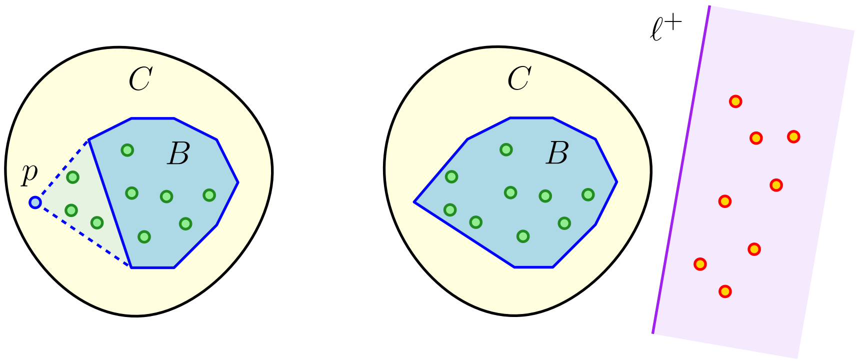


Figure 2.1 (I) Performing `expand`(p), and marking points inside C. (II) Performing `remove`(ℓ), and marking points outside C.

2.2.2 The algorithm

The algorithm repeatedly performs rounds, as described next, until the set of unclassified points is empty.

At every round, if the inner approximation B is empty, then the algorithm sets $U^+ = U$. Otherwise, the algorithm picks a line ℓ that is tangent to B with the largest number of points of U on the other side of ℓ than B. Let ℓ^- and ℓ^+ be the two closed halfspace bounded by ℓ, where $B \subseteq \ell^-$. The algorithm computes the point set $U^+ = U \cap \ell^+$. We have two cases:

(A) Suppose $|U^+|$ is of constant size. The algorithm queries the oracle for the status of each of these points. For every point $p \in U^+$, such that $p \in C$, the algorithm performs `expand`(p). Otherwise, the oracle returned a separating line ℓ, and the algorithm calls `remove`(ℓ^+).

(B) Otherwise, $|U+|$ does not have constant size. The algorithm computes a centerpoint $c \in \mathbb{R}^2$ for U^+, and asks the oracle for the status of c. There are two possibilities:
  - **(i)** If $c \in C$, then the algorithm performs `expand`(c).
  - **(ii)** If $c \notin C$, then the oracle returned a separating line $\hbar$, and the algorithm performs `remove`$(\hbar)$.

2.2.3 Analysis

Let B_i be the inner approximation at the start of the ith iteration, and let z be the first index where B_z is not an empty set. Similarly, let U_i be the set of unclassified points at the start of the ith iteration, where initially $U_1 = U$.

▶ **Lemma 1.** *The number of (initial) iterations in which the inner approximation is empty is $z = O(\log n)$.*

Proof. As soon as the oracle returns a point that is in C, the inner approximation is no longer empty. As such, we need to bound the initial number of iterations where the oracle returns that the query point is outside C. Let $f_i = |U_i|$, and note that $U_1 = P$ and $f_1 = |P| = n$. Let c_i be the centerpoint of U_i, which is the query point in the ith iteration (c_i is outside C). As such, the line separating c_i from C, returned by the oracle, has at least $f_i/3$ points of U_i on the same side as c_i, by the centerpoint property. All of these points get labeled in this iteration, and it follows that $f_{i+1} \leq (2/3)f_i$, which readily implies the claim, since $f_z < 1$, for $z = \lceil \log_{3/2} n \rceil + 1$. ◀

▶ **Definition 2** (Visibility graph)**.** *Consider the graph G_i over U_i, where two points $p, r \in U_i$ are connected $\iff$ the segment pr does not intersect the interior of B_i.*

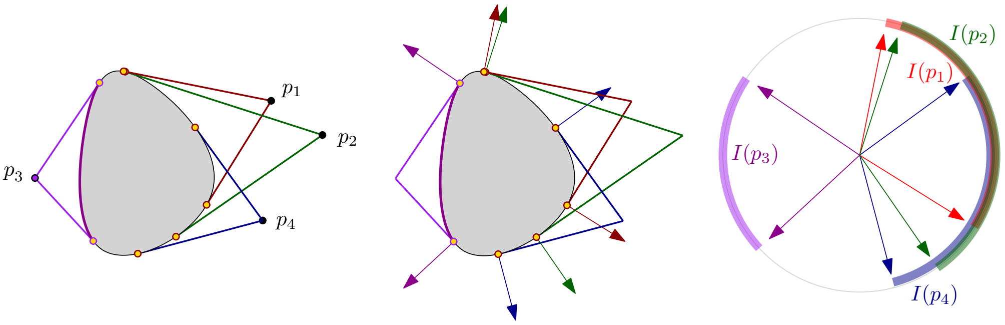


Figure 2.2 Four points and a convex body with their associated circular intervals.

The visibility graph as an interval graph

For a point $p \in U_i$, let $I_i(p)$ be the set of all directions v (i.e., vectors of length 1) such that there is a line perpendicular to v that separates p from B_i. Formally, a line ℓ separates p from B_i, if the interior of B_i is on one side of ℓ and p is on the (closed) other side of ℓ (if $p \in \ell$, the line is still considered to separate the two). Clearly, $I_i(p)$ is a circular interval on the unit circle. See Figure 2.2. The resulting set of intervals is $\mathcal{V}_i = \{I_i(p) \mid p \in U_i\}$. It is easy to verify that the intersection graph of $\mathcal{V}_i$ is G_i. Throughout the execution of the algorithm, the inner approximation B_i grows monotonically, this in turn implies that the visibility intervals shrink over time; that is, $I_i(p) \subseteq I_{i-1}(p)$, for all $p \in P$ and i. Intuitively, in each round, either many edges from G_i are removed (because intervals had shrunk and they no longer intersect), or many vertices are removed (i.e., the associated points are classified).

▶ **Definition 3.** *Given a set $\mathcal{V}$ of objects (e.g., intervals) in a domain D (e.g., unit circle), the* ***depth*** *of a point $p \in D$, is the number of objects in $\mathcal{V}$ that contain p. Let $\operatorname{depth}(\mathcal{V})$ be the maximum depth of any point in D.*

When it is clear, we use $\operatorname{depth}(G)$ to denote $\operatorname{depth}(\mathcal{V})$, where $G = (\mathcal{V}, E)$ is the intersection graph in Definition 2.

First, we bound the number of edges in this visibility graph G and then argue that in each iteration, either many edges of G are discarded or vertices are removed (as they are classified).

▶ **Lemma 4.** *Let $\mathcal{V}$ be a set of n intervals on the unit circle, and let $G = (\mathcal{V}, E)$ be the associated intersection graph. Then $|E| = O(\alpha\omega^2)$, where $\omega = \operatorname{depth}(\mathcal{V})$ and $\alpha = \alpha(G)$ is the size of the largest independent set in G. Furthermore, the upper bound on $|E|$ is tight.*

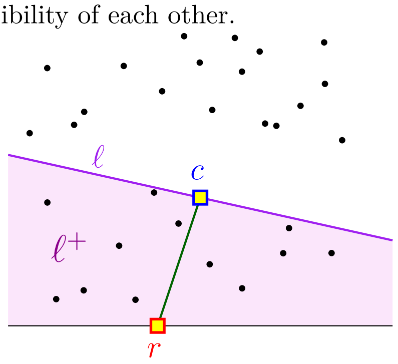
Proof. Let J be the largest independent set of intervals in G. The intervals of J divide the circle into $2|J|$ (atomic) circular arcs. Consider such an arc γ, and let $K(\gamma)$ be the set of all intervals of $\mathcal{V}$ that are fully contained in γ. All the intervals of $K(\gamma)$ are pairwise intersecting, as otherwise one could increase the size of the independent set. As such, all the intervals of $K(\gamma)$ must contain a common intersection point. It follows that $|K(\gamma)| \leq \omega$.

Let $K'(\gamma)$ be the set of all intervals intersecting γ. This set might contain up to 2ω additional intervals (that are not contained in γ), as each such additional interval must contain at least one of the endpoints of γ. Namely, $|K'(\gamma)| \leq 3\omega$. In particular, any two intervals intersecting inside γ both belong to $K'(\gamma)$. As such, the total number of edges contributed by $K'(\gamma)$ to G is at most $\binom{3\omega}{2} = O(\omega^2)$. Since there are $\leq 2\alpha$ arcs under consideration, the total number of edges in G is bounded by $O(\alpha\omega^2)$, which implies the claim.

The lower bound is easy to see by taking an independent set of intervals of size α, and replicating every interval ω times. ◀

▶ **Lemma 5.** *Let P be a set of n points in the plane lying above the x-axis, c be a centerpoint of P, and $S = \binom{P}{2}$ be set of all segments induced by P. Next, consider any point r on the x-axis. Then, the segment cr intersects at least $n^2/36$ segments of S.*

Proof. If the segment cr intersects the segment p_1p_2, for $p_1, p_2 \in P$, then we consider p_1 and p_2 to no longer be mutually visible. It suffices to lower bound the number of pairs of points which lose mutual visibility of each other.



Consider a line ℓ passing through the point c. Let ℓ^+ be the closed halfspace bounded by ℓ containing r. Note that $|P \cap \ell^+| \geq n/3$, since c is a centerpoint of P, and $c \in \ell$. Rotate ℓ around c until there are $\geq n/6$ points on each side of rc in the halfspace ℓ^+. To see why this rotation of ℓ exists, observe that the two halfspaces bounded by the line spanning rc, have zero points on one side, and at least $n/3$ points on the other side – a continuous rotation of ℓ between these two extremes, implies the desired property.

Observe that points in ℓ^+ and on opposite sides of the segment cr cannot see each other, as the segment connecting them must intersect cr. Consequently, the number of induced segments that cr intersects is at least $n^2/36$. ◀

▶ **Lemma 6.** *Let G_i be the intersection graph, in the beginning of the ith iteration, and let $m_i = |E(G_i)|$. After the ith iteration of the greedy algorithm, we have $m_{i+1} \leq m_i - \omega^2/36$, where $\omega = \mathrm{depth}(G_i)$.*

Proof. Recall that in the algorithm $U^+ = U_i \cap \ell^+$ is the current set of unclassified points and ℓ is the line tangent to B_i, where ℓ^+ is the closed halfspace that avoids the interior of B_i and contains the largest number of unlabeled points of U_i. We have that $\omega = |U^+|$.

If a `remove` operation was performed in the ith iteration, then the number of points of U^+ which are discarded is at least $\omega/3$. In this case, the oracle returned a separating line $\hbar$ between a centerpoint c of U^+ and the inner approximation. For the halfspace $\hbar^+$ containing c, we have $t_i = |U^+ \cap \hbar^+| \geq |U^+|/3 \geq \omega/3$. Furthermore, all the points of U^+ are pairwise mutually visible (in relation to the inner approximation B_i). Namely, $m_{i+1} = |E(G_i - (U^+ \cap \hbar^+))| \leq m_i - \binom{t_i}{2} \leq m_i - \omega^2/36$.

If an `expand` operation was performed, the centerpoint c of U^+ is added to the current inner approximation B_i. Let r be a point in $\ell \cap B_i$, and let c_i be the center point of U_i computed by the algorithm. By Lemma 5 applied to r, c and U^+, we have that at least $\omega^2/36$ pairs of points of U^+ are no longer mutually visible to each other in relation to B_{i+1}. We conclude, that at least $\omega^2/36$ edges of G_i are no longer present in G_{i+1}. ◀

▶ **Definition 7.** *A subset of points $X \subseteq P \subseteq \mathbb{R}^2$ are in **convex position**, if all the points of X are vertices of $\mathcal{CH}(X)$ (note that a point in the middle of an edge is not considered to be a vertex). The **index** of P, denoted by $\hexagon_P$, is the cardinality of the largest subset of P of points which are in convex position.*

▶ **Theorem 8.** *Let C be a convex body provided via a separation oracle, and let P be a set of n points in the plane. The greedy classification algorithm performs $O\big((\hexagon_P + 1)\log n\big)$ oracle queries. The algorithm correctly identifies all points in $P \cap C$ and $P \setminus C$.*

Proof. By Lemma 1, the number of iterations (and also queries) in which the inner approximation is empty is $O(\log n)$, and let $z = O(\log n)$ be the first iteration such that the inner approximation is not empty. It suffices to bound the number of queries made by the algorithm after the inner approximation becomes non-empty.

For $i \geq z$, let $G_i = (U_i, E_i)$ denote the visibility graph of the remaining unclassified points U_i in the beginning of the ith iteration. Any independent set in G_i corresponds to a set of points $X \subseteq P$ that do not see each other due to the presence of the inner approximation B_i. That is, X is in convex position, and furthermore $|X| \leq \hexagon_P$.

For $0 \leq t \leq n$, let $s(t)$ be the first iteration i, such that $\mathrm{depth}(G_i) \leq t$. Since the depth of G_i is a monotone decreasing function, this quantity is well defined. An **epoch** is a range of iterations between $s(t)$ and $s(t/2)$, for any parameter t. We claim that an epoch lasts $O(\hexagon_P)$ iterations (and every iteration issues only one oracle query). Since there are only $O(\log n)$ (non-overlapping) epochs till the algorithm terminates, as the depth becomes zero, this implies the claim.

So consider such an epoch starting at $i = s(t)$. We have $m = m_i = |E(G_i)| = O(\hexagon_P t^2)$, by Lemma 4, since $\hexagon_P$ is an upper bound on the size of the largest independent set in G_i. By Lemma 6, as long as the depth of the intervals is at least $t/2$, the number of edges removed from the graph at each iteration, during this epoch, is at least $\Omega(t^2)$. As such, the algorithm performs at most $O(m_i/t^2) = O(\hexagon_P)$ iterations in this epoch, till the maximum depth drops to $t/2$. ◀

2.2.4 Implementing the greedy algorithm

With the use of dynamic segment trees [17] we show that the greedy classification algorithm can be implemented efficiently.

▶ **Lemma 9.** *Let C be a convex body provided via a separation oracle, and let P be a set of n points in the plane. If an oracle query costs time T, then the greedy algorithm can be implemented in $O\big(n \log^2 n \log\log n + T \cdot \hexagon_P \log n\big)$ expected time.*

Proof. The algorithm follows the proof of Theorem 8. We focus on efficiently implementing the algorithm once inner approximation is no longer empty. Let $U \subseteq P$ be the subset of unclassified points. By binary searching on the vertices of the inner approximation B, we can compute the collection of visibility intervals $\mathcal{V}$ for all points in U in $O(|U| \log m) = O(n \log n)$ time (recall that $\mathcal{V}$ is a collection of circular intervals on the unit circle). We store these intervals in a dynamic segment tree $\mathcal{T}$ with the modification that each node v in $\mathcal{T}$ stores the maximum depth over all intervals contained in the subtree rooted at v. Note that $\mathcal{T}$ can be made fully dynamic to support updates in $O(\log n \log\log n)$ time [17].

An iteration of the greedy algorithm proceeds as follows. Start by collecting all points $U^+ \subseteq U$ realizing the maximum depth using $\mathcal{T}$. When $t = |U^+|$, this step can be done in $O(\log n + t)$ time by traversing $\mathcal{T}$. We compute the centerpoint of U^+ in $O(t \log t)$ expected time [4] and query the oracle using this centerpoint. Either points of U are classified (and we delete their associated intervals from $\mathcal{T}$) or we improve the inner approximation. The inner approximation (which is the convex hull of query points inside the convex body C) can be maintained in an online fashion with insert time $O(\log n)$ [19, Chapter 3]. When the inner approximation expands, the points of U^+ have their intervals shrink. As such, we recompute $I(p)$ for each $p \in U^+$ and reinsert $I(p)$ into $\mathcal{T}$.

As defined in the proof of Theorem 8, an epoch is the subset of iterations in which the maximum depth is in the range $[t/2, t]$, for some integer t. During such an epoch, we make two claims:

1. there are $\sigma = O(n)$ updates to $\mathcal{T}$, and
2. the greedy algorithm performs $O(n/t)$ centerpoint calculations on sets of size $O(t)$.

Both of these claims imply that a single epoch of the greedy algorithm can be implemented in expected time $O(\sigma \log n \log\log n + n \log n + T \cdot \bigcirc_P)$. As there are $O(\log n)$ epochs, the algorithm can be implemented in expected time $O(n \log^2 n \log\log n + T \cdot \bigcirc_P \log n)$.

We now prove the first claim. Recall that we have a collection of intervals $\mathcal{V}$ lying on the circle of directions. Partition the circle into k atomic arcs, where each arc contains $t/10$ endpoints of intervals in $\mathcal{V}$. Note that $k = 20n/t = O(n/t)$. For each circular arc γ, let $\mathcal{V}_\gamma \subseteq \mathcal{V}$ be the set of intervals intersecting γ. As the maximum depth is bounded by t, we have that $|\mathcal{V}_\gamma| \leq t + t/10 = 1.1t$. In particular, if $G[\mathcal{V}_\gamma]$ is the induced subgraph of the intersection graph G, then $G[\mathcal{V}_\gamma]$ has at most $\binom{|\mathcal{V}_\gamma|}{2} = O(t^2)$ edges.

In each iteration, the greedy algorithm chooses a point in an arc γ (we say that γ is *hit*) and edges are only deleted from $G[\mathcal{V}_\gamma]$. The key observation is that an arc γ can only be hit $O(1)$ times before all points of γ have depth below $t/2$, implying that it will not be hit again until the next epoch. Indeed, each time γ is hit, the number of edges in the induced subgraph $G[\mathcal{V}_\gamma]$ drops by a constant factor (Lemma 6). Additionally, when $G[\mathcal{V}_\gamma]$ has less than $\binom{t/2}{2}$ edges then any point on γ has depth less than $t/2$. These two facts imply that an arc is hit $O(1)$ times.

When an arc is hit, we must reinsert $|\mathcal{V}_\gamma| = O(t)$ intervals into $\mathcal{T}$. In particular, over a single epoch, the total number of hits over all arcs is bounded by $O(k)$. As such, $\sigma = O(kt) = O(n)$.

For the second claim, each time an arc is hit, a single centerpoint calculation is performed. Since each arc has depth at most t and is hit a constant number of times, there are $O(k) = O(n/t)$ such centerpoint calculations in a single epoch, each costing expected time $O(t \log t)$. ◀

In Section 4 we present an application of the greedy classification algorithm. Namely, we present an efficient algorithm for computing the discrete geometric median of a point set (Lemma 18).

2.3 The greedy algorithm in 3D

Consider the 3D variant of the 2D problem: Given a set of points P in $\mathbb{R}^3$ and a convex body C specified via a separation oracle, the task at hand is to classify, for all the points of P, whether or not they are in C, using the fewest oracle queries possible.

The greedy algorithm naturally extends, where at each iteration i a plane e_i is chosen that is tangent to the current inner approximation B_i, such that it's closed halfspace (which avoids the interior of B_i) contains the largest number of unclassified points from the set U_i.

If the queried centerpoint is outside, the oracle returns a separating plane and as such points can be discarded by the `remove` operation. Similarly, if the centerpoint is reported inside, then the algorithm calls the `expand` and updates the 3D inner approximation B_i.

The idea behind the analysis is similar to Theorem 8. The challenge in analyzing the greedy algorithm in 3D is that mutual visibility between pairs of points is not necessarily lost as the inner approximation grows. Thus we have to analyze mutual visibility between *triples* of points. The analysis considers both the intersection graph G_i between pairs of points, and a new hypergraph H_i, where there is an edge $\{p, q, r\}$ in H_i if the triangle in $\mathbb{R}^3$ formed by the points p, q, r avoids the inner approximation B_i. The main technical ingredient involves bounding the number of edges in H_i by the maximum depth and size of the largest independent set in G_i. Finally, we argue that in each iteration a constant number of edges are deleted from H_i by the centerpoint property. The full details are presented in [12]. We obtain the following result.

▶ **Theorem 10** (Proof in [12]). *Let $C \subseteq \mathbb{R}^3$ be a convex body provided via a separation oracle, and let P be a set of n points in $\mathbb{R}^3$. The greedy classification algorithm performs $O\big((\hexagon_P + 1)\log n\big)$ oracle queries. The algorithm correctly identifies all points in $P \cap C$ and $P \setminus C$.*

3 An instance-optimal approximation in two dimensions

Before discussing the improved algorithm, we present a lower bound on the number of oracle queries performed by any algorithm that classifies all the given points. We then present the the improved algorithm, which matches the lower bound up to a factor of $O(\log^2 n)$.

3.1 A lower bound

Given a set P of points in the plane, and a convex body C, the **outer fence** of P is a closed convex polygon F_{out} with minimum number of vertices, such that $C \subseteq F_{\text{out}}$ and $C \cap P = F_{\text{out}} \cap P$. Similarly, the **inner fence** is a closed convex polygon F_{in} with minimum number of vertices, such that $F_{\text{in}} \subseteq C$ and $C \cap P = F_{\text{in}} \cap P$. Intuitively, the outer fence separates $P \setminus C$ from ∂C, while the inner fence separates $P \cap C$ from ∂C. The **separation price** of P and C is

$$\circledcirc(P, C) = |F_{\text{in}}| + |F_{\text{out}}|,$$

where $|F|$ denotes the number of vertices of a polygon F. See Figure 3.1 for an example.

▶ **Lemma 11.** *Given a point set P and a convex body C in the plane, any algorithm that classifies the points of P in relation to C, must perform at least $\circledcirc(P, C)$ separation oracle queries.*

Proof. Consider the set Q of queries performed by the optimal algorithm (for this input), and split it, into the points inside and outside C. The set of points inside, $Q_{\text{in}} = Q \cap C$ has the property that $Q_{\text{in}} \subseteq C$, and furthermore $\mathcal{CH}(Q_{\text{in}}) \cap P = C \cap P$ – otherwise, there would be a point of $C \cap P$ that is not classified. Namely, the vertices of $\mathcal{CH}(Q_{\text{in}})$ are vertices of a fence that separates the points of P inside C from the boundary of C. As such, we have that $|Q_{\text{in}}| \geq |\mathcal{CH}(Q_{\text{in}})| \geq |F_{\text{in}}|$.

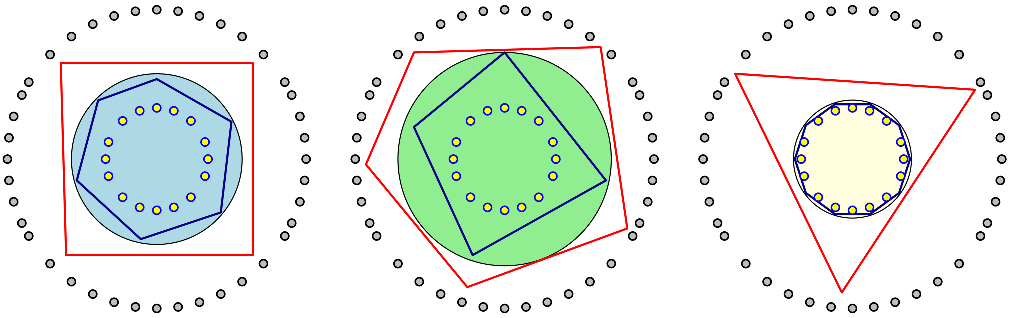


Figure 3.1 The separation price, for the same point set, is different depending on how "tight" the body is in relation to the inner and outer point set.

Similarly, each query in $Q_{\text{out}} = Q \setminus Q_{\text{in}}$ gives rise to a separating halfplane. The intersection of the corresponding halfplanes is a convex polygon H which contains C, and furthermore contains no point of $P \setminus C$. Namely, the boundary of H behaves like an outer fence. As such, we have $|Q_{\text{out}}| \geq |H| \geq |F_{\text{out}}|$.

Combining, we have that $|Q| = |Q_{\text{in}}| + |Q_{\text{out}}| \geq |F_{\text{in}}| + |F_{\text{out}}| = ⊚(P, C)$, as claimed. ◀

▶ Remarks.

1. Naturally the separation price, and thus the proof of the lower bound, generalizes to higher dimensions. See [12].
2. The lower bound only holds for $d \geq 2$. In 1D, the problem can be solved using $O(\log n)$ queries with binary search. The above would predict that any algorithm needs $\Omega(1)$ queries. However it is not hard to argue a stronger lower bound of $\Omega(\log n)$.
3. In [12], we show that when P is a set of n points chosen uniformly at random from a square and C is a smooth convex body, $\mathbf{E}[⊚(P, C)] = O(n^{1/3})$. Thus, when the points are randomly chosen, one can think of $⊚(P, C)$ as growing sublinearly in n.

3.2 A sketch of the improved algorithm

We refer to reader to [12] for a complete description of the improved algorithm in 2D. The idea of the algorithm is conceptually the same as the greedy algorithm of Section 2: at all times a current inner approximation $B \subseteq C$ and the set of unclassified points $U \subseteq P$ are maintained. We define a *pocket* to be a connected region of $\mathcal{CH}(U \cup B) \setminus B$ (see Figure 3.2). The algorithm will repeatedly choose points inside a pocket, and attempt to classify them, while simultaneously dividing the pocket into two smaller pockets. In this way, we improve the inner approximation B every time a pocket is handled. The algorithm continues in this fashion until all points are classified. The analysis involves a careful charging argument. Roughly speaking, whenever a pocket contains a vertex of F_{in} or F_{out}, we can charge the work of creating and splitting the pocket to such a vertex. Otherwise, a pocket contains no vertex from either fence. For such a pocket, we prove that when this pocket was created by the algorithm, all of the points contained in the pocket must lie outside C. When all points inside a pocket are outside C, we argue that they can all be classified as outside after $O(\log n)$ queries by using centerpoints as the oracle queries.

▶ **Theorem 12** (Proof in [12]). *Let C be a convex body provided via a separation oracle, and let P be a set of n points in the plane. The improved classification algorithm performs $O\big(\big[1 + ⊚(P, C)\big] \log^2 n\big)$ oracle queries. The algorithm correctly identifies all points in $P \cap C$ and $P \setminus C$.*

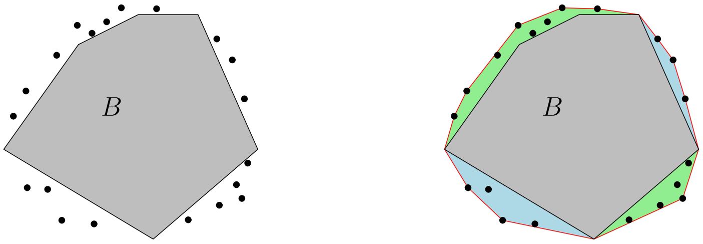


Figure 3.2 Unclassified points and their pockets.

4 Application: Minimizing a convex function

Suppose we are given a set of n points P in the plane and a convex function $f : \mathbb{R}^2 \to \mathbb{R}$. Our goal is to compute the point in P minimizing $\min_{p\in P} f(p)$. Given a point $p \in \mathbb{R}^2$, assuming that we can evaluate f and the derivative of f at p efficiently, we show that the point in P minimizing f can be computed using $O(\hexagon_P \log^2 n)$ evaluations to f or its derivative.

▶ **Definition 13.** *Let $f : \mathbb{R}^d \to \mathbb{R}$ be a convex function. For a number $c \in \mathbb{R}$, define the **level set of** f as $\mathcal{L}_f(c) = \left\{p \in \mathbb{R}^d \;\middle|\; f(p) \le c\right\}$. If f is a convex function, then $\mathcal{L}_f(c)$ is a convex set for all $c \in \mathbb{R}$.*

▶ **Definition 14.** *Let $f : \mathbb{R}^d \to \mathbb{R}$ be a convex (and possibly non-differentiable) function. For a point $p \in \mathbb{R}^d$, a vector $v \in \mathbb{R}^d$ is a **subgradient** of f at p if for all $q \in \mathbb{R}^d$, $f(q) \ge f(p) + \langle v, q - p\rangle$. The **subdifferential** of f at $p \in \mathbb{R}^d$, denoted by $\partial f(p)$, is the set of all subgradients $v \in \mathbb{R}^d$ of f at p.*

It is well known that when the domain for f is $\mathbb{R}^d$ and f is a convex function, then $\partial f(p)$ is a non-empty set of all $p \in \mathbb{R}^d$ (for example, see [8, Chapter 3]).

Let $\alpha = \min_{p\in P} f(p)$. We have that $\mathcal{L}_f(\alpha) \cap P = \{p \in P \mid f(p) = \alpha\}$ and $\mathcal{L}_f(\alpha') \cap P = \varnothing$ for all $\alpha' < \alpha$. Hence, the problem is reduced to determining the smallest value r such that $\mathcal{L}_f(r) \cap P$ is non-empty.

▶ **Lemma 15.** *Let P be a collection of n points in the plane. For a given value r, let $C_r = \mathcal{L}_f(r)$. The set $C_r \cap P$ can be computed using $O(\hexagon_P \log n)$ evaluations to f or its derivative. If T is the time needed to evaluate f or its derivative, the algorithm can be implemented in $O(n \log^2 n \log\log n + T \cdot \hexagon_P \log n)$ expected time.*

Proof. The Lemma follows by applying Theorem 8. Indeed, let $C_r = \mathcal{L}_f(r)$ be the convex body of interest. It remains to design a separation oracle for C_r.

Given a query point $q \in \mathbb{R}^2$, first compute $c = f(q)$. If $c \le r$, then report that $q \in C_r$. Otherwise, $c > r$. In this case, compute some gradient vector v in $\partial f(q)$. Using the vector v, we can obtain a line ℓ tangent to the boundary of $\mathcal{L}_f(c)$ at q. As $\mathcal{L}_f(r) \subseteq \mathcal{L}_f(c)$, ℓ is a separating line for q and C_r, as desired. As such, the number of separation oracle queries needed to determine $C_r \cap P$ is bounded by $O(\hexagon_P \log n)$ by Theorem 8.

The implementation details of Theorem 8 are given in Lemma 9. ◀

The algorithm

Let $\alpha = \min_{p\in P} f(p)$. For a given number $r \ge 0$, set $P_r = \mathcal{L}_f(r) \cap P$. We develop a randomized algorithm to compute α.

Set $P_0 = P$. In the ith iteration, the algorithm chooses a random point $p_i \in P_{i-1}$ and computes $r_i = f(p_i)$. Next, we determine P_{r_i} using Lemma 15. In doing so, we modify the separation oracle of Lemma 15 to store the collection of queries $S_i \subseteq P$ which satisfy $f(s) = r_i$ for all $s \in S_i$. We set $P_{i+1} = P_{r_i} \setminus S_i$. Observe that all points $p \in P_{i+1}$ have $f(p) < r_i$. The algorithm continues in this fashion until we reach an iteration j in which $|P_{j+1}| \leq 1$. If $P_{j+1} = \{q\}$ for some $q \in P$, output q as the desired point minimizing the geometric median. Otherwise $P_{j+1} = \varnothing$, implying that $P_{r_j} = S_j$, and the algorithm outputs any point in the set S_j.

Analysis

We analyze the running time of the algorithm. To do so, we argue that the algorithm invokes the algorithm in Lemma 15 only a logarithmic number of times.

▶ **Lemma 16.** *In expectation, the above algorithm terminates after $O(\log n)$ iterations.*

Proof. Let $V = \{f(p) \mid p \in P\}$ and $N = |V|$. For a number r, define $V_r = \{i \in V \mid i \leq r\}$. Notice that we can reinterpret the algorithm described above as the following random process. Initially set $r_0 = \max_{i \in V} i$. In the ith iteration, choose a random number $r_i \in V_{r_{i-1}}$. This process continues until we reach an iteration j in which $\left|V_{r_j}\right| \leq 1$.

We can assume without loss of generality that $V = \{1, 2, \ldots, N\}$. For an integer $i \leq N$, let $T(i)$ be the expected number of iterations needed for the random process to terminate on the set $\{1, \ldots, i\}$. We have that $T(i) = 1 + \frac{1}{i-1} \sum_{j=1}^{i-1} T(i-j)$, with $T(1) = 0$. This recurrence solves to $T(i) = O(\log i)$. As such, the algorithm repeats this random process $O(\log N) = O(\log n)$ times in expectation. ◀

▶ **Lemma 17.** *Let P be a set of n points in $\mathbb{R}^2$ and let $f : \mathbb{R}^2 \to \mathbb{R}$ be a convex function. The point in P minimizing f can be computed using $O(\hexagon_P \log^2 n)$ evaluations to f or its derivative. The bound on the number of evaluations holds in expectation. If T is the time needed to evaluate f or its derivative, the algorithm can be implemented in $O(n \log^3 n \log\log n + T \cdot \hexagon_P \log^2 n)$ expected time.*

Proof. The result follows by combining Lemma 15 and Lemma 16. ◀

4.1 The discrete geometric median

Let P be a set of n points in $\mathbb{R}^d$. For all $x \in \mathbb{R}^d$, define the function $f(x) = \sum_{q \in P-x} \|x - q\|_2$. The **discrete geometric median** is defined as the point in P minimizing the quantity $\min_{p \in P} f(p)$.

Note that f is convex, as it is the sum of convex functions. Furthermore, given a point p, we can compute $f(p)$ and the derivative of f at p in $O(n)$ time. As such, by Lemma 17, we obtain the following.

▶ **Lemma 18.** *Let P be a set of points in $\mathbb{R}^2$. Then the discrete geometric median of P can be computed in $O(n \log^2 n \cdot (\log n \log\log n + \hexagon_P))$ expected time.*

▶ Remark. For a set of n points P chosen uniformly at random from the unit square, it is known that in expectation $\hexagon_P = \Theta(n^{1/3})$ [1]. As such, the discrete geometric median for such a random set P can be computed in $O(n^{4/3} \log^2 n)$ expected time.

5 Conclusion and open problems

In this paper we have presented various algorithms for classifying points with oracle access to an unknown convex body. As far as the authors are aware, this problem has not been studied within the community previously. However we believe that this is an interesting and natural problem. We now pose some open problems.

(A) Develop a more natural instance-optimal algorithm in 2D which improves upon the $O(\log^2 n)$ approximation. Alternatively, develop algorithms in which the number of queries is parameterized by different functions of the input instance.

(B) An algorithm in 3D which is instance-optimal up to some additional factors (see [12] for the definition of the separation price in higher dimensions).

(C) Any results beyond three dimensions is unknown. The greedy algorithm (Theorem 8 and Theorem 10) easily extends to $\mathbb{R}^d$. However the analysis in higher dimensions will most likely reveal that the algorithm makes (ignoring logarithmic factors) of the order of $\hexagon_P{}^{O(d)}$ queries, which is only interesting when $\hexagon_P$ is much smaller than n.

References

1 G. Ambrus and I. Bárány. Longest convex chains. *Rand. Struct. & Alg.*, 35(2):137–162, 2009. doi:10.1002/rsa.20269.

2 Dana Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1987. doi:10.1007/BF00116828.

3 Imre Bárány and Zoltán Füredi. Computing the volume is difficulte. *Discrete Comput. Geom.*, 2:319–326, 1987. doi:10.1007/BF02187886.

4 T. M. Chan. An optimal randomized algorithm for maximum Tukey depth. In J. Ian Munro, editor, *Proc. 15th ACM-SIAM Sympos. Discrete Algs.* (SODA), pages 430–436. SIAM, 2004. URL: http://dl.acm.org/citation.cfm?id=982792.982853.

5 Michael B. Cohen, Yin Tat Lee, Gary L. Miller, Jakub Pachocki, and Aaron Sidford. Geometric median in nearly linear time. In Daniel Wichs and Yishay Mansour, editors, *Proc. 48th ACM Sympos. Theory Comput.* (STOC), pages 9–21. ACM, 2016. doi:10.1145/2897518.2897647.

6 David A. Cohn, Les E. Atlas, and Richard E. Ladner. Improving generalization with active learning. *Machine Learning*, 15(2):201–221, 1994. doi:10.1007/BF00993277.

7 Esther Ezra and Micha Sharir. A nearly quadratic bound for point-location in hyperplane arrangements, in the linear decision tree model. *Discrete Comput. Geom.*, 61(4):735–755, 2019. doi:10.1007/s00454-018-0043-8.

8 Juan Ferrera. *An introduction to nonsmooth analysis.* Academic Press, Boston, 2013. doi:10.1016/C2013-0-15234-8.

9 Yuhong Guo and Russell Greiner. Optimistic active-learning using mutual information. In *Proc. 20th Int. Joint Conf. on AI* (IJCAI), pages 823–829, 2007. URL: http://ijcai.org/Proceedings/07/Papers/132.pdf.

10 S. Har-Peled, M. Jones, and S. Rahul. An animation of the greedy classification algorithm in 2d. URL: https://www.youtube.com/watch?v=IZX0VQdIgNA.

11 S. Har-Peled, N. Kumar, D. M. Mount, and B. Raichel. Space exploration via proximity search. *Discrete Comput. Geom.*, 56(2):357–376, 2016. doi:10.1007/s00454-016-9801-7.

12 Sariel Har-Peled, Mitchell Jones, and Saladi Rahul. Active learning a convex body in low dimensions. *CoRR*, abs/1903.03693, 2019. arXiv:1903.03693.

13 David Haussler and Emo Welzl. ε-nets and simplex range queries. *Discrete & Computational Geometry*, 2:127–151, 1987. doi:10.1007/BF02187876.

14 Daniel M. Kane, Shachar Lovett, Shay Moran, and Jiapeng Zhang. Active classification with comparison queries. In *Proc. 58th Annu. IEEE Sympos. Found. Comput. Sci.* (FOCS), pages 355–366, 2017. doi:10.1109/FOCS.2017.40.

15 Andrey Kupavskii. The vc-dimension of k-vertex d-polytopes. *CoRR*, abs/2004.04841, 2020. arXiv:2004.04841.

16 J. Matoušek and U. Wagner. New constructions of weak epsilon-nets. In *Proceedings of the nineteenth annual symposium on Computational geometry*, pages 129–135. ACM, 2003.

17 Kurt Mehlhorn and Stefan Näher. Dynamic fractional cascading. *Algorithmica*, 5(2):215–241, 1990. doi:10.1007/BF01840386.

18 F. Panahi, A. Adler, A. F. van der Stappen, and K. Goldberg. An efficient proximity probing algorithm for metrology. In *Int. Conf. on Automation Science and Engineering, CASE 2013*, pages 342–349, 2013. doi:10.1109/CoASE.2013.6653995.

19 Franco P. Preparata and Michael Ian Shamos. *Computational Geometry - An Introduction*. Texts and Monographs in Computer Science. Springer, 1985. doi:10.1007/978-1-4612-1098-6.

20 Natan Rubin. An improved bound for weak epsilon-nets in the plane. In Mikkel Thorup, editor, *Proc. 59th Annu. IEEE Sympos. Found. Comput. Sci.* (FOCS), pages 224–235. IEEE Computer Society, 2018. doi:10.1109/FOCS.2018.00030.

21 Burr Settles. Active learning literature survey. Technical Report #1648, Computer Science, Univ. Wisconsin, Madison, January 2009. URL: https://minds.wisconsin.edu/bitstream/handle/1793/60660/TR1648.pdf?sequence=1&isAllowed=y.

22 V. N. Vapnik and A. Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory Probab. Appl.*, 16:264–280, 1971.

23 Wolfgang Weil, editor. *Random Polytopes, Convex Bodies, and Approximation*, pages 77–118. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. doi:10.1007/978-3-540-38175-4_2.

Node-Connectivity Terminal Backup, Separately-Capacitated Multiflow, and Discrete Convexity

Hiroshi Hirai
Department of Mathematical Informatics,
Graduate School of Information Science and Technology,
The University of Tokyo, Japan
hirai@mist.i.u-tokyo.ac.jp

Motoki Ikeda
Department of Mathematical Informatics,
Graduate School of Information Science and Technology,
The University of Tokyo, Japan
motoki_ikeda@mist.i.u-tokyo.ac.jp

Abstract

The *terminal backup problems* [Anshelevich and Karagiozova, 2011] form a class of network design problems: Given an undirected graph with a requirement on terminals, the goal is to find a minimum cost subgraph satisfying the connectivity requirement. The *node-connectivity terminal backup problem* requires a terminal to connect other terminals with a number of node-disjoint paths. This problem is not known whether is NP-hard or tractable. Fukunaga (2016) gave a 4/3-approximation algorithm based on LP-rounding scheme using a general LP-solver.

In this paper, we develop a combinatorial algorithm for the relaxed LP to find a half-integral optimal solution in $O(m\log(mUA)\cdot \mathrm{MF}(kn, m+k^2n))$ time, where m is the number of edges, k is the number of terminals, A is the maximum edge-cost, U is the maximum edge-capacity, and $\mathrm{MF}(n', m')$ is the time complexity of a max-flow algorithm in a network with n' nodes and m' edges. The algorithm implies that the 4/3-approximation algorithm for the node-connectivity terminal backup problem is also efficiently implemented. For the design of algorithm, we explore a connection between the node-connectivity terminal backup problem and a new type of a multiflow, called a *separately-capacitated multiflow*. We show a min-max theorem which extends Lovász–Cherkassky theorem to the node-capacity setting. Our results build on discrete convex analysis for the node-connectivity terminal backup problem.

2012 ACM Subject Classification Mathematics of computing → Combinatorial optimization

Keywords and phrases terminal backup problem, node-connectivity, separately-capacitated multiflow, discrete convex analysis

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.65

Category Track A: Algorithms, Complexity and Games

Funding *Hiroshi Hirai*: JSPS KAKENHI Grant Number JP17K00029 and JST PRESTO Grant Number JPMJPR192A, Japan.

Acknowledgements We thank the referees for helpful comments.

1 Introduction

Network design problems are central problems in combinatorial optimization. A large number of basic combinatorial optimization problems are network design problems. Examples are spanning tree, matching, TSP, and Steiner networks. They admit a typical formulation of a network design problem: Find a minimum-cost network satisfying given connectivity requirements. The present paper addresses a relatively new class of network design problems,

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 65; pp. 65:1–65:19
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

called *terminal backup problems*. The problem is to find a cheapest subnetwork in which each terminal can send a specified amount of flows to other terminals, i.e., the data in each terminal can be backed up, possibly in a distributed manner, in other terminals.

A mathematical formulation of the terminal backup problem is given as follows. Let $((V,E),S,u,c,a,r)$ be an undirected network, where (V,E) is a simple undirected graph, $S \subseteq V$ ($|S| \geq 3$) is a set of *terminals*, $u : E \to \mathbb{Z}_+$ is a nonnegative edge-capacity function, $c : V \setminus S \to \mathbb{Z}_+$ is a nonnegative node-capacity function, $a : E \to \mathbb{Z}_+$ is a nonnegative edge-cost function, and $r : S \to \mathbb{Z}_+$ is a nonnegative requirement function on terminals. The goal is to find a feasible edge-capacity function x of minimum cost $\sum_{e \in E} a(e)x(e)$. Here an edge-capacity function x is said to be *feasible* if $0 \leq x \leq u$ and each terminal $s \in S$ has a flow from s to $S \setminus \{s\}$, an $\{s\}$–$(S \setminus \{s\})$ flow, of total flow-value $r(s)$ in the network $((V,E),S,x,c)$ capacitated by the edge-capacity x and the node-capacity c.

The original formulation, due to Anshelevich and Karagiozova [1], is uncapacitated (i.e., u, c are infinity), requires x to be integer-valued, and assumes $r(s) = 1$ for all $s \in S$. They showed that an optimal solution can be obtained in polynomial time. Bernáth et al. [2] extended this polynomial time solvability to an arbitrary integer-valued requirement r. For the setting of general edge-capacity (and infinite node-capacity), which we call the *edge-connectivity terminal backup problem (ETB)*, it is unknown whether ETB is NP-hard or tractable.

Fukunaga [8] considered the above setting including both edge-capacity and node-capacity, which we call the *node-connectivity terminal backup problem (NTB)*, and explored intriguing features of its fractional relaxation. The *fractional ETB (FETB)* and *fractional NTB (FNTB)* are LP-relaxations obtained from ETB and NTB, respectively, by relaxing solution x to be real-valued. Fukunaga showed the half-integrality property of FNTB, that is, there always exists an optimal solution that is half-integer-valued. Based on this property, he developed a 4/3-approximation algorithm for NTB by rounding a half-integral (extreme) optimal solution. Moreover, he noticed a useful relationship between FETB and *multicommodity flow (multiflow)*. In fact, a solution of FETB is precisely the edge-support of a multiflow consisting of the $r(s)$ amount of $\{s\}$–$(S \setminus \{s\})$ flow for each $s \in S$. This is a consequence of Lovász–Cherkassky theorem [5, 21] in multiflow theory. In particular, FETB is equivalent to a minimum-cost multiflow problem, which is a variant of the one studied by Karzanov [19, 20] and Goldberg and Karzanov [10].

Utilizing this connection, Hirai [12] developed a combinatorial polynomial time algorithm for FETB and the corresponding multiflow problem. This algorithm uses a max-flow algorithm as a subroutine, and brings a combinatorial implementation of Fukunaga's 4/3-approximation algorithm for ETB, where he used a generic LP-solver (e.g., the ellipsoid method) to obtain a half-integral extreme optimal solution.

Our first contribution is the extension of this result to the NTB setting, implying that the 4/3-approximation algorithm for NTB is also efficiently implemented.

▶ **Theorem 1.** *A half-integral optimal solution of FNTB can be obtained in* $O(m \log(mUA) \cdot \mathrm{MF}(kn, m + k^2 n))$ *time.*

Here $n := |V|$, $m := |E|$, $k := |S|$, $U := \max_{e \in E} u(e)$, and $A := \max_{e \in E} a(e)$, and $\mathrm{MF}(n', m')$ is the time complexity of an algorithm for solving the max-flow problem in the network with n' nodes and m' edges.

As in the ETB case, we explore and utilize a new connection between NTB and a multiflow problem. We introduce a new notion of a *free multiflow with separate node-capacity constraints* or simply a *separately-capacitated multiflow*. Instead of the usual node-capacity constraints, this multiflow should satisfy the separate node-capacity constraints: For each terminal $s \in S$ and each node $i \in V$, the total flow-value of flows connecting s to the other terminals and flowing into i is at most the node capacity $c(i)$.

Our second contribution is a min-max theorem for separately-capacitated multiflows, which extends Lovász–Cherkassky theorem to the node-capacitated setting and implies that a solution of FNTB is precisely the edge-support of a separately-capacitated multiflow. This answers Fukunaga's comment: *how the computation should proceed in the node capacitated setting remains elusive* [8, p. 799].

▶ **Theorem 2.** *The maximum flow-value of a separately-capacitated multiflow is equal to* $(1/2)\sum_{s \in S} \nu_s$*, where* ν_s *is the minimum capacity of an* $\{s\}$*–*$(S \setminus \{s\})$ *cut. Moreover, a half-integral maximum multiflow exists, and it can be found in* $O(n \cdot \mathrm{MF}(kn, m + k^2 n))$ *time.*

Here, a T–T' *cut* is a union of an edge-subset $F \subseteq E$ and a node-subset $X \subseteq V \setminus (T \cup T')$ such that removing those subsets disconnects T and T', and its capacity is defined as $u(F) + c(X)$.

Our algorithm for Theorem 1 builds on the ideas of *Discrete Convex Analysis (DCA) beyond* $\mathbb{Z}^n$ – a theory of discrete convex functions on special graph structures generalizing $\mathbb{Z}^n$ (the grid graph), which has been recently differentiated from the original DCA [23] and has been successfully applied to algorithm design for well-behaved classes of multiflow and related network design problems [12, 13, 14, 16]. Indeed, the algorithm in [12] for FETB was designed as: Formulate the dual of FETB as a minimization of an *L-convex function* on the (Cartesian) product of trees, apply the framework of the *steepest descent algorithm (SDA)*, and show that it is implemented by using a max-flow algorithm as a subroutine.

We formulate the dual of FNTB as an optimization problem on the product of the spaces of all subtrees of a fixed tree (Section 2.1). We develop a simple cut-descent algorithm for this optimization problem (Sections 2.2 and 2.3). Then we prove that this coincides with SDA for an L-convex function defined on the graph structure on the space of all subtrees (Section 3). Then the number of descents is estimated by a general theory of SDA, and the cost-scaling method is naturally incorporated to derive the time complexity (Section 2.4). Theorem 2 is obtained as a byproduct of these arguments. Due to the space limitation, we omit most of technical proofs, which are given in the full version.

Related work

ETB is a *survivable network design problem (SND)* with a special skew-supermodular function, and NTB is a node connectivity version (NSND) with a special skew-supermodular biset function. In his influential paper [18], Jain devised the iterative rounding method, and obtains a 2-approximation algorithm for SND, provided that an extreme optimal solution of the LP-relaxation of SND (with modified skew-supermodular functions) is available. Fleischer, Jain, and Williamson [7] and Cheriyan, Vempala, and Vetta [4] extended this iterative rounding 2-approximation algorithm to some classes of NSND. One of important open problems in the literature is a design of a combinatorial 2-approximation algorithm for (V)SND with the skew-supermodular (biset) function associated with connectivity requirements. One approach is to devise a combinatorial polynomial time algorithm to find an extreme optimal solution of its LP-relaxation; the currently known only polynomial time algorithm is a general LP-solver (e.g., the ellipsoid method). Our algorithm for FNTB, though it is the LP-relaxation of a very special NSND, may give an insight on such a research direction. On this direction, Feldmann, Könemann, Pashkovich, and Sanità [6] gave a $(2+\epsilon)$-approximation algorithm for SND with a proper function by solving the LP-relaxation approximately via the multiplicative weights method [9].

The notion of a separately-capacitated multiflow, introduced in this paper, is a new variation of S-paths packing. As seen in [24, Chapter 73], S-paths packing is one of the well-studied subjects in combinatorial optimization. Recent work [17] developed a fast algorithm for half-integral *nonzero S-paths packing problem on a group-valued graph* (with unit-capacity). Our derivation of Theorem 2 is different with flow-augmenting arguments such as Cherkassky's T-operation or those in [17]. It is a future research to develop such an algorithm for a separately-capacitated multiflow. Also, exploring an integer version of Theorem 2, an analogue of Mader's theorem [22], is an interesting future direction.

Notations

Let $\mathbb{Z}, \mathbb{Z}_+, \mathbb{R}, \mathbb{R}_+$ be the set of integers, nonnegative integers, reals, and nonnegative reals, respectively. Let $\mathbb{Z}^*, \mathbb{Z}^*_+$ be the set of half-integers and nonnegative half-integers, respectively, i.e., $\mathbb{Z}^* := \mathbb{Z}/2$. Let $\overline{\mathbb{R}} := \mathbb{R} \cup \{+\infty\}$ and $\underline{\mathbb{R}} := \mathbb{R} \cup \{-\infty\}$. Let denote $(a)^+ := \max\{a, 0\}$ for $a \in \mathbb{R}$. For a finite set V, we often identify a function on V with a vector in $\mathbb{R}^V$. For $i \in V$, its characteristic function $\chi_i : V \to \mathbb{R}$ is defined by $\chi_i(j) = 1$ if $j = i$ and $\chi_i(j) = 0$ otherwise. For a function f on V and a subset $U \subseteq V$, we denote $f(U) := \sum_{i \in U} f(i)$.

In this paper, all graphs are simple and connected unless otherwise specified. For an undirected graph on nodes V, the set of edges connecting U_1 and U_2 ($U_1, U_2 \subseteq V$) is denoted by $\delta(U_1, U_2)$. If $U_2 = V \setminus U_1$, we simply denote it by δU_1. If U_1 is a singleton, i.e., $U_1 = \{i\}$, then we denote $\delta\{i\}$ by δi. An edge connecting i and j is denoted by ij.

2 Node-Connectivity Terminal Backup Problem

Let $((V, E), S, u, c, a, r)$ be a network. Assume that $S = \{1, \ldots, k\} \subseteq V = \{1, \ldots, n\}$. By a perturbation technique, we may assume that a is positive; see Remark 3.

A sufficient and necessity condition for the feasibility of NTB is easily derived from the Menger's theorem as follows. A *biset* is a pair of node subsets $X, X^+ \subseteq V$ with $X \subseteq X^+$. We write $\hat{X} = (X, X^+)$ for a biset. Let $\Gamma(\hat{X}) := X^+ \setminus X$, and let $\delta(\hat{X}) := \delta(X, V \setminus X^+)$. For $s \in S$, define a family $\mathcal{C}_s$ of bisets by

$$\mathcal{C}_s := \{(X, X^+) \mid \{s\} \subseteq X \subseteq X^+ \subseteq V \setminus (S \setminus \{s\})\}.$$

Let $\mathcal{C} := \bigcup_{s \in S} \mathcal{C}_s$. Then an edge-capacity $x : E \to \mathbb{Z}_+$ is feasible if and only if

$$x(\delta(\hat{X})) + c(\Gamma(\hat{X})) \geq r(s) \quad (\hat{X} \in \mathcal{C}_s,\ s \in S). \tag{1}$$

We assume that u satisfies (1) throughout the paper (otherwise NTB is infeasible).

Fukunaga [8] developed an approximation algorithm for NTB via the following relaxation problem FNTB:

$$\begin{aligned} \text{(FNTB)} \quad \text{Minimize} \quad & \sum_{e \in E} a(e)x(e) \\ \text{subject to} \quad & x(\delta\hat{X}) + c(\Gamma(\hat{X})) \geq r(s) \quad (s \in S,\ \hat{X} \in \mathcal{C}_s), && (2) \\ & 0 \leq x(e) \leq u(e) \quad (e \in E). && (3) \end{aligned}$$

From the assumption, the polytope defined by (2) and (3) is nonempty. Also, it is known [8, Corollary 3.3] that the polytope is half-integral. Thus FNTB has a half-integral optimal solution. This can be obtained by a general LP solver [8, Lemma 4.4].

▶ Remark 3. If $Z := \{e \in E \mid a(e) = 0\}$ is nonempty, then we use the following perturbation technique based on [10, 20]. Recall that U is the maximum edge capacity. Define a positive edge-cost a' by $a'(e) := 1$ for $e \in Z$ and $a'(e) := (2U|Z| + 1)a(e)$ for $e \notin Z$. Let x^* be a half-integral optimal solution of FNTB under the edge-cost a' (it exists by the half-integrality). We prove that x^* is also optimal under the original edge-cost a. It suffices to show that $\sum_{e\in E} a(e)x^*(e) \leq \sum_{e\in E} a(e)x(e)$ for any feasible half-integral edge-capacity x. It holds that $(2U|Z| + 1)(\sum_{e\in E} a(e)x^*(e) - \sum_{e\in E} a(e)x(e)) = \sum_{e\in E} a'(e)x^*(e) - \sum_{e\in E} a'(e)x(e) - x^*(Z) + x(Z) \leq U|Z|$ and thus $\sum_{e\in E} a(e)x^*(e) - \sum_{e\in E} a(e)x(e) \leq U|Z|/(2U|Z| + 1) < 1/2$. By the half-integrality, we obtain $\sum_{e\in E} a(e)x^*(e) - \sum_{e\in E} a(e)x(e) \leq 0$.

2.1 Combinatorial Duality for FNTB

We introduce a combinatorial duality theory for FNTB. For each $s \in S$, consider an infinite path graph P_s with one endpoint. Glue those $k\,(= |S|)$ endpoints, and denote the resulting graph by $\mathbb{T}$. We denote the set of nodes of P_s and $\mathbb{T}$ also by P_s and $\mathbb{T}$, respectively. We give length $1/2$ for each edge in $\mathbb{T}$. The glued endpoint is denoted by 0, and the point in P_s $(s \in S)$ having the distance l from 0 is denoted by (l, s). We denote the set of all subtrees of $\mathbb{T}$ by $\mathbb{S} = \mathbb{S}(\mathbb{T})$. If a subtree T does not contain 0, then it is contained in some P_s. Such a subtree T is said to be of *s-type* and is denoted by $[l, l']_s$, where (l, s) and (l', s) are the closest and farthest nodes from 0 in T, respectively. If a subtree T contains 0, then it is said to be of *0-type* and is denoted by $[l_1, l_2, \ldots, l_k] = [l_s]_{s\in S}$, where (l_s, s) is the node in $T \cap P_s$ farthest from 0 for each $s \in S$. We identify a node on $\mathbb{T}$ with a subtree consisting of this node only.

For a 0-type subtree $T = [l_s]_{s\in S} \in \mathbb{S}$, let $\mathrm{size}_s(T) := l_s$ for $s \in S$, and $\mathrm{size}(T) := \sum_{s=1}^{k} \mathrm{size}_s(T)$. For an s-type subtree $T = [l, l']_s \in \mathbb{S}$, let $\mathrm{size}(T) := l' - l$. For two subtrees $T, T' \in \mathbb{S}$, we denote the minimum distance between T and T' on $\mathbb{T}$ by $\mathrm{dist}(T, T')$, i.e., $\mathrm{dist}(T, T') := \min\{d_{\mathbb{T}}(v, v') \mid v \in T,\ v' \in T'\}$, where $d_{\mathbb{T}}$ is the shortest distance on $\mathbb{T}$.

We formulate a dual of FNTB as a problem of assigning a subtree for each node $i \in V$. That is, subtrees are viewed as node-potentials. So we use p_i and $p : V \to \mathbb{S}$ for denoting a subtree assigned for node $i \in V$ and a potential function, respectively. Formally, let us consider the following maximization problem DTB.

$$\begin{aligned}
\text{(DTB)}\quad \text{Maximize}\quad & \sum_{s\in S} r_s \,\mathrm{dist}(0, p_s) - \sum_{i\in V\setminus S} c_i \,\mathrm{size}(p_i) - \sum_{ij\in E} u_{ij}(\mathrm{dist}(p_i, p_j) - a_{ij})^+ \\
\text{subject to}\quad & p : V \to \mathbb{S}, \\
& p_s \in P_s \quad (s \in S). \qquad\qquad (4)
\end{aligned}$$

It turns out in the proof of Proposition 4 below that this seemingly strange formulation of DTB is essentially the LP-dual of FTB. If $p : V \to \mathbb{S}$ satisfies (4), then it is called a *potential.* See Figure 1 for an intuition for a subtree-valued potential p. A potential p is said to be *proper* if any p_i for $i \in V$ is contained in the minimal subtree that contains all p_s $(s \in S)$.

▶ **Proposition 4.** *The optimum value of FNTB is at least that of DTB. Moreover, there exists a proper optimal potential for DTB.*

Proof. Let $p : V \to \mathbb{S}$ be any potential (not necessarily proper). For each $s \in S$, suppose that p_s is written as $p_s = (M_s, s)$ for $M_s \in \mathbb{Z}^*_+$. Define a new proper potential $p' : V \to \mathbb{S}$ by

$$p'_i := \begin{cases} [\min\{l, M_s\}, \min\{l', M_s\}]_s & \text{if } p_i = [l, l']_s, \\ [\min\{l_1, M_1\}, \ldots, \min\{l_k, M_k\}] & \text{if } p_i = [l_1, \ldots, l_k]. \end{cases}$$

Then the objective function value of p' does not decrease. This implies the latter part of the statement.

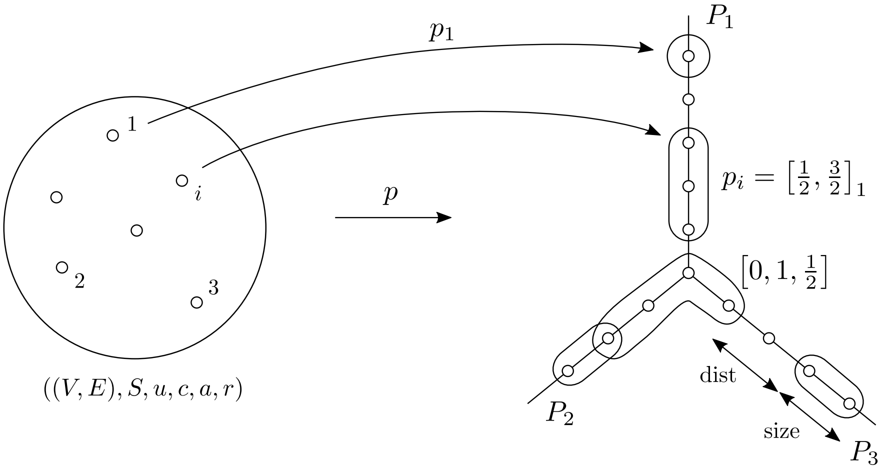

p_1
P_1
1
i
2
3
p
$p_i = \left[\frac{1}{2}, \frac{3}{2}\right]_1$
$\left[0, 1, \frac{1}{2}\right]$
dist
size
P_2
P_3
$((V,E),S,u,c,a,r)$

Figure 1 A subtree-valued potential p.

We next show the former part, i.e., the weak duality. The LP dual of FNTB is written as

$$\begin{aligned}&\text{Maximize} \quad \sum_{s\in S}\sum_{\hat{X}\in\mathcal{C}_s}(r_s - c(\Gamma(\hat{X})))\pi(\hat{X}) - \sum_{e\in E} u_e\left(\sum_{\hat{X}\in\mathcal{C}:e\in\delta\hat{X}}\pi(\hat{X}) - a_e\right)^+\\ &\text{subject to} \quad \pi : \mathcal{C}\to\mathbb{R}_+.\end{aligned}$$

We show that for any proper potential $p : V \to \mathbb{S}$, we can construct $\pi : \mathcal{C} \to \mathbb{R}_+$ such that

$$\sum_{\hat{X}\in\mathcal{C}_s}\pi(\hat{X}) = \mathrm{dist}(0, p_s) \quad (s \in S), \tag{5}$$

$$\sum_{\hat{X}\in\mathcal{C}:i\in\Gamma(\hat{X})}\pi(\hat{X}) = \mathrm{size}(p_i) \quad (i \in V \setminus S), \tag{6}$$

$$\sum_{\hat{X}\in\mathcal{C}:e\in\delta\hat{X}}\pi(\hat{X}) = \mathrm{dist}(p_i, p_j) \quad (ij \in E). \tag{7}$$

Then by $\sum_{\hat{X}\in\mathcal{C}} c(\Gamma(\hat{X}))\pi(\hat{X}) = \sum_{\hat{X}\in\mathcal{C}}\sum_{i\in\Gamma(\hat{X})} c_i\pi(\hat{X}) = \sum_{i\in V\setminus S} c_i \sum_{\hat{X}\in\mathcal{C}:i\in\Gamma(\hat{X})}\pi(\hat{X})$, the weak duality follows.

Let e be an edge in $\mathbb{T}$. We define a biset (X_e, X_e^+) as follows. When we remove e from $\mathbb{T}$, there appear two connected components. Let T_e be the component which does not contain $0\,(\in \mathbb{T})$. Define $X_e, X_e^+ \subseteq V$ by

$$X_e := \{i \in V \mid p_i \text{ is contained in } T_e\}, \quad X_e^+ := X_e \cup \{i \in V \mid p_i \text{ contains } e\}.$$

Observe that if e is an edge in P_s and $X_e \neq \emptyset$, then $(X_e, X_e^+) \in \mathcal{C}_s$. Then a potential function $\pi : \mathcal{C} \to \mathbb{R}_+$ defined by

$$\pi(\hat{X}) := \frac{1}{2}|\{e \mid \hat{X} = (X_e, X_e^+)\}| \quad (\hat{X} \in \mathcal{C})$$

satisfies (5)–(7). ◀

We remark that the technique used in the above proof is based on a tree representation of a laminar biset family; see also [11] for the relating argument that maps to each node a subtree as a potential. We also note that our algorithm below will give an algorithmic proof of the strong duality.

We next derive from Proposition 4 the complementary slackness condition. Let $p : V \to \mathbb{S}$ be a proper potential. By p, we decompose V into $S \cup V_0 \cup \bigcup_{s \in S} V_s$, where

$$V_0 := \{i \in V \setminus S \mid p_i \text{ is of 0-type}\},$$
$$V_s := \{i \in V \setminus S \mid p_i \text{ is of } s\text{-type}\} \quad (s \in S).$$

In the next lemma, we see that it is sufficient to only consider edges $ij \in E$ with $\mathrm{dist}(p_i, p_j) \geq a_{ij}$. Let denote the set of such edges by

$$E^* := \{ij \in E \mid \mathrm{dist}(p_i, p_j) \geq a_{ij}\}.$$

For $i \in V_0$ and $s \in S$, we denote a set of edges in E^* connecting i and V_s by

$$E^{i,s} := \{ij \in E^* \mid j \in V_s\} \quad (i \in V_0,\ s \in S).$$

By the positivity of a, we see that $(E^{i,1}, E^{i,2}, \ldots, E^{i,k})$ is a partition of $E^* \cap \delta i$. For $i \in V_s$ $(s \in S)$, there appear two connected components when we remove p_i from $\mathbb{T}$. Let $T_{i,0}$ be the component which includes $0\,(\in \mathbb{T})$, and let $T_{i,+}$ be the other component. Then we define the sets of edges $E^{i,0}$ and $E^{i,+}$ by

$$E^{i,0} := \{ij \in E^* \mid p_j \text{ is contained in } T_{i,0}\},$$
$$E^{i,+} := \{ij \in E^* \mid p_j \text{ is contained in } T_{i,+}\}.$$

By the positivity of a, we see that $(E^{i,0}, E^{i,+})$ is a partition of $E^* \cap \delta i$.

▶ **Lemma 5.** *Let $x : E \to \mathbb{R}_+$ be an edge-capacity function with $0 \leq x \leq u$, and let $p : V \to \mathbb{S}$ be a proper potential. If x and p satisfy the following conditions (A1–5), then x and p are optimal solutions for FNTB and DTB, respectively:*

(A1) *For each $ij \in E$, if $\mathrm{dist}(p_i, p_j) > a_{ij}$, then $x_{ij} = u_{ij}$.*
(A2) *For each $ij \in E$, if $\mathrm{dist}(p_i, p_j) < a_{ij}$, then $x_{ij} = 0$.*
(A3) *For each $i \in \bigcup_{s \in S} V_s$, it holds $x(E^{i,0}) = x(E^{i,+}) \leq c_i$. If $\mathrm{size}(p_i) > 0$, then $x(E^{i,0}) = x(E^{i,+}) = c_i$.*
(A4) *For each $i \in V_0$ and $s \in S$, it holds $x(E^{i,s}) \leq c_i$ and $x(E^{i,s}) \leq \sum_{s' \neq s} x(E^{i,s'})$. If $\mathrm{size}_s(p_i) > 0$, then $x(E^{i,s}) = c_i$.*
(A5) *For each $s \in S$, it holds $x(\delta s) \geq r_s$. If $\mathrm{dist}(0, p_s) > 0$, then $x(\delta s) = r_s$.*

Proof. Let x and p satisfy (A1–5). For the feasibility of x, it is sufficient to show that, for each $s \in S$, there exists a flow satisfying the capacities x and c that connects s and $S \setminus \{s\}$ with flow-value r_s. To prove this, we decompose x into a separately-capacitated multiflow. An *S-path* is a path connecting distinct terminals. Consider the following algorithm, which takes x as an input and outputs a function $\lambda : \mathcal{P} \to \mathbb{R}_+$, where $\mathcal{P}$ is a set of S-paths:

0. Let $\mathcal{P} = \emptyset$.
1. Take $s \in S$ and an edge sj satisfying $x(sj) > 0$. If such a pair does not exist, then stop the algorithm; output $(\mathcal{P}, \lambda)$. Otherwise, let $j_0 \leftarrow s,\ j_1 \leftarrow j,\ \mu \leftarrow x(sj),\ t \leftarrow 1$.
2. If j_t is a terminal, then add $P = (j_0, j_1, \ldots, j_t)$ to $\mathcal{P}$ and let $\lambda(P) := \mu > 0$. Update $x(e) \leftarrow x(e) - \mu$ on each edge e in P, and return to Step 1. Otherwise go to Step 3.
3. If $j_t \in \bigcup_{s \in S} V_s$, then $j_{t-1}j_t \in E^{j_t,+}$ or $j_{t-1}j_t \in E^{j_t,0}$ by (A2) and $x(j_{t-1}j_t) > 0$. In the former case, take $j_t j_{t+1} \in E^{j_t,0}$ with $x(j_t j_{t+1}) > 0$. Such an edge exists by the former part of (A3). In the latter case, take $j_t j_{t+1} \in E^{j_t,+}$ with $x(j_t j_{t+1}) > 0$. Update $\mu \leftarrow \min\{\mu, x(j_t j_{t+1})\},\ t \leftarrow t + 1$, and return to Step 2.

If $j_t \in V_0$, then $j_{t-1}j_t \in E^{j_t,s}$ (as we will show). Take $s' \neq s$ with maximum $x(E^{j_t,s'})$ ($>$ 0), and take $j_tj_{t+1} \in E^{j_t,s'}$ with $x(j_tj_{t+1}) > 0$. Such an edge exists by $x(j_{t-1}j_t) > 0$ and the former part of (A4). Update

$$\mu \leftarrow \min\left\{\mu, x(j_tj_{t+1}), \frac{\min\left\{\sum_{s'''\neq s''} x(E^{j_t,s'''}) - x(E^{j_t,s''}) \mid s'' \neq s, s'\right\}}{2}\right\},$$

and $t \leftarrow t+1$. Note that $\mu > 0$ by the maximality of $x(E^{j_t,s'})$. Return to Step 2.

Suppose that we add $(j_0, j_1, \ldots, j_\ell)$ to $\mathcal{P}$ in Step 2. Observe that j_{t+1} is at a side opposite to j_{t-1} based on j_t for each $t = 1, \ldots, \ell-1$. By the positivity of a and (A2), $\{j_{t-1}, j_t, j_{t+1}\}$ are distinct and

$$\mathrm{dist}(p_{j_{t-1}}, p_{j_{t+1}}) = \mathrm{dist}(p_{j_{t-1}}, p_{j_t}) + \mathrm{size}(p_{j_t}) + \mathrm{dist}(p_{j_t}, p_{j_{t+1}})$$

if $j_t \in \bigcup_{s\in S} V_s$, and

$$\mathrm{dist}(p_{j_{t-1}}, p_{j_{t+1}}) = \mathrm{dist}(p_{j_{t-1}}, p_{j_t}) + \mathrm{size}_s(p_{j_t}) + \mathrm{size}_{s'}(p_{j_t}) + \mathrm{dist}(p_{j_t}, p_{j_{t+1}})$$

if $j_t \in V_0$, where $j_{t-1} \in V_s$ and $j_{t+1} \in V_{s'}$ $(s \neq s')$. Since $\mathbb{T}$ is a tree, we can show

$$\mathrm{dist}(p_{j_0}, p_{j_\ell}) = \sum_{t=0}^{\ell-1} \mathrm{dist}(p_{j_t}, p_{j_{t+1}}) + \sum_{1\leq t\leq \ell-1,\ t\neq t'} \mathrm{size}(p_{j_t}) + \mathrm{size}_{j_0}(p_{j_{t'}}) + \mathrm{size}_{j_\ell}(p_{j_{t'}}) \quad (8)$$

by an induction, where $j_{t'} \in V_0$ (if exists); see also [12, Lemma 3.9]. Hence $(j_0, j_1, \ldots, j_\ell)$ is a "shortest path on $\mathbb{T}$" from j_0 to j_ℓ, and $j_0, \ldots, j_\ell$ are distinct.

Thus after $|V|$ executions of Step 3, the algorithm adds a path P to $\mathcal{P}$ in Step 2. Also the algorithm keeps (A2) and the former parts of (A3–4). To see it for (A4), suppose that the algorithm adds a path $(j_0, j_1, \ldots, j_t, \ldots, j_\ell)$ to $\mathcal{P}$ in Step 2, where $j_0 = s \in S$, $j_t \in V_0$ and $j_\ell = s' \in S$. By the above argument, such t is uniquely determined (if exists). Then for all $s'' \neq s, s'$, we have $\sum_{s'''\neq s''} x(E^{j_t,s'''}) - x(E^{j_t,s''}) \geq 2\mu$. Thus after the decrease of the value of x along with P, it satisfies that $\sum_{s'''\neq s''} x(E^{j_t,s'''}) - x(E^{j_t,s''}) \geq 0$.

After the decrease of the value of x along with a path, it becomes $x(e) = 0$ for at least one edge $e \in E$, or becomes $\sum_{s'\neq s} x(E^{i,s'}) - x(E^{i,s}) = 0$ for at least one pair of $i \in V_0$ and $s \in S$. The algorithm keeps those values to be zero in the remaining execution, implying that it terminates after adding at most $O(m + kn)$ paths to $\mathcal{P}$. To see it, suppose that after the decrease of the value of x along with a path, it becomes $\sum_{s'\neq s} x(E^{i,s'}) - x(E^{i,s}) = 0$ for $i \in V_0$ and $s \in S$. If the algorithm chooses a path $(j_0, \ldots, j_t = i, \ldots, j_\ell)$ for adding to $\mathcal{P}$ in the remaining execution, then by the maximality of $x(E^{i,s})$, it should satisfy that $j_{t-1}j_t \in E^{i,s}$ or $j_tj_{t+1} \in E^{i,s}$. Thus $\sum_{s'\neq s} x(E^{i,s'}) - x(E^{i,s})$ does not change by the decrease of the value of x along with $(j_0, \ldots, j_\ell)$.

We have shown the algorithm always terminates in finite steps. For the output $f = (\mathcal{P}, \lambda)$, let $f(e) := \sum_{P\in\mathcal{P}:e\in P} \lambda(P)$ for $e \in E$, and let $f(i) := \sum_{P\in\mathcal{P}:i\in P} \lambda(P)$ for $i \in V$. Also let $\mathcal{P}_s \subseteq \mathcal{P}$ be the subset of paths connecting s to other terminals, and let $f_s = (\mathcal{P}_s, \lambda_s)$ for $s \in S$. Clearly, it holds that $f(e) \leq x(e) \leq u(e)$ for $e \in E$. For $i \in V_s$ $(s \in S)$, if a path $P \in \mathcal{P}$ goes through i, then P must be contained in $\mathcal{P}_s$. Thus by the former part of (A3), $f_s(i) = f(i) \leq x(E^{i,0})$ $(= x(E^{i,+})) \leq c(i)$. Also, $f_{s'}(i) \leq f_s(i) \leq c(i)$ for any $s' \neq s$. On the other hand, for $i \in V_0$, if a path in $\mathcal{P}_s$ $(s \in S)$ goes through i, then it must include an edge contained in $E^{i,s}$. Thus by the former part of (A4), we have $f_s(i) \leq x(E^{i,s}) \leq c(i)$. Therefore f is a separately-capacitated multiflow. Moreover, f_s satisfies the requirement r by the former part of (A5). Thus x is a feasible solution of FNTB.

We next show the optimality of x and p. First observe that when the algorithm terminates, all edges $e \in E$ satisfy $x(e) = 0$. In fact, if there exists an edge $e \in E$ with $x(e) > 0$, then we can construct an S-path with edges having positive x-values by repeating to apply the former parts of (A3–4). Thus $f(e) = x(e)$ $(e \in E)$ for the original input x. We see that

$$\begin{aligned}
&\sum_{ij\in E} a_{ij}x_{ij} - \sum_{s\in S} r_s \operatorname{dist}(0,p_s) + \sum_{i\in V\setminus S} c_i \operatorname{size}(p_i) + \sum_{ij\in E} u_{ij}(\operatorname{dist}(p_i,p_j) - a_{ij})^+ \\
&= \sum_{ij\in E}(\operatorname{dist}(p_i,p_j) - a_{ij})^+(u_{ij} - x_{ij}) + \sum_{ij\in E}(a_{ij} - \operatorname{dist}(p_i,p_j))^+ x_{ij} + \sum_{ij\in E} x_{ij}\operatorname{dist}(p_i,p_j) \\
&\quad + \sum_{i\in V\setminus S} c_i \operatorname{size}(p_i) - \sum_{s\in S} r_s \operatorname{dist}(0,p_s) \\
&= \sum_{ij\in E}(\operatorname{dist}(p_i,p_j) - a_{ij})^+(u_{ij} - x_{ij}) + \sum_{ij\in E}(a_{ij} - \operatorname{dist}(p_i,p_j))^+ x_{ij} \\
&\quad + \sum_{s\in S}\sum_{i\in V_s}(c_i - f(i))\operatorname{size}(p_i) + \sum_{i\in V_0}\sum_{s\in S}(c_i - f_s(i))\operatorname{size}_s(p_i) + \sum_{s\in S}(f(s) - r_s)\operatorname{dist}(0,p_s), \qquad (9)
\end{aligned}$$

where we use $a + (d-a)^+ = d + (a-d)^+$ for $a, d \in \mathbb{R}$ and

$$\begin{aligned}
&\sum_{ij\in E} f(ij)\operatorname{dist}(p_i,p_j) + \sum_{s\in S}\sum_{i\in V_s} f(i)\operatorname{size}(p_i) + \sum_{i\in V_0}\sum_{s\in S} f_s(i)\operatorname{size}_s(p_i) \\
&= \sum_{ij\in E}\sum_{P\in\mathcal{P}, ij\in E(P)} \lambda(P)\operatorname{dist}(p_i,p_j) \\
&\qquad + \sum_{s\in S}\sum_{i\in V_s}\sum_{P\in\mathcal{P}, i\in V(P)} \lambda(P)\operatorname{size}(p_i) + \sum_{i\in V_0}\sum_{s\in S}\sum_{P\in\mathcal{P}_s, i\in V(P)} \lambda_s(P)\operatorname{size}_s(p_i) \\
&= \sum_{st}\sum_{P\in\mathcal{P}: P \text{ connects } st} \lambda(P)\operatorname{dist}(p_s,p_t) = \sum_{s\in S} f(s)\operatorname{dist}(0,p_s)
\end{aligned}$$

by (8). We see $f(i) = x(E^{i,0})$ $(= x(E^{i,+}))$ for $i \in \bigcup_{s\in S} V_s$, and $f_s(i) = x(E^{i,s})$ for $i \in V_0$ and $s \in S$. Also $f(s) = x(\delta s)$ for $s \in S$. Then (9) is zero by (A1–2) and the latter parts of (A3–5). By Proposition 4, we conclude that x and p are both optimal. ◀

▶ Remark 6. Suppose the input edge-capacity x satisfies $x(\delta i) \in \mathbb{Z}_+$ for any $i \in V$. Then μ is always half-integral, and the integrality of $x(\delta i)$ is also kept in the execution of the algorithm. Thus the output multiflow is half-integer-valued. This argument will be used for proving a min-max theorem (Theorem 2) for a separately-capacitated multiflow later.

The decomposition algorithm is based on [11, Lemma 4.5]; see also [14, Lemma 3.3].

The existence of an edge-capacity x satisfying (A1–5) can be checked by solving the *undirected circulation problem.* This fact leads a simple descent algorithm for DTB and FNTB. Notice that a potential $p : V \to \mathbb{S}$ can be identified with a vector in $\mathbb{S}^n$. For brevity we write $p \in \mathbb{S}^n$ below. Let $h_a = h : \mathbb{S}^n \to \overline{\mathbb{R}}$ be a function defined by

$$h(p) := -\sum_{s\in S} r_s \operatorname{dist}(0,p_s) + \sum_{i\in V\setminus S} c_i \operatorname{size}(p_i) + \sum_{ij\in E} u_{ij}(\operatorname{dist}(p_i,p_j) - a_{ij})^+ \qquad (10)$$

if $p \in \mathbb{S}^n$ is a potential and $h(p) := \infty$ otherwise. Then DTB is precisely a minimization of h over $\mathbb{S}^n$. Consider the following algorithm DESCENT:

Algorithm 1 DESCENT.

```
0. Initialize $p \equiv 0$ (i.e., $p(i) = 0$ for any $i \in V$).
1. Check the sufficiency of the optimality of $p$ by searching $x$ satisfying (A1–5).
2. If $x$ is found, then $x$ and $p$ are optimal; stop.
3. Otherwise find $q \in \mathbb{S}^n$ with $h(q) < h(p)$; update $p$ by $q$ and go to Step 1.
```

We give more details of DESCENT in Section 2.3. As for Step 1, we can also do Step 3 by the undirected circulation problem; q is computed by the certificate of the nonexistence of x. In the following subsections, we introduce the undirected circulation problem and discuss how to find x or q in each case.

2.2 Checking the Optimality

Let (U, F) be an undirected graph, and let $\underline{b} : F \to \underline{\mathbb{R}}$ and $\overline{b} : F \to \overline{\mathbb{R}}$ be lower and upper capacity functions satisfying $\underline{b}(e) \leq \overline{b}(e)$ for each $e \in F$. The graph (U, F) may contain self-loops but no multiedges. The *circulation problem* on $((U, F), \underline{b}, \overline{b})$ is the problem of finding an edge-weight $y : F \to \mathbb{R}$ satisfying $\underline{b}(e) \leq y(e) \leq \overline{b}(e)$ for each $e \in F$ and $\sum_{ij \in F} y(ij) = 0$ for each $i \in U$. Such a y is called a *circulation*.

Let 3^U denote the set of pairs (Y, Z) of two subsets $Y, Z \subseteq U$ with $Y \cap Z = \emptyset$. For $(Y, Z) \in 3^U$, let $\chi_{Y,Z} := \sum_{i \in Y} \chi_i - \sum_{i \in Z} \chi_i \in \mathbb{R}^U$. Define the *cut function* $\kappa : 3^U \to \underline{\mathbb{R}}$ by

$$\kappa(Y, Z) := \sum_{ij \in F} \{(\chi_{Y,Z}(\{i, j\}))^+ \underline{b}(ij) - (\chi_{Z,Y}(\{i, j\}))^+ \overline{b}(ij)\} \quad ((Y, Z) \in 3^U).$$

It is well-known that the feasibility of the circulation problem is characterized via the cut function. We can show it by reducing to Hoffman's circulation theorem. A cut $(Y, Z) \in 3^U$ with $\kappa(Y, Z) > 0$ is called *violating*, and is called *maximum violating* if it attains the maximum $\kappa(Y, Z)$ among all violating cuts.

▶ **Lemma 7** (see, e.g., [16, Theorems 2.4, 2.7]). *Let $((U, F), \underline{b}, \overline{b})$ be an undirected network.*

(1) *The circulation problem is feasible if and only if $\kappa(Y, Z) \leq 0$ for any $(Y, Z) \in 3^U$.*

(2) *If $\underline{b}$ and $\overline{b}$ are integer-valued, then there exists a feasible half-integer-valued circulation $y : E \to \mathbb{Z}^*_+$.*

(3) *Under the same assumption, we can obtain a feasible half-integer-valued circulation or a maximum violating cut in $O(\mathrm{MF}(|U|, |F|))$ time.*

Let us return to our problem. For a given proper potential $p \in \mathbb{S}^n$, the existence of $x : E \to \mathbb{R}_+$ satisfying (A1–5) reduces to the undirected circulation problem on the following network $\mathcal{N}_p := ((U, F), \underline{c}, \overline{c})$. See Figure 2 for the following construction.

For each $i \in \bigcup_{s \in S} V_s$, divide i into two nodes $U_i := \{i^0, i^+\}$, and connect nodes by an edge $i^0 i^+$. For representing (A3), let $\underline{c}(i^0 i^+) := -c_i$, and let $\overline{c}(i^0 i^+) := 0$ if $\mathrm{size}(p_i) = 0$ and $\overline{c}(i^0 i^+) := -c_i$ if $\mathrm{size}(p_i) > 0$. For each $i \in V_0$, divide i into $2k$ nodes $U_i := U_i^0 \cup U_i^+$, where $U_i^0 := \{i^{1,0}, i^{2,0}, \ldots, i^{k,0}\}$ and $U_i^+ := \{i^{1,+}, i^{2,+}, \ldots, i^{k,+}\}$, and connect them by edges $i^{s,0} i^{s,+}$ for $s \in S$ and $i^{s,0} i^{s',0}$ for distinct $s, s' \in S$. For representing (A4), let $\underline{c}(i^{s,0} i^{s,+}) := -c_i$, and let $\overline{c}(i^{s,0} i^{s,+}) := 0$ if $\mathrm{size}_s(p_i) = 0$ and $\overline{c}(i^{s,0} i^{s,+}) := -c_i$ if $\mathrm{size}_s(p_i) > 0$. Also let $\underline{c}(i^{s,0} i^{s',0}) := 0$ and $\overline{c}(i^{s,0} i^{s',0}) := \infty$. For each $s \in S$, let $s^0 := s$ and $U_s := \{s^0\}$, and add a self-loop $s^0 s^0$. For representing (A5), let $\underline{c}(s^0 s^0) := -\infty$ if $\mathrm{dist}(0, p_s) = 0$ and $\underline{c}(s^0 s^0) := -r_s$ if $\mathrm{dist}(0, p_s) > 0$, and let $\overline{c}(s^0 s^0) := -r_s$.

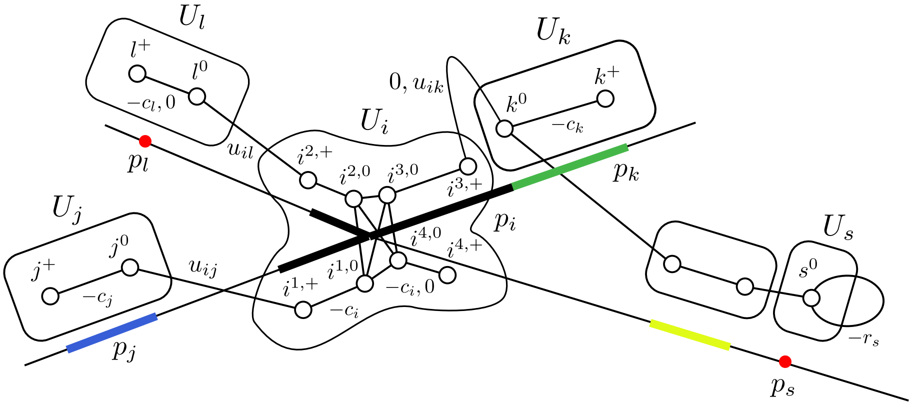


Figure 2 The undirected network $\mathcal{N}_p$.

For each edge $ij \in E$, if $\mathrm{dist}(p_i, p_j) < a_{ij}$, then $x_{ij} = 0$ by (A2). Thus we remove those edges. Let $E_>$ be the set of edges $ij \in E$ with $\mathrm{dist}(p_i, p_j) > a_{ij}$, and let $E_=$ be the set of edges $ij \in E$ with $\mathrm{dist}(p_i, p_j) = a_{ij}$. We replace endpoints of each edge $ij \in E_> \cup E_=$. If $i \in V_0$ and $j \in V_s$, then replace ij with $i^{s,+}j^0$. If $i \in V_s$ and $j \in V_{s'}$ $(s \neq s')$, then replace ij with $i^0 j^0$. If $i, j \in V_s$ and p_i is closer to 0 than p_j, i.e., $\mathrm{dist}(0, p_i) < \mathrm{dist}(0, p_j)$, then replace ij with $i^+ j^0$. We identify those replaced edges with the original edges. Let $\underline{c}(ij) := 0$ if $ij \in E_=$ and $\underline{c}(ij) := u_{ij}$ if $ij \in E_>$, and let $\overline{c}(ij) := u_{ij}$. U and F are defined as the union of all nodes and edges in the above, respectively.

▶ **Theorem 8.** *Let $\mathcal{N}_p = ((U, F), \underline{c}, \overline{c})$ be the undirected network constructed from a proper potential $p \in \mathbb{S}^n$. If it has a (half-integer-valued) circulation $y : F \to \mathbb{R}$, then an edge-capacity function $x : E \to \mathbb{R}_+$ defined by*

$$x(e) := \begin{cases} y(e) & \text{if } e \in E_> \cup E_=, \\ 0 & \text{otherwise } (e = ij \text{ with } \mathrm{dist}(p_i, p_j) < a_{ij}) \end{cases}$$

satisfies (A1–5).

Proof. We can obtain (A1–5) from definitions immediately. For example, the former part of (A4) follows from $x(E^{i,s}) = -y(i^{s,0} i^{s,+}) \leq -\underline{c}(i^{s,0} i^{s,+}) = c_i$ and

$$x(E^{i,s}) = -y(i^{s,0} i^{s,+}) = \sum_{s' \neq s} y(i^{s,0} i^{s',0}) \leq \sum_{s' \neq s} -y(i^{s',0} i^{s',+}) = \sum_{s' \neq s} x(E^{i,s'}),$$

and the latter part of (A4) follows from $-y(i^{s,0} i^{s,+}) \geq -\overline{c}(i^{s,0} i^{s,+}) = c_i$ for $i \in V_0$ and $s \in S$ with $\mathrm{size}_s(p_i) > 0$. ◀

2.3 Finding a Descent Direction

If the algorithm in Lemma 7 outputs a circulation in $\mathcal{N}_p$, then an optimal edge-capacity is computed from the circulation, and p is optimal by Lemma 5 and Theorem 8. Otherwise the algorithm outputs a maximum violating cut. We show that we can find $q \in \mathbb{S}^n$ with $h(q) < h(p)$ using the maximum violating cut. A basic idea is to modify each subtree p_i, according to the intersection pattern of the maximum violating cut with U_i, so that the objective function h decreases. This implies the necessity of Lemma 5 and the strong duality of Proposition 4.

We begin with introducing the notion of *basic moves* for a subtree. For an s-type subtree $T = [l, l']_s$, we denote its endpoints by $v_0(T) := (l, s) \in \mathbb{T}$ and $v_+(T) := (l', s) \in \mathbb{T}$. When we remove T from $\mathbb{T}$, there appear two connected components. Let T'_0 be the component containing $0\,(\in \mathbb{T})$, and T'_+ be the other. We can expand the subtree T by adding a node next to T. There are two nodes next to T, one is contained in T'_0 and the other is contained in T'_+. The 0-*expansion* is the operation to add that node contained in $T_{i,0}$ to T, and the $+$-*expansion* is the operation to add that node contained in $T_{i,+}$ to T. If T satisfies $\mathrm{size}(T) > 0$, then we can shrink T by removing $v_0(T)$ or $v_+(T)$ from T. The 0-*shrinkage* is the operation to remove $v_0(T)$ from T, and the $+$-*shrinkage* is the operation to remove $v_+(T)$ from T.

For a 0-type subtree $T = [l_s]_{s\in S}$, we denote its endpoints by $v_s(T) := (l_s, s) \in \mathbb{T}$ for $s \in S$. When we remove T from $\mathbb{T}$, there appear k $(= |S|)$ connected components. For $s \in S$, let T'_s be the component which is contained in P_s. As above, we can expand the subtree T by adding a node next to T. There are k nodes next to T, and each T'_s $(s \in S)$ contains exactly one such a node. The $(s, +)$-*expansion* for $s \in S$ is the operation to add that node contained in T'_s to T. If T satisfies $\mathrm{size}_s(T) > 0$ for $s \in S$, then we can shrink T by removing $v_s(T)$ from T. The $(s, +)$-*shrinkage* for $s \in S$ with $\mathrm{size}_s(T) > 0$ is the operation to remove $v_s(T)$ from T. For $s \in S$, if $\mathrm{size}_{s'}(T) = 0$ for any other $s' \in S$, then we can shrink T by removing $0\,(\in \mathbb{T})$ from T. The $(s, 0)$-*shrinkage* for such $s \in S$ is the operation to remove 0 from T. We call these expansion and shrinkages *basic moves*.

Let $(Y, Z) \in 3^U$ be a cut. From (Y, Z), the modification $p^{Y,Z}$ of p is defined as follows. For $s \in S$, do:

- If $s^0 \in Y$, then 0-expand and $+$-shrink p_s.
- If $s^0 \in Z$, then $+$-expand and 0-shrink p_s.

For $i \in \bigcup_{s\in S} V_s$, do:

- If $i^0 \in Y$, then 0-expand p_i. If $i^0 \in Z$, then 0-shrink p_i.
- If $i^+ \in Y$, then $+$-expand p_i. If $i^+ \in Z$, then $+$-shrink p_i.

For $i \in V_0$, do:

- If $U_i^0 \cap (Y \cup Z) = \emptyset$, then we do the following for each $s \in S$:
  - If $i^{s,+} \in Y$, then $(s, +)$-expand p_i. If $i^{s,+} \in Z$, then $(s, +)$-shrink p_i.
- If $i^{s,0} \in Z$ for some $s \in S$, then $(s, 0)$-shrink p_i. Also do the following:
  - If $i^{s,+} \in Y$, then $(s, +)$-expand p_i. If $i^{s,+} \in Z$, then $(s, +)$-shrink p_i.

There may exists $i \in V$ that such a move cannot be defined, e.g., $i \in \bigcup_{s\in S} V_s$ with $\mathrm{size}(p_i) \le 1/2$ and $\{i^0, i^+\} \subseteq Z$, or $j \in V_0$ with $\{j^{s,0}, j^{s',0}\} \subseteq Z$. If the moves can be defined for all $i \in V$, then the cut (Y, Z) is called *movable*. For a movable cut $(Y, Z) \in 3^U$, we denote the modified potential by $p^{Y,Z}$.

We cal a node $(l, s) \in \mathbb{T}$ *even* if the number of edges between (l, s) and 0 is even, and *odd* otherwise. A basic move is said to be *upward* if the added node is even or the removed node is odd. A basic move is said to be *downward* if the added node is odd or the removed node is even. A movable cut $(Y, Z) \in 3^U$ is *upward-movable* (resp. *downward-movable*) if all basic moves occurring in the modification from p to $p^{Y,Z}$ are basic upward moves (resp. basic downward moves). Let denote the sets of all upward-movable cuts and downward-movable cuts by $\mathcal{M}^\uparrow$ and $\mathcal{M}^\downarrow$, respectively.

▶ **Lemma 9.** *For $(Y, Z) \in \mathcal{M}^\uparrow \cup \mathcal{M}^\downarrow$, it holds $h(p^{Y,Z}) - h(p) = -\kappa(Y, Z)/2$.*

Thus we are motivated to obtain an upward- or downward-movable cut (Y, Z) with a positive $\kappa(Y, Z)$ value. The following lemma says that we can do this efficiently given a maximum violating cut.

▶ **Lemma 10.** *Given a maximum violating cut, we can obtain an upward-movable cut* $(Y, Z) \in \mathcal{M}^{\uparrow}$ *and a downward-movable cut* $(Y', Z') \in \mathcal{M}^{\downarrow}$ *satisfying*

$$\kappa(Y, Z) = \max_{(Y'', Z'') \in \mathcal{M}^{\uparrow}} \kappa(Y'', Z''), \quad \kappa(Y', Z') = \max_{(Y'', Z'') \in \mathcal{M}^{\downarrow}} \kappa(Y'', Z'') \tag{11}$$

in $O(kn)$ *time. Moreover, at least one of* $\kappa(Y, Z)$ *and* $\kappa(Y', Z')$ *is positive.*

▶ **Theorem 11.** *Let* $\mathcal{N}_p := ((U, F), \underline{c}, \overline{c})$ *be the undirected network constructed from a proper potential* $p \in \mathbb{S}^n$*. Suppose that the instance is infeasible. Given a maximum violating cut, we can obtain a proper potential* $q \in \mathbb{S}^n$ *with* $h(q) < h(p)$ *in* $O(kn)$ *time.*

Proof. By Lemma 10, we can obtain an upward-movable cut $(Y, Z) \in \mathcal{M}^{\uparrow}$ and a downward-movable cut $(Y', Z') \in \mathcal{M}^{\downarrow}$ satisfying (11) in $O(kn)$ time. Let (Y'', Z'') be the cut that attains maximum κ-value among $\{(Y, Z), (Y', Z')\}$, and let $q := p^{Y'', Z''}$. Then $h(q) < h(p)$ by Lemmas 9 and 10. We can make q proper by the procedure given in the first part of the proof of Proposition 4. ◀

Now we are ready to present the details of DESCENT. First construct $\mathcal{N}_p$ from the current proper potential $p \in \mathbb{S}^n$, and run the algorithm given in Lemma 7 to solve the circulation problem; this corresponds to Step 1 given in the procedure at the end of Section 2.1. If a feasible half-integer-valued circulation is obtained, then a half-integral optimal edge-capacity x is computed by Theorem 8; this corresponds to Step 2. Otherwise a maximum violating cut is obtained, and then a proper potential $q \in \mathbb{S}^n$ with $h(q) < h(p)$ is computed by Theorem 11; this corresponds to Step 3. One iteration of this algorithm can be done in $O(\mathrm{MF}(kn, m + k^2 n))$ time.

The value $-h(p)$ is at most mUA (by Proposition 4) and $-h(p) \in \mathbb{Z}^*_+$. Thus the number of iterations is at most $O(mUA)$. Actually, this analysis of the time complexity is not tight. In fact, the number of iterations can be evaluated as $O(nA)$.

If a potential $q \in \mathbb{S}^n$ is obtained from a potential p by a modification defined by a movable cut on $\mathcal{N}_p$, then we say that q is a *neighbor* of p, that is, there exists a movable cut $(Y', Z') \in 3^U$ such that $q = p^{Y', Z'}$. For $p, q \in \mathbb{S}^n$, define a distance $\tilde{d}_{\mathbb{S}^n}(p, q)$ by the minimum length of a sequence $(p = p_0, p_1, \dots, p_\ell = q)$ such that p_t is a neighbor of p_{t-1} for all $t = 1, \dots, \ell$. Let $\mathrm{opt}(h)$ denote the set of minimizers of h, and let $\tilde{d}_{\mathbb{S}^n}(p, \mathrm{opt}(h)) := \min_{q \in \mathrm{opt}(h)} \tilde{d}_{\mathbb{S}^n}(p, q)$.

▶ **Lemma 12.** *Starting with an initial potential* $p_0 \in \mathbb{S}^n$*, DESCENT finds an optimal potential at most* $\tilde{d}_{\mathbb{S}^n}(p_0, \mathrm{opt}(h)) + 2$ *iterations.*

Lemma 12 can be shown by using *DCA beyond* $\mathbb{Z}^n$. We will discuss it in Section 3.

▶ **Lemma 13.** *There exists an optimal potential* $p \in \mathrm{opt}(h)$ *satisfying that for any* $i \in V$*,* p_i *is contained in* $(2nA, 2nA, \dots, 2nA) \in \mathbb{S}$.

▶ **Theorem 14.** *DESCENT solves FNTB in* $O(nA \cdot \mathrm{MF}(kn, m + k^2 n))$ *time.*

Proof. We can only consider the potentials satisfying the condition in Lemma 13. Any pair of such potentials $p, q \in \mathbb{S}$ satisfies $\tilde{d}_{\mathbb{S}^n}(p, q) = O(nA)$. Then the statement follows from Lemma 12. ◀

We note that Theorem 14 is shown under the positivity assumption of the edge-cost a. We prove Theorem 2 using Theorem 14.

Proof of Theorem 2. Let $f = (\mathcal{P}, \lambda)$ be a separately-capacitated multiflow. Recall that $f_s = (\mathcal{P}_s, \lambda|_{\mathcal{P}_s})$, where $\mathcal{P}_s \subseteq \mathcal{P}$ is a subset of paths connecting s to other terminals. Let $\operatorname{val} f := \sum_{P \in \mathcal{P}} \lambda(P)$ and $\operatorname{val} f_s := \sum_{P \in \mathcal{P}_s} \lambda(P)$ for $s \in S$. Then $\operatorname{val} f_s$ is at most the capacity of any $\{s\}$–$(S \setminus \{s\})$ cut. Thus $\operatorname{val} f = (1/2) \sum_{s \in S} \operatorname{val} f_s \leq (1/2) \sum_{s \in S} \nu_s$.

Consider an instance $((V,E), S, u, c, a, r)$ of FNTB, where $a \equiv 1$ and $r_s := \nu_s$ for each $s \in S$. Since u clearly satisfies (1), this instance is feasible. Then DESCENT outputs a half-integral optimal edge-capacity x and an optimal potential p. Since x and p satisfy the conditions (A1–5), we can apply the decomposition algorithm in the proof of Lemma 5 for x, and obtain a separately-capacitated multiflow f. Then $\operatorname{val} f = (1/2) \sum_{s \in S} f(s) \geq (1/2) \sum_{s \in S} r_s = (1/2) \sum_{s \in S} \nu_s$. Moreover, since x comes from a half-integral circulation (Theorem 8), x satisfies $x(\delta i) \in \mathbb{Z}_+$ for any $i \in V \setminus S$. In fact, for $i \in \bigcup_{s \in S} V_s$, it is observed from $x(\delta i) = -2y(i^0 i^+)$, and for $i \in V_0$, it is observed from $x(\delta i) = \sum_{s \in S} -y(i^{s,0} i^{s,+}) = 2 \sum_{s < s'} y(i^{s,0}, i^{s',0})$. Then by Remark 6, the decomposition algorithm outputs a half-integer-valued multiflow.

The time complexity result follows from that FNTB can be solved in $O(n \cdot \mathrm{MF}(kn, m + k^2 n))$ time by Theorem 14, and the decomposition algorithm runs in $O((m + kn)n)$ time. ◀

2.4 Scaling Algorithm

The time complexity of DESCENT is pseudo-polynomial. We improve it by combining with a (cost-)scaling method.

Let $\gamma \in \mathbb{Z}_+$ be an integer such that $2^\gamma \geq A$. The scaling algorithm consists of $\gamma + 1$ phases. In t-th phase, solve DTB with an edge-cost $a_t : E \to \mathbb{Z}_+$ defined by $a_t(e) := \lceil a(e)/2^t \rceil$ $(e \in E)$, i.e., minimize h_{a_t}. (Recall h_a is defined by (10).) Here $\lceil \cdot \rceil$ is the round-up operator. Note that all $a_t(e)$ are positive. Begin with $t = \mu$, and decrease t one-by-one. Then, when $t = 0$, the problem coincides with the original DTB. In each t-phase, we use DESCENT to minimize h_{a_t}. At the initial phase $t = \mu$, we run DESCENT with the starting point $p_0 \in \mathbb{S}^n$, where $(p_0)_i = 0$ for all $i \in V$. For t-phase with $t \leq \mu - 1$, the starting point is determined from the obtained optimal potential in the previous phase. Let $2[l, l']_s := [2l, 2l']_s$ and $2[l_s]_{s \in S} := [2l_s]_{s \in S}$. For a potential $p \in \mathbb{S}^n$, define a new potential $2p \in \mathbb{S}^n$ by $(2p)_i := 2p_i$ for $i \in V$.

▶ **Lemma 15.** *Let $p \in \mathbb{S}^n$ be an optimal potential for t-phase ($t = 1, \ldots, \mu$). Then the potential $2p \in \mathbb{S}^n$ is optimal for DTB with an edge-cost $2a_t$.*

Proof. By the strong duality of Proposition 4, there exists a solution $x : E \to \mathbb{R}$ for FNTB, such that $\sum_{e \in E} a_t(e) x(e) = -h_{a_t}(p)$. Then $\sum_{e \in E} 2a_t(e) x(e) = -h_{2a_t}(2p)$ holds, which implies the optimality of $2p$ by (the weak duality of) Proposition 4. ◀

Observe that $a_{t-1} = 2a_t - \sum_{e \in F} \chi_e$, where $F := \{e \in E \mid a_{t-1}(e) \text{ is odd}\}$. The key property is the following sensitivity result.

▶ **Lemma 16.** *Let $a : E \to \mathbb{Z}_+$ be a positive edge-cost. Let $e \in E$ be an edge satisfying $a(e) \geq 2$, and $a' := a - \chi_e$. Let $p \in \operatorname{opt}(h_a)$. Then $\tilde{d}_{\mathbb{S}^n}(p, \operatorname{opt}(h_{a'})) \leq 2$.*

We prove Lemma 16 in Section 3.3 using the notion of discrete convexity.

Proof of Theorem 1. For the initial phase $t = \mu$, an optimal potential can be obtained in $O(n)$ iterations of DESCENT by Lemmas 12 and 13. For each remaining phase, an optimal potential can be obtained in $O(m)$ iterations of DESCENT by Lemmas 12, 15 and 16. Thus $O(n + m \log A) = O(m \log A)$ iterations of DESCENT are sufficient. Recall that we assume the positivity of the edge-cost a. When a is not positive, the perturbation (Remark 3) is needed. Thus the maximum of edge-costs is $O(mUA)$. Then the theorem follows. ◀

3 Discrete Convex Analysis for Node-Connectivity Terminal Backup

The theory of DCA beyond $\mathbb{Z}^n$ gives an algorithm, called the steepest descent algorithm (SDA), for minimizing L-convex functions on certain graph structures. We first introduce the L-convexity and SDA, and next show that DESCENT is precisely SDA for an L-convex function. Then Lemma 12 immediately follows. Finally, we discuss a sensitivity argument, which shows Lemma 16.

3.1 A General Theory

In this subsection, we briefly introduce a theory of discrete convexity on graph structures specialized to median graphs. See [15] for further details.

We use basic terminologies of poset and lattice. Let $\mathcal{L}$ be a poset (partially ordered set) with a partial order $\preceq$. The *principal filter* $\mathcal{F}_x$ and the *principal ideal* $\mathcal{I}_x$ of $x \in \mathcal{L}$ are defined as $\{y \in \mathcal{L} \mid y \succeq x\}$ and $\{y \in \mathcal{L} \mid y \preceq x\}$, respectively. For $x, y \in \mathcal{L}$ with $x \preceq y$, the *interval* $[x, y]$ is defined as the set of $z \in \mathcal{L}$ satisfying $x \preceq z \preceq y$. We consider a (meet-)semilattice having the minimum element. A *median semilattice* $\mathcal{L}$ is a semilattice that every principal ideal is a distributive lattice and for any $x, y, z \in \mathcal{L}$, the join $x \vee y \vee z$ exists if $x \vee y$, $y \vee z$, and $z \vee x$ exist. A *Boolean semilattice* is a median semilattice that every principal ideal is a Boolean lattice.

Let G be a (possibly infinite) undirected graph. We denote the set of nodes also by G. Let $d = d_G$ be the shortest path metric on G. The *(metric) interval* $I(u, v)$ of $u, v \in G$ is the set of $w \in G$ satisfying $d(u, v) = d(u, w) + d(w, v)$. A *median* graph G is a graph that for any $u, v, w \in G$, $I(u, v) \cap I(v, w) \cap I(w, u)$ is a singleton.

We consider an *orientation* on edges of a median graph G, that takes $u \searrow v$ or $u \swarrow v$ on each edge uv. An orientation is *admissible* if for any 4-cycle (u_1, u_2, u_3, u_4), $u_1 \searrow u_2$ implies $u_4 \searrow u_3$. It is known [13, Lemma 2.4] that an admissible orientation on a median graph is acyclic. Thus we can define a poset on G by the admissible orientation, i.e., if an edge uv is oriented as $u \swarrow v$, then $u \preceq v$. G with an admissible orientation is *well-oriented* if $[u, v]$ is a Boolean lattice for any u, v with $u \preceq v$. In a well-oriented median graph G, it is known [15, Proposition 2] that every principal filter of G is a Boolean semilattice, and every principal ideal of G is a Boolean semilattice with the reversed order.

We can define an L-convex function on a well-oriented median graph G. For a function $f : G \to \overline{\mathbb{R}}$, define the *effective domain* of f as $\{u \in G \mid f(u) < \infty\}$ and denote by $\operatorname{dom} f$. If a sequence of nodes $(u = u_0, u_1, \ldots, u_\ell = v)$ satisfies that for any $i = 1, \ldots, \ell$, there exist $u', v' \in G$ with $u' \preceq v'$ such that $\{u_{i-1}, u_i\} \subseteq [u', v']$, then the sequence is said to be a Δ-*path* connecting u and v. A subset $X \subseteq G$ is Δ-*connected* if for any $u, v \in X$, there exists a Δ-path in X connecting u and v. A function $f : G \to \overline{\mathbb{R}}$ is called *L-convex* if $\operatorname{dom} f$ is Δ-connected and the restrictions of f to every principal filter and ideal are submodular. Here the *submodularity* on a median semilattice is a rather complicated notion; we give a formal definition in the full version.

The global optimality of an L-convex function f can be characterized by a *local* condition; $u \in \operatorname{dom} f$ is a minimizer of f if and only if u is a minimizer of f restricted to $\mathcal{F}_u \cup \mathcal{I}_u$. This induces a natural minimization algorithm, called the *steepest descent algorithm* (SDA):

Algorithm 2 SDA.

```
0. Initialize $u \in G$ with $f(u) < \infty$.
1. Find a local minimizer $v \in \mathcal{F}_u \cup \mathcal{I}_u$ of $f$.
2. If $f(v) = f(u)$, then stop; output $u$. Otherwise update $u$ by $v$ and go to Step 1.
```

The number of iterations of SDA is bounded by the Δ-distance from the initial point u and minimizers of f. Here the *Δ-distance* $d^{\Delta}(u,v)$ of $u,v \in G$ is the minimum length of a Δ-path connecting u to v. Let $\mathrm{opt}(f)$ denote the set of minimizers of f, and let $d^{\Delta}(u, \mathrm{opt}(f)) := \min_{v \in \mathrm{opt}(f)} d^{\Delta}(u,v)$.

▶ **Theorem 17** ([15, Theorem 4.3]). *The number of iterations of SDA with the initial point $u \in G$ is at most $d^{\Delta}(u, \mathrm{opt}(f)) + 2$.*

3.2 Discrete Convexity in Node-Connectivity Terminal Backup

We show that the dual objective function h defined in (10) is actually an L-convex function, and the algorithm DESCENT is precisely SDA. Define a graph on $\mathbb{S}$ by connecting two nodes (subtrees) $T, T' \in \mathbb{S}$ such that T and T' can transform to each other by a basic move. If we can move T to T' by a basic downward-move (equivalently, we can move T' to T by a basic upward-move), we give an orientation $T \searrow T'$. The graph $\mathbb{S}$ is a median graph, but not well-oriented. To make the graph well-oriented, we add a virtual subtree connecting to nodes (l,s) and $(l+1/2, s)$ for each $l \in \mathbb{Z}_+^*$ and $s \in S$. We denote such a virtual subtree by $[l+1/2, l]_s$. Give a natural orientation to each added edge. Let $\overline{\mathbb{S}} := \mathbb{S} \cup \{[l+1/2,l]_s \mid l \in \mathbb{Z}_+^*, s \in S\}$. Extend h to be a function on $\overline{\mathbb{S}}^n$ by $h(p) := \infty$ if there exists $i \in V$ such that $p_i \in \overline{\mathbb{S}} \setminus \mathbb{S}$.

▶ **Proposition 18.**

(1) *$\overline{\mathbb{S}}$ is a well-oriented median graph, and so is $\overline{\mathbb{S}}^n$.*

(2) *h is an L-convex function on $\overline{\mathbb{S}}^n$.*

(3) *For $p, q \in \mathbb{S}^n$, $\tilde{d}_{\mathbb{S}^n}(p,q) = d^{\Delta}(p,q)$.*

(4) *The map $(Y,Z) \mapsto p^{Y,Z}$ is a bijection between $\mathcal{M}^{\uparrow}$ and $\mathcal{F}_p \cap \mathrm{dom}\, h$, and $\mathcal{M}^{\downarrow}$ and $\mathcal{I}_p \cap \mathrm{dom}\, h$.*

Proof of Lemma 12. By Lemma 9 and Proposition 18 (4), the cuts (Y,Z) and (Y',Z') in Lemma 10 are minimizers of h on $\mathcal{F}_p$ and $\mathcal{I}_p$, respectively. Therefore DESCENT is precisely SDA for h. Thus the number of iterations can be evaluated by Theorem 17, and the statement follows from Proposition 18 (3). ◀

3.3 Sensitivity

To prove Lemma 16, we transform the instance $((V,E), S, u, c, a, r)$ of FNTB to an edge-uncapacitated one by a standard technique: Divide each edge $e \in E$ into two edges e_1, e_2, and add a new node v_e into the middle of these two edges. Let the edge-costs of e_1 and e_2 be the same as the original edge-cost of e, and let the edge-capacities of e_1 and e_2 be ∞. Let the node-capacity of the added node be $u(e)$. The number of vertices in the new instance is $|V| + |E| = n + m$, and the number of edges is $2|E| = 2m$. We denote the new instance by $((\bar{V}, \bar{E}), S, \bar{u}, \bar{c}, \bar{a}, r)$.

We consider the dual problem DTB for the edge-uncapacitated instance. In this case, we say that $\bar{p} \in \mathbb{S}^{n+m}$ is a *potential* for an edge-cost $\bar{a}$ if it satisfies (4) and $\mathrm{dist}(\bar{p}_i, \bar{p}_j) \leq \bar{a}_{ij}$ for any $ij \in \bar{E}$. Then DTB is a minimization of a function $h_a : \mathbb{S}^n \to \overline{\mathbb{R}}$ defined by

$$\bar{h}_{\bar{a}}(\bar{p}) := -\sum_{s \in S} r_s \,\mathrm{dist}(0, \bar{p}_s) + \sum_{i \in \bar{V} \setminus S} \bar{c}_i \,\mathrm{size}(\bar{p}_i) \tag{12}$$

if $\bar{p}$ is a potential for $\bar{a}$ and $\bar{h}_{\bar{a}}(\bar{p}) := \infty$ otherwise.

Let $p \in \mathbb{S}^n$ be a potential for the original instance. We can extend p to a potential $\bar{p}$ for the edge-uncapacitated instance as follows: For $v = i \in V$, let $\bar{p}_v := 2p_i$. For $v = v_{ij}$ ($ij \in E$), we have two cases $\mathrm{dist}(p_i, p_j) \leq a_{ij}$ and $\mathrm{dist}(p_i, p_j) > a_{ij}$. For the former case, let $\bar{p}_v$ be any point in $\mathbb{T}$ (i.e., $\mathrm{size}(\bar{p}_v) = 0$) satisfying $\mathrm{dist}(\bar{p}_i, \bar{p}_v) \leq a_{ij}$ and $\mathrm{dist}(\bar{p}_v, \bar{p}_j) \leq a_{ij}$. For the latter case, let $\bar{p}_v$ satisfy $\mathrm{dist}(\bar{p}_i, \bar{p}_v) = a_{ij}$, $\mathrm{dist}(\bar{p}_v, \bar{p}_j) = a_{ij}$ and $\mathrm{size}(\bar{p}_v) = 2(\mathrm{dist}(\bar{p}_v, \bar{p}_j) - a_{ij}) > 0$.

▶ **Proposition 19.** *Let $p \in \mathbb{S}^n$ be an optimal potential for the original instance. Then the extended potential $\bar{p} \in \mathbb{S}^{n+m}$ defined above is optimal for the edge-uncapacitated instance.*

We first show Lemma 16 for an edge-uncapacitated instance. For brevity, we assume that the original instance $((V, E), S, u, c, a, r)$ is already an edge-capacitated instance. By Proposition 18 (3), the following is equivalent to Lemma 16.

▶ **Lemma 20.** *Let $a : E \to \mathbb{Z}_+$ be a positive edge-cost. Let $ij \in E$ be an edge satisfying $a(ij) \geq 2$, and $a' := a - \chi_{ij}$. Then for any $p \in \mathrm{opt}(h_a)$, it holds $d^\Delta(p, \mathrm{opt}(h_{a'})) \leq 2$.*

We prove Lemma 20 via the notion of *normal Δ-paths*. Let G be an oriented median graph. For nodes $u, v \in G$ with $d^\Delta(u, v) = 1$, let $\langle\langle u, v\rangle\rangle$ be the minimum interval $[u', v']$ such that $\{u, v\} \subseteq [u', v']$. A Δ-path $(u = u_0, u_1, \ldots, u_\ell = v)$ is the *normal Δ-path* from u to v if for any $t = 1, \ldots, \ell - 1$ and any interval $[u', v']$ with $\{u_{t-1}, u_t\} \subseteq [u', v']$ it holds $[u', v'] \cap \langle\langle u_t, u_{t+1}\rangle\rangle = \{u_t\}$. The normal Δ-path from u to v is uniquely determined, and the length ℓ equals to ${d_G}^\Delta(u, v)$ [3, Theorem 6.24]. Let $u \to v$ denote u_1, and let $u \twoheadrightarrow v$ denote $u_{\ell-1}$. Also Let $u \to^t v$ denote u_t for $t = 0, \ldots, \ell$.

▶ **Lemma 21.** *Let $p, q \in \mathrm{dom}\, h_a$. Then*

$$h_a(p) + h_a(q) \geq h_a(p \to q) + h_a(q \to p), \tag{13}$$

$$h_a(p) + h_a(q) \geq h_a(p \twoheadrightarrow q) + h_a(q \twoheadrightarrow p). \tag{14}$$

▶ **Lemma 22.** *Let $p, q \in \overline{\mathbb{S}}^n$ and $i, j \in V$. Suppose that $\mathrm{dist}(q_i, q_j) < \mathrm{dist}((q \to p)_i, (q \to p)_j)$ and $\mathrm{dist}(q_i, q_j) < \mathrm{dist}((p \twoheadrightarrow q)_i, (p \twoheadrightarrow q)_j)$. Then for any $t = 1, \ldots, d^\Delta(p, q)$, it holds $\mathrm{dist}((p \to^t q)_i, (p \to^t q)_j) + 1/2 \leq \mathrm{dist}((p \to^{t-1} q)_i, (p \to^{t-1} q)_j)$.*

Proof of Lemma 20. If p is a potential for a', then $p \in \mathrm{opt}(h_{a'})$. Suppose that p is not a potential for a'. Take $q \in \mathrm{opt}(h_{a'})$ having the minimum Δ-distance from p. Then $q \in \mathrm{dom}\, h_a$. Thus by (13) and $p \in \mathrm{opt}(h_a)$, we have $h_a(q) \geq h_a(q \to p)$. If $(q \to p) \in \mathrm{dom}\, h_{a'}$, then $h_{a'}(q \to p) = h_a(q \to p) \leq h_a(q) = h_{a'}(q)$ and thus $(q \to p) \in \mathrm{opt}(h_{a'})$; a contradiction to the minimality of q. Hence $(q \to p) \notin \mathrm{dom}\, h_{a'}$, and $\mathrm{dist}((q \to p)_i, (q \to p)_j) \geq a'_{ij} + 1/2 > a'_{ij} \geq \mathrm{dist}(q_i, q_j)$ (by the half-integrality of $\mathrm{dist}(\cdot, \cdot)$). Similarly we have $\mathrm{dist}((p \twoheadrightarrow q)_i, (p \twoheadrightarrow q)_j) > \mathrm{dist}(q_i, q_j)$. Then we can apply Lemma 22 and obtain

$$\begin{aligned} \mathrm{dist}(p_i, p_j) &\geq \mathrm{dist}((p \to q)_i, (p \to q)_j) + 1/2 \\ &\geq \mathrm{dist}((p \to^2 q)_i, (p \to^2 q)_j) + 2/2 \\ &\geq \cdots \geq \mathrm{dist}((p \twoheadrightarrow q)_i, (p \twoheadrightarrow q)_j) + (d^\Delta(p, q) - 1)/2. \end{aligned}$$

By $\mathrm{dist}(p_i, p_j) \leq a_{ij}$ and $\mathrm{dist}((p \twoheadrightarrow q)_i, (p \twoheadrightarrow q)_j) \geq a'_{ij} + 1/2 = a_{ij} - 1/2$, we have

$$d^\Delta(p, q) \leq 1 + 2(\mathrm{dist}(p_i, p_j) - \mathrm{dist}((p \twoheadrightarrow q)_i, (p \twoheadrightarrow q)_j)) \leq 2. \qquad ◀$$

We give a sketch of a proof of Lemma 16 for an edge-capacitated instance. First construct the edge-uncapacitated instance $((\bar{V}, \bar{E}), S, \bar{u}, \bar{c}, \bar{a}, r)$ as above. Then an optimal potential $\bar{p} \in \mathbb{S}^{n+m}$ is obtained from p by Proposition 19, and $e \in E$ is divided into two edges

$e_1, e_2 \in \bar{E}$. By Lemma 20 for e_1 and e_2, there exists an optimal potential $\bar{p}'$ for the edge-uncapacitated instance with $d^{\Delta}(\bar{p}, \bar{p}') \leq 4$. By halving $\bar{p}'$, a "quarter-integral" optimal potential $p' \in \mathrm{opt}(h_{a'})$ is obtained. Lemma 16 is then shown by rounding quarter-integral components to half-integral.

References

1 E. Anshelevich and A. Karagiozova. Terminal backup, 3D matching, and covering cubic graphs. *SIAM J. Comput.*, 40(3):678–708, 2011. doi:10.1137/090752699.

2 A. Bernáth, Y. Kobayashi, and T. Matsuoka. The generalized terminal backup problem. *SIAM J. Discrete Math.*, 29(3):1764–1782, 2015. doi:10.1137/140972858.

3 J. Chalopin, V. Chepoi, H. Hirai, and D. Osajda. Weakly modular graphs and nonpositive curvature. To appear in *Mem. Amer. Math. Soc.*

4 J. Cheriyan, S. Vempala, and A. Vetta. Network design via iterative rounding of setpair relaxations. *Combinatorica*, 26(3):255–275, 2006. doi:10.1007/s00493-006-0016-z.

5 B. V. Cherkassky. A solution of a problem of multicommodity flows in a network. *Ekonomika i Matematicheskie Metody*, 13:143–151, 1977. in Russian.

6 A. E. Feldmann, J. Könemann, K. Pashkovich, and L Sanità. Fast approximation algorithms for the generalized survivable network design problem. In *Proceedings of the 27th International Symposium on Algorithms and Computation*, pages 33:1–33:12, 2016. doi:10.4230/LIPIcs.ISAAC.2016.33.

7 L. Fleischer, K. Jain, and D. P. Williamson. Iterative rounding 2-approximation algorithms for minimum-cost vertex connectivity problems. *J. Comput. Syst. Sci.*, 72(5):838–867, 2006. doi:10.1016/j.jcss.2005.05.006.

8 T. Fukunaga. Approximating the generalized terminal backup problem via half-integral multiflow relaxation. *SIAM J. Discrete Math.*, 30(2):777–800, 2016. doi:10.1137/151004288.

9 N. Garg and J. Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. *SIAM Journal on Computing*, 37(2):630–652, 2007. doi:10.1137/S0097539704446232.

10 A. V. Goldberg and A. V. Karzanov. Scaling methods for finding a maximum free multiflow of minimum cost. *Math. Oper. Res.*, 22(1):90–109, 1997. doi:10.1287/moor.22.1.90.

11 H. Hirai. Half-integrality of node-capacitated multiflows and tree-shaped facility locations on trees. *Math. Program. A*, 137(1):503–530, 2013.

12 H. Hirai. L-extendable functions and a proximity scaling algorithm for minimum cost multiflow problem. *Discrete Optim.*, 18:1–37, 2015. doi:10.1016/j.disopt.2015.07.001.

13 H. Hirai. Discrete convexity and polynomial solvability in minimum 0-extension problems. *Math. Program. A*, 155(1):1–55, 2016. doi:10.1007/s10107-014-0824-7.

14 H. Hirai. A dual descent algorithm for node-capacitated multiflow problems and its applications. *ACM Trans. Algorithms*, 15(1):15:1–15:24, 2018. doi:10.1145/3291531.

15 H. Hirai. L-convexity on graph structures. *J. Oper. Res. Soc. Jap.*, 61(1):71–109, 2018. doi:10.15807/jorsj.61.71.

16 H. Hirai and M. Ikeda. A cost-scaling algorithm for minimum-cost node-capacitated multiflow problem, 2019. arXiv:1909.01599.

17 Y. Iwata, Y. Yamaguchi, and Y. Yoshida. 0/1/all CSPs, half-integral A-path packing, and linear-time FPT algorithms. In *Proceedings of the 59th IEEE Annual Symposium on Foundations of Computer Science*, pages 462–473, 2018. doi:10.1109/FOCS.2018.00051.

18 K. Jain. A factor 2 approximation algorithm for the generalized Steiner network problem. *Combinatorica*, 21(1):39–60, 2001. doi:10.1007/s004930170004.

19 A. V. Karzanov. A minimum cost maximum multiflow problem. In *Combinatorial Methods for Flow Problems*, pages 138–156. Institute for System Studies, Moscow, 1979. in Russian.

20 A. V. Karzanov. Minimum cost multiflows in undirected networks. *Math. Program.*, 66(1):313–325, 1994. doi:10.1007/BF01581152.

21 L. Lovász. On some connectivity properties of Eulerian graphs. *Acta Math. Acad. Sci. Hung.*, 28(1–2):129–138, 1976. doi:10.1007/BF01902503.

22 W. Mader. Über die Maximalzahl kreuzungsfreier *H*-Wege,. *Archiv. Math.*, 31(1):387–402, 1978. doi:10.1007/BF01226465.

23 K. Murota. *Discrete Convex Analysis*. SIAM, Philadelphia, 2003.

24 A. Schrijver. *Combinatorial Optimization—Polyhedra and Efficiency*. Springer-Verlag, Berlin, 2003.

A Dichotomy for Bounded Degree Graph Homomorphisms with Nonnegative Weights

Artem Govorov[1]
Department of Computer Sciences, University of Wisconsin-Madison, Madison, WI, USA
hovarau@cs.wisc.edu

Jin-Yi Cai
Department of Computer Sciences, University of Wisconsin-Madison, Madison, WI, USA
jyc@cs.wisc.edu

Martin Dyer
School of Computing, University of Leeds, UK
M.E.Dyer@leeds.ac.uk

Abstract

We consider the complexity of counting weighted graph homomorphisms defined by a symmetric matrix A. Each symmetric matrix A defines a graph homomorphism function $Z_A(\cdot)$, also known as the partition function. Dyer and Greenhill [10] established a complexity dichotomy of $Z_A(\cdot)$ for symmetric $\{0,1\}$-matrices A, and they further proved that its #P-hardness part also holds for bounded degree graphs. Bulatov and Grohe [4] extended the Dyer-Greenhill dichotomy to nonnegative symmetric matrices A. However, their hardness proof requires graphs of arbitrarily large degree, and whether the bounded degree part of the Dyer-Greenhill dichotomy can be extended has been an open problem for 15 years. We resolve this open problem and prove that for nonnegative symmetric A, either $Z_A(G)$ is in polynomial time for all graphs G, or it is #P-hard for bounded degree (and simple) graphs G. We further extend the complexity dichotomy to include nonnegative vertex weights. Additionally, we prove that the #P-hardness part of the dichotomy by Goldberg et al. [12] for $Z_A(\cdot)$ also holds for simple graphs, where A is any real symmetric matrix.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness

Keywords and phrases Graph homomorphism, Complexity dichotomy, Counting problems

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.66

Category Track A: Algorithms, Complexity and Games

Related Version `https://arxiv.org/abs/2002.02021`

Funding *Artem Govorov*: Supported by NSF CCF-1714275.
Jin-Yi Cai: Supported by NSF CCF-1714275.
Martin Dyer: Supported by EPSRC grant EP/S016562/1.

1 Introduction

The modern study of graph homomorphisms originates from the work by Lovász and others several decades ago and has been a very active area [18, 14]. If G and H are two graphs, a graph homomorphism (GH) is a mapping $f\colon V(G) \to V(H)$ that preserves vertex adjacency, i.e., whenever (u, v) is an edge in G, $(f(u), f(v))$ is also an edge in H. Many combinatorial problems on graphs can be expressed as graph homomorphism problems. Well-known examples include the problems of finding a proper vertex coloring, vertex cover, independent

[1] Artem Govorov is the author's preferred spelling of his name, rather than the official spelling Artsiom Hovarau.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 66; pp. 66:1–66:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

set and clique. For example, if $V(H) = \{0,1\}$ with an edge between 0 and 1 and a loop at 0, then $f\colon V(G) \to \{0,1\}$ is a graph homomorphism iff $f^{-1}(1)$ is an independent set in G; similarly, proper vertex colorings on G using at most m colors correspond to homomorphisms from G to $H = K_m$ (with no loops).

More generally, one can consider weighted graphs H and aggregate all homomorphisms from G to H into a weighted sum. This is a powerful graph invariant which can express many graph properties. Formally, for a symmetric $m \times m$ matrix A, the *graph homomorphism function* on a graph $G = (V, E)$ is defined as follows:

$$Z_A(G) = \sum_{\xi: V \to [m]} \prod_{(u,v) \in E} A_{\xi(u),\xi(v)}.$$

Note that if H is unweighted, and A is its $\{0,1\}$-adjacency matrix, then each product $\prod_{(u,v)\in E} A_{\xi(u),\xi(v)}$ is 0 or 1, and is 1 iff ξ is a graph homomorphism. Thus in this case $Z_A(G)$ counts the number of homomorphisms from G to H. One can further allow H to have vertex weights. In this case, we can similarly define the function $Z_{A,D}(\cdot)$ (see Definition 4).

These sum-of-product functions $Z_A(\cdot)$ and $Z_{A,D}(\cdot)$ are referred to as the *partition functions* in statistical physics [3]. Various special cases of GH have been studied there extensively, which include the Ising, Potts, hardcore gas, Beach, Widom-Rowlinsom models, etc. [3].

The computational complexity of $Z_A(\cdot)$ has been studied systematically. Dyer and Greenhill [10, 11] proved that, for a symmetric $\{0,1\}$-matrix A, $Z_A(\cdot)$ is either in polynomial time or #P-complete, and they gave a succinct condition for this complexity dichotomy: if A satisfies the condition then $Z_A(\cdot)$ is computable in polynomial time (we also call it *tractable*), otherwise it is #P-complete. Bulatov and Grohe [4] (see also [22, 13]) generalized the Dyer-Greenhill dichotomy to $Z_A(\cdot)$ for nonnegative symmetric matrices A. It was further extended by Goldberg et al. [12] to arbitrary real symmetric matrices, and finally by Cai, Chen and Lu [7] to arbitrary complex symmetric matrices. In the last two dichotomies, the tractability criteria are not trivial to state. Nevertheless, both tractability criteria are decidable in polynomial time (in the size of A).

The definition of the partition function $Z_A(\cdot)$ can be easily extended to directed graphs G and arbitrary (not necessarily symmetric) matrices A corresponding to directed edge weighted graphs H. Concerning the complexity of counting directed GH, we currently have the *decidable* dichotomies by Dyer, Goldberg and Paterson [9] for $\{0,1\}$-matrices corresponding to (unweighted) simple acyclic graphs H, and by Cai and Chen [6] for all nonnegative matrices A.

Dyer and Greenhill in the same paper [10] proved a stronger statement that if a $\{0,1\}$-matrix A fails the tractability condition then $Z_A(G)$ is #P-complete even when restricted to bounded degree graphs G. We note that the complexity of GH for bounded degree graphs is particularly interesting as much work has been done on the approximate complexity of GH focused on bounded degree graphs and approximate algorithms are achieved for them [8, 25, 21, 20, 17, 1, 2, 19, 15]. However, for fifteen years the worst case complexity for bounded degree graphs in the Bulatov-Grohe dichotomy was open. Since this dichotomy is used essentially in almost all subsequent work, e.g., [12, 7], this has been a stumbling block.

Our main contribution in this paper is to resolve this 15-year-old open problem. We prove that the #P-hardness part of the Bulatov-Grohe dichotomy still holds for *bounded degree graphs*. It can be further strengthened to apply to bounded degree *simple* graphs. We actually prove a broader dichotomy for $Z_{A,D}(\cdot)$, where in addition to the nonnegative symmetric edge weight matrix A there is also a nonnegative diagonal vertex weight matrix D. We will give an explicit tractability condition such that, if (A, D) satisfies the condition then $Z_{A,D}(G)$ is computable in polynomial time for all G, and if it fails the condition then $Z_{A,D}(G)$ is #P-hard even restricted to *bounded degree simple graphs* G. $Z_A(G)$ is the special case of $Z_{A,D}(G)$ when D is the identity matrix. Additionally, we prove that the #P-hardness part of the dichotomy by Goldberg et al. [12] for all real symmetric edge weight matrices A still holds for *simple graphs*. (Although in this case, whether under the same condition on A the #P-hardness still holds for bounded degree graphs is not resolved in the present paper.)

In order to prove the dichotomy theorem on bounded degree graphs, we have to introduce a nontrivial extension of the well-developed interpolation method [24]. We use some of the well-established techniques in this area of research such as stretchings and thickenings. But the main innovation is an overall design of the interpolation for a more abstract target polynomial than $Z_{A,D}$. To carry out the proof there is an initial condensation step where we combine vertices that have proportionately the same neighboring edge weights (technically defined by pairwise linear dependence) into a super vertex with a combined vertex weight. Note that this creates vertex weights even when initially all vertex weights are 1. When vertex weights are present, an approach in interpolation proof is to arrange things well so that in the end one can redistribute vertex weights to edge weights. However, when edge weights are not 0-1, any gadget design must deal with a quantity at each vertex that cannot be directly *redistributed*. This dependence has the form $\sum_{j=1}^{m_{\zeta(w)}} \alpha_{\zeta(w)j}\mu_{\zeta(w)j}^{\deg(w)}$, resulting from combining pairwise linearly dependent rows and columns, that depends on the vertex degree $\deg(w)$ in a complicated way. (We note that in the 0-1 case all $\mu_{\zeta(w)j} \in \{0, 1\}$, making it in fact degree *independent*.)

We overcome this difficulty by essentially introducing a virtual level of interpolation – an interpolation to realize some "virtual gadget" that cannot be physically realized, and yet its "virtual" vertex weights are suitable for redistribution. Technically we have to define an auxiliary graph G', and express the partition function in an extended framework, called $Z_{\mathscr{A},\mathscr{D}}$ on G' (see Definition 6). In a typical interpolation proof, there is a polynomial with coefficients that have a clear combinatorial meaning defined in terms of G, usually consisting of certain sums of exponentially many terms in some target partition function. Here, we will define a target polynomial with certain coefficients; however these coefficients do not have a direct combinatorial meaning in terms of $Z_{A,D}(G)$, but rather they only have a direct combinatorial meaning in terms of $Z_{\mathscr{A},\mathscr{D}}$ on G'. In a suitable "limiting" sense, a certain aggregate of these coefficients forms some useful quantity in the final result. This introduces a concomitant "virtual" vertex weight which depends on the vertex degree that is "just-right" so that it can be redistributed to become part of the incident edge weight, thus effectively killing the vertex weight. This leads to a reduction from $Z_C(\cdot)$ (without vertex weight) to $Z_{A,D}(\cdot)$, for some C that inherits the hardness condition of A, thus proving the #P-hardness of the latter. This high level description will be made clearer in Section 4. The nature of the degree dependent vertex weight introduces a substantial difficulty; in particular a direct adaptation of the proof in [10] does not work.

Our extended vertex-weighted version of the Bulatov-Grohe dichotomy can be used to correct a crucial gap in the proof by Thurley [23] for a dichotomy for $Z_A(\cdot)$ with Hermitian edge weight matrices A, where this degree dependence was also at the root of the difficulty. [2]

[2] In [23], the proof of Lemma 4.22 uses Lemma 4.24. In Lemma 4.24, A is assumed to have pairwise linearly independent rows while Lemma 4.22 does not assume this, and the author appeals to a twin reduction step in [10]. However, unlike in the 0-1 case [10], such a step incurs degree dependent vertex weights. This gap is fixed by our Theorem 8.

2 Preliminaries

In order to state all our complexity results in the strict notion of Turing computability, we adopt the standard model [16] of computation for partition functions, and require that all numbers be from an arbitrary but fixed algebraic extension of $\mathbb{Q}$. We use $\mathbb{R}$ and $\mathbb{C}$ to denote the sets of real and complex algebraic numbers. Many statements remain true in other fields or rings if arithmetic operations can be carried out efficiently in a model of computation (see [5] for more discussions on this issue).

For a positive integer n, we use $[n]$ to denote the set $\{1, \dots, n\}$. When $n = 0$, $[0] = \emptyset$. We use $[m : n]$, where $m \le n$, to denote $\{m, m + 1, \dots, n\}$.

In this paper, we consider undirected graphs unless stated otherwise. Following standard definitions, the graph G is allowed to have multiple edges but no loops. (However, we will touch on this issue a few times when G is allowed to have loops.) The graph H can have multiple edges and loops, or more generally, edge weights. For the graph H, we treat its loops as edges.

An edge-weighted graph H on m vertices can be identified with a symmetric $m \times m$ matrix A in the obvious way. We write this correspondence by $H = H_A$ and $A = A_H$.

▶ **Definition 1.** *Let $A \in \mathbb{C}^{m\times m}$ be a symmetric matrix. The problem* EVAL(A) *is defined as follows: Given an undirected graph $G = (V, E)$, compute*

$$Z_A(G) = \sum_{\xi: V \to [m]} \prod_{(u,v)\in E} A_{\xi(u),\xi(v)}.$$

The function $Z_A(\cdot)$ is called a *graph homomorphism function* or a *partition function*. When A is a symmetric $\{0, 1\}$-matrix, i.e., when the graph $H = H_A$ is unweighted, $Z_A(G)$ counts the number of homomorphisms from G to H. In this case, we denote EVAL(H) = EVAL(A_H), and this problem is also known as the #H-coloring problem.

▶ **Theorem 2** (Dyer and Greenhill [10]). *Let H be a fixed undirected graph. Then* EVAL(H) *is in polynomial time if every connected component of H is either (1) an isolated vertex, or (2) a complete graph with all loops present, or (3) a complete bipartite graph with no loops present. Otherwise, the problem* EVAL(H) *is #P-complete.*

Bulatov and Grohe [4] extended Theorem 2 to EVAL(A) where A is a symmetric matrix with nonnegative entries. In order to state their result, we need to define a few notions first.

We say a nonnegative symmetric $m \times m$ matrix A is rectangular if there are pairwise disjoint nonempty subsets of $[m]$: $T_1, \dots, T_r, P_1, \dots, P_s, Q_1, \dots, Q_s$, for some $r, s \ge 0$, such that $A_{i,j} > 0$ iff

$$(i, j) \in \bigcup_{k\in[r]} (T_k \times T_k) \cup \bigcup_{l\in[s]} [(P_l \times Q_l) \cup (Q_l \times P_l)].$$

We refer to $T_k \times T_k, P_l \times Q_l$ and $Q_l \times P_l$ as blocks of A. Further, we say a nonnegative symmetric matrix A is *block-rank-*1 if A is rectangular and every block of A has rank one.

▶ **Theorem 3** (Bulatov and Grohe [4]). *Let A be a symmetric matrix with nonnegative entries. Then* EVAL(A) *is in polynomial time if A is block-rank-*1*, and is #P-hard otherwise.*

There is a natural extension of EVAL(A) involving the use of vertex weights. Both papers [10, 4] use them in their proofs. A graph H on m vertices with vertex and edge weights is identified with a symmetric $m \times m$ edge weight matrix A and a diagonal $m \times m$ vertex weight matrix $D = \text{diag}(D_1, \dots, D_m)$ in a natural way. Then the problem EVAL(A) can be generalized to EVAL(A, D) for vertex-edge-weighted graphs.

▶ **Definition 4.** *Let $A \in \mathbb{C}^{m\times m}$ be a symmetric matrix and $D \in \mathbb{C}^{m\times m}$ a diagonal matrix. The problem* EVAL(A, D) *is defined as follows: Given an undirected graph $G = (V, E)$, compute*

$$Z_{A,D}(G) = \sum_{\xi:V\to[m]} \prod_{w\in V} D_{\xi(w)} \prod_{(u,v)\in E} A_{\xi(u),\xi(v)}.$$

Note that EVAL(A) is the special case EVAL(A, I_m). We also need to define another EVAL problem where the vertex weights are specified by the degree.

▶ **Definition 5.** *Let $A \in \mathbb{C}^{m\times m}$ be a symmetric matrix and $\mathfrak{D} = \{D^{[\![i]\!]}\}_{i=0}^{\infty}$ a sequence of diagonal matrices in $\mathbb{C}^{m\times m}$. The problem* EVAL$(A, \mathfrak{D})$ *is defined as follows: Given an undirected graph $G = (V, E)$, compute*

$$Z_{A,\mathfrak{D}}(G) = \sum_{\xi:V\to[m]} \prod_{w\in V} D^{[\![\deg(w)]\!]}_{\xi(w)} \prod_{(u,v)\in E} A_{\xi(u),\xi(v)}.$$

Finally, we need to define a general EVAL problem, where the vertices and edges can individually take specific weights. Let $\mathscr{A}$ be a set of (edge weight) $m \times m$ matrices and $\mathscr{D}$ a set of diagonal (vertex weight) $m \times m$ matrices. A GH-grid $\Omega = (G, \rho)$ consists of a graph $G = (V, E)$ with possibly both directed and undirected edges, and loops, and ρ assigns to each edge $e \in E$ or loop an $A^{(e)} \in \mathscr{A}$ and to each vertex $v \in V$ a $D^{(v)} \in \mathscr{D}$. (A loop is just an edge of the form (v, v).) If $e \in E$ is a directed edge then the tail and head correspond to rows and columns of $A^{(e)}$, respectively; if $e \in E$ is an undirected edge then $A^{(e)}$ must be symmetric.

▶ **Definition 6.** *The problem* EVAL$(\mathscr{A}, \mathscr{D})$ *is defined as follows: Given a GH-grid $\Omega = \Omega(G)$, compute*

$$Z_{\mathscr{A},\mathscr{D}}(\Omega) = \sum_{\xi\colon V\to[m]} \prod_{w\in V} D^{(w)}_{\xi(w)} \prod_{e=(u,v)\in E} A^{(e)}_{\xi(u),\xi(v)}$$

We remark that $Z_{\mathscr{A},\mathscr{D}}$ is introduced only as a tool to express a certain quantity in a "virtual" interpolation; the dichotomy theorems do not apply to this. Definitions 5 and 6 are carefully crafted in order to carry out the #P-hardness part of the proof of Theorem 8. Notice that the problem EVAL$(\mathscr{A}, \mathscr{D})$ generalizes both problems EVAL(A) and EVAL(A, D), by taking $\mathscr{A}$ to be a single symmetric matrix, and by taking $\mathscr{D}$ to be a single diagonal matrix. But EVAL$(A, \mathfrak{D})$ is not naturally expressible as EVAL$(\mathscr{A}, \mathscr{D})$ because the latter does not force the vertex-weight matrix on a vertex according to its degree.

We refer to $[m]$ as the domain of the corresponding EVAL problem. If $\mathscr{A} = \{A\}$ or $\mathscr{D} = \{D\}$, then we simply write $Z_{A,\mathscr{D}}(\cdot)$ or $Z_{\mathscr{A},D}(\cdot)$, respectively.

We use a superscript (Δ) and/or a subscript simp to denote the restriction of a corresponding EVAL problem to degree-Δ bounded graphs and/or simple graphs. E.g., EVAL$^{(\Delta)}(A)$ denotes the problem EVAL(A) restricted to degree-Δ bounded graphs, EVAL$_{\text{simp}}(A, \mathfrak{D})$ denotes the problem EVAL$(A, \mathfrak{D})$ restricted to simple graphs, and both restrictions apply in EVAL$^{(\Delta)}_{\text{simp}}(A, \mathfrak{D})$.

Working within the framework of EVAL(A, D), we define an edge gadget to be a graph with two distinguished vertices, called u^* and v^*. An edge gadget $G = (V, E)$ has a signature (edge weight matrix) expressed by an $m \times m$ matrix F, where

$$F_{ij} = \sum_{\substack{\xi\colon V\to[m] \\ \xi(u^*)=i,\, \xi(v^*)=j}} \prod_{z\in V\setminus\{u^*,v^*\}} D_{\xi(z)} \prod_{(x,y)\in E} A_{\xi(x),\xi(y)}$$

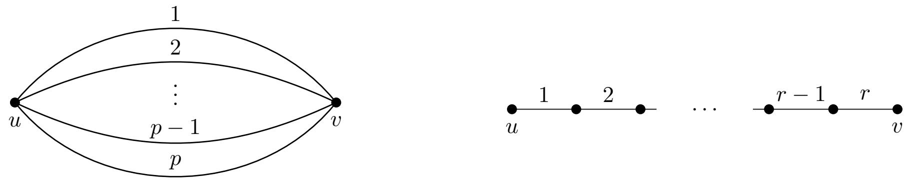


Figure 1 The thickening T_pe and the stretching S_re of an edge $e = (u, v)$.

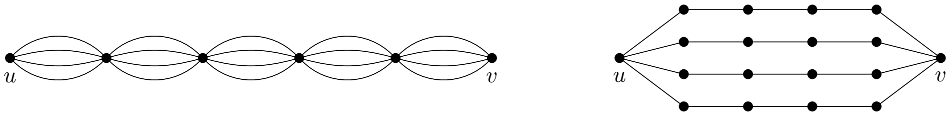


Figure 2 The graphs T_4S_5e (on the left) and S_5T_4e (on the right) where $e = (u, v)$.

for $1 \le i, j \le m$. When this gadget is placed in a graph identifying u^* and v^* with two vertices u and v in that graph, then F is the signature matrix for the pair (u, v). Note that the vertex weights corresponding to u and v are excluded from the product in the definition of F. Similar definitions can be introduced for $\mathrm{EVAL}(A)$, $\mathrm{EVAL}(A, \mathfrak{D})$ and $\mathrm{EVAL}(\mathscr{A}, \mathscr{D})$.

We use $\le_{\mathrm{T}}^{\mathrm{P}}$ (and $\equiv_{\mathrm{T}}^{\mathrm{P}}$) to denote polynomial-time Turing reductions (and equivalences, respectively).

Two simple operations are known as *thickening* and *stretching*. Let $p, r \ge 1$ be integers. A *p-thickening* of an edge replaces it by p parallel edges, and a *r-stretching* replaces it by a path of length r. In both cases we retain the endpoints u, v. The *p-thickening* or *r-stretching* of G with respect to $F \subseteq E(G)$, denoted respectively by $T_p^{(F)}(G)$ and $S_r^{(F)}(G)$, are obtained by *p-thickening* or *r-stretching* each edge from F, respectively. Other edges, if any, are unchanged in both cases. When $F = E(G)$, we call them the *p-thickening* and *r-stretching* of G and denote them by $T_p(G)$ and $S_r(G)$, respectively. T_pe and S_re are the special cases when the graph consists of a single edge e. See Figure 1 for an illustration. Thickenings and stretchings can be combined in any order. Examples are shown in Figure 2.

For a matrix A, we denote by $A^{\odot p}$ the matrix obtained by replacing each entry of A with its pth power. Clearly, $Z_A(T_pG) = Z_{A^{\odot p}}(G)$ and $Z_A(S_rG) = Z_{A^r}(G)$. More generally, for the vertex-weighted case, we have $Z_{A,D}(T_pG) = Z_{A^{\odot p},D}(G)$ and $Z_{A,D}(S_rG) = Z_{A(DA)^{r-1},D}(G)$. Here $(DA)^0 = I_m$ if A and D are $m \times m$.

3 Dichotomy for bounded degree graphs

In addition to the Dyer-Greenhill dichotomy (Theorem 2), in the same paper [10] they also proved that the #P-hardness part of their dichotomy holds for bounded degree graphs. The bounded degree case of the Bulatov-Grohe dichotomy (Theorem 3) was left open, and all known proofs [4, 22, 13] of its #P-hardness part require unbounded degree graphs. All subsequent dichotomies that use the Bulatov-Grohe dichotomy, e.g., [12, 7] also explicitly or implicitly (because of their dependence on the Bulatov-Grohe dichotomy) require unbounded degree graphs. In this paper, we extend the #P-hardness part of the Bulatov-Grohe dichotomy to bounded degree graphs.

▶ **Theorem 7.** *Let A be a symmetric nonnegative matrix. If A is not block-rank-1, then for some $\Delta > 0$, the problem $\mathrm{EVAL}^{(\Delta)}(A)$ is #P-hard.*

The degree bound Δ proved in Theorem 7 depends on A, as is the case in Theorem 2. The authors of [10] conjectured that a universal bound $\Delta = 3$ works for Theorem 2; whether a universal bound exists for both Theorems 2 and 7 is open. For general symmetric real or complex A, it is open whether bounded degree versions of the dichotomies in [12] and [7] hold. Xia [26] proved that a universal bound does not exist for complex symmetric matrices A, assuming #P does not collapse to P.

We prove a broader dichotomy than Theorem 7, which also includes arbitrary nonnegative vertex weights.

▶ **Theorem 8.** *Let A and D be $m \times m$ nonnegative matrices, where A is symmetric, and D is diagonal. Let A' be the matrix obtained from A by striking out rows and columns that correspond to 0 entries of D on the diagonal. If A' is block-rank-1, then the problem $\mathrm{EVAL}(A, D)$ is in polynomial time. Otherwise, for some $\Delta > 0$, the problem $\mathrm{EVAL}^{(\Delta)}_{\mathrm{simp}}(A, D)$ is #P-hard.*

Every 0 entry of D on the diagonal effectively nullifies the corresponding domain element in $[m]$, so the problem becomes an equivalent problem on the reduced domain. Thus, for a nonnegative diagonal D, without loss of generality, we may assume the domain has already been reduced so that D is positive diagonal. In what follows, we will make this assumption.

In Section 5, we will prove the tractability part of Theorem 8. This follows easily from known results. In Section 6, we will present two technical lemmas, Lemma 9 and Lemma 10 to be used in Section 4. Finally, in Section 7 we prove Theorem 11, showing that the #P-hardness part of the dichotomy for counting GH by Goldberg et al. [12] for real symmetric matrix (with mixed signs) is also valid for simple graphs.

4 Hardness proof

We proceed to prove the #P-hardness part of Theorem 8. Let A and D be $m \times m$ matrices, where A is nonnegative symmetric but not block-rank-1, and D is positive diagonal. The first step is to eliminate pairwise linearly dependent rows and columns of A. (We will see that this step will naturally create nontrivial vertex weights even if we initially start with the vertex unweighted case $D = I_m$.)

If A has a zero row or column i, then for any connected input graph G other than a single isolated vertex, no map $\xi : V(G) \to [m]$ having a nonzero contribution to $Z_{A,D}(G)$ can map any vertex of G to i. So, by crossing out all zero rows and columns (they have the same index set since A is symmetric) we may assume that A has no zero rows or columns. We then delete the same set of rows and columns from D, thereby expressing the problem $\mathrm{EVAL}^{(\Delta)}_{\mathrm{simp}}(A, D)$ for $\Delta \geq 0$ on a smaller domain. Also permuting the rows and columns of both A and D simultaneously by the same permutation does not change the value of $Z_{A,D}(\cdot)$, and so it does not change the complexity of $\mathrm{EVAL}^{(\Delta)}_{\mathrm{simp}}(A, D)$ for $\Delta \geq 0$ either. Having no zero rows and columns implies that pairwise linear dependence is an equivalence relation, and so we may assume that the pairwise linearly dependent rows and columns of A are contiguously arranged. Then, after renaming the indices, the entries of A are of the following form: $A_{(i,j),(i',j')} = \mu_{ij}\mu_{i'j'}A'_{i,i'}$, where A' is a nonnegative symmetric $s \times s$ matrix with all columns nonzero and pairwise linearly independent, $1 \leq i, i' \leq s$, $1 \leq j \leq m_i$, $1 \leq j' \leq m_{i'}$, $\sum_{i=1}^{s} m_i = m$, and all $\mu_{ij} > 0$. We also rename the indices of the matrix D so that the diagonal entries of D are of the following form: $D_{(i,j),(i,j)} = \alpha_{ij} > 0$ for $1 \leq i \leq s$ and $1 \leq j \leq m_i$. As $m \geq 1$ we get $s \geq 1$.

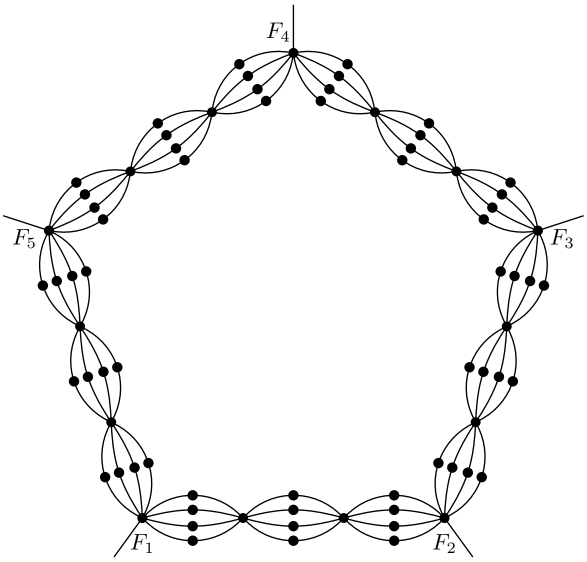


Figure 3 The gadget $\mathcal{R}_{5,3,4}$.

Then the partition function $Z_{A,D}(\cdot)$ can be written in a compressed form

$$Z_{A,D}(G) = \sum_{\zeta:V(G)\to[s]} \left(\prod_{w\in V(G)} \sum_{j=1}^{m_{\zeta(w)}} \alpha_{\zeta(w)j}\mu_{\zeta(w)j}^{\deg(w)} \right) \prod_{(u,v)\in E(G)} A'_{\zeta(u),\zeta(v)} = Z_{A',\mathfrak{D}}(G)$$

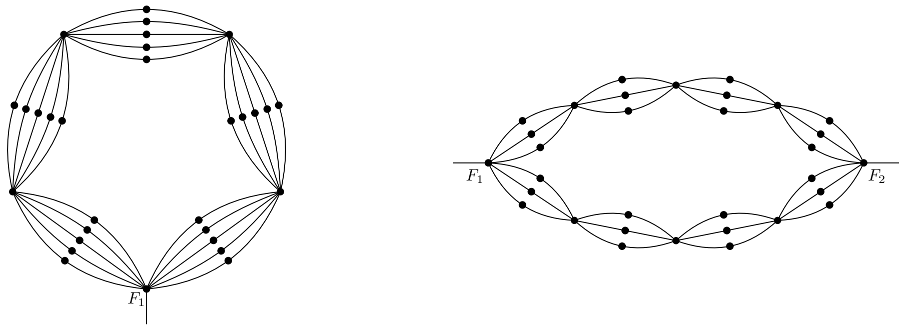
where $\mathfrak{D} = \{D^{[\![k]\!]}\}_{k=0}^{\infty}$ with $D_i^{[\![k]\!]} = \sum_{j=1}^{m_i} \alpha_{ij}\mu_{ij}^k > 0$ for $k \ge 0$ and $1 \le i \le s$. Then all matrices in $\mathfrak{D}$ are positive diagonal. Note the dependence on the vertex degree $\deg(w)$ for $w \in V(G)$. Since the underlying graph G remains unchanged, this way we obtain the equivalence $\text{EVAL}_{\text{simp}}^{(\Delta)}(A, D) \equiv_{\text{T}}^{\text{P}} \text{EVAL}_{\text{simp}}^{(\Delta)}(A', \mathfrak{D})$ for any $\Delta \ge 0$. Here the subscript simp can be included or excluded, and the same is true for the superscript (Δ), the statement remains true in all cases. We also point out that the entries of the matrices $D^{[\![k]\!]} \in \mathfrak{D}$ are computable in polynomial time in the input size of (A, D) as well as in k.

4.1 Gadgets $\mathcal{P}_{n,p}$ and $\mathcal{R}_{d,n,p}$

We first introduce the *edge gadget* $\mathcal{P}_{n,p}$, for all $p, n \ge 1$. It is obtained by replacing each edge of a path of length n by the gadget in Figure 5 from Lemma 10. More succinctly $\mathcal{P}_{n,p}$ is $S_2T_pS_ne$, where e is an edge.

To define the gadget $\mathcal{R}_{d,n,p}$, for all $d, p, n \ge 1$, we start with a cycle on d vertices $F_1, \dots, F_d$ (call it a d-cycle), replace every edge of the d-cycle by a copy of $\mathcal{P}_{n,p}$, and append a dangling edge at each vertex F_i of the d-cycle. To be specific, a 2-cycle has two vertices with 2 parallel edges between them, and a 1-cycle is a loop on one vertex. The gadget $\mathcal{R}_{d,n,p}$ always has d dangling edges. Note that all $\mathcal{R}_{d,n,p}$ are loopless simple graphs (i.e., without parallel edges or loops), for $d, n, p \ge 1$. An example of a gadget $\mathcal{R}_{d,n,p}$ is shown in Figure 3. For the special cases $d = 1, 2$, examples of gadgets $\mathcal{R}_{d,n,p}$ can be seen in Figure 4.

We note that vertices in $\mathcal{P}_{n,p}$ have degrees at most $2p$, and vertices in $\mathcal{R}_{d,n,p}$ have degrees at most $2p + 1$, taking into account the dangling edges. Clearly $|V(\mathcal{R}_{d,n,p})| = dn(p + 1)$ and $|E(\mathcal{R}_{d,n,p})| = (2np + 1)d$, including the dangling edges.



(a) $\mathcal{R}_{1,5,5}$

(b) $\mathcal{R}_{2,4,3}$

Figure 4 Examples of gadgets $\mathcal{R}_{d,n,p}$ for $d = 1, 2$.

By Lemma 10, we can fix some $p \geq 1$ such that $B = (A' D^{[\![2]\!]} A')^{\odot p}$ is nondegenerate, where the superscript $[\![2]\!]$ is from the stretching operator S_2 which creates those degree 2 vertices, and the superscript $\odot p$ is from the thickening operator T_p, followed by S_2, which creates those parallel paths of length 2. The edge gadget $\mathcal{P}_{n,p}$ has the edge weight matrix

$$L^{(n)} = \underbrace{BD^{[\![2p]\!]}B \dots BD^{[\![2p]\!]}B}_{D^{[\![2p]\!]} \text{ appears } n-1 \,\geq\, 0 \text{ times}} = B(D^{[\![2p]\!]}B)^{n-1} \tag{1}$$

$$= (D^{[\![2p]\!]})^{-1/2}((D^{[\![2p]\!]})^{1/2}B(D^{[\![2p]\!]})^{1/2})^{n}(D^{[\![2p]\!]})^{-1/2}, \tag{2}$$

where in the notation $L^{(n)}$ we suppress the index p. The $n-1$ occurrences of $D^{[\![2p]\!]}$ in (1) are due to those $n-1$ vertices of degree $2p$. Here $(D^{[\![2p]\!]})^{1/2}$ is a diagonal matrix with the positive square roots of the corresponding entries of $D^{[\![2p]\!]}$ on the main diagonal, and $(D^{[\![2p]\!]})^{-1/2}$ is its inverse. The vertices F_i are of degree $2p+1$ each, but the contributions by its vertex weights are not included in $L^{(n)}$.

The constraint function induced by $\mathcal{R}_{d,n,p}$ is more complicated to write down. When it is placed as a part of a graph, for any given assignment to the d vertices F_i, we can express the contribution of the gadget $\mathcal{R}_{d,n,p}$ in terms of d copies of $L^{(n)}$, *together with* the vertex weights incurred at the d vertices F_i which will depend on their degrees.

4.2 Interpolation using $\mathcal{R}_{d,n,p}$

Assume for now that G does not contain isolated vertices. We will replace every vertex $u \in V(G)$ of degree $d = d_u = \deg(u) \geq 1$ by a copy of $\mathcal{R}_{d,n,p}$, for all $n, p \geq 1$. The replacement operation can be described in two steps: In step one, each $u \in V(G)$ is replaced by a d-cycle on vertices $F_1, \dots, F_d$, each having a dangling edge attached. The d dangling edges will be identified one-to-one with the d incident edges at u. If u and v are adjacent vertices in G, then the edge (u, v) in G will be replaced by merging a pair of dangling edges, one from the d_u-cycle and one from the d_v-cycle. Thus in step one we obtain a graph G', which basically replaces every vertex $u \in V(G)$ by a cycle of $\deg(u)$ vertices. Then in step two, for every cycle in G' that corresponds to some $u \in V(G)$ we replace each edge on the cycle by a copy of the edge gadget $\mathcal{P}_{n,p}$.

Let $G_{n,p}$ denote the graph obtained from G by the replacement procedure above. Since all gadgets $\mathcal{R}_{d,n,p}$ are loopless simple graphs, so are $G_{n,p}$ for all $n, p \geq 1$, even if G has multiple edges (or had multiloops, if we view a loop as adding degree 2 to the incident vertex). As a technical remark, if G contains vertices of degree 1, then the intermediate graph G' has loops but all graphs $G_{n,p}$ $(n, p \geq 1)$ do not. Also note that all vertices in $G_{n,p}$ have degree at most $2p+1$, which is independent of n.

Next, it is not hard to see that

$$|V(G_{n,p})| = \sum_{u \in V(G)} d_u n(p+1) = 2n(p+1)|E(G)|,$$
$$|E(G_{n,p})| = |E(G)| + \sum_{u \in V(G)} 2npd_u = (4np+1)|E(G)|.$$

Hence the size of the graphs $G_{n,p}$ is polynomially bounded in the size of G, n and p.

Since we chose a fixed p, and will choose n to be bounded by a polynomial in the size of G, whenever something is computable in polynomial time in n, it is also computable in polynomial time in the size of G (we will simply say in polynomial time).

We consider $Z_{A',\mathfrak{D}}(G)$, and substitute G by $G_{n,p}$. We will make use of the edge weight matrix $L^{(n)}$ of $\mathcal{P}_{n,p}$ in (2). The vertices F_i are of degree $2p+1$ each in $G_{n,p}$, so will each contribute a vertex weight according to the diagonal matrix $D^{[\![2p+1]\!]}$ to the partition function, which are not included in $L^{(n)}$, but now must be accounted for in $Z_{A',\mathfrak{D}}(G_{n,p})$.

Since B is real symmetric and $D^{[\![2p]\!]}$ is positive diagonal, the matrix

$$\widetilde{B} = (D^{[\![2p]\!]})^{1/2} B (D^{[\![2p]\!]})^{1/2}$$

is real symmetric. Then $\widetilde{B}$ is orthogonally diagonalizable over $\mathbb{R}$, i.e., there exist a real orthogonal matrix S and a real diagonal matrix $J = \operatorname{diag}(\lambda_i)_{i=1}^{s}$ such that $\widetilde{B} = S^T J S$. Then $\widetilde{B}^n = S^T J^n S$ so the edge weight matrix for $\mathcal{P}_{n,p}$ becomes

$$L^{(n)} = (D^{[\![2p]\!]})^{-1/2} \widetilde{B}^n (D^{[\![2p]\!]})^{-1/2} = (D^{[\![2p]\!]})^{-1/2} S^T J^n S (D^{[\![2p]\!]})^{-1/2}.$$

Note that $L^{(n)}$ as a matrix is defined for any $n \geq 0$, and $L^{(0)} = (D^{[\![2p]\!]})^{-1}$, even though there is no physical gadget $\mathcal{P}_{0,p}$ that corresponds to it. However, it is precisely this "virtual" gadget we wish to "realize" by interpolation.

Clearly, $\widetilde{B}$ is nondegenerate as B and $(D^{[\![2p]\!]})^{1/2}$ both are, and so is J. Then all $\lambda_i \neq 0$. We can also write $L_{ij}^{(n)} = \sum_{\ell=1}^{s} a_{ij\ell} \lambda_\ell^n$ for every $n \geq 0$ and some real $a_{ij\ell}$'s which depend on S, $D^{[\![2p]\!]}$, but not on J and n, for all $1 \leq i, j, \ell \leq s$. By the formal expansion of the symmetric matrix $L^{(n)}$ above, we have $a_{ij\ell} = a_{ji\ell}$. Note that for all $n, p \geq 1$, the gadget $\mathcal{R}_{d_v,n,p}$ for $v \in V(G)$ employs exactly d_v copies of $\mathcal{P}_{n,p}$. Let $t = \sum_{v \in V(G)} d_v = 2|E(G)|$; this is precisely the number of edge gadgets $\mathcal{P}_{n,p}$ in $G_{n,p}$.

In the evaluation of the partition function $Z_{A',\mathfrak{D}}(G_{n,p})$, we stratify the vertex assignments in $G_{n,p}$ as follows. Denote by $\kappa = (k_{ij})_{1 \leq i \leq j \leq s}$ a tuple of nonnegative integers, where the indexing is over all $s(s+1)/2$ ordered pairs (i,j). There are a total of $\binom{t+s(s+1)/2-1}{s(s+1)/2-1}$ such tuples that satisfy $\sum_{1 \leq i \leq j \leq s} k_{ij} = t$. For a fixed s, this is a polynomial in t, and thus a polynomial in the size of G. Denote by $\mathcal{K}$ the set of all such tuples κ. We will stratify all vertex assignments in $G_{n,p}$ by $\kappa \in \mathcal{K}$, namely all assignments such that there are exactly k_{ij} many constituent edge gadgets $\mathcal{P}_{n,p}$ with the two end points (in either order of the end points) assigned i and j respectively.

For each $\kappa \in \mathcal{K}$, the edge gadgets $\mathcal{P}_{n,p}$ in total contribute $\prod_{1 \leq i \leq j \leq s} (L_{ij}^{(n)})^{k_{ij}}$ to the partition function $Z_{A',\mathfrak{D}}(G_{n,p})$. If we factor this product out for each $\kappa \in \mathcal{K}$, we can express $Z_{A',\mathfrak{D}}(G_{n,p})$ as a linear combination of these products over all $\kappa \in \mathcal{K}$, with polynomially many coefficient values c_κ that are independent of all edge gadgets $\mathcal{P}_{n,p}$. Another way to define these coefficients c_κ is to think in terms of G': For any $\kappa = (k_{ij})_{1 \leq i \leq j \leq s} \in \mathcal{K}$, we say a vertex assignment on G' is consistent with κ if it assigns exactly k_{ij} many cycle edges of G' (i.e., those that belong to the cycles that replaced vertices in G) as ordered pairs of vertices to the values (i,j) or (j,i). (For any loop in G', as a cycle of length 1 that came from a degree 1 vertex of G, it can only be assigned (i,i) for some $1 \le i \le s$.) Let L' be any symmetric edge signature to be assigned on each of these cycle edges in G', and keep the edge signature A' on the merged dangling edges between any two such cycles, and the suitable vertex weights specified by $\mathfrak{D}$, namely each vertex receives its vertex weight according to $D^{[\![2p+1]\!]}$. Then c_κ is the sum, over all assignments consistent with κ, of the products of all edge weights and vertex weights *other than* the contributions by L', in the evaluation of the partition function on G'. In other words, for each $\kappa \in \mathcal{K}$,

$$c_\kappa = \sum_{\substack{\zeta\colon V(G') \to [s] \\ \zeta \text{ is consistent with } \kappa}} \prod_{w \in V(G')} D^{[\![2p+1]\!]}_{\zeta(w)} \prod_{(u,v) \in \widetilde{E}} A'_{\zeta(u),\zeta(v)},$$

where $\widetilde{E} \subseteq E(G')$ are the non-cycle edges of G' that are in 1-1 correspondence with $E(G)$.

In particular, the values c_κ are independent of n. Thus for some polynomially many values c_κ, where $\kappa \in \mathcal{K}$, we have for all $n \ge 1$,

$$Z_{A',\mathfrak{D}}(G_{n,p}) = \sum_{\kappa \in \mathcal{K}} c_\kappa \prod_{1 \le i \le j \le s} (L_{ij}^{(n)})^{k_{ij}} = \sum_{\kappa \in \mathcal{K}} c_\kappa \prod_{1 \le i \le j \le s} (\sum_{\ell=1}^{s} a_{ij\ell} \lambda_\ell^n)^{k_{ij}}. \tag{3}$$

Expanding out the last sum and rearranging the terms, for some values $b_{i_1,\ldots,i_s}$ independent of n, we get

$$Z_{A',\mathfrak{D}}(G_{n,p}) = \sum_{\substack{i_1+\ldots+i_s=t \\ i_1,\ldots,i_s \ge 0}} b_{i_1,\ldots,i_s} (\prod_{j=1}^{s} \lambda_j^{i_j})^n$$

for all $n \ge 1$.

This represents a linear system with the unknowns $b_{i_1,\ldots,i_s}$ with the rows indexed by n. The number of unknowns is clearly $\binom{t+s-1}{s-1}$ which is polynomial in the size of the input graph G since s is a constant. The values $\prod_{j=1}^{s} \lambda_j^{i_j}$ can be clearly computed in polynomial time. We show how to compute the value

$$\sum_{\substack{i_1+\ldots+i_s=t \\ i_1,\ldots,i_s \ge 0}} b_{i_1,\ldots,i_s}$$

from the values $Z_{A',\mathfrak{D}}(G_{n,p})$, $n \ge 1$ in polynomial time. The coefficient matrix of this system is a Vandermonde matrix. However, it can have repeating columns so it might not be of full rank because the coefficients $\prod_{j=1}^{s} \lambda_j^{i_j}$ do not have to be pairwise distinct. However, when they are equal, say, $\prod_{j=1}^{s} \lambda_j^{i_j} = \prod_{j=1}^{s} \lambda_j^{i'_j}$, we replace the corresponding unknowns $b_{i_1,\ldots,i_s}$ and $b_{i'_1,\ldots,i'_s}$ with their sum as a new variable. Since all $\lambda_i \ne 0$, we have a Vandermonde system of full rank after all such combinations. Therefore we can solve this linear system in polynomial time and find the desired value $\sum_{\substack{i_1+\ldots+i_s=t \\ i_1,\ldots,i_s \ge 0}} b_{i_1,\ldots,i_s}$.

Now we will consider a problem in the framework of $Z_{\mathscr{A},\mathscr{D}}$ according to Definition 6. Let $G_{0,p}$ be the (undirected) GH-grid, with the underlying graph G', and every edge of the cycle in G' corresponding to a vertex in $V(G)$ is assigned the edge weight matrix $(D^{[\![2p]\!]})^{-1}$, and we keep the vertex-weight matrices $D^{[\![2p+1]\!]}$ at all vertices F_i. The other edges, i.e., the original edges of G, each keep the assignment of the edge weight matrix A'. (So in the specification of $Z_{\mathscr{A},\mathscr{D}}$, we have $\mathscr{A} = \{(D^{[\![2p]\!]})^{-1}, A'\}$, and $\mathscr{D} = \{D^{[\![2p+1]\!]}\}$. We note that G' may have loops, and Definition 6 specifically allows this.) Then

$$Z_{\{(D^{[\![2p]\!]})^{-1},A'\},D^{[\![2p+1]\!]}}(G_{0,p}) = \sum_{\substack{i_1+\ldots+i_s=t \\ i_1,\ldots,i_s \ge 0}} b_{i_1,\ldots,i_s} (\prod_{j=1}^{s} \lambda_j^{i_j})^0 = \sum_{\substack{i_1+\ldots+i_s=t \\ i_1,\ldots,i_s \ge 0}} b_{i_1,\ldots,i_s}$$

and we have just computed this value in polynomial time in the size of G from the values $Z_{A',\mathfrak{D}}(G_{n,p})$, for $n \geq 1$. In other words, we have achieved it by querying the oracle $\mathrm{EVAL}(A', \mathfrak{D})$ on the instances $G_{n,p}$, for $n \geq 1$, in polynomial time.

Equivalently, we have shown that we can simulate a virtual "gadget" $\mathcal{R}_{d,0,p}$ replacing every occurrence of $\mathcal{R}_{d,n,p}$ in $G_{n,p}$ in polynomial time. The virtual gadget $\mathcal{R}_{d,0,p}$ has the edge signature $(D^{\llbracket 2p \rrbracket})^{-1}$ in place of $(D^{\llbracket 2p \rrbracket})^{-1/2}\widetilde{B}^n(D^{\llbracket 2p \rrbracket})^{-1/2}$ in each $\mathcal{P}_{n,p}$, since

$$(D^{\llbracket 2p \rrbracket})^{-1/2}\widetilde{B}^0(D^{\llbracket 2p \rrbracket})^{-1/2} = (D^{\llbracket 2p \rrbracket})^{-1/2} I_s (D^{\llbracket 2p \rrbracket})^{-1/2} = (D^{\llbracket 2p \rrbracket})^{-1}.$$

Additionally, each F_i retains the vertex-weight contribution with the matrix $D^{\llbracket 2p+1 \rrbracket}$ in $\mathcal{R}_{d,0,p}$. We view it as having "virtual" degree $2p+1$. This precisely results in the GH-grid $G_{0,p}$.

However, even though $G_{0,p}$ still retains the cycles, since $(D^{\llbracket 2p \rrbracket})^{-1}$ is a diagonal matrix, each vertex F_i in a cycle is forced to receive the same vertex assignment value in the domain set $[s]$; all other vertex assignments contribute zero in the evaluation of $Z_{\{(D^{\llbracket 2p \rrbracket})^{-1}, A'\}, D^{\llbracket 2p+1 \rrbracket}}(G_{0,p})$. This can be easily seen by traversing the vertices $F_1, \ldots, F_d$ in a cycle. Hence we can view each cycle employing the virtual gadget $\mathcal{R}_{d,0,p}$ as a single vertex that contributes only a diagonal matrix of positive vertex weights $P^{\llbracket d \rrbracket} = (D^{\llbracket 2p+1 \rrbracket}(D^{\llbracket 2p \rrbracket})^{-1})^d$, where d is the vertex degree in G. Contracting all the cycles to a single vertex each, we arrive at the original graph G. Let $\mathfrak{P} = \{P^{\llbracket i \rrbracket}\}_{i=0}^{\infty}$, where we let $P^{\llbracket 0 \rrbracket} = I_s$, and for $i > 0$, we have $P_j^{\llbracket i \rrbracket} = w_j^i$ where $w_j = \sum_{k=1}^{m_j} \alpha_{jk}\mu_{jk}^{2p+1} / \sum_{k=1}^{m_j} \alpha_{jk}\mu_{jk}^{2p} > 0$ for $1 \leq j \leq s$. This shows that we now can interpolate the value $Z_{A',\mathfrak{P}}(G)$ using the values $Z_{A',\mathfrak{D}}(G_{n,p})$ in polynomial time in the size of G. The graph G is arbitrary but without isolated vertices here. We show next how to deal with the case when G has isolated vertices.

Given an arbitrary graph G, assume it has $h \geq 0$ isolated vertices. Let G^* denote the graph obtained from G by their removal. Then G^* is of size not larger than G and $h \leq |V(G)|$. Obviously, $Z_{A',\mathfrak{P}}(G) = (\sum_{i=1}^{s} P_i^{\llbracket 0 \rrbracket})^h Z_{A',\mathfrak{P}}(G^*) = s^h Z_{A',\mathfrak{P}}(G^*)$. Here the integer s is a constant, so the factor $s^h > 0$ can be easily computed in polynomial time. Thus, knowing the value $Z_{A',\mathfrak{P}}(G^*)$ we can compute the value $Z_{A',\mathfrak{P}}(G)$ in polynomial time. Further, since we only use the graphs $G_{n,p}, n \geq 1$ during the interpolation, each being simple of degree at most $2p+1$, combining it with the possible isolated vertex removal step, we conclude $\mathrm{EVAL}(A', \mathfrak{P}) \leq_{\mathrm{T}}^{\mathrm{P}} \mathrm{EVAL}_{\mathrm{simp}}^{(2p+1)}(A', \mathfrak{D})$.

Next, it is easy to see that for an arbitrary graph G

$$\begin{aligned}
Z_{A',\mathfrak{P}}(G) &= \sum_{\zeta: V(G) \to [s]} \prod_{z \in V(G)} P_{\zeta(z)}^{\llbracket \deg(z) \rrbracket} \prod_{(u,v) \in E(G)} A'_{\zeta(u),\zeta(v)} \\
&= \sum_{\zeta: V(G) \to [s]} \prod_{z \in V(G)} w_{\zeta(z)}^{\deg(z)} \prod_{(u,v) \in E(G)} A'_{\zeta(u),\zeta(v)} \\
&= \sum_{\zeta: V(G) \to [s]} \prod_{(u,v) \in E(G)} w_{\zeta(u)} w_{\zeta(v)} A'_{\zeta(u),\zeta(v)} \\
&= \sum_{\zeta: V(G) \to [s]} \prod_{(u,v) \in E(G)} C_{\zeta(u),\zeta(v)} = Z_C(G).
\end{aligned}$$

Here C is an $s \times s$ matrix with the entries $C_{ij} = A'_{ij} w_i w_j$ where $1 \leq i, j \leq s$. Clearly, C is a nonnegative symmetric matrix. In the above chain of equalities, we were able to *redistribute the weights w_i and w_j into the edge weights A'_{ij}* which resulted in the edge weights C_{ij}, so that precisely each edge (u, v) in G gets two factors $w_{\zeta(u)}$ and $w_{\zeta(v)}$ since the vertex weights at u and v were $w_{\zeta(u)}^{\deg(u)}$ and $w_{\zeta(v)}^{\deg(v)}$ respectively. (This is a crucial step in our proof.)

Because the underlying graph G is arbitrary, it follows that $\mathrm{EVAL}(A', \mathfrak{P}) \equiv_{\mathrm{T}}^{\mathrm{P}} \mathrm{EVAL}(C)$. Combining this with the previous EVAL-reductions and equivalences, we obtain

$$\mathrm{EVAL}(C) \equiv_{\mathrm{T}}^{\mathrm{P}} \mathrm{EVAL}(A', \mathfrak{P}) \leq_{\mathrm{T}}^{\mathrm{P}} \mathrm{EVAL}_{\mathrm{simp}}^{(2p+1)}(A', \mathfrak{D}) \equiv_{\mathrm{T}}^{\mathrm{P}} \mathrm{EVAL}_{\mathrm{simp}}^{(2p+1)}(A, D),$$

so that $\mathrm{EVAL}(C) \leq_{\mathrm{T}}^{\mathrm{P}} \mathrm{EVAL}_{\mathrm{simp}}^{(\Delta)}(A, D)$, by taking $\Delta = 2p + 1$.

Remembering that our goal is to prove the #P-hardness for the matrices A, D not satisfying the tractability conditions of Theorem 8, we finally use the assumption that A is not block-rank-1. Next, noticing that all $\mu_{ij} > 0$, by construction A' is not block-rank-1 either. Finally, because all $w_i > 0$ nor is C block-rank-1 implying that $\mathrm{EVAL}(C)$ is #P-hard by Theorem 3. Hence $\mathrm{EVAL}_{\mathrm{simp}}^{(2p+1)}(A, D)$ is also #P-hard. This completes the proof of the #P-hardness part of Theorem 8.

We remark that one important step in our interpolation proof happened at the stratification step before (3). In the proof we have the goal of redistributing vertex weights to edge weights; but this redistribution is sensitive to the degree of the vertices. This led us to define the auxiliary graph G' and the coefficients c_κ. Usually in an interpolation proof there are some coefficients that have a clear combinatorial meaning in terms of the original problem instance. Here these values c_κ do not have a clear combinatorial meaning in terms of $Z_{A',\mathfrak{D}}(G)$, rather they are defined in terms of an intermediate problem instance G', which is neither G nor the actual constructed graphs $G_{n,p}$. It is only in a "limiting" sense that a certain combination of these values c_κ allows us to compute $Z_{A',\mathfrak{D}}(G)$.

5 Tractability part

The tractability part of Theorem 8 follows easily from known results. For completeness we outline a proof here. Let A and D be $m \times m$ matrices, where A is nonnegative symmetric block-rank-1 and D is positive diagonal.

First, $Z_{A,D}(G)$ can be reduced to the connected components $G_1, \dots, G_t$ of G,

$$Z_{A,D}(G) = \prod_{i=1}^{t} Z_{A,D}(G_i),$$

so we may as well assume G is connected. We permute the rows and columns of A, D by the same permutation and then cross out zero rows and columns of A. This does not change $Z_{A,D}$. We may assume that $A = \mathrm{diag}(A_i)_{i=1}^{k}$ is block diagonal with nonzero blocks $A_1, \dots, A_k$, where each block A_i is either a symmetric matrix of rank 1 with no zero entries, or a symmetric bipartite matrix of the form $\left(\begin{smallmatrix} 0 & B \\ B^T & 0 \end{smallmatrix}\right)$ where B has rank 1 and no zero entries. Then we can write $D = \mathrm{diag}(D_i)_{i=1}^{k}$ where each D_i is positive diagonal of the corresponding size. As A is block diagonal and G is connected,

$$Z_{A,D}(G) = \sum_{i=1}^{k} Z_{A_i,D_i}(G).$$

So we may as well assume that A is one of these blocks. Also let $D = \mathrm{diag}(\alpha_i)_{i=1}^{m}$.

1) A is a symmetric matrix of rank 1 with no zero entries. We can write $A = x^T x$ for some positive row vector $x = (x_i)_{i=1}^{m}$. Then

$$Z_{A,D}(G) = \prod_{u \in V(G)} \sum_{i=1}^{m} \alpha_i x_i^{\deg(u)}.$$

2) $A = \left(\begin{smallmatrix} 0 & B \\ B^T & 0 \end{smallmatrix}\right)$, where B is $\ell \times (m-\ell)$ (for some $1 \le \ell < m$) has rank 1 and no zero entries. We can write $B = x^T y$ for some positive row vectors $x = (x_i)_{i=1}^{\ell}$ and $y = (y_j)_{j=\ell+1}^{m}$. Since G is connected, $Z_{A,D}(G) = 0$ unless G is bipartite. If G is bipartite with a vertex bipartization $V_1 \cup V_2$, then we only need to consider maps $\xi\colon G \to [m]$ such that either $\xi(V_1) \subseteq [\ell]$, $\xi(V_2) \subseteq [\ell+1:m]$ or $\xi(V_1) \subseteq [\ell+1:m]$, $\xi(V_2) \subseteq [\ell]$, with all other maps contribute zero to $Z_{A,D}(G)$. Then

$$\begin{aligned} Z_{A,D}(G) = &\left(\prod_{u \in V_1} \sum_{i=1}^{\ell} \alpha_i x_i^{\deg(u)}\right)\left(\prod_{v \in V_2} \sum_{j=\ell+1}^{m} \alpha_j y_j^{\deg(v)}\right) \\ &+ \left(\prod_{u \in V_1} \sum_{j=\ell+1}^{m} \alpha_j y_j^{\deg(u)}\right)\left(\prod_{v \in V_2} \sum_{i=1}^{\ell} \alpha_i x_i^{\deg(v)}\right). \end{aligned}$$

6 Two technical lemmas

We need two technical lemmas. The following lemma is from [10] (Lemma 3.6); for the convenience of readers we give a proof here.

▶ **Lemma 9.** *Let A and D be $m \times m$ matrices, where A is real symmetric with all columns nonzero and pairwise linearly independent, and D is positive diagonal. Then all columns of ADA are nonzero and pairwise linearly independent.*

Proof. The case $m = 1$ is trivial. Assume $m \ge 2$. Let $D = \operatorname{diag}(\alpha_i)_{i=1}^m$, and $\Pi = \operatorname{diag}(\sqrt{\alpha_i})_{i=1}^m$. Then $\Pi^2 = D$. We have $ADA = Q^T Q$, where $Q = \Pi A$. Let q_i denote the ith column of Q. Then Q has pairwise linearly independent columns. By the Cauchy-Schwartz inequality,

$$q_i^T q_j < \left((q_i^T q_i)(q_j^T q_j)\right)^{1/2},$$

whenever $i \ne j$. Then for any $1 \le i < j \le m$, the ith and jth columns of ADA contain a submatrix

$$\begin{bmatrix} q_i^T q_i & q_i^T q_j \\ q_i^T q_j & q_j^T q_j \end{bmatrix},$$

so they are linearly independent. ◀

The following is also adapted from [10] (Theorem 3.1).

▶ **Lemma 10.** *Let A and D be $m \times m$ matrices, where A is real symmetric with all columns nonzero and pairwise linearly independent, and D is positive diagonal. Then for all sufficiently large positive integers p, the matrix $B = (ADA)^{\odot p}$ corresponding to the edge gadget in Figure 5 is nondegenerate.*

Proof. If $m = 1$, then any $p \ge 1$ works. Let $m \ge 2$. Following the proof of Lemma 9, we have $q_i^T q_j < \sqrt{(q_i^T q_i)(q_j^T q_j)}$, for all $1 \le i < j \le m$. Let

$$\gamma = \max_{1 \le i < j \le m} \frac{q_i^T q_j}{\sqrt{(q_i^T q_i)(q_j^T q_j)}} < 1.$$

Let $A' = ADA = Q^T Q$ so $A'_{ij} = q_i^T q_j$. Then $A'_{ij} \le \gamma \sqrt{A'_{ii} A'_{jj}}$ for all $i \ne j$. Consider the determinant of A'. Each term of $\det(A')$ has the form

$$\pm \prod_{i=1}^{m} A'_{i\sigma(i)},$$

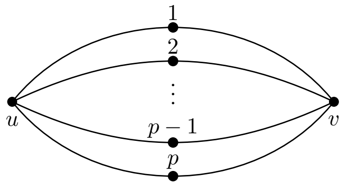


Figure 5 The edge gadget S_2T_pe, $e=(u,v)$ with the edge weight matrix $(ADA)^{\odot p}$.

where σ is a permutation of $[m]$. Denote $t(\sigma) = |\{i \mid \sigma(i) \neq i\}|$. Then

$$\prod_{i=1}^{m} A'_{i\sigma(i)} \leq \gamma^{t(\sigma)} \prod_{i=1}^{m} \sqrt{A'_{ii}} \prod_{i=1}^{m} \sqrt{A'_{\sigma(i)\sigma(i)}} = \gamma^{t(\sigma)} \prod_{i=1}^{m} A'_{ii}.$$

Consider the p-thickening of A' for $p \geq 1$. Each term of $\det\left((A')^{\odot p}\right)$ has the form $\pm \prod_{i=1}^{m} A'^{\,p}_{i\sigma(i)}$ for some permutation σ of $[m]$. Now

$$|\{\sigma \mid t(\sigma) = j\}| \leq \binom{m}{j} j! \leq m^j,$$

for $0 \leq j \leq m$. By separating out the identity permutation and all other terms, for $p \geq \lfloor \ln(2m)/\ln(1/\gamma) \rfloor + 1$, we have $2m\gamma^p < 1$, and

$$\begin{aligned} \det\left((A')^{\odot p}\right) &\geq \left(\prod_{i=1}^{m} A'_{ii}\right)^p - \left(\prod_{i=1}^{m} A'_{ii}\right)^p \sum_{j=1}^{m} m^j \gamma^{pj} \\ &\geq \left(\prod_{i=1}^{m} A'_{ii}\right)^p \left(1 - \frac{m\gamma^p}{1-m\gamma^p}\right) = \left(\prod_{i=1}^{m} A'_{ii}\right)^p \left(\frac{1-2m\gamma^p}{1-m\gamma^p}\right) > 0. \end{aligned}$$ ◀

7 Hardness for $Z_A(\cdot)$ on simple graphs for real symmetric A

There is a more direct approach to prove the #P-hardness part of the Bulatov-Grohe dichotomy (Theorem 3) for simple graphs. Although this method does not handle degree-boundedness, we can apply it more generally to the problem EVAL(A, D) when the matrix A is real symmetric and D is positive diagonal. In particular, we will prove the #P-hardness part of the dichotomy for counting GH by Goldberg et al. [12] (the problem EVAL(A) without vertex weights, where A is a real symmetric matrix) for simple graphs.

We first prove the following theorem.

▶ **Theorem 11.** *Let A and D be $m \times m$ matrices, where A is real symmetric and D is positive diagonal. Then* EVAL$(A, D) \leq_{\mathrm{T}}^{\mathrm{P}}$ EVAL$_{\mathrm{simp}}(A, D)$.

Proof. We may assume A is not identically 0, for otherwise the problem is trivial. Let $G = (V, E)$ be an input graph to the problem EVAL(A, D). For any $n \geq 1$, let $G_n = S_n^{(F)}(G)$ where $F \subseteq E$ is the subset consisting of the edges of G each of which is parallel to at least one other edge. In other words, we obtain G_n by replacing every parallel edge e by its n-stretching $S_n e$. We will refer to these as paths of length n in G_n. Note that $G_1 = G$. Moreover, for every $n \geq 2$, the graph G_n is simple and loopless, and has polynomial size in the size of G and n.

A path of length $n \geq 1$ has the edge weight matrix

$$M^{(n)} = \underbrace{ADA \ldots ADA}_{D \text{ appears } n-1 \,\geq\, 0 \text{ times}} = A(DA)^{n-1} = D^{-1/2}(D^{1/2}AD^{1/2})^n D^{-1/2}.$$

Here $D^{1/2}$ is a diagonal matrix with the positive square roots of the corresponding entries of D on the main diagonal, and $D^{-1/2}$ is its inverse.

Since A is real symmetric and D is positive diagonal, the matrix $\widetilde{A} = D^{1/2}AD^{1/2}$ is real symmetric. Then $\widetilde{A}$ is orthogonally diagonalizable over $\mathbb{R}$, i.e., there exist a real orthogonal matrix S and a real diagonal matrix $J = (\lambda_i)_{i=1}^m$ such that $\widetilde{A} = S^T JS$. If A has rank r, then $1 \leq r \leq m$, and we may assume that $\lambda_i \neq 0$ for $1 \leq i \leq r$ and $\lambda_i = 0$ for $i > r$.

We have $\widetilde{A}^n = S^T J^n S$, so the edge weight matrix for a path of length $n \geq 1$ can be written as

$$M^{(n)} = D^{-1/2}\widetilde{A}^n D^{-1/2} = D^{-1/2}S^T J^n S D^{-1/2}.$$

We can write $M_{ij}^{(n)} = \sum_{\ell=1}^r a_{ij\ell}\lambda_\ell^n$ by a formal expansion, for every $n \geq 1$ and some real $a_{ij\ell}$'s that are dependent on D and S, but independent of n and λ_ℓ, where $1 \leq i, j \leq m$ and $1 \leq \ell \leq r$. By the formal expansion of the symmetric matrix $M^{(n)}$ above, we have $a_{ij\ell} = a_{ji\ell}$. Let $t = |F|$, which is the number of edges in G subject to the stretching operator S_n to form G_n.

In the evaluation of the partition function $Z_{A,D}(G_n)$, we stratify the vertex assignments in G_n as follows. Denote by $\kappa = (k_{ij})_{1 \leq i \leq j \leq m}$ a nonnegative tuple with entries indexed by ordered pairs of nonnegative numbers that satisfy $\sum_{1 \leq i \leq j \leq m} k_{ij} = t$. Let $\mathcal{K}$ denote the set of all such possible tuples κ. In particular, $|\mathcal{K}| = \binom{t+m(m+1)/2-1}{m(m+1)/2-1}$. For a fixed m, this is a polynomial in t, and thus a polynomial in the size of G. Let c_κ be the sum over all assignments of all vertex and edge weight products in $Z_{A,D}(G_n)$, except the contributions by the paths of length n formed by stretching parallel edges in G, such that the endpoints of precisely k_{ij} constituent paths of length n receive the assignments (i, j) (in either order of the end points) for every $1 \leq i \leq j \leq m$. Technically we can call a vertex assignment on G consistent with κ (where $\kappa \in \mathcal{K}$), if it satisfies the stated property. Note that the contribution by each such path does not include the vertex weights of the two end points (but does include all vertex weights of the internal $n - 1$ vertices of the path). We can write

$$c_\kappa = \sum_{\substack{\xi\colon V(G)\to[m] \\ \xi \text{ is consistent with } \kappa}} \prod_{w \in V} D_{\xi(w)} \prod_{(u,v) \in E \setminus F} A_{\xi(u),\xi(v)}$$

for $\kappa \in \mathcal{K}$.

In particular, the values c_κ are independent of n. Thus for some polynomially many values c_κ, where $\kappa \in \mathcal{K}$, we have for all $n \geq 1$,

$$Z_{A,D}(G_n) = \sum_{\kappa \in \mathcal{K}} c_\kappa \prod_{1 \leq i \leq j \leq m} (M_{ij}^{(n)})^{k_{ij}} = \sum_{\kappa \in \mathcal{K}} c_\kappa \prod_{1 \leq i \leq j \leq m} (\sum_{\ell=1}^r a_{ij\ell}\lambda_\ell^n)^{k_{ij}}.$$

Expanding out the last sum and rearranging the terms, for some values $b_{i_1,\ldots,i_r}$ independent of n, we get

$$Z_{A,D}(G_n) = \sum_{\substack{i_1+\ldots+i_r=t \\ i_1,\ldots,i_r \geq 0}} b_{i_1,\ldots,i_r} (\prod_{\ell=1}^r \lambda_\ell^{i_\ell})^n \tag{4}$$

for all $n \geq 1$.

This can be viewed as a linear system with the unknowns $b_{i_1,\ldots,i_r}$ with the rows indexed by n. The number of unknowns is $\binom{t+r-1}{r-1}$ which is polynomial in the size of the input graph G, since $r \le m$ is a constant. The values $\prod_{\ell=1}^r \lambda_\ell^{i_\ell}$ can all be computed in polynomial time.

We show how to compute the value $Z_{A,D}(G) = \sum_{\substack{i_1+\ldots+i_r=t \\ i_1,\ldots,i_r \ge 0}} b_{i_1,\ldots,i_r} \prod_{\ell=1}^r \lambda_\ell^{i_\ell}$, from the values $Z_{A,D}(G_n)$ where $n \ge 2$ in polynomial time (recall that G_n is simple and loopless for $n \ge 2$). The coefficient matrix of the linear system (4) is a Vandermonde matrix. However, it might not be of full rank because the coefficients $\prod_{\ell=1}^r \lambda_\ell^{i_\ell}$ do not have to be pairwise distinct, and therefore it can have repeating columns. Nevertheless, when there are two repeating columns we replace the corresponding unknowns $b_{i_1,\ldots,i_r}$ and $b_{i'_1,\ldots,i'_r}$ with their sum as a new variable; we repeat this replacement procedure until there are no repeating columns. Since all $\lambda_\ell \ne 0$, for $1 \le \ell \le r$, after the replacement, we have a Vandermonde system of full rank. Therefore we can solve this modified linear system in polynomial time. This allows us to obtain the value $Z_{A,D}(G) = \sum_{\substack{i_1+\ldots+i_r=t \\ i_1,\ldots,i_r \ge 0}} b_{i_1,\ldots,i_r} \prod_{\ell=1}^r \lambda_\ell^{i_\ell}$, which also has exactly the same pattern of repeating multipliers $\prod_{\ell=1}^r \lambda_\ell^{i_\ell}$.

We have shown how to compute the value $Z_{A,D}(G)$ in polynomial time by querying the oracle $\mathrm{EVAL}(A, D)$ on polynomially many instances G_n, for $n \ge 2$. It follows that $\mathrm{EVAL}(A, D) \le_{\mathrm{T}}^{\mathrm{P}} \mathrm{EVAL}_{\mathrm{simp}}(A, D)$. ◀

We are ready to prove the #P-hardness part of the dichotomy by Goldberg et al. [12] (Theorem 1.1) for simple graphs. Let A be a real symmetric $m \times m$ matrix. Assuming that A does not satisfy the tractability conditions of the dichotomy theorem of Goldberg et al., the problem $\mathrm{EVAL}(A)$ is #P-hard. By Theorem 11 (with $D = I_m$), $\mathrm{EVAL}(A) \le_{\mathrm{T}}^{\mathrm{P}} \mathrm{EVAL}_{\mathrm{simp}}(A)$. It follows that $\mathrm{EVAL}_{\mathrm{simp}}(A)$ is #P-hard.

Hence the dichotomy theorem by Goldberg et al. can improve to apply to simple graphs.

▶ **Theorem 12.** *Let A be a real symmetric matrix. Then either $\mathrm{EVAL}(A)$ is in polynomial time or $\mathrm{EVAL}_{\mathrm{simp}}(A)$ is #P-hard (a fortiori, $\mathrm{EVAL}(A)$ is #P-hard).*

Moreover, there is a polynomial time algorithm that, given the matrix A, decides which case of the dichotomy it is.

▶ Remark 13. The interpolation argument in Theorem 11 works even if G is a multigraph possibly with multiple loops at any vertex in the following sense. In Definition 4, we treat the loops of G as edges. We think of them as mapped to the entries A_{ii} in the evaluation of the partition function $Z_{A,D}$. However, we need to slightly change the way we define the graphs G_n. In addition to n-stretching the parallel edges of G, we also need to n-stretch each loop of G (i.e., replacing a loop by a closed path of length n). Now F is the set of parallel edges *and* loops in G. This way each $G_n = S_n^{(F)}(G)$ for $n \ge 2$ is simple and loopless. The rest of the proof goes through. In other words, the statement of Theorem 11 extends to a reduction from the $\mathrm{EVAL}(A, D)$ problem that allows input G to have multiloops, to the standard problem $\mathrm{EVAL}_{\mathrm{simp}}(A, D)$ not allowing loops.

References

1 Alexander I. Barvinok. *Combinatorics and Complexity of Partition Functions*, volume 30 of *Algorithms and combinatorics*. Springer, 2017.

2 Alexander I. Barvinok and Pablo Soberón. Computing the partition function for graph homomorphisms. *Combinatorica*, 37(4):633–650, 2017.

3 Rodney J. Baxter. The six and eight-vertex models revisited. *Journal of Statistical Physics*, 116(1):43–66, 2004.

4 Andrei Bulatov and Martin Grohe. The complexity of partition functions. *Theoretical Computer Science*, 348(2-3):148–186, 2005.

5 Jin-Yi Cai and Xi Chen. *Complexity Dichotomies for Counting Problems*, volume 1: Boolean Domain. Cambridge University Press, 2017. doi:10.1017/9781107477063.

6 Jin-Yi Cai and Xi Chen. A decidable dichotomy theorem on directed graph homomorphisms with non-negative weights. *Computational Complexity*, 28(3):345–408, 2019.

7 Jin-Yi Cai, Xi Chen, and Pinyan Lu. Graph homomorphisms with complex values: A dichotomy theorem. *SIAM Journal on Computing*, 42(3):924–1029, 2013.

8 Martin E. Dyer, Alan M. Frieze, and Mark Jerrum. On counting independent sets in sparse graphs. *SIAM Journal on Computing*, 31(5):1527–1541, 2002.

9 Martin E. Dyer, Leslie A. Goldberg, and Mike Paterson. On counting homomorphisms to directed acyclic graphs. *Journal of the ACM*, 54(6):27, 2007.

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

11 Martin E. Dyer and Catherine S. Greenhill. Corrigendum: The complexity of counting graph homomorphisms. *Random Structures and Algorithms*, 25(3):346–352, 2004.

12 Leslie A. Goldberg, Martin Grohe, Mark Jerrum, and Marc Thurley. A complexity dichotomy for partition functions with mixed signs. *SIAM Journal on Computing*, 39(7):3336–3402, 2010.

13 Martin Grohe and Marc Thurley. Counting homomorphisms and partition functions. In *Model Theoretic Methods in Finite Combinatorics*, volume 558 of *Contemporary Mathematics*, pages 243–292. American Mathematical Society, 2011.

14 Pavol Hell and Jaroslav Nešetřil. *Graphs and homomorphisms*, volume 28 of *Oxford lecture series in mathematics and its applications*. Oxford University Press, 2004.

15 Tyler Helmuth, Will Perkins, and Guus Regts. Algorithmic Pirogov-Sinai theory. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 1009–1020, 2019.

16 Hendrik W. Lenstra Jr. Algorithms in algebraic number theory. *Bulletin of the American Mathematical Society*, 26(2):211–244, 1992.

17 Liang Li, Pinyan Lu, and Yitong Yin. Correlation decay up to uniqueness in spin systems. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 67–84, 2013.

18 László Lovász. Operations with structures. *Acta Mathematica Hungarica*, 18(3-4):321–328, 1967.

19 Han Peters and Guus Regts. Location of zeros for the partition function of the Ising model on bounded degree graphs. *arXiv preprint*, 2018. arXiv:1810.01699.

20 Alistair Sinclair, Piyush Srivastava, and Marc Thurley. Approximation algorithms for two-state anti-ferromagnetic spin systems on bounded degree graphs. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 941–953, 2012.

21 Allan Sly. Computational transition at the uniqueness threshold. In *Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 287–296, 2010.

22 Marc Thurley. *The Complexity of Partition Functions*. PhD thesis, Humboldt Universität zu Berlin, 2009.

23 Marc Thurley. The complexity of partition functions on Hermitian matrices. *arXiv preprint*, 2010. arXiv:1004.0992.

24 Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.

25 Dror Weitz. Counting independent sets up to the tree threshold. In *Proceedings of the 38th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 140–149, 2006.

26 Mingji Xia. Holographic reduction: A domain changed application and its partial converse theorems. *International Journal of Software and Informatics*, 5(4):567–577, 2011.

Sublinear-Space Lexicographic Depth-First Search for Bounded Treewidth Graphs and Planar Graphs[†]

Taisuke Izumi[1]
Nagoya Institute of Technology, Japan
t-izumi@nitech.ac.jp

Yota Otachi
Nagoya University, Japan
otachi@nagoya-u.jp

Abstract

The lexicographic depth-first search (Lex-DFS) is one of the first basic graph problems studied in the context of space-efficient algorithms. It is shown independently by Asano et al. [ISAAC 2014] and Elmasry et al. [STACS 2015] that Lex-DFS admits polynomial-time algorithms that run with $O(n)$-bit working memory, where n is the number of vertices in the graph. Lex-DFS is known to be P-complete under logspace reduction, and giving or ruling out polynomial-time sublinear-space algorithms for Lex-DFS on general graphs is quite challenging. In this paper, we study Lex-DFS on graphs of bounded treewidth. We first show that given a tree decomposition of width $O(n^{1-\epsilon})$ with $\epsilon > 0$, Lex-DFS can be solved in sublinear space. We then complement this result by presenting a space-efficient algorithm that can compute, for $w \leq \sqrt{n}$, a tree decomposition of width $O(w\sqrt{n}\log n)$ or correctly decide that the graph has a treewidth more than w. This algorithm itself would be of independent interest as the first space-efficient algorithm for computing a tree decomposition of moderate (small but non-constant) width. By combining these results, we can show in particular that graphs of treewidth $O(n^{1/2-\epsilon})$ for some $\epsilon > 0$ admits a polynomial-time sublinear-space algorithm for Lex-DFS. We can also show that planar graphs admit a polynomial-time algorithm with $O(n^{1/2+\epsilon})$-bit working memory for Lex-DFS.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness; Theory of computation → Complexity classes; Theory of computation → Graph algorithms analysis; Mathematics of computing → Graph algorithms

Keywords and phrases depth-first search, space complexity, treewidth

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.67

Category Track A: Algorithms, Complexity and Games

Funding *Taisuke Izumi*: JSPS KAKENHI Grant Number 19K11824
Yota Otachi: JSPS KAKENHI Grant Numbers JP18H04091, JP18K11168, JP18K11169

1 Introduction

1.1 Background and Motivation

Depth-First Search (DFS) is one of the most fundamental and elementary graph search algorithms with a huge number of applications. Lexicographic DFS (Lex-DFS) is a popular variant of DFS, which requires the search head always moves to the *first* undiscovered neighbor in the adjacency list of the current vertex (as long as it exists). Recently, the space-efficient implementation of fundamental graph algorithms, including (Lex-)DFS, receives much attention [3, 6, 15, 16, 23, 26, 30]. These researches are roughly motivated by the

[1] Corresponding author.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 67; pp. 67:1–67:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

two aspects as follows: First, the space matter is serious in the big-data (i.e., too large inputs) and/or IoT (i.e., too small computational devices) era. Second, the challenge of proving space-complexity lower bounds for problems within class P still lies at the core of computational complexity theory. One of the ultimate goals on this research direction is to prove or disprove the seminal P $\neq$ L conjecture [2]. We focus on the space complexity of Lex-DFS, particularly the algorithms using memory below the trivial $O(n \log n)$-bit bound. This can be motivated from both sides but much leans against the second one. The sublinear-space Lex-DFS problem is formulated as the one of outputting the Lex-DFS ordering of all vertices in streaming way, and its space complexity is measured by the required working-memory size, as the classical read-only model [25].

To argue the complexity of sublinear-space algorithms, the notion of P-completeness under logspace reduction plays an important role, which is analogous to NP-completeness in P $\neq$ NP conjecture. Reif [40] shows that Lex-DFS is P-complete under logspace reduction. It implies that no Lex-DFS algorithm only using $O(\log n)$-bit working memory exists unless P = L holds. A counterpart from the upper-bound side is recently obtained by a few literatures [3, 6, 23, 26]. They focus on the implementation of (Lex-)DFS achieving both polynomial time and $o(n \log n)$-bit space complexity, where n is the number of vertices in the input graph. Initiated by Asano et al. [3] and Elmasry et al. [23], a series of papers by several authors explore the time-space tradeoffs of (Lex-)DFS in the area of $o(n \log n)$-bit space-complexity. The state-of-the-art bounds are threefolds, $O(m \log^* n)$ running time and $O(n)$-bit working memory, $O(m + n)$ running time and $O(n \log \log(4 + m/n))$-bit working memory, and $O(m + n)$ running time and $O(n \log^{(k)} n)$-bit working memory for any integer $k > 1$, which are all proposed by Hagerup [26]. Looking at hidden coefficients, the smallest-space algorithm is the one by Asano et al. [3], which achieves a polynomial running time (with a large exponent) using the working memory of $n + o(n)$ bits. No algorithm so far attains cn-bit space complexity for $c < 1$, and obtaining such an algorithm is commonly recognized as a very challenging problem. This open problem is also supported from yet another context of computational complexity theory. Lex-DFS on directed graphs is at least as hard as the directed s-t reachability problem, which is known to be NL-complete and thus its space-efficient (ideally, logspace) solution is closely related to the seminal L = NL? problem. In fact, any directed Lex-DFS algorithm achieving $O(n^{1-\epsilon})$-bit space complexity for any small constant $\epsilon > 0$ would be a breakthrough result on this research line.

1.2 Our Result

In the context of directed s-t reachability, there are many attempts of attaining $O(n^{1-\epsilon})$-bit space complexity for a specific graph class such as planar or bounded treewidth graphs [1, 4, 5, 12, 13, 27, 29], which naturally yields the interest to the feasibility of sublinear-space Lex-DFS for those classes. It should be noted that Lex-DFS is P-complete even for planar graphs [2], and thus its difficulty under log-space solvability is the same as the general case. One of the main results presented in this paper is a sublinear-space Lex-DFS algorithm for bounded treewidth graphs. The first theorem is stated as follows.

▶ **Theorem 1.** *Let $0 < \epsilon < 1$ be an arbitrary positive constant, G be any n-vertex directed graph of treewidth w, and $(\mathcal{T}, \{B_x\}_{x \in V_\mathcal{T}})$ be its tree decomposition of width $w' \geq w$, where $\mathcal{T} = (V_\mathcal{T}, E_\mathcal{T})$ is a tree and each node x in $\mathcal{T}$ is associated with a subset B_x of vertices in G. Assume a polynomial-time algorithm Alg enumerating the vertices in B_x for all $x \in V_\mathcal{T}$ and the edges in $E_\mathcal{T}$. Then there exists a Lex-DFS algorithm of running time $O(n^{O(1/\epsilon)})$ using $O(\epsilon^{-1} w' n^\epsilon \log n)$-bit memory (except for the space used by Alg).*

[2] L is the class of problems decidable with $O(\log n)$-bit working space (and thus in poly(n) time).

It should be noted that Lex-DFS does not necessarily lie on the seminal framework known as Courcelle's theorem [19] and its logspace version [22] because the output depends on the order of vertices in the adjacency list of the input graph.

To figure out a "purely" sublinear-space Lex-DFS algorithm, it is necessary to implement a tree decomposition algorithm using only sublinear space. Elberfeld et al. [22] presents a logspace tree-decomposition algorithm for $w = O(1)$, but no sublinear-space algorithm covering the case of $w = \omega(1)$ has been known so far. Our second theorem provides a sublinear-space solution for tree decomposition:

▶ **Theorem 2.** *There exists an algorithm that, given a graph G of n vertices and $w \leq n^{1/2}$, either provides a tree decomposition of width $O(wn^{1/2} \log n)$ or correctly decides that the treewidth of G is more than w. This algorithm runs in a polynomial time and uses $O(wn^{1/2} \log^2 n)$-bit space.*

To the best of our knowledge, this is the first non-trivial tree-decomposition algorithm attaining both sublinear-space and polynomial time for $w = \omega(1)$. It is also worth noting that planar graphs admit a space-efficient algorithm of finding a balanced separator of size $O(\sqrt{n})$ using $\tilde{O}(\sqrt{n})$-bit space [27], which can be translated into a small-space tree decomposition algorithm of width $O(\sqrt{n} \log n)$. Putting all the results above together, we obtain the following consequence:

▶ **Corollary 3.** *Let $\epsilon > 0$ be any positive constant. There exist the polynomial-time Lex-DFS algorithms respectively satisfying the following properties*[3]*.*

- *Using $O(n^{\epsilon})$-bit working memory for directed graphs of treewidth $w = O(1)$.*
- *Using $O(wn^{1/2+\epsilon})$-bit working memory for directed graphs of treewidth $w = O(n^{1/2})$.*
- *Using $O(n^{1/2+\epsilon})$-bit working memory for directed planar graphs.*

1.3 Related Work

As stated above, the space complexity of Lex-DFS is one of the classical problems in the context of logspace computability. Following the P-completeness result by Reif [40], Anderson and Mayr [2] also shows a weaker variant of Lex-DFS (lexicographically first maximal path) is also P-complete even for planar graphs. The main interest of those earlier results is closely related to the s-t reachability problem. It is known that the space complexity of undirected s-t connectivity drops into $O(\log n)$ bits for any input graphs, which is proved in Reingold's celebrating paper [41]. The best space upper bound of all polynomial-time directed s-t reachability algorithms is $O(n/2^{\Omega(\sqrt{\log n})})$ bits by Barnes et al. [9]. Its near optimality within a (naturally) restricted class of algorithms, called NNJAG [39], is also shown by Edmonds et al. [21].

More recently, the space-complexity matter of the directed s-t reachability problem for specific graph classes receives much attention, and a number of papers try to expand the graph class allowing sublinear-space directed reachability algorithms. Grid graphs [1, 5, 28], planar graphs [1, 4, 13, 27], bounded-genus graphs [12], and bounded-treewidth graphs [29] have been considered so far. Notice that the algorithms presented in [12] and [29] for bounded-genus graphs and bounded-treewidth graphs respectively require the surface embedding and the tree decomposition of the input graph (as Theorem 1), but it is not addressed how to compute them using sublinear space. Our tree-decomposition algorithm (by Theorem 2) yields the first

[3] Since one can choose an arbitrary $\epsilon > 0$, polylog(n) factors are absorbed in the part of n^{ϵ} in the statements of this corollary.

purely sublinear-space directed reachability algorithm for graphs of treewidth $w = O(n^{1/2-\epsilon})$. Very recently, an $O(1)$-approximate tree decomposition algorithm using $O(wn)$-bit space is presented [31], which attains a non-trivial space complexity for $w = o(\log n)$.

Despite relatively rich literatures on directed s-t reachability, the space complexity of Lex-DFS receives less attention until recently. After the two concurrent results by Asano et al. [3] and Elmasry et al. [23], a few follow-up papers propose fundamental graph algorithms using only $o(n \log n)$-bit working memory, which cover Lex-BFS [6, 23], single-source shortest path [23], biconnected component decomposition [15, 30], s-t numbering [15], maximum cardinality search [16], and so on. It is also becoming active to consider sublinear-space algorithms for fundamental non-graph problems [7, 20, 34, 36, 42].

On the side of computational models, the read-only model is one of the classical models to consider the complexity of working memory. An earlier topic in this model is the time-space tradeoffs for sorting and/or selection [10, 17, 18, 25, 37, 38]. Recently, more unconventional models are also investigated; Stream model [33], restore model (algorithms can manipulate input memory but after the computation the initial input data must be recovered) [14, 32], and catalytic model (algorithms can use a large memory which are already used for other purpose, and after the computation the memory state must be recovered to the initial one) [11]. Some of the results in those models allow a Lex-DFS algorithm using only a small (exclusive) working memory, but they are incomparable to the ones in the standard read-only model. Barba et al. [8] provides a general scheme of realizing stack machines using only a small memory space. While the dominant part of the memory usage in Lex-DFS algorithms is the storage for a stack, the technique by Barba et al. only applies to the algorithms whose access pattern to stacks are non-adaptive. Thus that scheme does not work for saving the space complexity of Lex-DFS algorithms.

1.4 Organization of Paper

In Section 2, we introduce the model, notations, and several auxiliary matters for our Lex-DFS problem. Sections 3 and 4 respectively show the proofs of Theorem 1 and 2. Finally the paper is concluded in Section 5.

2 Preliminaries

2.1 Model and Notation

As stated in the introduction, this paper adopts the *read-only model* [25], where the space complexity of an algorithm is measured by the number of bits used for the working space, excluding the memory for inputs and outputs. The input memory is read-only, and the output memory is write-only. The memory-access model follows the standard RAM of $(\log n)$-bit words. Let G be any directed input graph of n vertices and m edges, which is stored in the form of the adjacency list A_G. For any graph G, we denote the sets of the vertices and edges in G by V_G and E_G respectively. We assume $V_G = [0, n-1]$, that is, each vertex in V_G is uniquely identified by an integer in $[0, n-1]$. Letting $N_G(v) \subseteq V_G$ be the set of v's neighbors in G, we refer to the neighbor list of $v \in V_G$ as $A_{G,v}$ and denote the i-th vertex in $A_{G,v}$ by $A_{G,v}[i]$ (index i starts from value zero). We use notation $u <_v u'$ for $u, u' \in N_G(v)$ if u precedes u' in $A_{G,v}$. We define the inverse mapping of $A_{G,v}$ as $A^{-1}_{G,v}$, that is, for any neighbor u of v, $A^{-1}_{G,v}[u]$ returns the position of u in $A_{G,v}$. When we consider a subgraph $H \subseteq G$, the adjacency list of H is inherited from that of G. More precisely, when we delete an edge $(u, v) \in E_G$, the adjacency list $A_{H,u}$ after deletion is defined as the one such that

$A_{H,u}[i] = A_{G,u}[i]$ for any $i < A^{-1}_{G,u}[v]$ and $A_{H,u}[i-1] = A_{G,u}[i]$ for any $i > A^{-1}_{G,u}[v]$. Since any subgraph is obtained by iterative deletion of edges (and removal of isolated vertices), the specification of the adjacency list after deletion of one edge also specifies the adjacency list A_H for any subgraph H.

Letting X be a set of vertices or edges, we denote by $G - X$ the graph obtained from G by removing all the elements in X (if X is a vertex set, all the edges incident to a vertex in X are also deleted). The subgraph of G induced by X is denoted by $G[X]$. If there is a polynomial-time algorithm *Alg* enumerating all the elements in X, it naturally provides an access to the adjacency list A_H for $H = G[X]$ or $H = G - X$ without explicitly constructing it in the working memory (i.e., *Alg* works as a filter extracting the elements in $G[X]$ or $G - X$ from the adjacency list of A_G). Then we call *Alg* an *emulator* of H. The overhead of accessing to A_H is a polynomial time (depending on the running time of *Alg*) per one access. Thus we can run any polynomial-time algorithm taking H as its input within a polynomial time. In the following argument, we often omit the subscript G of the notations defined above if there is no ambiguity.

2.2 Lex-DFS

In what follows, we fix a starting vertex s of Lex-DFS tasks. A Lex-DFS algorithm is presented in Algorithm 1. In the algorithm, we introduce the notion of time. At each time, the search head v_{cur} moves to a neighbor decided by the algorithm. The search starts at time $t = 1$ and finishes at time $t = 2n$. We define h_t as the vertex pointed by the search head at the beginning of time t in the Lex-DFS on G. For vertex $v \in V$, the *discovery time* $d(v)$ of v is defined as the first time when the search head moves to v. Similarly, we define the *leaving time* $l(v)$ of v as the last time when the search head moves from v. The discovery time of s is defined as zero. Following the terminology in [3], a vertex v is called a *gray* vertex at t if $d(v) \leq t \leq l(v)$ holds. The (path) subgraph corresponding to the sequence of all gray vertices at time t sorted by their discovery times is called the *gray path* at t, which is denoted by S_t[4]. For any vertex $u \in V_{S_t}$, we also denote by $p_t(u)$ and $s_t(u)$ the (immediate) predecessor and successor of u in S_t, and by $S_t(u)$ the prefix of S_t terminating at u. We define $p_t(s) = \perp$ and $s_t(h_t) = \perp$.

In Algorithm 1, we encapsulate the space-consuming parts of the algorithm by two abstract procedures called Pivot(t) and Parent(t). The procedure Pivot(t) tries to find the first undiscovered neighbor of h_t with respect to the order $<_{h_t}$. If there is no undiscovered neighbor, it returns -1. The procedure Parent(t) returns the predecessor $p_t(h_t)$ in the current gray path. It returns -1 if $h_t = s$ holds. It is obvious that Lex-DFS is implemented with the working memory of $f(n) + O(\log n)$ bits if both of Pivot(t) and Parent(t) are implemented with $f(n)$ bits.

2.3 Tree Decomposition and Balanced Separator

We first present the definition of tree decomposition.

▶ **Definition 4.** *A tree decomposition of an undirected graph G is a pair $(\mathcal{T}, \{B_x\}_{x \in V_{\mathcal{T}}})$, where $\mathcal{T}$ is a tree and each node $x \in V_{\mathcal{T}}$ is associated with a subset B_x of vertices in V_G (called the bag x) satisfying the following conditions:*

- *Any edge in G is covered by at least one bag, i.e., $\forall (u, v) \in E_G : \exists x : u, v \in B_x$.*
- *Letting $\mathcal{T}(u)$ be the subgraph of $\mathcal{T}$ induced by the bags containing u, for any $u \in V_G$, $\mathcal{T}(u)$ is non-empty and connected.*

[4] Intuitively, the gray path S_t is the path from s to h_t in the Lex-DFS tree of G.

Algorithm 1 Lex-DFS Algorithm for graph G starting from s.

```
$v_{cur} \leftarrow s$; $t \leftarrow 1$                          ▷ $v_{cur}$ is the search head
while true do
    $v \leftarrow$ PIVOT($t$)                      ▷ Find the first undiscovered neighbor of $v_{cur}$ in $A_{G,s}$.
    if $v = -1$ then                               ▷ All neighbors have been already visited
        $v \leftarrow$ PARENT($t$)                 ▷ Find the parent in the gray path
        if $v = -1$ then halt                      ▷ All vertices are visited
    else
        Output $v$                                 ▷ Discovery of a new vertex
    $v_{cur} \leftarrow v$
    $t \leftarrow t + 1$
```

Note that tree decomposition is defined for undirected graphs. When we consider the tree decomposition of directed graphs G, we naturally adapt the same definition to the undirected graph obtained from G by omitting the orientation of all edges[5]. Each bag is identified by an integer value in $[0, |V_{\mathcal{T}}| - 1]$. Throughout this paper, we assume that any decomposition tree $\mathcal{T}$ is rooted, and that a tree decomposition is encoded as the sequence $(B_1, q(1)), (B_2, q(2)), \ldots, (B_x, q(x)), \ldots, (B_{|V_{\mathcal{T}}|-1}, q(|V_{\mathcal{T}}| - 1))$, where $q(x)$ is the ID of the parent of x in $\mathcal{T}$. The parent of the root bag is defined as -1. A sublinear-space tree decomposition algorithm must output this sequence in a streaming way. The *width* w of a tree decomposition $(\mathcal{T}, \{B_x\}_{x \in V_{\mathcal{T}}})$ is defined as the maximum bag size minus one, i.e., $w = (\max_{x \in V_{\mathcal{T}}} |B_x|) - 1$. The *treewidth* of a graph G is the minimum width over all tree decompositions of G. It is a fundamental property that the removal of the vertices in any (non-leaf) bag B_x from G splits G into several connected components, each of which corresponds to a subtree of $\mathcal{T}$ obtained by the removal of x from $\mathcal{T}$.

Tree decomposition is closely related to the notion of balanced vertex separators. Let $G = (V, E)$ be any directed graph and $\mu \colon V_G \to \mathbb{N}$ be any vertex-weight function. We define $\mu(X) = \sum_{v \in X} \mu(v)$ for any $X \subseteq V_G$. A vertex subset $U \subseteq V$ is called a *weighted α-balanced separator of G with respect to μ* if any weakly-connected component C in $G - U$ satisfies $\mu(V_C)/\mu(V_G) \leq \alpha$. If μ is a constant function, it is simply called an *α-balanced separator of G*. Throughout this paper, we often consider a subgraph obtained by recursively removing separators. Let $cc(H, U)$ be the set of connected components in $H - U$ for any graph H and its vertex subset $U \subseteq V_H$, and $vcc(H, U) = \{V_C \mid C \in cc(H, U)\}$. The following lemma holds.

▶ **Lemma 5.** *Let $H_0, H_1, H_2, \ldots, H_{k-1}$ and $U_0, U_1, \ldots, U_{k-1}$ be respectively the sequences of subgraphs of G and their vertex subsets such that $H_i \in cc(H_{i-1}, U_{i-1})$ holds. Assuming k algorithms respectively enumerating the vertices in U_i for each $i \in [0, k-1]$, there exists a logspace algorithm of enumerating all the vertex subsets in $vcc(H_{k-1}, U_{k-1})$ using them as black-box subroutines.*

Proof. Since the straightforward recursive emulation of H_{k-1} takes the overhead exponential of k, such an approach applies only to the case of $k = O(1)$. Instead of emulating $H_{i-1} - U_{i-1}$ recursively, we use the emulation of $G - U$ for $U = \bigcup_{0 \leq i \leq k-1} U_i$. Since we assume the algorithms of enumerating U_i for all $i \in [0, k-1]$, this emulation works with a polynomial-time overhead independent of k. We have $cc(H_{k-1}, U_{k-1}) \subseteq cc(G, U)$ obviously. While

[5] Precisely, if two directed edges (u, v) and (v, u) exist, omitting their orientation causes two multiedges between u and v. Then those edges are merged into a single undirected edge.

$cc(G, U)$ might contain a connected component not in $cc(H_{k-1}, U_{k-1})$, one can identify $C \in cc(H_{k-1}, U_{k-1})$ by checking if C has an outgoing edge to a neighbor in U_{k-1} because any component $C \notin cc(H_{k-1}, U_{k-1})$ is separated from H_{k-1} by $U_0, U_1, \ldots$ or U_{k-2}. Thus, letting ∂U_{k-1} be the set of vertices in $G - \bigcup_{0 \leq i \leq k-1} U_i$ adjacent to a node in U_{k-1}, it suffices to obtain an algorithm enumerating all the components in $cc(G, U)$ intersecting ∂U_{k-1}. It is realized by the following procedure.

1. Let $c = 1$, and $v_0, v_1, \ldots, v_{l-1}$ be the sequence of the vertices in ∂U_{k-1} sorted by their IDs, which can be enumerated using logarithmic space and the algorithms for $U_0, U_1, \ldots, U_{k-1}$.
2. For each v_i, check if a vertex v_j satisfying $j < i$ is reachable to v_i in $G - U$. If such a vertex exists, repeat this step for v_{i+1} (unless $i = l - 1$). Otherwise, go to step 3.
3. Enumerate all the vertices reachable from v_i as the vertices in the c-th component in $cc(H_{k-1}, U_{k-1})$. After the enumeration, increment c by one, and go back to step 2 for v_{i+1}.

The procedure above is implemented with $O(\log n)$-bit space by utilizing the logspace undirected s-t connectivity algorithm [41] (since $cc(H_{k-1}, U_{k-1})$ is a set of weakly-connected components, undirected s-t connectivity suffices). Letting v_j be the vertex with the minimum ID in $V_C \cap \partial U_{K-1}$ for a component $C \in cc(H_{k-1}, U_{k-1})$, C is necessarily enumerated when the procedure above processes v_j. In addition, it is never enumerated twice because any other vertex in $V_C \cap \partial U_{k-1}$ has an ID larger than v_j and is reachable to v_j in $G - U$. ◀

The lemma above implies that one can associate an unique integer ID with each connected component in $H_{k-1} - U_{k-1}$, and can emulate with a polynomial-time overhead the connected component in $H_{k-1} - U_{k-1}$ specified by a given ID. We also have the lemma below.

▶ **Lemma 6.** *Let $G = (V, E)$ be any graph of treewidth w, and $0 < \delta < 1$ be an arbitrary positive constant. Assume an algorithm outputting a tree decomposition of width at most w' for G. Then, there exists an algorithm outputting an $O(n^{-\delta})$-balanced vertex separator U of size $O(w'n^{\delta})$ for G, which uses only $O(w'n^{\delta} \log n)$-bit space (except for the space used by the tree-decomposition algorithm).*

Proof. Let $(\mathcal{T}, \{B_x\}_{x \in V_{\mathcal{T}}})$ be the (rooted) tree decomposition constructed by the algorithm. For any subgraph $\mathcal{T}' \subseteq \mathcal{T}$ and a subset $X \subseteq V_G$, we define $\mathrm{vol}(\mathcal{T}', X) = |\bigcup_{y \in V_{\mathcal{T}'}} B_y \setminus X|$. We also define $\mathcal{T}(x)$ as the subtree of $\mathcal{T}$ rooted by $x \in V_{\mathcal{T}}$.

The proof is constructive. The algorithm finds the $O(n^{\delta})$ bags whose removal splits $\mathcal{T}$ into a small subtrees $\mathcal{T}_0, \mathcal{T}_1, \ldots, \mathcal{T}_{M-1}$ satisfying $\mathrm{vol}(\mathcal{T}_i, U) \leq n^{1-\delta}$ for any $i \in [1, M]$. The algorithm manages two sets U' and U. The set U' stores the set of bag IDs constituting the vertex subset U, i.e., $U = \bigcup_{x \in U'} B_x$. The construction of U is done by iteratively adding a vertex in $V_{\mathcal{T}}$ to U'. Let x_i be the vertex added to U' at the i-th iteration, and U_i and U'_i be respectively the contents of U and U' when $|U'| = i$ holds. We further define $\mathcal{T}^R_i$ as the connected component of $\mathcal{T} - U'_i$ containing the root. The algorithm chooses as x_i the deepest vertex in $\mathcal{T}^R_{i-1}$ such that $\mathrm{vol}(\mathcal{T}^R_{i-1}(x_i), U_{i-1}) \geq n^{1-\delta}$ holds. The iteration terminates when $\mathrm{vol}(\mathcal{T}^R_i, U_i)$ becomes smaller than $n^{1-\delta}$. Since one iteration decreases $\mathrm{vol}(\mathcal{T}^R_*, U_*)$ by at least $n^{1-\delta}$, the algorithm terminates within n^{δ} iterations. In addition, for any children y of x_i, we have $\mathrm{vol}(\mathcal{T}^R_{i-1}(y), U_{i-1}) < n^{1-\delta}$ because x_i is the deepest vertex. It implies that any connected component in $G - U$ contains at most $n^{1-\delta}$ vertices.

The remaining issue is the time and space complexities for implementing the algorithm. Since the tree decomposition algorithm provides the whole topological information on $\mathcal{T}$, one can use it as the adjacency list of $\mathcal{T}$ incurring a polynomial-time overhead. Since U' is stored in the working memory, it is possible to emulate $\mathcal{T}^R_i$ for any i. The computation of $\mathrm{vol}(\mathcal{T}_i^R(x), U_i)$ for any $x \in V_{\mathcal{T}_i^R}$ can be done in a polynomial time using the membership test of $v \in B_y$ for all pairs of $v \in V_G \setminus U_i$ and $y \in V_{\mathcal{T}_i^R(x)} \setminus \{x\}$. Consequently, the proposed algorithm can be implemented with the storage cost for U and U'. Since each bag contains at most w' vertices, the space complexity is $O(w' n^\delta \log n)$ bits. ◀

3 Small-Space Lex-DFS Algorithm for Graphs of Bounded Treewidth

3.1 Reduction to (Approximate) Gray-Path Membership

The algorithms shown in [3] reduces the procedures of $\textsc{Pivot}(t)$ and $\textsc{Parent}(t)$ to a single abstract task called $\textsc{IsGray}(u, v)$, which tests if v belongs to the prefix of the gray path by u (i.e., tests if $v \in V_{S_{d(u)}}$ holds or not). The following lemma is proved in [3].

▶ **Lemma 7** (Asano et al. [3]). *Assume that there exists a polynomial-time algorithm $\textsc{IsGray}(u, v)$ which is executable at any time $d(u) \le t \le l(u)$ and determines if $v \in V_{S_{d(u)}}$ holds or not. Letting $f(n)$ be the space complexity of that algorithm and $g(n)$ be the space-complexity of solving the directed s-t reachability problem, we have two polynomial-time algorithms which respectively implement $\textsc{Pivot}(t)$ and $\textsc{Parent}(t)$ using the memory of $f(n) + g(n)$ bits.*

We slightly extend this lemma by introducing a new primitive called $\textsc{AIsGray}(u, v)$ (approximate testing of gray vertex), which is a one-sided-error version of $\textsc{IsGray}(u, v)$ satisfying the following two conditions:

1. If $v \in V_{S_{d(u)}}$ holds, $\textsc{AIsGray}(u, v)$ always returns true.
2. If $d(v) > d(u)$ holds, $\textsc{AIsGray}(u, v)$ always returns false.

The following lemma implies that we can replace $\textsc{IsGray}(u, v)$ in Lemma 7 by $\textsc{AIsGray}(u, v)$.

▶ **Lemma 8.** *Assume a polynomial-time algorithm $\textsc{AIsGray}(u, v)$ executable at any time $d(u) \le t \le l(u)$ using $f(n)$-bit space, and a polynomial-time tree decomposition algorithm outputting the decomposition of width at most w' for any input graphs of treewidth w. Then we have the polynomial-time algorithm which implements $\textsc{IsGray}(u, v)$ using the space of $f(n) + O(w' \log n)$ bits (except for the space used by the tree decomposition algorithm).*

Proof. To implement $\textsc{IsGray}(u, v)$, it suffices to enumerate all the vertices in $S_t(u)$, which can be realized by repeatedly using an algorithm outputting $s_t(x)$ for given $x \in V_{S_t(u)} \setminus \{h_t\}$. Let $V'(x)$ be the set of the vertices v' such that $\textsc{AIsGray}(x, v')$ returns true. It has been shown in [3] that $s_t(x)$ is the first vertex $y \in N(x)$ with respect to the order of A_x such that y is reachable to h_t in $G - V_{S_t(x)}$. We first show that this fact still holds even if we replace $V_{S_t(x)}$ by $V'(x)$. Any y preceding $s_t(x)$ in A_x is unreachable to h_t in $G - V'(x)$ because it is unreachable in $G - V_{S_t(x)}$ and $V_{S_t(x)} \subseteq V'(x)$ holds by the first condition of $\textsc{AIsGray}$. Letting X be the graph corresponding to the suffix of S_t from $s_t(x)$ to h_t, any vertex in V_X has a discovery time larger than $d(x)$, and thus $V_X \cap V'(x) = \emptyset$ holds by the second condition of $\textsc{AIsGray}$. It implies that $s_t(x)$ is reachable to h_t in $G - V'(x)$, and concludes that $s_t(x)$ is the first vertex $y \in N(x)$ such that y is reachable to h_t in $G - V'(x)$. Since the procedure $\textsc{AIsGray}(x, v')$ works as the emulator of $G - V'(x)$, we can obtain an algorithm of computing $s_t(x)$ using any directed s-t reachability algorithm. The algorithm by Jain et al. [29] matches our goal. It uses any tree decomposition of width w' as a side information, and runs in a polynomial time using $O(w' \log n)$-bit space. The memory complexity is $f(n)$ bits for $\textsc{AIsGray}$, and $O(w' \log n)$ bits for deciding s-t reachability and for managing a constant number of pointers to vertices in V_G. ◀

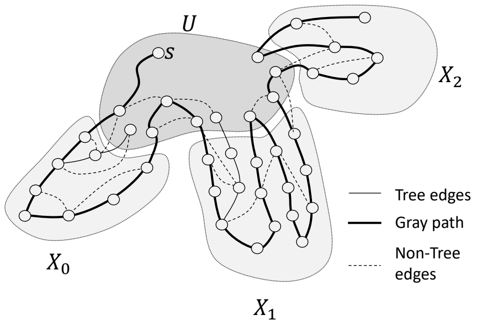


Figure 1 An example of decomposition by U.

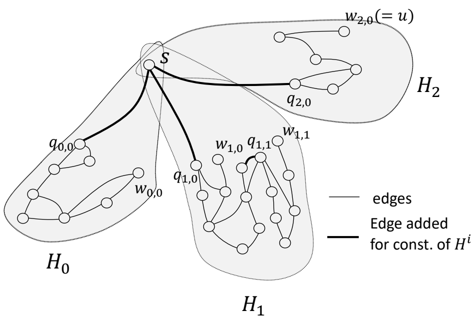


Figure 2 Construction of H_i.

3.2 Implementation of AIsGray(u, v)

As utilized in the proof of Lemma 8, directed s-t reachability is solvable using $O(w' \log n)$-bit space with the side information of a tree decomposition of width w' [29]. Thus we have Lemma 7 with $g(n) = O(w' \log n)$. The remaining part of our algorithm is to implement AIsGray(u, v) executable at any $d(u) \leq t \leq l(u)$. Let U be the set shown in Lemma 6, and $\mathcal{X} = \{X_0, X_1, \ldots, X_{N-1}\}$ be the set of the connected components in $G - U$ (Figure 1). At each time t, our algorithm keeps track of the information of $(d(x), p_t(x), s_t(x))$ for any $x \in V_{S_t} \cap U$. Specifically, we prepare the dictionary Z which maps any vertex x in U to the corresponding triple $(d(x), p_t(x), s_t(x))$ if $x \in S_t$ holds. We refer to the three elements in the triple for x as $Z[x].d$, $Z[x].p$, and $Z[x].s$ respectively. The contents of Z is updated in the main routine of Lex-DFS when the search head moves. Let ℓ_i be the number of connected components in the subgraph $S_{d(u)}[V_{X_i}]$. For each connected component C in $S_{d(u)}[V_{X_i}]$, we define its *entrance* and *exit* as the vertices with the minimum and maximum discovery times in V_C respectively. Let $\mathcal{C}_i = C_{i,0}, C_{i,1}, \ldots, C_{i,\ell_i-1}$ be the sequence of the connected components in $S_{d(u)}[V_{X_i}]$ sorted by the discovery times of their entrances. We also define the exit of $C_{i,-1}$ as s, which works as a sentinel value. Let $Q_i = (q_{i,0}, q_{i,1}, \ldots, q_{i,\ell_i-1})$ and $W_i = (w_{i,-1}, w_{i,0}, w_{i,1}, \ldots, w_{i,\ell_i-1})$ be the sequences of the entrances and exits associated with each component in $\mathcal{C}_i$ respectively. Now we construct the graph H_i from G by the following procedure:

1. Remove all the vertices not in $V_{X_i} \cup \{s\}$ as well as their incident edges.
2. For all $0 \leq j \leq \ell_i - 1$, contract the gray path from $w_{i,j-1}$ to $q_{i,j}$ into an edge. The positions of $q_{i,j}$ in $A_{w_{i,j-1}}$ and $w_{i,j-1}$ in $A_{q_{i,j}}$ are equal to those of $s_t(w_{i,j-1})$ and $p_t(q_{i,j})$ respectively.

Note that $s = w_{i,-1} = q_{i,0}$ holds if $s \in V_{X_i}$. We illustrate an example of the construction in Figure 2. Consider the run of any Lex-DFS algorithm in H_i until the discovery of w_{i,ℓ_i-1}, which outputs a vertex set $L_i \subseteq V_{X_i} \cup \{s\}$. Let $L(u) = L_0 \cup L_1 \cup \cdots \cup L_{N-1} \cup (S_{d(u)} \cap U)$. An important fact is that AIsGray(v, u) can be implemented using the query if $v \in L(u)$ or not. The following lemma holds.

▶ **Lemma 9.** *Let L_i be the output sequence of the Lex-DFS running in H_i until the discovery of w_{i,ℓ_i-1}. Any vertex in $V_{S_{d(u)}[V_{X_i}]}$ is contained in L_i, and $d(x) \leq d(w_{i,\ell_i-1}) \leq d(u)$ holds for any $x \in L_i$.*

Proof. For any $v \in V$, let $\mathcal{P}_{G,v}$ be the set of all simple paths from s to v in G, and $\mathcal{P}_G = \bigcup_{v \in V_G} \mathcal{P}_{G,v}$. For any path $P = s, u_1, \ldots, u_j, v$ in $\mathcal{P}_{G,v}$, we define its *word* $\gamma_G(P)$ as the sequence $A_s^{-1}[u_1], A_{u_1}^{-1}[u_2], \ldots, A_{u_k}^{-1}[v]$. Letting $\prec$ be the lexicographic order over all words, the *minimum path* $\pi_G(v) \in \mathcal{P}_{G,v}$ of a vertex $v \in V$ is defined as the one satisfying

$\gamma_G(\pi_G(v)) \prec \gamma_G(P)$ for any $P \in \mathcal{P}_{G,v}$. For any two vertices $v, v' \in V_{H_i}$, the gray paths in H_i to v and v' are respectively obtained by contracting several common subpaths in the gray paths to v and v' in G. Hence it is easy to check $\gamma_G(\pi_G(v)) \prec \gamma_G(\pi_G(v'))$ holds if and only if $\gamma_{H_i}(\pi_{H_i}(v)) \prec \gamma_{H_i}(\pi_{H_i}(v'))$ holds for any $v, v' \in V_{H_i}$. Since it is well-known that the total ordering of V_G with respect to $\prec$ over $\{\gamma_G(\pi_G(v))\}_{v \in V_G}$ is equivalent to the Lex-DFS ordering of V_G, this fact implies that L_i contains all the vertices in V_{X_i} discovered earlier than $w_{i,\ell-1}$ in the Lex-DFS search in G, and contains no vertex in V_{X_i} whose discovery time in the Lex-DFS search in G is later than $d(w_{i,\ell-1})$. The lemma is proved. ◀

Since H_i is a minor of G, its treewidth is also bounded by w. Thus we can perform our Lex-DFS algorithm recursively for graph H_i of $O(n^{1-\epsilon})$ vertices to output L_i. The graph H_i can be emulated using the subset U and the information stored in Z. Outputting L_i for all $i \in [0, N-1]$ can answer the query if $v \in L(u)$ holds or not.

3.3 Algorithm Details for Lex-DFS

The pseudocode of our algorithm is given in Algorithm 2. It is defined as a recursive procedure Lex-DFS(G, s, u), which outputs the Lex-DFS sequence of G starting from s until u is discovered. If the procedure runs with $u \notin V_G$, it outputs the whole Lex-DFS sequence of G starting from s. Note that the dictionary Z is independently defined in each recursive call for the computed separator U. In addition, the size $O(n^\epsilon)$ of separator U is fixed independently of recursion depth. That is, the variable n in the size parameter $O(n^\epsilon)$ is always the number of vertices in the original input graph, not the number of vertices in the input graph taken as an argument of Lex-DFS. The main routine Lex-DFS almost follows Algorithm 1, except for using AIsGray to compute Pivot and Parent and managing Z. The core of the algorithm is the implementation of AIsGray, in particular, the emulation of H_i for each V_i $(0 \leq i \leq N-1)$. That part corresponds to the lines 23-30. First, we identify the set Q_i and W_i, which can be done by extracting the nodes $x \in U$ satisfying $Z[x].s \in V_i$ as a member of Q_i (or those satisfying $Z[x].p \in V_i$ as W_i). Since each node in $S_t(u) \cap U$ stores its discovery time in Z, we can add the nodes into Q_i or W_i in the order of their discovery times. Following the order of Q_i and W_i, we create the edge set F, which corresponds to the edges obtained by the contraction of gray subpaths in the step 2 of the construction.

3.4 Complexity

Since Pivot and Parent are called at most $2n$ times in each recursive call, a polynomial-time invocations of AIsGray suffices to implement them. Let n^c be the upper bound for the number of invocations of AIsGray in one execution of Pivot or Parent. One invocation of AIsGray calls Lex-DFS at most n times. Putting all them together, n^{c+2} recursive invocations of Lex-DFS occur per one call of Lex-DFS. Since the recursion depth is obviously bounded by $O(1/\epsilon)$, at most $O(n^{(c+2)/\epsilon})$ calls of Lex-DFS are invoked in total. By Lemma 5, the input graph to each recursive call can be emulated with a polynomial-time overhead, and thus one invocation of Lex-DFS excepting the run of recursive calls has a polynomially-bounded running time. Consequently, the total running time is $n^{O(1/\epsilon)}$.

In each recursive call, the information on U and Z is stored in the working memory. The space for storing U and Z are bounded by $O(w'n^\epsilon \log n)$ bits. Except for the space used by the tree decomposition algorithm, the space of $O(w' \log n)$ bits is necessary for implementing Parent and Pivot from AIsGray. Since the recursion depth is $O(1/\epsilon)$, the total space complexity is $O(w'\epsilon^{-1}n^\epsilon \log n)$ bits.

Algorithm 2 Lex-DFS Algorithm for graph G of treewidth k (starting from s).

```
function Lex-DFS$(G, s, u)$
    if $|V_G| \leq n^\epsilon$ then run the standard Lex-DFS algorithm and halt
    Find a separator $U$ of size $O(n^\epsilon)$
    Initialize $Z : U \to [0, n-1] \times V_G \times V_G$
    $v_{cur} \leftarrow s$; $t \leftarrow 1$
    output $s$; $Z[s] \leftarrow (1, \perp, \perp)$
    while true do
        $v \leftarrow$ Pivot$(t)$ using AIsGray$(v_{cur}, \cdot)$
        if $v = -1$ then                         ▷ All neighbors have been already visited
            $v \leftarrow$ Parent$(t)$ using AIsGray$(v_{cur}, \cdot)$
            $Z[v_{cur}] \leftarrow$ null; $Z[v] \leftarrow (Z[v].d, Z[v].p, \perp)$
            if $v = -1$ then halt                ▷ All vertices are visited
        else
            $Z[v_{cur}] \leftarrow (Z[v_{cur}].d, Z[v_{cur}].p, v)$; $Z[v] \leftarrow (t, v_{cur}, \perp)$
            Output $v$
            if $v = u$ then halt
        $v_{cur} \leftarrow v$
        $t \leftarrow t + 1$
    function AIsGray$(u, v)$
        if $Z[v] \neq$ null then return true
        Let $X_0, X_1, \ldots, X_{N-1}$ be the connected components in $G - U$
        for $i = 0, 1, \ldots, N-1$ do
            $Q \leftarrow ()$; $W \leftarrow (s)$
            for $\forall x \in U$: $Z[x] \neq$ null in ascending order of $Z[x].d$ do
                if $Z[x].s \in V_{X_i}$ then append $Z[x].s$ to $Q$
                if $Z[x].p \in V_{X_i}$ then append $Z[x].p$ to $W$
            if $u \in V_{X_i}$ then append $u$ to $W$
            $\ell_i \leftarrow |Q|$
            for $j = 0, 1, \ldots, \ell_i - 1$ do
                $F \leftarrow F \cup \{(W[j], Q[j])\}$
            $H_i \leftarrow G[V_i] + F$                ▷ Not explicitly constructed
            if $v \in$ Lex-DFS$(H_i, s, W[\ell_i - 1])$ then return true
        return false
```

4 Tree Decomposition using Small Space

In this section, we present a tree-decomposition algorithm, which uses $O(wn^{1/2} \log n)$-bit space and outputs a decomposition of width $O(wn^{1/2} \log n)$ for any undirected graph $G = (V, E)$ of treewidth $w \leq \sqrt{n}$. We first introduce a space-saving variant of the known weighted balanced separator algorithm.

▶ **Lemma 10** (Extended from Theorem 1.1 of Fomin et al. [24]). *There exists a polynomial-time algorithm that, given a graph G on n vertices, any vertex-weight function μ, and a positive integer k, either provides a weighted $O(1)$-balanced separator of G with respect to μ consisting* *of $O(k^2)$ vertices, or concludes that the treewidth of G is more than k. The algorithm uses the memory space required for finding the minimum unweighted s-t vertex cut in G plus $O(k^2 \log n)$ bits.*

Proof. We refer to the algorithm proposed in Theorem 1.1 of [24] as SEP. Except for the space complexity matter, the correctness of the lemma completely follows that of SEP presented in [24]. Thus it suffices to show how SEP is implemented using the memory space claimed in this lemma. The algorithm SEP roughly works as follows. Let G be any input graph of treewidth at most k.

1. First the algorithm SEP constructs any rooted spanning tree T of G, and decomposes it into a set $\mathcal{X}$ of $\Theta(k)$ connected subtrees such that at most $O(k)$ vertices can belong to two or more subtrees in $\mathcal{X}$: Let $T(x)$ be the subtree of T rooted by x. The algorithm SEP starts with $T' = T$, and for $i = 0, 1, \dots,$ iteratively finds the deepest vertex x_i such that $|V_{T'(x_i)}| \geq \alpha n/k$ holds for an appropriate constant α. Then the subtree $T'(x_i)$ is split into several subtrees of size $\Theta(n/k)$ sharing x_i, each of which becomes a member of $\mathcal{X}$. After updating T' as $T' \leftarrow T' - T'(x_i)$, the algorithm proceeds to the next iteration.
2. For any $X_i, X_j \in \mathcal{X}$ such that $V_{X_i} \cap V_{X_j} = \emptyset$, SEP emulates the graph $H_{i,j}$ obtained from G by contracting X_i and X_j into two vertices x_i and x_j. Then it computes the minimum x_i-x_j vertex cut in $H_{i,j}$. If there exists a pair (i, j) such that the output cut contains at most k vertices, the algorithm adds it to the separator set U.
3. The steps 1 and 2 are iteratively applied to the largest connected component after the removal of the computed vertex cut, until U becomes an $O(1)$-balanced separator of G. It is proved in [24] that this iteration terminates within $O(k)$ times if the treewidth of the input graph is at most k.

The small-space implementation of step 1 is very similar with the algorithm shown in the proof of Lemma 6. With the support of the emulator of T, finding x_i and the emulation of T' can be done in the same way as the proof of Lemma 6. The spanning tree T can be emulated using the logspace undirected connectivity: We introduce an arbitrary logspace-computable edge-weight function $g \colon E_G \to \mathbb{N}$ which assigns all edges with different weights. Let $e_0, e_1, \dots, e_{m-1}$ be the sequence of all edges sorted in the ascending order of their weights, and $E_i = \{e_0, e_1, \dots, e_{i-1}\}$. Then an edge $e_i = (u, v)$ is contained in the minimum spanning tree of G with respect to g if and only if u and v is connected in $G[E_i]$, which directly deduces the emulator of the minimum spanning tree.

Assuming an algorithm computing the set $\mathcal{X}$, $H_{i,j}$ can be emulated with a polynomial-time overhead. Thus the step 2 can be implemented within a polynomial time using $O(k^2 \log n)$ bits (except for the space used by the vertex-cut algorithm). ◀

4.1 A Small-Space Balanced Separator Algorithm

Let $I(G)$ be the maximum independent set of G, (if two or more maximum independent sets exist, an arbitrary one is chosen), and $\overline{I}(G) = V \setminus I(G)$ for short. The first key ingredient of our algorithm is a space-saving algorithm for the minimum s-t vertex cut problem.

▶ **Lemma 11.** *Let G be any n-vertex undirected graph. For any $s, t \in V_G$, the minimum (unweighted) s-t vertex cut of G can be found in a polynomial time using $O(|\overline{I}(G)| \log n)$ bits.*

Proof. The algorithm basically follows the vertex-cut version of Ford-Fulkerson algorithm, which manages a set of augmenting paths for recognizing the current residual graph (see Section 3.5 in [35] for example). In the case of unweighted vertex cuts, any set of augmenting paths is a set of vertex-disjoint s-t paths in G. Letting L be the maximum total length of managed s-t paths, the algorithm can be implemented using $O(L \log n)$-bit space. Thus it suffices to show that $L = O(|\overline{I}(G)|)$ holds for any instance G. Let $R_1, R_2, \ldots, R_y$ be any set of vertex-disjoint s-t paths in G. Since no two vertices in $I(G)$ consecutively appears in any path, we have $|V_{R_i} \cap I(G)| \leq |V_{R_i} \cap \overline{I}(G)| + 1$ for any $i \in [1, y]$. Then $|V_{R_i}| \leq 3|V_{R_i} \cap \overline{I}(G)|$ holds. Since V_{R_i} for all $i \in [1, y]$ are mutually disjoint, it follows $\sum_{1 \leq i \leq y} |V_{R_i}| = O(|\overline{I}(G)|)$. The lemma is proved. ◀

Consider a partition of V_G into a family $\mathcal{P} = \{P_0, P_2, \ldots, P_{N-1}\}$ of N subsets such that $G[P_i]$ is a connected subgraph of G. We denote by $G/\mathcal{P}$ the graph obtained by contracting each subgraph P_i into a single vertex u_i with weight $\mu(u_i) = |V_{P_i}|$ (parallel edges are merged into the single one). The second key ingredient is to reduce the (approximate) tree decomposition of G into that of another graph $G/\mathcal{P}$ for an appropriate partition $\mathcal{P}$ such that $|\overline{I}(G/\mathcal{P})|$ becomes small. Since treewidth never increases by edge contraction, $G/\mathcal{P}$ also has a treewidth at most k. Thus we can run the balanced-separator algorithm obtained from Lemmas 10 and 11 on $G/\mathcal{P}$ using only $O((|\overline{I}(G/\mathcal{P})| + k^2) \log n)$-bit space. For the computed separator B, we replace each $u_i \in B$ by V_{P_i}. That is, we create a vertex subset $B' = \bigcup_{u_i \in B} V_{P_i}$, which is obviously a balanced separator of G consisting of $O(k^2 \max_i\{|V_{P_i}|\})$ vertices. To attain the space complexity of Theorem 2 following this approach, we have to construct a polynomial-time algorithm outputting the partition $\mathcal{P}$ satisfying $|\overline{I}(G/\mathcal{P})| = O(kn^{1/2})$ and $|P_i| = O(n^{1/2}/k)$ for any $P_i \in \mathcal{P}$. In addition, we have to guarantee that the algorithm uses only $O(kn^{1/2} \log n)$ bits.

We argue the implementation of such an algorithm. Let us define an arbitrary total ordering of edges in E_G. This can be easily realized by any ordering function $f \colon E \to \mathbb{N}$ which can be computed in logarithmic space (e.g., the lexicographic ordering of endpoint ID pairs). Let $e_1, e_2, \ldots, e_m$ be the sequence of all edges sorted in this order, and $E_i = \{e_1, e_2, \ldots, e_i\}$. For any subgraph $H \subseteq G$, we also define $S(H, v, i)$ as the set of the vertices which are reachable from v in $H[E_i]$. The partition is constructed by the following algorithm:

1. Let $\mathcal{P} = \emptyset$, $R \leftarrow \emptyset$, and $H \leftarrow G$.
2. Find the minimum ℓ such that the largest connected component C in $H[E_\ell]$ contains at least $n^{1/2}/k$ vertices, and add V_C to $\mathcal{P}$. Letting v be the vertex with the smallest ID in V_C, we store the pair (v, ℓ) into R.
3. Update $H \leftarrow G - \bigcup_{(v,\ell) \in R} S(G, v, \ell)$.
4. Repeat steps 2 and 3 until the size of any connected component in H becomes less than $n^{1/2}/k$.
5. Letting $Q = \bigcup_{P_i \in \mathcal{P}} P_i$, add $V_{C'}$ to $\mathcal{P}$ for any connected component C' in $G - Q$.

The actual algorithm does not store $\mathcal{P}$ explicitly. Except for step 5, the set $\mathcal{P}$ is write-only, and it is easy to verify that steps 1-4 can be implemented only with the space for storing R. The matter of the space complexity relies on how to restore the set Q in step 5 only from the information of R. Let C_i be the connected component found in the i-th iteration of step 2, and (u_i, j_i) be the entries added to R then. We denote by H_i the graph stored in H immediately after the i-th iteration of step 2. It is easy to enumerate the vertices in $S(G, u, j)$ by the logspace undirected s-t connectivity algorithm [41] (recall that we omit the orientation of edges in considering the tree decomposition for directed graphs), and thus we obtain an emulator of H_i using the information in R and extra $O(\log n)$-bit space. It also yields an algorithm for enumerating $C_i = S(H_{i-1}, u_i, j_i)$. Consequently, this algorithm works using only $O(|R| \log n)$-bit space. The following lemma guarantees the correctness of the output.

▶ **Lemma 12.** *Let $\mathcal{P} = \{P_0, P_1, \ldots, P_{N-1}\}$ be the partition outputted by the algorithm above. Then for any $P_i \in \mathcal{P}$, $|P_i| \leq 2n^{1/2}/k$ holds. In addition, $|\overline{I}(G/\mathcal{P})| \leq kn^{1/2}$ holds.*

Proof. We first show that $\mathcal{P}$ is actually a partition of V_G. By step 5, it is obvious that any vertex in V_G is contained in at least one subset $P_i \in \mathcal{P}$. The subset added in step 5 does not intersect other subsets. Consider any two subsets C_i and C_k added in step 2 ($i < k$). By the construction of C_i and C_k, we have $C_i = S(H_{i-1}, u_i, j_i)$ and $C_k = S(H_{k-1}, u_k, j_k)$. Since $i < k$ holds, H_{k-1} does not contain any vertex in $S(G, u_i, j_i)$. It implies H_{k-1} does not contain any vertex in $C_i = S(H_{i-1}, u_j, j_i)$ because of $H_{i-1} \subseteq G$. That is, C_i and C_k are mutually disjoint. Next, we bound the size of each P_i. Any subset added in step 5 has a cardinality less than $n^{1/2}/k$. Since the algorithm finds the smallest ℓ such that the cardinality of C becomes at least $n^{1/2}/k$, any connected component in $H_{i-1}[E_{\ell-1}]$ has a size less than $n^{1/2}/k$. Thus the size of any connected component in $H_{i-1}[E_\ell]$ is at most $2n^{1/2}/k$. Finally, we show $|\overline{I}(G/\mathcal{P})| \leq kn^{1/2}$. We call a subset $P_i \in \mathcal{P}$ a *red subset* if P_i is added in step 5, and also call the corresponding vertex $u_i \in V_{G/\mathcal{P}}$ a *red vertex*. It is obvious that any red subset P_i has no outgoing edge to other red subsets in G, the set of all red vertices forms an independent set of $G/\mathcal{P}$. Since the cardinality of any non-red subset is at least $n^{1/2}/k$, at most $kn^{1/2}$ non-red vertices exist in $G/\mathcal{P}$. It implies $\overline{I}(G/\mathcal{P}) \leq kn^{1/2}$. The lemma is proved. ◀

This lemma also implies that the size of R is at most $kn^{1/2}$. Thus the space complexity of the algorithm is bounded by $O(kn^{1/2} \log n)$ bits. The combination of Lemmas 10, 11, and 12 yields the following lemma.

▶ **Lemma 13.** *There exists a polynomial-time algorithm that, given a graph G on n vertices and a positive integer k, either provides an $O(1)$-balanced separator of G consisting of $O(kn^{1/2})$ vertices, or concludes that the treewidth of G is more than k. The algorithm uses $O(kn^{1/2} \log n)$-bit space.*

The remaining part is to transform the separator algorithm into a tree-decomposition algorithm. The following lemma obviously deduces Theorem 2.

▶ **Lemma 14.** *Assume an algorithm Alg which outputs an $O(1)$-balanced separator of size $k(w, n)$ for any graph G of treewidth w using $g(n)$-bit space. Then there exists a polynomial-time tree decomposition algorithm which outputs a tree decomposition of width $O(k(w, n) \log n)$ using $O(g(n) + k(w, n) \log^2 n)$-bit working memory.*

Proof. The proof is constructive. We refer to the constructed algorithm as *Alg*. The algorithm *Alg* first computes an $O(1)$-balanced separator U of G, and then recursively constructs the tree decomposition of each connected component in $G - U$ whose size is larger than $k(w, n)$. A component having at most $k(w, n)$ vertices is treated as the subgraph with the tree decomposition consisting of a single bag of the whole component. Let $H_0, H_1, \ldots, H_{\ell-1}$ be the connected components in $G - U$ sorted in the order specified by the enumeration algorithm of Lemma 5, and T_i be the output sequence of the recursive call for H_i. Defining the binary operator $\circ$ for concatenating two sequences, let $T = T_0 \circ T_1 \circ \cdots \circ T_{\ell-1}$. We further define $m_i = \sum_{0 \leq j \leq i} |T_i|$. The algorithm *Alg* modifies the sequence T in the following way: Any pair $(B, q) \in T_i$ is replaced by $(B \cup U, q + m_i)$ if $q \neq -1$ or $(B \cup U, m_{\ell-1})$ otherwise. Finally, we append the pair $(U, -1)$ at the tail of T. Intuitively, this modification is for relabeling bag identifiers to guarantee their uniqueness, and for merging all the subgraph decompositions into a single one rooted by the last bag $(U, -1)$. It is easy to verify that the modified sequence is a tree decomposition of G. By Lemma 5, the input graph at any recursion level is emulated with a polynomial-time overhead. Thus the running time at any recursion level is a polynomial time. One recursive call remove one bag from the input graph, the number of recursive calls is bounded by n. Totally the algorithm *Alg* finishes within a polynomial time.

Let w_i be the maximum width of all the output tree decompositions at the i-th recursion level. Then we have the inequality $w_{i+1} \leq k(w,n) + w_i$. Since the separator is $O(1)$-balanced, the recursion finishes at the depth of $O(\log n)$. It implies that the maximum bag size is $O(k(w,n)\log n)$. The modification of the sequence T can be done in a streaming way. Thus the space complexity of *Alg* is $O(|U|\log n) = O(k(w,n)\log n)$ bits per one recursion. The largest space consumption is at the bottom-level recursion, where *Alg* uses $O(k(w,n)\log^2 n)$ bits in total. ◀

The lemma above also deduces the consequence for planar graphs in Corollary 3. Since planar graphs admit a $\widetilde{O}(\sqrt{n})$-bit space $O(1)$-balanced separator algorithm [27], we can use it instead of Lemma 13.

5 Conclusion

In this paper, we presented a Lex-DFS algorithm for directed graphs of bounded treewidth w. It is not only the first algorithm solving Lex-DFS using sublinear space for $w = \omega(1)$, but also the first algorithm solving directed s-t reachability in the same situation. One of the key tools is a new sublinear-space tree decomposition algorithm covering the case of moderate (small but non-constant) treewidth. The authors believe that this is a strong tool for designing small-space algorithms for other fundamental graph problems on bounded-treewidth graphs.

References

1 Eric Allender, David A. Mix Barrington, Tanmoy Chakraborty, Samir Datta, and Sambuddha Roy. Planar and grid graph reachability problems. *Theory of Computing Systems.*, 45(4):675–723, 2009.

2 Richard Anderson and Ernst W. Mayr. Parallelism and the maximal path problem. *Information Processing Letters*, 24(2):121–126, 1987. doi:10.1016/0020-0190(87)90105-0.

3 Tetsuo Asano, Taisuke Izumi, Masashi Kiyomi, Matsuo Konagaya, Hirotaka Ono, Yota Otachi, Pascal Schweitzer, Jun Tarui, and Ryuhei Uehara. Depth-first search using $O(n)$ bits. In *International Symposium on Algorithms and Computation (ISAAC)*, pages 553–564, 2014.

4 Tetsuo Asano, David Kirkpatrick, Kotaro Nakagawa, and Osamu Watanabe. $\widetilde{O}(\sqrt{n})$-space and polynomial-time algorithm for planar directed graph reachability. In *International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 45–56, 2014.

5 Ryo Ashida and Kotaro Nakagawa. $\widetilde{O}(n^{1/3})$-space algorithm for the grid graph reachability problem. In *International Symposium on Computational Geometry (SoCG)*, pages 5:1–5:13, 2018.

6 Niranka Banerjee, Sankardeep Chakraborty, and Venkatesh Raman. Improved space efficient algorithms for BFS, DFS and applications. In *International Computing and Combinatorics Conference (COCOON)*, pages 119–130, 2016.

7 Bahareh Banyassady, Matias Korman, Wolfgang Mulzer, André van Renssen, Marcel Roeloffzen, Paul Seiferth, and Yannik Stein. Improved Time-Space Trade-Offs for Computing Voronoi Diagrams. In *International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 9:1–9:14, 2017. doi:10.4230/LIPIcs.STACS.2017.9.

8 Luis Barba, Matias Korman, Stefan Langerman, Kunihiko Sadakane, and Rodrigo I. Silveira. Space-time trade-offs for stack-based algorithms. *Algorithmica*, 72(4):1097–1129, 2015. doi:10.1007/s00453-014-9893-5.

9 Greg Barnes, Jonathan F. Buss, Walter L. Ruzzo, and Baruch Schieber. A sublinear space, polynomial time algorithm for directed s-t connectivity. *SIAM Journal on Computing*, 27(5):1273–1282, 1998. doi:10.1137/S0097539793283151.

10 Allan B. Borodin and Stephan Cook. A time-space tradeoff for sorting on a general sequential model of computation. *SIAM Journal on Computing*, 11(2):287–297, 1982. doi:10.1137/0211022.

11 Harry Buhrman, Richard Cleve, Michal Koucký, Bruno Loff, and Florian Speelman. Computing with a full memory: Catalytic space. In *ACM Symposium on Theory of Computing (STOC)*, pages 857–866, 2014. doi:10.1145/2591796.2591874.

12 Diptarka Chakraborty, A. Pavan, Raghunath Tewari, N. Variyam Vinodchandran, and Lin Forrest Yang. New Time-Space Upperbounds for Directed Reachability in High-genus and H-minor-free Graphs. In *International Conference on Foundation of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 585–595, 2014.

13 Diptarka Chakraborty and Raghunath Tewari. An $O(n^\epsilon)$ space and polynomial time algorithm for reachability in directed layered planar graphs. *ACM Transactions on Computation Theory*, 9(4), 2017.

14 Sankardeep Chakraborty, Anish Mukherjee, Venkatesh Raman, and Srinivasa Rao Satti. A Framework for In-place Graph Algorithms. In *European Symposium on Algorithms (ESA)*, volume 112, pages 13:1–13:16, 2018. doi:10.4230/LIPIcs.ESA.2018.13.

15 Sankardeep Chakraborty, Venkatesh Raman, and Srinivasa Rao Satti. Biconnectivity, Chain Decomposition and st-Numbering Using O(n) Bits. In *International Symposium on Algorithms and Computation (ISAAC)*, volume 64, pages 22:1–22:13, 2016. doi:10.4230/LIPIcs.ISAAC.2016.22.

16 Sankardeep Chakraborty and Srinivasa Rao Satti. Space-efficient algorithms for maximum cardinality search, its applications, and variants of bfs. *Jouanal of Combinatorial Optimization*, 37(2):465–481, 2019. doi:10.1007/s10878-018-0270-1.

17 Timothy M. Chan. Comparison-based time-space lower bounds for selection. *ACM Transactions on Algorithms*, 6(2):26:1–26:16, 2010. doi:10.1145/1721837.1721842.

18 Timothy M. Chan, J. Ian Munro, and Venkatesh Raman. Faster, space-efficient selection algorithms in read-only memory for integers. In *International Symposium on Algorithms and Computation (ISAAC)*, pages 405–412, 2013. doi:10.1007/978-3-642-45030-3_38.

19 Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

20 Omar Darwish and Amr Elmasry. Optimal time-space tradeoff for the 2D convex-hull problem. In *European Symposium on Algorithms (ESA)*, pages 284–295, 2014.

21 Jeff Edmonds, Chung Keung Poon, and Dimitris Achlioptas. Tight lower bounds for st-connectivity on the nnjag model. *SIAM Journal on Computing*, 28(6):2257–2284, 1999. doi:10.1137/S0097539795295948.

22 Michael Elberfeld, Andreas Jakoby, and Till Tantau. Logspace versions of the theorems of Bodlaender and Courcelle. In *IEEE 51st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 143–152, 2010.

23 Amr Elmasry, Torben Hagerup, and Frank Kammer. Space-efficient Basic Graph Algorithms. In *International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 288–301, 2015. doi:10.4230/LIPIcs.STACS.2015.288.

24 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, Michał Pilipczuk, and Marcin Wrochna. Fully polynomial-time parameterized computations for graphs and matrices of low treewidth. *ACM Transactions on Algorithms*, 14(3), 2018.

25 Greg N. Frederickson. Upper bounds for time-space trade-offs in sorting and selection. *Journal of Computer and System Sciences*, 34(1):19–26, 1987. doi:10.1016/0022-0000(87)90002-X.

26 Torben Hagerup. Space-efficient DFS and applications to connectivity problems: Simpler, leaner, faster. *Algorithmica*, 82(4):1033–1056, 2020. doi:10.1007/s00453-019-00629-x.

27 Tatsuya Imai, Kotaro Nakagawa, Aduri Pavan, N. Variyam Vinodchandran, and O. Watanabe. An $O(n^{1/2+\epsilon})$-space and polynomial-time algorithm for directed planar reachability. In *IEEE Conference on Computational Complexity (CCC)*, pages 277–286, 2013.

28 Rahul Jain and Raghunath Tewari. An $O(n^{1/4+\epsilon})$ Space and Polynomial Algorithm for Grid Graph Reachability. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 19:1–19:14, 2019.

29 Rahul Jain and Raghunath Tewari. Reachability in High Treewidth Graphs. In *International Symposium on Algorithms and Computation (ISAAC)*, volume 149, pages 12:1–12:14, 2019. doi:10.4230/LIPIcs.ISAAC.2019.12.

30 Frank Kammer, Dieter Kratsch, and Moritz Laudahn. Space-efficient biconnected components and recognition of outerplanar graphs. *Algorithmica*, 81(3):1180–1204, 2019. doi:10.1007/s00453-018-0464-z.

31 Frank Kammer, Johannes Meintrup, and Andrej Sajenko. Space-efficient vertex separators for treewidth. *CoRR*, abs/1907.00676, 2019. arXiv:1907.00676.

32 Frank Kammer and Andrej Sajenko. Linear-time in-place dfs and bfs on the word ram. In *International Conference on Algorithms and Complexity (CIAC)*, pages 286–298, 2019.

33 Shahbaz Khan and Shashank K. Mehta. Depth First Search in the Semi-streaming Model. In *International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 126, pages 42:1–42:16, 2019. doi:10.4230/LIPIcs.STACS.2019.42.

34 Masashi Kiyomi, Hirotaka Ono, Yota Otachi, Pascal Schweitzer, and Jun Tarui. Space-efficient algorithms for longest increasing subsequence. *Theory of Computing Systems*, 2019. doi:10.1007/s00224-018-09908-6.

35 Tom Leighton and Satish Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM*, 46(6):787–832, 1999. doi:10.1145/331524.331526.

36 Andrea Lincoln, Virginia Vassilevska Williams, Joshua R. Wang, and R. Ryan Williams. Deterministic Time-Space Trade-Offs for k-SUM. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 55, pages 58:1–58:14, 2016. doi:10.4230/LIPIcs.ICALP.2016.58.

37 J.Ian Munro and Mike Paterson. Selection and sorting with limited storage. *Theoretical Computer Science*, 12(3):315–323, 1980. doi:10.1016/0304-3975(80)90061-4.

38 Jakob Pagter and Theis Rauhe. Optimal time-space trade-offs for sorting. In *ACM Symposium on Theory of Computer Science (STOC)*, pages 264–268, 1998. doi:10.1109/SFCS.1998.743455.

39 Chung Keung Poon. Space bounds for graph connectivity problems on node-named jags and node-ordered jags. In *IEEE 34th Annual Symposium on Foundations of Computer Science (FOCS)*, page 218–227, 1993.

40 John H. Reif. Depth-first search is inherently sequential. *Information Processing Letters*, 20(5):229–234, 1985. doi:10.1016/0020-0190(85)90024-9.

41 Omer Reingold. Undirected connectivity in log-space. *Journal of the ACM*, 55(4), 2008. doi:10.1145/1391289.1391291.

42 Joshua R. Wang. Space-efficient randomized algorithms for k-sum. In *European Symposium on Algorithms (ESA)*, pages 810–829, 2014.

Scheduling in the Random-Order Model

Susanne Albers
Department of Computer Science, Technical University of Munich, Germany
albers@in.tum.de

Maximilian Janke
Department of Computer Science, Technical University of Munich, Germany
maximilian.janke@in.tum.de

Abstract

Makespan minimization on identical machines is a fundamental problem in online scheduling. The goal is to assign a sequence of jobs to m identical parallel machines so as to minimize the maximum completion time of any job. Already in the 1960s, Graham showed that *Greedy* is $(2-1/m)$-competitive [18]. The best deterministic online algorithm currently known achieves a competitive ratio of 1.9201 [14]. No deterministic online strategy can obtain a competitiveness smaller than 1.88 [34].

In this paper, we study online makespan minimization in the popular random-order model, where the jobs of a given input arrive as a random permutation. It is known that *Greedy* does not attain a competitive factor asymptotically smaller than 2 in this setting [32]. We present the first improved performance guarantees. Specifically, we develop a deterministic online algorithm that achieves a competitive ratio of 1.8478. The result relies on a new analysis approach. We identify a set of properties that a random permutation of the input jobs satisfies with high probability. Then we conduct a worst-case analysis of our algorithm, for the respective class of permutations. The analysis implies that the stated competitiveness holds not only in expectation but with high probability. Moreover, it provides mathematical evidence that job sequences leading to higher performance ratios are extremely rare, pathological inputs. We complement the results by lower bounds for the random-order model. We show that no deterministic online algorithm can achieve a competitive ratio smaller than 4/3. Moreover, no deterministic online algorithm can attain a competitiveness smaller than 3/2 with high probability.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases Scheduling, makespan minimization, online algorithm, competitive analysis, lower bound, random-order

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.68

Category Track A: Algorithms, Complexity and Games

Related Version `https://arxiv.org/abs/2006.00386`

Funding Work supported by the European Research Council, Grant Agreement No. 691672, project APEG.

1 Introduction

We study one of the most basic scheduling problems. Consider a sequence of jobs $\mathcal{J} = J_1, \ldots, J_n$ that has to be assigned to m identical parallel machines. Each job J_t has an individual processing time p_t, $1 \leq t \leq n$. Preemption of jobs is not allowed. The goal is to minimize the *makespan*, i.e. the maximum completion time of any job in the constructed schedule. Both the offline and online variants of this problem have been studied extensively, see e.g. [4, 11, 14, 18, 20, 33] and references therein.

We focus on the online setting, where jobs arrive one by one. Whenever a job J_t is presented, its processing time p_t is revealed. The job has to be scheduled immediately on one of the machines without knowledge of any future jobs J_s, with $s > t$. Given a job sequence $\mathcal{J}$,

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 68; pp. 68:1–68:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

let $A(\mathcal{J})$ denote the makespan of an online algorithm A on $\mathcal{J}$. Let $OPT(\mathcal{J})$ be the optimum makespan. A deterministic online algorithm A is *c-competitive* if $A(\mathcal{J}) \leq c \cdot OPT(\mathcal{J})$ holds for all $\mathcal{J}$ [38]. The best competitive ratio that can be achieved by deterministic online algorithms is in the range $[1.88, 1.9201]$. No randomized online algorithm is known that beats deterministic ones, for general m.

In this paper we investigate online makespan minimization in the random-order model. Here an input instance / job sequence is chosen by an adversary. Then a random permutation of the input elements / jobs arrives. The random-order model was considered by Dynkin [10] and Lindley [28] for the secretary problem. Over the last years the framework has received quite some research interest and many further problems have been studied. These include generalized secretary problems [2, 3, 13, 27, 28], the knapsack problem [2, 27], bin packing [25], facility location [30], matching problems [16, 21, 29], packing LPs [26] and convex optimization [19].

We present an in-depth study of online makespan minimization in the random-order model. As a main contribution we devise a new deterministic online algorithm that achieves a competitive ratio of 1.8478. After almost 20 years this is the first progress for the pure online setting, where an algorithm does not resort to extra resources in handling a job sequence.

Previous work. We review the most important results relevant to our work and first address the standard setting where an online algorithm must schedule an arbitrary, worst-case job sequence. Graham in 1966 showed that the famous *Greedy* algorithm, which assigns each job to a least loaded machine, is $(2 - \frac{1}{m})$-competitive. Using new deterministic strategies the competitiveness was improved in a series of papers. Galambos and Woeginger [15] gave an algorithm with a competitive ratio of $(2 - \frac{1}{m} - \epsilon_m)$, where ϵ_m tends to 0 as $m \to \infty$. Bartal et al. [4] devised a 1.986-competitive algorithm. The bound was improved to 1.945 [22] and 1.923 [1]. Fleischer and Wahl [14] presented an algorithm that attains a competitive ratio of 1.9201 as $m \to \infty$. Chen et al. [7] gave an algorithm whose competitiveness is at most $1 + \varepsilon$ times the best possible factor, but no explicit bound was provided. Lower bounds on the competitive ratio of deterministic online algorithms were shown in [1, 5, 12, 17, 34, 35]. For general m, the bound was raised from 1.707 [12] to 1.837 [5] and 1.854 [17]. Rudin [34] showed that no deterministic strategy has a competitiveness smaller than 1.88.

For randomized online algorithms, there is a significant gap between the best known upper and lower bounds. For $m = 2$ machines, Bartal et al. [4] presented an algorithm that achieves an optimal competitive ratio of 4/3. To date, there exists no randomized algorithm whose competitiveness is smaller than the deterministic lower bound, for general m. The best known lower bound on the performance of randomized online algorithms tends to $e/(e-1) \approx 1.581$ as $m \to \infty$ [6, 37].

Recent research on makespan minimization has examined settings where an online algorithm is given extra resources when processing a job sequence. Specifically, an algorithm might have a buffer to reorder the incoming job sequence [11, 24] or is allowed to migrate jobs [36]. Alternatively, an algorithm has information on the job sequence [8, 9, 23, 24], e.g. it might know the total processing time of the jobs or even the optimum makespan.

In the random-order model only one result is known for makespan minimization on identical machines. Osborn und Torng [32] showed that *Greedy* does not achieve a competitive ratio smaller than 2 as $m \to \infty$. Recently Molinaro [31] studied online load balancing with the objective to minimize the l_p-norm of the machine loads. He considers a general scenario with machine-dependent job processing times. For makespan minimization he presents an algorithm that, in the worst case, is $O(\log m/\varepsilon)$-competitive and, in the random-order model, has an expected makespan of $(1+\varepsilon)OPT(\mathcal{J}) + O(\log m/\varepsilon)$, for any $\varepsilon \in (0, 1]$.

Our contribution. We investigate online makespan minimization in the random-order model, a sensible and widely adopted input model to study algorithms beyond the worst case. Specifically, we develop a new deterministic algorithm that achieves a competitive ratio of 1.8478 as $m \to \infty$. This is the first improved performance guarantee in the random-order model. The competitiveness is substantially below the best known ratio of 1.9201 in the worst-case setting and also below the corresponding lower bound of 1.88 in that framework.

A new feature of our algorithm is that it schedules an incoming job on one of three candidate machines in order to maintain a certain load profile. The best strategies in the worst-case setting use two possible machines, and it is not clear how to take advantage of additional machines in that framework. The choice of our third, extra machine is quite flexible: An incoming job is placed either on a least loaded, a heavily loaded or – as a new option – on an intermediate machine. The latter one is the $(h+1)$-st least loaded machine, where h may be any integer with $h \in \omega(1)$ and $h \in o(\sqrt{m})$.

When assigning a job to a machine different from the least loaded one, an algorithm has to ensure that the resulting makespan does not exceed c times the optimum makespan, for the targeted competitive ratio c. All previous strategies in the literature lower bound the optimum makespan by the current average load on the machines. Our new algorithm works with a refined lower bound that incorporates the processing times of the largest jobs seen so far. The lower bound is obvious but has not been employed by previous algorithms.

The analysis of our algorithm proceeds in two steps. First we define a class of *stable job sequences*. These are sequences that reveal information on the largest jobs as processing volume is scheduled. More precisely, once a certain fraction of the total processing volume $\sum_{t=1}^{n} p_t$ has arrived, one has a good estimate on the h-th largest job and has encountered a certain number of the $m+1$ largest jobs in the input. The exact parameters have to be chosen carefully.

We prove that with high probability, a random permutation of a given input of jobs is stable. We then conduct a worst-case analysis of our algorithm on stable sequences. Using their properties, we show that if the algorithm generates a flat schedule, like *Greedy*, and can be hurt by a huge job, then the input must contain many large jobs so that the optimum makespan is also high. A new ingredient in the worst-case analysis is the processing time of the h-th largest job in the input. We will relate it to machine load in the schedule and to the processing time of the $(m+1)$-st largest job; twice the latter value is a lower bound on the optimum makespan.

The analysis implies that the competitive ratio of 1.8478 holds with high probability. Input sequences leading to higher performance ratios are extremely rare. We believe that our analysis approach might be fruitful in the study of other problems in the random-order model: Identify properties that a random permutation of the input elements satisfies with high probability. Then perform a worst-case analysis.

Finally in this paper we devise lower bounds for the random-order model. We prove that no deterministic online algorithm achieves a competitive ratio smaller than 4/3. Moreover, if a deterministic online algorithm is c-competitive with high probability, then $c \geq 3/2$.

2 Strong competitiveness in the random-order model

We define competitiveness in the random-order model and introduce a stronger measure of competitiveness that implies high-probability bounds. Recall that traditionally a deterministic online algorithm A is c-competitive if $A(\mathcal{J}) \leq c \cdot OPT(\mathcal{J})$ holds for all job sequences $\mathcal{J} = J_1, \ldots, J_n$. We will refer to this worst-case model also as the *adversarial model*.

In the *random-order model* a job sequence $\mathcal{J} = J_1, \ldots, J_n$ is given, which may be specified by an adversary. (Alternatively, a set of jobs could be specified.) Then a random permutation of the jobs arrives. We define the expected cost / makespan of a deterministic online algorithm. Let S_n be the permutation group of the integers from 1 to n, which we consider a probability space under the uniform distribution, i.e. each permutation in S_n is chosen with probability $1/n!$. Given $\sigma \in S_n$, let $\mathcal{J}^\sigma = J_{\sigma(1)}, \ldots, J_{\sigma(n)}$ be the *job sequence permuted by* σ. The expected makespan of A on $\mathcal{J}$ in the random-order model is $A^{\text{rom}}(\mathcal{J}) = \mathbf{E}_{\sigma\sim S_n}[A(\mathcal{J}^\sigma)] = \frac{1}{n!}\sum_{\sigma\in S_n} A(\mathcal{J}^\sigma)$. The algorithm A is *c-competitive in the random-order model* if $A^{\text{rom}}(\mathcal{J}) \le c \cdot OPT(\mathcal{J})$ holds for all job sequences $\mathcal{J}$.

We next define the notion of a deterministic online algorithm A being *nearly c-competitive*. The second condition in the following definition requires that the probability of A not meeting the desired performance ratio must be arbitrarily small as m grows and a random permutation of a given job sequence arrives. The subsequent Lemma 2 states that a nearly c-competitive algorithm is c-competitive in the random-order model.

▶ **Definition 1.** *A deterministic online algorithm A is called* nearly c-competitive *if the following two conditions hold.*

- *The algorithm A achieves a constant competitive ratio in the adversarial model.*
- *For every $\varepsilon > 0$, there exists an $m(\varepsilon)$ such that for all machine numbers $m \ge m(\varepsilon)$ and all job sequences $\mathcal{J}$ there holds $\mathbf{P}_{\sigma\sim S_n}[A(\mathcal{J}^\sigma) \ge (c+\varepsilon)OPT(\mathcal{J})] \le \varepsilon$.*

▶ **Lemma 2.** *If a deterministic online algorithm is nearly c-competitive, then it is c-competitive in the random-order model as $m \to \infty$.*

Proof. Let C be the constant such that A is C-competitive in the adversarial model. We may assume that $C > c$. Given $0 < \delta \le C - c$, we show that there exists an $m(\delta)$ such that, for all $m \ge m(\delta)$, we have $A^{\text{rom}}(\mathcal{J}) \le (c+\delta)OPT(\mathcal{J})$ for every job sequences $\mathcal{J}$. Let $\varepsilon = \delta/(C - c + 1)$. Since A is nearly c-competitive, there exists an $m(\varepsilon)$ such that, for all $m \ge m(\varepsilon)$ and all inputs $\mathcal{J}$, there holds $P_\varepsilon(\mathcal{J}) = \mathbf{P}_{\sigma\sim S_n}[A(\mathcal{J}^\sigma) \ge (c+\varepsilon)OPT(\mathcal{J})] \le \varepsilon$. Set $m(\delta) = m(\varepsilon)$. We obtain

$$\begin{aligned} A^{\text{rom}}(\mathcal{J}) &\le (1 - P_\varepsilon(\mathcal{J}))(c+\varepsilon)OPT(\mathcal{J}) + P_\varepsilon(\mathcal{J}) \cdot C \cdot OPT(\mathcal{J}) \\ &\le ((1-\varepsilon)(c+\varepsilon) + \varepsilon C)OPT(\mathcal{J}) \\ &\le (c + \varepsilon(C - c + 1))OPT(\mathcal{J}) \\ &= (c+\delta)OPT(\mathcal{J}). \end{aligned}$$ ◀

3 Description of the new algorithm

The deficiency of *Greedy* is that it tends to generate a flat, balanced schedule in which all the machines have approximately the same load. An incoming large job can then enforce a high makespan relative to the optimum one. It is thus crucial to try to avoid flat schedules and maintain steep schedules that exhibit a certain load imbalance among the machines.

However, in general, this is futile. Consider a sequence of m identical jobs with a processing time of, say, P_{m+1} (refering to the size of the $(m+1)$-st largest job in an input). Any online algorithm that is better than 2-competitive must schedule these m jobs on separate machines, obtaining the flattest schedule possible. An incoming even larger job of processing time $p_{\max}$ will now enforce a makespan of $P_{m+1} + p_{\max}$. Observe that $\text{OPT} \ge \max\{2P_{m+1}, p_{\max}\}$ since there must be one machine containing two jobs. In particular $P_{m+1} + p_{\max} \le 1.5\text{OPT}$. Hence sensible online algorithms do not perform badly on this sequence.

This example summarizes the quintessential strategy of online algorithms that are good on all sequences: Ensure that in order to create a schedule that is very flat, i.e. such that all machines have high load λ, the adversary must present m jobs that all are large relative to λ. In order to exploit this very flat schedule and cause a high makespan the adversary needs to follow up with yet another large job. But with these $m+1$ jobs, the optimum scheduler runs into the same problem as in the example: Of the $m+1$ large jobs, two have to be scheduled on the same machine. Thus the optimum makespan is high, compensating to the high makespan of the algorithm.

Effectively realizing the aforementioned strategy is highly non-trivial. In fact it is the central challenge in previous works on adversarial makespan minimization that improve upon *Greedy* [1, 4, 14, 15, 22]. These works gave us clear notions of how to avoid flat schedules, which form the basis for our approaches. Instead of simply rehashing these ideas, we want to outline next how we profit from random-order arrival in particular.

3.1 How random-order arrival helps

The first idea to profit from random-order arrival addresses the lower bound on OPT sophisticated online algorithms need. In the literature only the current average load has been considered, but under random-order arrival another bound comes to mind: The largest job seen so far. In order for an algorithm to perform badly, a large job needs to come close to the end of the sequence. Under random-order arrival, it is equally likely for such a job to arrive similarly close to the beginning of the sequence. In this case, the algorithm knows a better lower bound for OPT. The main technical tool will be our *Load Lemma*, which allows us to relate what a job sequence should reveal early from an analysis perspective to the actual fraction of jobs scheduled. This idea does not work for worst-case orders since they tend to order jobs by increasing processing times.

Recall that the general challenge of our later analysis will be to establish that there had to be m large jobs once the schedule gets very flat. In classical analyses, which consider worst-case orders, these jobs appear with increasing density towards the end of the sequence. In random orders this is unlikely, which can be exploited by the algorithm.

The third idea improves upon the first idea. Suppose, that we were to modify our algorithm such that it could handle one very large job arriving close to the end of the sequence. In fact, assume that it could only perform badly when confronted with h very large jobs. We can then disregard any sequence which contains fewer such jobs. Recall that the first idea requires one very large job to arrive sufficiently close to the beginning. Now, as h grows, the probability of the latter event grows as well and approaches 1. This will not only improve our competitive ratio tremendously, it also allows us to adhere to the stronger notion of nearly competitiveness introduced in Section 2. Let us discuss how such a modification is possible: The first step is to design our algorithm in a way that it is reluctant to use the h least loaded machines. Intuitively, if the algorithm tries to retain machines of small load it will require very large jobs to fill them. In order to force these filling jobs to actually be large enough, our algorithm needs to use a very high lower bound for OPT. In fact, here it uses another lower bound for the optimum makespan, $2P^t_{m+1}$, twice the $(m+1)$-st largest job seen so far at time t. Common analysis techniques can only make predictions about P^t_{m+1} at the very end of the sequence. It requires very subtle use of the random-order model to work around this.

3.2 Formal definition

Formally our algorithm ALG is nearly c-competitive, where c is the unique real root of the polynomial $Q[x] = 4x^3 - 14x^2 + 16x - 7$, i.e.

$$c = \frac{7+\sqrt[3]{28-3\sqrt{87}}+\sqrt[3]{28+3\sqrt{87}}}{6} < 1.8478.$$

Given $\mathcal{J}$, ALG is presented with a job sequence/permutation $\mathcal{J}^\sigma = J_{\sigma(1)}, \ldots, J_{\sigma(n)}$ that must be scheduled in this order. Throughout the scheduling process ALG always maintains a list of the machines sorted in non-increasing order of current *load*. At any time the load of a machine is the sum of the processing times of the jobs already assigned to it. After ALG has processed the first t jobs $J_{\sigma(1)}, \ldots, J_{\sigma(t)}$, let $M_1^t, \ldots, M_m^t$ be any ordering of the m machines according to non-increasing load. More specifically, let l_j^t denote the load of machine M_j^t. Then $l_1^t \geq \ldots \geq l_m^t$ and l_1^t is the makespan of the current schedule.

ALG places each incoming job $J_{\sigma(t)}$, $1 \leq t \leq n$, on one of three candidate machines. The choice of one machine, having an intermediate load, is flexible. Let $h = h(m)$ be an integer with $h(m) \in \omega(1)$ and $h(m) \in o(\sqrt{m})$. We could use e.g. $h(m) = \lfloor \sqrt[3]{m} \rfloor$ or $h(m) = \lfloor \log m \rfloor$. Let

$$i = \lceil (2c - 3)m \rceil + h \approx 0.6956m.$$

ALG will assign the incoming job to the machine with the smallest load, the $(h + 1)$-st smallest load or the i-th largest load.

When scheduling a job on a machine that is different from the least loaded one, an algorithm has to ensure that the resulting makespan does not exceed c^* times the optimum makespan, where c^* is the desired competitiveness. All previous algorithms lower bound the optimum makespan by the current average machine load. Algorithm ALG works with a refined lower bound that incorporates the processing time of the largest job and twice the processing time of the $(m + 1)$-st largest job seen so far. These lower bounds on the optimum makespan are immediate but have not been used in earlier strategies.

Formally, for $j = 1, \ldots, m$, let L_j^t be the *average load* of the $m - j + 1$ least loaded machines $M_j^t, \ldots, M_m^t$, i.e. $L_j^t = \frac{1}{m-j+1}\sum_{r=j}^m l_r^t$. We let $L^t = L_1^t = \frac{1}{m}\sum_{s=1}^t p_s$ be the average load of all the machines. For any $j = 1, \ldots, n$, let P_j^t be the processing time of the j-th largest job among the first t jobs $J_{\sigma(1)}, \ldots, J_{\sigma(t)}$ in $\mathcal{J}^\sigma$. If $t < j$, we set $P_j^t = 0$. We let $p_{\max}^t = P_1^t$ be the processing time of the largest job among the first t jobs in $\mathcal{J}^\sigma$. Finally, let $L = L^n$, $P_j = P_j^n$ and $p_{\max} = p_{\max}^n$.

The value $O^t = \max\{L^t, p_{\max}^t, 2P_{m+1}^t\}$ is a common lower bound on the optimum makespan for the first t jobs and hence $OPT(\mathcal{J})$, see Proposition 5 in the next section. Note that immediately before $J_{\sigma(t)}$ is scheduled, ALG can compute L^t and hence O^t because L^t is $1/m$ times the total processing time of the jobs that have arrived so far.

We next characterize load imbalance. Let

$$k = 2i - m \approx (4c - 7)m \approx 0.3912m$$

and

$$\alpha = \frac{2(c-1)}{2c-3} \approx 2.7376.$$

The *schedule at time t* is the one immediately before $J_{\sigma(t)}$ has to be assigned. The schedule is *flat* if $l_k^{t-1} < \alpha L_{i+1}^{t-1}$. Otherwise it is *steep*. Job $J_{\sigma(t)}$ is scheduled *flatly* (*steeply*) if the schedule at time t is flat (steep).

ALG handles each incoming job $J_{\sigma(t)}$, with processing time $p_{\sigma(t)}$, as follows. If the schedule at time t is steep, the job is placed on the least loaded machine M_m^{t-1}. On the other hand, if the schedule is flat, the machines M_i^{t-1}, M_{m-h}^{t-1} and M_m^{t-1} are probed in this order. If $l_i^{t-1} + p_{\sigma(t)} \le c \cdot O^t$, then the new machine load on M_i^{t-1} will not violate the desired competitiveness. The job is placed on this machine M_i^{t-1}. Otherwise, if the latter inequality is violated, *ALG* checks if a placement on M_{m-h}^{t-1} is safe, i.e. if $l_{m-h}^{t-1} + p_{\sigma(t)} \le c \cdot O^t$. If this is the case, the job is put on M_{m-h}^{t-1}. Otherwise, $J_{\sigma(t)}$ is finally scheduled on the least loaded machine M_m^{t-1}. A pseudo-code description of *ALG* is given below. The job assignment rules are also illustrated in Figures 1 and 2.

Algorithm 1 The scheduling algorithm *ALG*.

```
Let $J_{\sigma(t)}$ be the next job to be scheduled.
if the schedule at time $t$ is steep then
    Assign $J_{\sigma(t)}$ to the least loaded machine $M_m^{t-1}$;
else // the schedule is flat
    if $l_i^{t-1} + p_{\sigma(t)} \le c \cdot O^t$ then Assign $J_{\sigma(t)}$ to $M_i^{t-1}$;
    else if $l_{m-h}^{t-1} + p_{\sigma(t)} \le c \cdot O^t$ then Assign $J_{\sigma(t)}$ to $M_{m-h}^{t-1}$;
    else Assign $J_{\sigma(t)}$ to $M_m^{t-1}$;
```

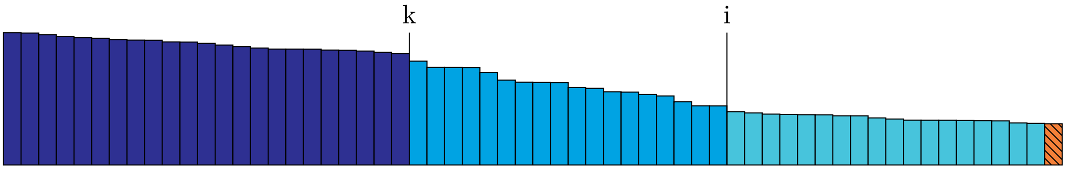


Figure 1 A steep schedule. *ALG* only considers the least loaded machine.

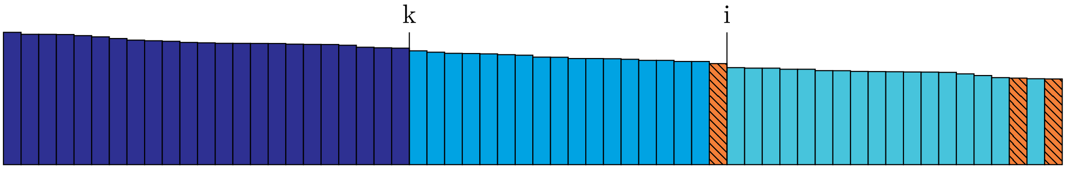


Figure 2 A flat schedule. The three machines considered by *ALG* are marked for $h = 2$.

In the next section we will prove the following theorem, Theorem 3, which uses the notion from Section 2. Lemma 2 then immediately gives the main result, Corollary 4.

▶ **Theorem 3.** *ALG is nearly c-competitive, with $c < 1.8478$ defined as above.*

▶ **Corollary 4.** *ALG is c-competitive in the random-order model as $m \to \infty$.*

From our analysis it can be verified that the number of machines required to be $(c + \varepsilon)$-competitive is bounded by a small polynomial of degree 4 in $1/\varepsilon$. For ease of presentation, we made no optimizations in that regard.

4 Analysis of the algorithm

4.1 Analysis basics

We present some results for the adversarial model so that we can focus on the true random-order analysis of *ALG* in the next sections. First, recall the three common lower bounds used for online makespan minimization.

▶ **Proposition 5.** *For any $\mathcal{J}$, there holds $OPT(\mathcal{J}) \geq \max\{L, p_{\max}, 2P_{m+1}\}$. Moreover, for any permutation J^σ, there holds $O^1 \leq O^2 \leq \ldots \leq O^n \leq OPT(\mathcal{J})$.*

Proof. The optimum makespan $OPT(\mathcal{J})$ cannot be smaller than the average machine load L for the input, even if all the jobs are distributed evenly among the m machines. Moreover, the job with the largest processing time $p_{\max}$ must be scheduled non-preemptively on one of the machines in an optimal schedule. Thus $OPT(\mathcal{J}) \geq p_{\max}$. Finally, among the $m+1$ largest jobs of the input, two must be placed on the same machine in an optimal solution. Hence $OPT(\mathcal{J}) \geq 2P_{m+1}$. For any permutation J^σ, the value O^t cannot decrease as jobs J_t arrive. ◀

For any job sequence $\mathcal{J} = J_1, \ldots, J_n$, let $R(\mathcal{J}) = \min\{\frac{L}{p_{\max}}, \frac{p_{\max}}{L}\}$. Intuitively, this measures the complexity of $\mathcal{J}$.

▶ **Proposition 6.** *For any $\mathcal{J} = J_1, \ldots, J_n$, there holds $ALG(\mathcal{J}) \leq \max\{1+R(\mathcal{J}), c\}OPT(\mathcal{J})$.*

Proof. Let $\mathcal{J} = J_1, \ldots, J_n$ be an arbitrary job sequence and let J_t be the job that defines ALG's makespan. If the makespan exceeds $c \cdot OPT(\mathcal{J})$, then it exceeds $c \cdot O^t$. Thus ALG placed J_t on machine M_m^{t-1}, cf. lines 4 and 5 of the algorithm. This machine was a least loaded one, having a load of at most L. Hence $ALG(\mathcal{J}) \leq L + p_t \leq L + p_{\max} \leq \frac{L+p_{\max}}{\max\{L, p_{\max}\}} \cdot OPT(\mathcal{J}) = (1 + R(\mathcal{J})) \cdot OPT(\mathcal{J})$. ◀

Since $R(\mathcal{J}) \leq 1$ we immediately obtain the following result, which ensures that ALG satisfies the first condition of a nearly c-competitive algorithm, see Definition 1.

▶ **Corollary 7.** *ALG is 2-competitive in the adversarial model.*

We next identify a class of *plain* job sequences that we do not need to consider in the random-order analysis because ALG's makespan is upper bounded by c times the optimum on these inputs.

▶ **Definition 8.** *A job sequence $\mathcal{J} = J_1, \ldots, J_n$ is called* plain *if $n \leq m$ or if $R(\mathcal{J}) \leq c - 1$. Otherwise it is called* proper*.*

Let $\mathcal{J} = J_1, \ldots, J_n$ be any job sequence that is processed/scheduled in this order. Observe that if it contains at most m jobs, i.e. $n \leq m$, and ALG cannot place a job J_t on machines M_i^{t-1} or M_{m-h}^{t-1} because the resulting load would exceed $c \cdot O^t$, then the job is placed on an empty machine. Using Proposition 6 we derive the following fact.

▶ **Lemma 9.** *For any plain job sequence $\mathcal{J} = J_1, \ldots, J_n$, there holds $ALG(\mathcal{J}) \leq c \cdot OPT(\mathcal{J})$.*

If a job sequence $\mathcal{J}$ is plain (proper), then every permutation of it is. Hence, given Lemma 9, we may concentrate on proper job sequences in the remainder of the analysis. We finally state a fact that relates to the second condition of a nearly c-competitive algorithm, see again Definition 1. The proof is given in the full version.

▶ **Lemma 10.** *Let $\mathcal{J} = J_1, \ldots, J_n$ be any job sequence that is scheduled in this order and let J_t be a job that causes ALG's makespan to exceed $(c + \varepsilon)OPT(\mathcal{J})$, for some $\epsilon \geq 0$. Then both the load of ALG's least loaded machine at the time of the assignment as well as p_t exceed $(c - 1 + \varepsilon)OPT(\mathcal{J})$.*

Proof. ALG places J_t on machine M_m^{t-1}, which is a least loaded machine when the assignment is done. If l_m^{t-1} or p_t were upper bounded by $(c - 1 + \varepsilon)OPT(\mathcal{J})$, then the resulting load would be $l_m^{t-1} + p_t \leq (c - 1 + \varepsilon)OPT(\mathcal{J}) + \max\{L, p_t\} \leq (c - 1 + \varepsilon)OPT(\mathcal{J}) + OPT(\mathcal{J}) = (c + \varepsilon)OPT(\mathcal{J})$. ◀

4.2 Stable job sequences

We define the class of stable job sequences. These sequences are robust in that they will admit an adversarial analysis of ALG. Intuitively, the sequences reveal information on the largest jobs when a significant fraction of the total processing volume $\sum_{t=1}^{n} p_t$ has been scheduled. More precisely, one gets an estimate on the processing time of the h-th largest job in the entire sequence and encounters a relevant number of the $m+1$ largest jobs. If a job sequence is unstable, large jobs occur towards the very end of the sequence and can cause a high makespan relative to the optimum one.

We will show that ALG is adversarially $(c+\varepsilon)$-competitive on stable sequences, for a given $\varepsilon > 0$. Therefore, the definition of stable sequences is formulated for a fixed $\varepsilon > 0$. Given $\mathcal{J}$, let $\mathcal{J}^\sigma = J_{\sigma(1)}, \ldots, J_{\sigma(n)}$ be any permutation of the jobs. Furthermore, for every $j \leq n$ and in particular $j \in \{h, m+1\}$, the *set of the j largest jobs* is a fixed set of cardinality j such that no job outside this set has a strictly larger processing time than any job inside the set.

▶ **Definition 11.** *A job sequence $\mathcal{J}^\sigma = J_{\sigma(1)}, \ldots, J_{\sigma(n)}$ is* stable *if the following conditions hold.*

- *There holds $n > m$.*
- *Once $L^t \geq (c-1)\frac{i}{m}L$, there holds $p^t_{\max} \geq P_h$.*
- *For every $j \geq i$, the sequence ending once we have $L^t \geq (\frac{j}{m} + \frac{\varepsilon}{2})L$ contains at least $j+h+2$ many of the $m+1$ largest jobs in $\mathcal{J}$.*
- *The sequence ending right before either (a) $L^t \geq \frac{i}{m}(c-1)\varepsilon L$ holds or (b) the h-th largest job of $\mathcal{J}$ is scheduled contains at least $h+1$ many of the $m+1$ largest jobs in $\mathcal{J}$.*

Otherwise the job sequence is unstable.

Given $\varepsilon > 0$ and m, let $P_\varepsilon(m)$ be the infimum, over all proper job sequences $\mathcal{J}$, that a random permutation of $\mathcal{J}$ is stable, i.e.

$$P_\varepsilon(m) = \inf_{\mathcal{J} \text{ proper}} \mathbf{P}_{\sigma \sim S_n}[\mathcal{J}^\sigma \text{ is stable}].$$

As the main result of this section we will prove that this probability tends to 1 as $m \to \infty$.

▶ **Main Lemma 1.** *For every $\varepsilon > 0$, there holds $\lim_{m \to \infty} P_\varepsilon(m) = 1$.*

The above lemma implies that for any $\varepsilon > 0$ there exists an $m(\varepsilon)$ such that, for all $m \geq m(\varepsilon)$ and all $\mathcal{J}$, there holds $\mathbf{P}_{\sigma \sim S_n}[\mathcal{J}^\sigma \text{ is stable}] \geq 1 - \varepsilon$. In Section 4.3 we will show that ALG is $(c+\varepsilon)$-competitive on stable job sequences. This implies $\mathbf{P}_{\sigma \sim S_n}[ALG(\mathcal{J}^\sigma) \geq (c+\varepsilon)OPT(\mathcal{J})] \leq \varepsilon$. Given Lemma 7, we obtain the following corollary.

▶ **Corollary 12.** *If ALG is adversarially $(c+\varepsilon)$-competitive on stable sequences, for every $\varepsilon > 0$ and $m \geq m(\varepsilon)$ sufficiently large, then it is nearly c-competitive.*

In the remainder of this section we describe how to establish Main Lemma 1. Full proofs of all the lemmas of this section are given in the full version. We need some notation. In Section 3 the value L^t_j was defined with respect to a fixed job sequence that was clear from the context. We adopt the notation $L^t_j[\mathcal{J}^\sigma]$ to make this dependence visible. We adopt a similar notation for the variables L, P^t_j, P_j, $p^t_{\max}$ and $p_{\max}$. For an input $\mathcal{J}$ and $\sigma \in S_n$, we will use the notation $L^t_j[\sigma] = L^t_j[\mathcal{J}^\sigma]$. Again, we use a similar notation for the variables P^t_j and $p^t_{\max}$.

At the heart of the proof of Main Lemma 1 is the Load Lemma. Observe that after t time steps in a random permutation of an input $\mathcal{J}$, each job has arrived with probability t/n. Thus the expected total processing time of the jobs seen so far is $t/n \cdot \sum_{s=1}^{n} p_s$. Equivalently, in expectation L^t equals $t/n \cdot L$. The Load Lemma proves that this relation holds with high probability. We set $t = \varphi n$.

▶ **Load Lemma.** *Given any $\varepsilon > 0$ and $\varphi \in (0,1]$, there exists an $m(\varepsilon, \varphi)$ such that for all $m \geq m(\varepsilon, \varphi)$ and all proper sequences $\mathcal{J}$, there holds*

$$\mathbf{P}_{\sigma \sim S_n}\left[\left|\frac{L^{\lfloor \varphi n \rfloor}[\mathcal{J}^\sigma]}{\varphi L[\mathcal{J}^\sigma]} - 1\right| \geq \varepsilon\right] \leq \varepsilon.$$

Proof sketch. By scaling all job processing times by a common factor we may assume that $p_{\max} = 1$. Then $L = \Theta(m)$ because $\mathcal{J}$ is proper. The main idea of the proof is to show that the variance of the random variable $L^{\lfloor \varphi n \rfloor}[\mathcal{J}^\sigma]$ lies in $O(m) = O(L)$. Using Chebyshev's inequality we show that the probability of $L^{\lfloor \varphi n \rfloor}[\mathcal{J}^\sigma]$ deviating by its expected value φL by more than some term in $\Theta(m^{-1/4})$ is in $O(m^{-1/2})$. The lemma then follows by choosing m sufficiently large. ◀

We note that the Load Lemma does not hold for general sequences. A counterexample is a job sequence in which one job carries all the load, while all the other jobs have a negligible processing time. The proof of the Load Lemma relies on a lower bound of $R(\mathcal{J})$, which is $c - 1$ for proper sequences.

We present two consequences of the Load Lemma that will allow us to prove that stable sequences reveal information on the largest jobs when a certain processing volume has been scheduled. Consider a proper $\mathcal{J}$. Given $\mathcal{J}^\sigma = J_{\sigma(1)}, \ldots, J_{\sigma(n)}$ and $\varphi > 0$, let $N(\varphi)[\mathcal{J}^\sigma]$ be the number of jobs $J_{\sigma(t)}$ that are among the $m+1$ largest jobs in $\mathcal{J}$ and such that $L^t \leq \varphi L$.

▶ **Lemma 13.** *Let $\varepsilon > 0$ and $\varphi \in (0,1]$. Then there holds*

$$\lim_{m \to \infty} \inf_{\mathcal{J} \text{ proper}} \mathbf{P}_{\sigma \sim S_n}\left[N(\varphi + \varepsilon)[\mathcal{J}^\sigma] \geq \lfloor \varphi m \rfloor + h + 2\right] = 1.$$

Proof sketch. The Load Lemma basically matches load ratios L^t/L with ratios t/n on the time line of job arrivals, up to some margin of error. We can then infer that at least $\lfloor \varphi m \rfloor + h + 1$ of the $m+1$ largest jobs are among the first $(\varphi + \varepsilon)n$ jobs in a job sequence $\mathcal{J}^\sigma$, with a probability that tends to 1 as $m \to \infty$. In expectation $(\varphi + \varepsilon)(m+1)$ of the $m+1$ largest jobs occur in this prefix, which is strictly more than $\lfloor \varphi m \rfloor + h + 1$, for m large enough. Formally, we show that (a slight variant of) the random variable $N(\varphi + \varepsilon)[\mathcal{J}^\sigma]$ is hypergeometrically distributed and has variance at most $m+1$. Using Chebyshev's inequality we derive Lemma 13. ◀

▶ **Lemma 14.** *Let $\varepsilon > 0$ and $\varphi \in (0,1]$. Then there holds*

$$\lim_{m \to \infty} \inf_{\mathcal{J} \text{ proper}} \mathbf{P}_{\sigma \sim S_n}\left[\forall_{\tilde{\varphi} \geq \varphi}\ N(\tilde{\varphi} + \varepsilon)[\mathcal{J}^\sigma] \geq \lfloor \tilde{\varphi} m \rfloor + h + 2\right] = 1.$$

Proof sketch. By rounding the values $\tilde{\varphi}$ we may restrict ourselves to finitely many $\tilde{\varphi}$. Using the Union Bound and Lemma 13 we can prove Lemma 14. ◀

Proof sketch for Main Lemma 1. We consider the properties in the definition of stable job sequences. Since $\mathcal{J}$ is proper, there holds $n > m$. By the Load Lemma the second property translates to one of the h largest jobs being among the first $(c-1)\frac{i}{m}n$ jobs in the permuted sequence $\mathcal{J}^\sigma$. The corresponding probability is roughly $1 - (1 - (c-1)\frac{i}{m})^h$ and (quickly) approaches 1 as m and thus h tends to infinity. The third property is a consequence of Lemma 14. For the fourth property we use Lemma 13, considering the sequence ending once $L^t \geq (\frac{j}{m} + \frac{\varepsilon}{2})L$ holds. Finally, the probability of the h-th largest job being preceded by at least $h+1$ of the $m+1$ largest jobs approaches 1 since $h \in o(\sqrt{m})$. Again a full proof is given in the full version. ◀

4.3 An adversarial analysis

In this section we prove the following main result.

▶ **Main Lemma 2.** *For every $\varepsilon > 0$ and $m \geq m(\varepsilon)$ sufficiently large, ALG is adversarially $(c+\varepsilon)$-competitive on stable job sequences.*

Consider a fixed $\varepsilon > 0$. Given Lemma 7, we may assume that $0 < \varepsilon < 2 - c$. Suppose that there was a stable job sequence $\mathcal{J}^\sigma$ such that $ALG(\mathcal{J}^\sigma) > (c+\varepsilon)OPT(\mathcal{J}^\sigma)$. We will derive a contradiction, given that m is large. In order to simplify notation, in the following let $\mathcal{J} = \mathcal{J}^\sigma$ be the stable job sequence violating the performance ratio of $c+\varepsilon$. Let $\mathcal{J} = J_1, \ldots, J_n$ and $OPT = OPT(\mathcal{J})$.

Let $J_{n'}$ be the first job that causes ALG to have a makespan greater than $(c+\varepsilon)OPT$ and let $b_0 = l_m^{n'-1}$ be the load of the least loaded machine $M_m^{n'-1}$ right before $J_{n'}$ is scheduled on it. The makespan after $J_{n'}$ is scheduled, called the *critical makespan*, is at most $b_0 + p_{n'} \leq b_0 + OPT$. In particular $b_0 > (c-1+\varepsilon)OPT$ as well as $p_{n'} > (c-1+\varepsilon)OPT$, see Lemma 10. Let

$$\lambda_{\text{start}} = \tfrac{c-1}{1+2c(2-c)} \approx 0.5426 \quad \text{and} \quad \lambda_{\text{end}} = \tfrac{1}{2(c-1+\varepsilon)} \approx 0.5898.$$

There holds $\lambda_{\text{start}} < \lambda_{\text{end}}$. The critical makespan of ALG is bounded by $b_0 + OPT < (1 + \frac{1}{c-1+\varepsilon})b_0 = (c+\varepsilon)\frac{b_0}{c-1+\varepsilon} = (c+\varepsilon)2\lambda_{\text{end}}b_0$. Since ALG does not achieve a performance ratio of $c+\varepsilon$ on $\mathcal{J}$ we have

$$P_{m+1} \leq OPT/2 < \lambda_{\text{end}}b_0. \tag{1}$$

Our main goal is to derive a contradiction to this inequality.

The impact of the variable P_h

A new, crucial aspect in the analysis of ALG is P_h, the processing time of the h-th largest job in the sequence $\mathcal{J}$. Initially, when the processing of $\mathcal{J}$ starts, we have no information on P_h and can only infer $P_{m+1} \geq \lambda_{\text{start}}b_0$. The second property in the definition of stable job sequences ensures that $p^t_{\max} \geq P_h$ once the load ratio L^t/L is sufficiently large. Note that ALG then also works with this estimate because $P_h \leq p^t_{\max} \leq O^t$. This will allow us to evaluate the processing time of flatly scheduled jobs. In order prove that P_{m+1} is large, we will relate P_{m+1} and P_h, i.e. we will lower bound P_{m+1} in terms of P_h and vice versa. Using the relation we can then conclude $P_{m+1} \geq \lambda_{\text{end}}b_0$. In the analysis we repeatedly use the properties of stable job sequences and will explicitly point to it when this is the case. The omitted proofs of propositions and lemmas are given in the full version of the paper.

We next make the relationship between P_h and P_{m+1} precise. Given $0 < \lambda$, let $f(\lambda) = 2c\lambda - 1$ and given $w > 0$, let $g(w) = (c(2c-3)-1)w + 4 - 2c \approx 0.2854 \cdot w + 0.3044$. We set $g_b(\lambda) = g\left(\frac{\lambda}{b}\right)b$ and $f_b(w) = f\left(\frac{w}{b}\right)b$, for any $b > 0$. Then we will lower bound P_{m+1} by $g_{b_0}(P_h)$ and P_h by $f_{b_0}(P_{m+1})$. We state two technical propositions.

▶ **Proposition 15.** *For $\lambda > \lambda_{\text{start}}$, we have $g(f(\lambda)) > \lambda$.*

▶ **Proposition 16.** *For $0 < \varepsilon \le 1$, we have $g(1-\varepsilon) > \lambda_{\text{end}}$.*

We leave the proof of Proposition 15 to the full version of the paper. Proposition 16 determines the choice of our competitive ratio c. Recall that c is chosen minimal such that $Q[c] = 4c^3 - 14c^2 + 16c - 7 \ge 0$.

Proof of Proposition 16. We calculate that

$$\begin{aligned} g(1-\varepsilon) - \lambda_{\text{end}} &= (c(2c-3)-1)(1-\varepsilon) + 4 - 2c - \frac{1}{2(c-1+\varepsilon)} \\ &= \frac{2(c-1+\varepsilon)(2c^2-5c+3-(2c^2-3c-1)\varepsilon) - 1}{2(c-1+\varepsilon)} \\ &= \frac{4c^3-14c^2+16c-7+(4-2c)\varepsilon - 2(2c^2-3c-1)\varepsilon^2}{2(c-1+\varepsilon)}. \end{aligned}$$

Recall that $Q[c] = 4c^3 - 14c^2 + 16c - 7 = 0$. For $0 < \varepsilon \le 1$ we have

$$(4-2c)\varepsilon - (2c^2-3c-1)\varepsilon^2 \approx 0.3044 \cdot \varepsilon - 0.2854 \cdot \varepsilon^2 > 0.$$

Thus we see that $g(1-\varepsilon) - \lambda_{\text{end}} > 0$ and can conclude the lemma. ◀

4.3.1 Analyzing large jobs towards lower bounding P_h and P_{m+1}

Let $b > (c-1+\varepsilon)OPT$ be a value such that immediately before $J_{n'}$ is scheduled at least $m-h$ machines have a load of at least b. Note that $b = b_0$ satisfies this condition but we will be interested in larger values of b as well. We call a machine *b-full* once its load is at least b; we call a job J a *b-filling job* if it causes the machine it is scheduled on to become b-full. We number the b-filling jobs according to their order of arrival $J^{(1)}, J^{(2)}, \ldots$ and let $t(j)$ denote the time of arrival of the j-th filling job $J^{(j)}$.

Recall that our main goal is to show that $P_{m+1} \ge \lambda_{\text{end}} b_0$ holds. To this end we will prove that the *b_0-filling jobs* have a processing time of at least $\lambda_{\text{end}} b_0$. As there are m such jobs, the bound on P_{m+1} follows by observing that $J_{n'}$ arrives after all *b_0-filling jobs* are scheduled and that its processing time exceeds $\lambda_{\text{end}} b_0$ as well. In fact, since $OPT \ge b_0$, we have

$$p_{n'} > (c-1)OPT > 0.847 \cdot OPT > \lambda_{\text{end}} b_0 \approx 0.5898 \cdot b_0. \tag{2}$$

We remark that different to previous analyses in the literature we do not solely rely on lower bounding the processing time of filling jobs. By using the third property of stable job sequences, we can relate load and the size of the $(m+1)$-st largest job at specific points in the time horizon, cf. Lemma 22.

In the following we regard b as fixed and omit it from the terms *filling job* and *full*. Let $\lambda = \max\{\lambda_{\text{start}} b, \min\{g_b(P_h), \lambda_{\text{end}} b\}\}$. We call a job *large* if it has a processing time of at least λ. Let $\tilde{t} = t(m-h)$ be the time when the $(m-h)$-th filling job arrived. The remainder of this section is devoted to showing the following important Lemma 17. Some of the underlying lemmas, but not all of them, hold if $m \ge m(\varepsilon)$ is sufficiently large. We will make the dependence clear.

▶ **Lemma 17.** *At least one of the following statements holds:*

- *All filling jobs are large.*
- *If $m \ge m(\varepsilon)$, there holds $P^{\tilde{t}}_{m+1} \ge \lambda = \max\{\lambda_{\text{start}} b, \min\{g_b(P_h), \lambda_{\text{end}} b\}\}$, i.e. there are at least $m+1$ large jobs once the $(m-h)$-th filling job is scheduled.*

Before we prove the lemma we derive two important implications towards a lower bound of P_{m+1}.

▶ **Lemma 18.** *We have* $P_{m+1} \geq \lambda = \max\{\lambda_{\text{start}} b_0, \min\{g_{b_0}(P_h), \lambda_{\text{end}} b_0\}\}$.

Proof. Apply the last lemma, taking into account that $b \geq b_0$, and use that there are m many b_0-filling jobs followed by $J_{n'}$. The latter has size at least λ by inequality (2). ◀

We also want to lower bound the processing time of the $(m+1)$-st largest job at time $\tilde{t}$. However, at that time only $m-h$ filling jobs have arrived. The next lemma ensures that, if additionally P_h is not too large, this is not a problem.

▶ **Lemma 19.** *If* $P_h \leq (1-\varepsilon)b$ *and* $m \geq m(\varepsilon)$, *the second statement in Lemma 17 holds, i.e.* $P_{m+1}^{\tilde{t}} \geq \lambda = \max\{\lambda_{\text{start}} b, \min\{g_b(P_h), \lambda_{\text{end}} b\}\}$.

The proof of the lemma makes use of the fourth property of stable job sequences.

We introduce late and early filling jobs. We need a certain condition to hold, see Lemma 22, in order to show that the early filling jobs are large. We show that if this condition is not met, the fact that the given job sequence is stable ensures that $P_m^{\tilde{t}} \geq \lambda$.

Let s be chosen maximal such that the s-th filling job is scheduled steeply. If $s \leq i$, then set $s = i+1$ instead. We call all filling jobs $J^{(j)}$ with $j > i$ that are scheduled flatly *late filling jobs*. All other filling jobs are called *early filling jobs*. In particular the job $J^{(s+1)}$ and the filling jobs afterwards are late filling jobs. The following proposition implies that the fillings jobs after $J^{(m-h)}$, if they exist, are all late, i.e. scheduled flatly.

▶ **Proposition 20.** *We have* $s \leq m-h$ *if* $m \geq m(\varepsilon)$.

We need a technical lemma. For any time t, let $\overline{L}_s^t = \frac{1}{m-h-s+1} \sum_{j=s}^{m-h} l_j^t$ be the average load on the machines numbered s to $m-h$.

▶ **Lemma 21.** *If* $\overline{L}_s^{t(s)-1} \geq \alpha^{-1} b$ *holds and* $m \geq m(\varepsilon)$, *we have* $L^{t(s)-1} > \left(\frac{s}{m} + \frac{\varepsilon}{2}\right) \cdot L$.

▶ **Lemma 22.** *If the late filling jobs are large,* $\overline{L}_s^{t(s)-1} \geq \alpha^{-1} b$ *and* $m \geq m(\varepsilon)$, *we have* $P_{m+1}^{\tilde{t}} \geq \lambda$.

Proof. Assume that the conditions of the lemma hold. By Lemma 21 we have $L^{t(s)-1} > \left(\frac{s}{m} + \frac{\varepsilon}{2}\right) \cdot L$. By the third property of stable sequences, at most $m+1-(s+h+2) = m-s-h-1$ of the largest $m+1$ jobs appear in the sequence starting after time $t(s)-1$. However, this sequence contains $m-h-s$ late filling jobs. Thus there exists a late filling job that is not among the $m+1$ largest jobs. As it has a processing time of at least λ, by the assumption of the lemma, $P_{m+1} \geq \lambda$ holds.

Now consider the $m+1$ largest jobs of the entire sequence that arrive before $J^{(s)}$ as well as the jobs $J^{(s+1)}, \ldots, J^{(m-h)}$. There are at least $s+h+2$ of the former and $m-h-s$ of the latter. Thus we have found a set of at least $m+1$ jobs arriving before (or at) time $\tilde{t} = t(m-h)$. Moreover, we argued that all these jobs have a processing time of at least λ. Hence $P_{m+1}^{\tilde{t}} \geq \lambda$ holds true. ◀

We are ready to evaluate the processing time of filling jobs to prove Lemma 17.

▶ **Lemma 23.** *The processing time of late filling jobs strictly exceeds* $\max\{\lambda_{\text{start}} b, g_b(P_h)\}$.

▶ **Lemma 24.** *If* $\overline{L}_s^{t(s)-1} < \alpha^{-1} b$ *holds, the early filling jobs have a processing time of at least* $\lambda_{\text{end}} b$.

Before proving Lemma 24 let us observe the following, strengthening its condition.

▶ **Lemma 25.** *We have*

$$L_{i+1}^{t(i+1)-1} \leq L_{i+2}^{t(i+2)-1} \leq \dots L_s^{t(s)-1}.$$

Proof. Let $i+1 \leq j < s$. It suffices to verify that

$$L_j^{t(j)-1} \leq L_{j+1}^{t(j)} \leq L_{j+1}^{t(j+1)-1}.$$

The second inequality is obvious because for every r the loads l_r^t can only increase as t increases. For the first inequality we note that by definition the job $J^{(j)}$ was scheduled steeply and hence on a least loaded machine. This machine became full. Thus it is not among the $m-j$ least loaded machines at time $t(j)$. In particular $L_{j+1}^{t(j)}$, the average over the $m-j$ smallest loads at time $t(j)$, is also the average of the $m-j+1$ smallest loads excluding the smallest load at time $t(j)-1$. Therefore it cannot be less than $L_j^{t(j)-1}$. ◀

Proof of Lemma 24. Let $i < j \leq s$ such that $J^{(j)}$ was an early filling job. By Lemma 25 we have $L_j^{t(j)-1} \leq L_s^{t(s)-1} < \alpha^{-1}b = b - \frac{b}{2(c-1)} < b - \lambda_{\text{end}}b$. By definition $J^{(j)}$ was scheduled on a least loaded machine $M_m^{t(j)-1}$ which had load less than $L_j^{t(j)-1} < b - \lambda_{\text{end}}b$ before and at least b afterwards because it became full. In particular $J^{(j)}$ had size $\lambda_{\text{end}}b$.

For $k < j \leq i$ the job $J^{(j)}$ is scheduled steeply because we have by Lemma 25

$$l_k^{t(j)-1} \geq b > \alpha L_s^{t(s)-1} \geq \alpha L_{i+1}^{t(i+1)-1} \geq \alpha L_{i+1}^{t(j)-1}.$$

Thus for $k < j \leq i$ the job $J^{(j)}$ is scheduled on the least loaded machine $M_m^{t(j)-1}$, whose load $l_m^{t(j)-1}$ is bounded by

$$l_m^{t(j)-1} \leq L_{i+1}^{t(j)-1} \leq L_s^{t(s)-1} < \alpha^{-1}b = b - \frac{b}{2(c-1)} < b - \lambda_{\text{end}}b.$$

Hence the job $J^{(j)}$ had a size of at least $\lambda_{\text{end}}b$. We also observe that we have

$$l_i^{t(k)-1} \leq l_{i+1}^{t(k+1)-1} \leq \dots \leq l_{i+(m-i)}^{t(k+(m-i))-1} = l_m^{t(i)-1} < b - \lambda_{\text{end}}b.$$

In particular for $1 \leq j \leq k$ any filling job $J^{(j)}$ filled a machine with a load of at most $\max\{l_m^{t(k)}, l_i^{t(k)}\} = l_i^{t(k)} < b - \lambda_{\text{end}}b$. Hence it had a size of at least $\lambda_{\text{end}}b$. ◀

We now conclude the main lemma of this subsection, Lemma 17.

Proof of Lemma 17. By Lemma 23, all late filling jobs are large. We distinguish two cases depending on whether or not $\overline{L}_s^{t(s)-1} < \alpha^{-1}b$ holds. If it does, all filling jobs are large by Lemma 24 and the first statement in Lemma 17 holds. Otherwise, the second statement in Lemma 17 holds by Lemma 22. ◀

4.3.2 Lower bounding P_h and P_{m+1}

In this section we establish the following relations on P_h and P_{m+1}.

▶ **Lemma 26.** *There holds $P_h > (1-\varepsilon)b_0$ or $P_{m+1} \geq \lambda_{\text{end}}b_0$ if $m \geq m(\varepsilon)$.*

For the proof we need a way to lower bound the processing time of a job J_t depending on P_{m+1}^t:

▶ **Lemma 27.** *Let J_t be any job scheduled flatly on the least loaded machine and let $b = l_{m-h}^{t-1}$ be the load of the $(h+1)$-th least loaded machine. Then J_t has a processing time of at least $f_b(P_{m+1}^t)$.*

Proof. From the fact that J_t was not scheduled on the $(h+1)$-th least loaded machine M^t_{m-h} we derive that $p_t > c \cdot O^t - b \geq c \cdot P^t_{m+1} - b = f_b(P^t_{m+1})$ holds. ◀

Proof of Lemma 26. Assume for a contradiction that we had $P_h \leq (1-\varepsilon)b_0$. Let $J = J_t$ be the smallest among the h last b_0-filling jobs. Then J has a processing time $p \leq P_h$. We want to derive a contradiction to that. Let $b_1 = l^{t-1}_{m-h}$ be the load of the $(m-h)$-th machine right before J was scheduled. Because this machine was b_0-full at that time we know that $b_1 \geq b_0 > (c - 1 + \varepsilon)OPT$ holds and it makes sense to consider b_1-filling jobs. Let $\tilde{t}$ be the time the $(m-h)$-th b_1-filling job arrived. By Lemma 17 we have $P^{\tilde{t}}_{m+1} \geq \lambda = \max\{\lambda_{\text{start}} b_1, \min\{g_{b_1}(P_h), \lambda_{\text{end}} b_1\}\}$.

If we have $\lambda = \lambda_{\text{end}} b_1 \geq \lambda_{\text{end}} b_0$ we have already proven $P_{m+1} \geq \lambda_{\text{end}} b_0$ and the lemma follows. So we are left to treat the case that we have $P^{\tilde{t}}_{m+1} \geq \lambda = \max\{\lambda_{\text{start}} b_1, g_{b_1}(P_h)\}$.

Now we can derive the following contradiction:

$$P^{\tilde{t}}_{m+1} \geq g_{b_1}(P_h) \geq g_{b_1}(p) \geq g_{b_1}\left(f_{b_1}\left(P^{\tilde{t}}_{m+1}\right)\right) = g\left(f\left(\frac{P^{\tilde{t}}_{m+1}}{b_1}\right)\right) b_1 > P^{\tilde{t}}_{m+1}.$$

For the second inequality, we use the monotonicity of $g_{b_1}(-)$. The third inequality follows from Lemma 27 and the last one from Proposition 15. ◀

4.3.3 Establishing Main Lemma 2

Let $m \geq m(\varepsilon)$ be sufficiently large. The machine number $m(\varepsilon)$ is determined by the proofs of Proposition 20 and Lemma 21, and then carries over to the subsequent lemmas. Let us assume for a contradiction sake that there was a stable sequence $\mathcal{J}$ such that $ALG(\mathcal{J}) > (c+\varepsilon)OPT(\mathcal{J})$. As argued in the beginning of Section 4.3, see (1), it suffices to show that $P_{m+1} \geq \lambda_{\text{end}} b_0$. If this was not the case, we would have $P_h \geq (1-\varepsilon)b_0$ by Lemma 26. In particular by Proposition 16 we had $g_{b_0}(P_h) = g(1-\varepsilon)b_0 > \lambda_{\text{end}} b_0$. But now Lemma 18 shows that $P_{m+1} \geq \max\{\lambda_{\text{start}} b_0, \min\{g_{b_0}(P_h), \lambda_{\text{end}} b_0\}\} = \lambda_{\text{end}} b_0$.

We conclude, by Corollary 12, that ALG is nearly c-competitive.

5 Lower bounds

We present lower bounds on the competitive ratio of any deterministic online algorithm in the random-order model. Theorem 29 implies that if a deterministic online algorithm is c-competitive with high probability as $m \to \infty$, then $c \geq 3/2$.

▶ **Theorem 28.** *Let A be a deterministic online algorithm that is c-competitive in the random-order model. Then $c \geq 4/3$ if $m \geq 8$.*

▶ **Theorem 29.** *Let A be a deterministic online algorithm that is nearly c-competitive. Then $c \geq 3/2$.*

A basic family of inputs are job sequences that consist of jobs having an identical processing time of, say, 1. We first analyze them and then use the insight to derive our lower bounds. Let $m \geq 2$ be arbitrary. For any deterministic online algorithm A, let $r(A,m)$ be the maximum number in $\mathbb{N} \cup \{\infty\}$ such that A handles a sequence consisting of $r(A,m) \cdot m$ jobs with an identical processing time of 1 by scheduling each job on a least loaded machine.

▶ **Lemma 30.** *Let $m \geq 2$ be arbitrary. For every deterministic online algorithm A, there exists a job sequence $\mathcal{J}$ such that $A^{\text{rom}}(\mathcal{J}) \geq (1 + \frac{1}{r(A,m)+1})OPT(\mathcal{J})$. We use the convention that $\frac{1}{\infty+1} = 0$.*

Proof. For $r(A,m) = \infty$ there is nothing to show. For $r(A) < \infty$, consider the sequence $\mathcal{J}$ consisting of $(r(A,m)+1) \cdot m$ identical jobs, each having a processing time of 1. It suffices to analyze the algorithm adversarially as all permutations of the job sequence are identical. After having handled the first $r(A,m) \cdot m$ jobs, the algorithm A has a schedule in which every machine has load of $r(A,m)$. By the maximality of $r(A,m)$, the algorithm A schedules one of the following m jobs on a machine that is not a least loaded one. The resulting makespan is $r(A,m)+2$. The lemma follows since the optimal makespan is $r(A,m)+1$. ◀

Proof of Theorem 28. Let $m \geq 8$ be arbitrary. Consider any deterministic online algorithm A. If $r(A,m) \leq 2$, then, by Lemma 30, there exists a sequence $\mathcal{J}$ such that $A^{\text{rom}}(\mathcal{J}) \geq \frac{4}{3} \cdot OPT(\mathcal{J})$. Therefore, we may assume that $r(A,m) \geq 3$. Consider the input sequence $\mathcal{J}$ consisting of $4m-4$ identical small jobs of processing time 1 and one large job of processing time 4. Obviously $\text{OPT}(\mathcal{J}) = 4$.

Let i be the number of small jobs preceding the large job in $\mathcal{J}^\sigma$. The random variable i takes any (integer) value between 0 and $4m-4$ with probability $\frac{1}{4m-3}$. Since $r(A,m) \geq 3$ the least loaded machine has load of at least $l = \left\lfloor \frac{i}{m} \right\rfloor$ when the large job arrives. Thus $A(\mathcal{J}^\sigma) \geq l + 4$. The load l takes the values 0, 1 and 2 with probability $\frac{m}{4m-3}$ and the value 3 with probability $\frac{m-3}{4m-3}$. Hence the expected makespan of algorithm A is at least

$$A^{\text{rom}}(\mathcal{J}) \geq \frac{m}{4m-3} \cdot (0+1+2) + \frac{m-3}{4m-3} \cdot 3 + 4 = \frac{6m-9}{4m-3} + 4 > \frac{16}{3} = \frac{4}{3}\text{OPT}(\mathcal{J}).$$

For the last inequality we use that $m \geq 8$. ◀

Proof of Theorem 29. Let $m \geq 2$ be arbitrary and let A be any deterministic online algorithm. If $r(A,m) = 0$, then consider the sequence $\mathcal{J}$ consisting of m jobs with a processing time of 1 each. On every permutation of $\mathcal{J}$ algorithm A has a makespan of 2, while the optimum makespan is 1. If $r(A,m) \geq 1$, then consider the sequence $\mathcal{J}$ consisting of $2m-2$ small jobs having a processing time of 1 and one large job with a processing time of 2. Obviously $OPT(\mathcal{J}) = 2$. If the permuted sequence starts with m small jobs, the least loaded machine has load 1 once the large job arrives. Under such permutations $A(\mathcal{J}^\sigma) \geq 3 = \frac{3}{2} \cdot \text{OPT}(\mathcal{J})$ holds true. The probability of this happening is $\frac{m-1}{2m-1}$. The probability approaches $\frac{1}{2}$ and in particular does not vanish, for $m \to \infty$. Thus, if A is nearly c-competitive, then $c \geq 3/2$. ◀

References

1 S. Albers. Better bounds for online scheduling. *SIAM Journal on Computing*, 29(2):459–473, 1999. Publisher: SIAM.

2 M. Babaioff, N. Immorlica, D. Kempe, and Robert Kleinberg. A knapsack secretary problem with applications. In *Proc. 10th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 16–28. Springer, 2007.

3 M. Babaioff, N. Immorlica, David Kempe, and R. Kleinberg. Matroid Secretary Problems. *J. ACM*, 65(6):1–26, 2018. Publisher: ACM New York, NY, USA.

4 Y. Bartal, A. Fiat, H. Karloff, and R. Vohra. New algorithms for an ancient scheduling problem. In *Proc. 24th ACM Symposium on Theory of Computing (STOC)*, pages 51–58, 1992.

5 Y. Bartal, H. Karloff, and Y. Rabani. A better lower bound for on-line scheduling. *Inf. Process. Lett.*, 50(3):113–116, 1994.

6 B. Chen, A. van Vliet, and G. Woeginger. A lower bound for randomized on-line scheduling algorithms. *Inf. Process. Lett.*, 51(5):219–222, 1994. Publisher: Elsevier.

7 L. Chen, D. Ye, and G. Zhang. Approximating the optimal algorithm for online scheduling problems via dynamic programming. *Asia-Pacific Journal of Operational Research*, 32(01):1540011, 2015. Publisher: World Scientific.

8 T.C.E. Cheng, H. Kellerer, and V. Kotov. Semi-on-line multiprocessor scheduling with given total processing time. *Theoretical computer science*, 337(1-3):134–146, 2005. Publisher: Elsevier.

9 J. Dohrau. Online makespan scheduling with sublinear advice. In *41st International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM)*, pages 177–188. Springer, 2015.

10 E.B. Dynkin. The optimum choice of the instant for stopping a Markov process. *Soviet Mathematics*, 4:627–629, 1963.

11 M. Englert, D. Özmen, and M. Westermann. The power of reordering for online minimum makespan scheduling. In *Proc. 49th 676 IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 603–612. IEEE, 2008.

12 U. Faigle, W. Kern, and G. Turán. On the performance of on-line algorithms for partition problems. *Acta cybernetica*, 9(2):107–119, 1989.

13 M. Feldman, O. Svensson, and R. Zenklusen. A simple O (log log (rank))-competitive algorithm for the matroid secretary problem. In *Proc. 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1189–1201. SIAM, 2014.

14 R. Fleischer and M. Wahl. On-line scheduling revisited. *Journal of Scheduling*, 3(6):343–353, 2000. Publisher: Wiley Online Library.

15 G. Galambos and G. Woeginger. An on-line scheduling heuristic with better worst-case ratio than Graham's list scheduling. *SIAM Journal on Computing*, 22(2):349–355, 1993. Publisher: SIAM.

16 G. Goel and A. Mehta. Online budgeted matching in random input models with applications to Adwords. In *SODA*, volume 8, pages 982–991, 2008.

17 T. Gormley, N. Reingold, E. Torng, and J. Westbrook. Generating adversaries for request-answer games. In *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, pages 564–565, 2000.

18 R. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45(9):1563–1581, 1966. Publisher: Wiley Online Library.

19 Anupam Gupta, Ruta Mehta, and Marco Molinaro. Maximizing Profit with Convex Costs in the Random-order Model. In *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, volume 107, pages 71:1–71:14, 2018. doi:10.4230/LIPIcs.ICALP.2018.71.

20 D.S. Hochbaum and D.B. Shmoys. Using dual approximation algorithms for scheduling problems theoretical and practical results. *J. ACM*, 34(1):144–162, 1987. Publisher: ACM New York, NY, USA.

21 C. Karande, A. Mehta, and P. Tripathi. Online bipartite matching with unknown distributions. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 587–596, 2011.

22 D.R. Karger, S.J. Phillips, and E. Torng. A better algorithm for an ancient scheduling problem. *Journal of Algorithms*, 20(2):400–430, 1996. Publisher: Elsevier.

23 H. Kellerer and V. Kotov. An efficient algorithm for bin stretching. *Operations Research Letters*, 41(4):343–346, 2013. Publisher: Elsevier.

24 H. Kellerer, V. Kotov, M.G. Grazia Speranza, and Z. Tuza. Semi on-line algorithms for the partition problem. *Operations Research Letters*, 21(5):235–242, 1997. Publisher: Elsevier.

25 C. Kenyon. Best-Fit Bin-Packing with Random Order. In *SODA*, volume 96, pages 359–364, 1996.

26 T. Kesselheim, A. Tönnis, K. Radke, and B. Vöcking. Primal beats dual on online packing LPs in the random-order model. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 303–312, 2014.

27 R.D. Kleinberg. A multiple-choice secretary algorithm with applications to online auctions. In *SODA*, volume 5, pages 630–631, 2005.

28 O. Lachish. O (log log rank) competitive ratio for the matroid secretary problem. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 326–335. IEEE, 2014.

29 M. Mahdian and Q. Yan. Online bipartite matching with random arrivals: an approach based on strongly factor-revealing lps. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 597–606, 2011.

30 A. Meyerson. Online facility location. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pages 426–431. IEEE, 2001.

31 M. Molinaro. Online and random-order load balancing simultaneously. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1638–1650. SIAM, 2017.

32 C.J. Osborn and E. Torng. List's worst-average-case or WAC ratio. *Journal of Scheduling*, 11(3):213–215, 2008. Publisher: Springer.

33 K. Pruhs, J. Sgall, and E. Torng. *Online scheduling.* CRC Press, 2004.

34 J.F. Rudin III. Improved bounds for the on-line scheduling problem, 2001.

35 J.F. Rudin III and R. Chandrasekaran. Improved bounds for the online scheduling problem. *SIAM Journal on Computing*, 32(3):717–735, 2003. Publisher: SIAM.

36 P. Sanders, N. Sivadasan, and M. Skutella. Online scheduling with bounded migration. *Mathematics of Operations Research*, 34(2):481–498, 2009. Publisher: INFORMS.

37 J. Sgall. A lower bound for randomized on-line multiprocessor scheduling. *Inf. Process. Lett.*, 63(1):51–55, 1997. Publisher: Citeseer.

38 D.D. Sleator and R.E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985. Publisher: ACM New York, NY, USA.

Online Algorithms for Weighted Paging with Predictions

Zhihao Jiang[1]
Tsinghua University, Beijing, China
jzh16@mails.tsinghua.edu.cn

Debmalya Panigrahi
Duke University, Durham, NC, USA
debmalya@cs.duke.edu

Kevin Sun
Duke University, Durham, NC, USA
ksun@cs.duke.edu

Abstract

In this paper, we initiate the study of the weighted paging problem with predictions. This continues the recent line of work in online algorithms with predictions, particularly that of Lykouris and Vassilvitski (ICML 2018) and Rohatgi (SODA 2020) on unweighted paging with predictions. We show that unlike unweighted paging, neither a fixed lookahead nor knowledge of the next request for every page is sufficient information for an algorithm to overcome existing lower bounds in weighted paging. However, a combination of the two, which we call the strong per request prediction (SPRP) model, suffices to give a 2-competitive algorithm. We also explore the question of gracefully degrading algorithms with increasing prediction error, and give both upper and lower bounds for a set of natural measures of prediction error.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases Online algorithms, paging

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.69

Category Track A: Algorithms, Complexity and Games

Related Version https://arxiv.org/abs/2006.09509

Funding *Debmalya Panigrahi*: Supported in part by NSF grants CCF-1535972, CCF-1955703, an NSF CAREER Award CCF-1750140, and the Indo-US Virtual Networked Joint Center on Algorithms under Uncertainty.
Kevin Sun: Supported in part by NSF grants CCF-1535972, CCF-1527084, CCF-1955703, and an NSF CAREER Award CCF-1750140.

1 Introduction

The paging problem is among the most well-studied problems in online algorithms. In this problem, there is a set of n pages and a cache of size $k < n$. The online input comprises a sequence of requests for these pages. If the requested page is already in the cache, then the algorithm does not need to do anything. But, if the requested page is not in the cache, then the algorithm suffers what is known as a *cache miss* and must bring the requested page into the cache. If the cache is full, then an existing page must be evicted from the cache to make room for the new page. The goal of the online algorithm is to minimize the total number of cache misses in the *unweighted paging* problem, and the total weight of the evicted pages in the *weighted paging* problem. It is well-known that for both problems, the best deterministic algorithms have a competitive ratio of $O(k)$ and the best randomized algorithms have a competitive ratio of $O(\log k)$ (see, e.g., [4, 2]).

[1] Work done while visiting Duke University.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 69; pp. 69:1–69:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Although the paging problem is essentially solved from the perspective of competitive analysis, it also highlights the limitations of this framework. For instance, it fails to distinguish between algorithms that perform nearly optimally in practice such as the *least recently used* (LRU) rule and very naïve strategies such as *flush when full* that evicts all pages whenever the cache is full. In practice, paging algorithms are augmented with predictions about the future (such as those generated by machine learning models) to improve their empirical performance. To model this, for unweighted paging, several lookahead models have been proposed where only a partial prediction of the future leads to algorithms that are significantly better than what can be obtained in traditional competitive analysis. But, to the best of our knowledge, no such results were previously known for the weighted paging problem. In this paper, we initiate the study of the *weighted paging problem with future predictions*.

For unweighted paging, it is well-known that evicting the page whose next request is *farthest in the future* (also called Belady's rule) is optimal. As a consequence, it suffices for an online algorithm to simply predict the next request of every page (we call this *per request prediction* or PRP in short) in order to match offline performance. In fact, Lykouris and Vassilvitskii [9] (see also Rohatgi [13]) showed recently that in this prediction model, one can simultaneously achieve a competitive ratio of $O(1)$ if the predictions are accurate, and $O(\log k)$ regardless of the quality of the predictions. Earlier, Albers [1] used a different prediction model called *ℓ-strong lookahead*, where we predict a sequence of future requests that includes ℓ distinct pages (excluding the current request). For $\ell = n-1$, this prediction is stronger than the PRP model, since the algorithm can possibly see multiple requests for a page in the lookahead sequence. But, for $\ell < n-1$, which is typically the setting that this model is studied in, the two models are incomparable. The main result in [1] is to show that one can obtain a constant approximation for unweighted paging for $\ell \geq k-2$.

Somewhat surprisingly, we show that neither of these models are sufficient for weighted paging. In particular, we show a lower bound of $\Omega(k)$ for deterministic algorithms and $\Omega(\log k)$ for randomized algorithms in the PRP model. These lower bounds match, up to constants, standard lower bounds for the online paging problem (without prediction) (see, e.g., [11]), hence establishing that the PRP model does not give any advantage to the online algorithm beyond the strict online setting. Next, we show that for ℓ-strong lookahead, even with $\ell = k$, there are lower bounds of $\Omega(k)$ for deterministic algorithms and $\Omega(\log k)$ for randomized algorithms, again asymptotically matching the lower bounds from online paging without prediction. Interestingly, however, we show that a combination of these prediction models is sufficient: if $\ell = n-1$ in the strong lookahead setting, then we get predictions that subsume both models; and, in this case, we give a simple deterministic algorithm with a competitive ratio of 2 for weighted paging, thereby overcoming the online lower bounds.

Obtaining online algorithms with predictions, however, is fraught with the risk that the predictions are inaccurate which renders the analysis of the algorithms useless. Ideally, one would therefore, want the algorithms to also be robust, in that their performance gracefully degrades with increasing prediction error. Recently, there has been significant interest in designing online algorithms with predictions that achieve both these goals, of matching nearly offline performance if the predictions are correct, and of gracefully degrading as the prediction error increases. Originally proposed for the (unweighted) paging problem [9], this model has gained significant traction in the last couple of years and has been applied to problems in data structures [10], online decision making [12, 6], scheduling theory [12, 8], frequency estimation [7], etc. Our final result contributes to this line of research.

First, if the online algorithm and offline optimal solution both use a cache of size k, then we show that no algorithm can asymptotically benefit from the predictions while achieving sublinear dependence on the prediction error. Moreover, if we make the relatively modest assumption that the algorithm is allowed a cache that contains just 1 extra slot than that of the optimal solution, then we can achieve constant competitive ratio when the prediction error is small.

1.1 Overview of models and our results

Our first result is a lower bound for weighted paging in the PRP model. Recall that in the PRP model, in addition to the current page request, the online algorithm is provided the time-step for the next request of the same page. For instance, if the request sequence is $(a, b, a, c, d, b, \ldots)$, then at time-step 1, the algorithm sees request a and is given position 3, and at time-step 2, the algorithm sees request b and is given position 6.

▶ **Theorem 1.** *For weighted paging with PRP, any deterministic algorithm is $\Omega(k)$-competitive, and any randomized algorithm is $\Omega(\log k)$-competitive.*

Note that these bounds are tight, because there exist online algorithms without prediction whose competitive ratios match these bounds (see Chrobak *et al.* [4] and Bansal *et al.* [2]).

Next, for the ℓ-strong lookahead model, we show lower bounds for weighted paging. Recall that in this model, the algorithm is provided a lookahead into future requests that includes ℓ distinct pages. For instance, if $\ell = 3$ and the request sequence is $(a, b, a, c, d, b, \ldots)$, then at time-step 1, the algorithm sees request a and is given the lookahead sequence (b, a, c) since it includes 3 distinct pages. At time step 2, the algorithm sees request b and is given (a, c, d). Note the difference with the PRP model, which would not be give the information that the request in time-step 5 is for page d, but does give the information that the request in time-step 6 is for page b.

▶ **Theorem 2.** *For weighted paging with ℓ-strong lookahead where $\ell \leq n-k$, any deterministic algorithm is $\Omega(k)$-competitive, and any randomized algorithm is $\Omega(\log k)$-competitive.*

For weighted paging with ℓ-strong lookahead where $n-k+1 \leq \ell \leq n-1$, any deterministic algorithm is $\Omega(n-\ell)$-competitive, and any randomized algorithm is $\Omega(\log(n-\ell))$-competitive.

In contrast to these lower bounds, we show that a prediction model that combines features of these individual models gives significant benefits to an online algorithm. In particular, combining PRP and ℓ-strong lookahead, we define the following prediction model:

SPRP ("strong per-request prediction"): On a request for page p, the predictor gives the next time-step when p will be requested *and all page requests till that request.*

This is similar to $(n-1)$-strong lookahead, but is slightly weaker in that it does not provide the first request of every page at the outset. After each of the n pages has been requested, SPRP and $(n-1)$-strong lookahead are equivalent.

▶ **Theorem 3.** *There is a deterministic 2-competitive for weighted paging with SPRP.*

So far, all of these results assume that the prediction model is completely correct. However, in general, predictions can have errors, and therefore, it is desirable that an algorithm gracefully degrades with increase in prediction error. To this end, we also give upper and lower bounds in terms of the prediction error.

For unweighted paging, Lykouris and Vassilvitski [9] basically considered two measures of prediction error. The first, called ℓ_{pd} in this paper, is defined as follows: For each input request p_t, we increase ℓ_{pd} by $w(p_t)$ times the absolute difference between the predicted next-arrival time and the actual next-arrival time. For unweighted paging, Lykouris and Vassilvitskii [9] gave an algorithm with cost $O(\mathsf{OPT} + \sqrt{\ell_{pd} \cdot \mathsf{OPT}})$. Unfortunately, we rule out an analogous result for weighted paging.

▶ **Theorem 4.** *For weighted paging with SPRP, there is no deterministic algorithm whose cost is $o(k)\cdot\mathsf{OPT}+o(\ell_{pd})$, and there is no randomized algorithm whose cost is $o(\log k)\cdot\mathsf{OPT}+o(\ell_{pd})$.*

It turns out that the ℓ_{pd} error measure is closely related to another natural error measure that we call the ℓ_1 measure. This is defined as follows: for each input request p_t, if the prediction q_t is not the same as p_t, then increase ℓ_1 by the sum of weights $w(p_t)+w(q_t)$. (This is the ℓ_1 distance between the predictions and actual requests in the standard weighted star metric space for the weighted paging problem.) The lower bound for ℓ_{pd} continues to hold for ℓ_1 as well, and is tight.

▶ **Theorem 5.** *For weighted paging with SPRP, there is no deterministic algorithm whose cost is $o(k)\cdot\mathsf{OPT}+o(\ell_1)$, and there is no randomized algorithm whose cost is $o(\log k)\cdot\mathsf{OPT}+o(\ell_1)$. Furthermore, there is a deterministic algorithm with SPRP with cost $O(\mathsf{OPT}+\ell_1)$.*

One criticism of both the ℓ_{pd} and ℓ_1 error measures is that they are not robust to insertions or deletions from the prediction stream. To counter this, Lykouris and Vassilvitski [9] used a variant of the classic edit distance measure, and showed a constant competitive ratio for this error measure. For weighted paging, we also consider a variant of edit distance, called ℓ_{ed} and formally defined in Section 5, which allows insertions and deletions between the predicted and actual request streams.[2] Unfortunately, as with ℓ_{pd} and ℓ_1, we rule out algorithms that asymptoticaly benefit from the predictions while achieving sublinear dependence on ℓ_{ed}. Furthermore, if the algorithm were to use a cache with even one extra slot than the optimal solution, then we show that even for weighted paging, we can achieve a constant competitive algorithm. We summarize these results in the next theorem.

▶ **Theorem 6.** *For weighted paging with SPRP, there is no deterministic algorithm whose cost is $o(k)\cdot\mathsf{OPT}+o(\ell_{ed})$, and there is no randomized algorithm whose cost is $o(\log k)\cdot\mathsf{OPT}+o(\ell_{ed})$. In the same setting, there exists a randomized algorithm that uses a cache of size $k+1$ whose cost is $O(\mathsf{OPT}+\ell_{ed})$, where OPT uses a cache of size k.*

1.2 Related work

We now give a brief overview of the online paging literature, highlighting the results that consider a prediction model for future requests. For unweighted paging, the optimal offline algorithm is Belady's algorithm, which always evicts the page that appears farthest in the future [3]. For online paging, Sleator and Tarjan [14] gave a deterministic k-competitive algorithm, and Fiat et al. [5] gave a randomized $O(\log k)$-competitive algorithm; both results were also shown to be optimal. For weighted online paging, Chrobak *et al.* [4] gave a deterministic k-competitive algorithm, and Bansal *et al.* [2] gave an $O(\log k)$-competitive randomized algorithm, which are also optimal by extension.

Recently, Lykouris and Vassilvitskii [9] introduced a prediction model that we call PRP in this paper: on each request p, the algorithm is given a prediction of the next time at which p will be requested. For unweighted paging, they gave a randomized algorithm, based on the "marker" algorithm of Fiat et al. [5], with competitive ratio $O(\min(\sqrt{\ell_{pd}/\mathsf{OPT}},\log k))$. Here, ℓ_{pd} is the absolute difference between the predicted arrival and actual arrival times of requests, summed across all requests. They also perform a tighter analysis yielding a competitive ratio of $O(\min(\eta_{ed}/\mathsf{OPT},\log k))$, where η_{ed} is the edit distance

[2] For technical reasons, neither ℓ_{ed} in this paper nor the edit distance variant in [9] exactly match the classical definition of edit distance.

between the predicted sequence and the actual input. Subsequently, Rohatgi [13] improved the former bound to $O(1 + \min((\ell_{pd}/\mathsf{OPT})/k, 1) \log k)$ and also proved a lower bound of $\Omega(\log \min((\ell_{pd}/\mathsf{OPT})/(k \log k), k))$.

Albers [1] studied the ℓ-strong lookahead model: on each request p, the algorithm is shown the next ℓ distinct requests after p and all pages within this range. For unweighted paging, Albers [1] gave a deterministic $(k - \ell)$-competitive algorithm and a randomized $2H_{k-\ell}$-competitive algorithm. Albers also showed that these bounds are essentially tight: if $l \leq k - 2$, then any deterministic algorithm has competitive ratio at least $k - \ell$, and any randomized algorithm has competitive ratio at least $\Omega(\log(k - \ell))$.

Finally, we review the paging model in which the offline adversary is restricted to a cache of size $h < k$, while the online algorithm uses a larger cache of size k. For this model, Young [16] gave a deterministic algorithm with competitive ratio $k/(k - h + 1)$ and showed that this is optimal. In another paper, Young [15] showed that the randomized "marker" algorithm is $O(\log(k/k - h))$-competitive and this bound is optimal up to constants.

Roadmap

In Section 2, we show the lower bounds stated in Theorem 1 for the PRP model. The lower bounds for the ℓ-strong lookahead model stated in Theorem 2 are proven in Section 3. In Section 4, we state and analyze the algorithm for the SPRP model with no error, thereby proving Theorem 3. Finally, in Section 5, we consider the SPRP model with errors, and focus on the upper and lower bounds in Theorems 4, 5, and 6. Detailed proofs of these bounds appear in the full version of this paper.

2 The Per-Request Prediction Model (PRP)

In this section, we give the lower bounds stated in Theorem 1 for the PRP model. Our strategy, at a high level, will be the same in both the deterministic and randomized cases: we consider the special case where the cache size is exactly one less than the number of distinct pages. We then provide an algorithm that generates a specific input. In the deterministic case, this input will be adversarial, based on the single page not being in the cache at any time. In the randomized case, the input will be oblivious to the choices made by the paging algorithm but will be drawn from a distribution. We will give a brief overview of the main ideas that are common to both lower bound constructions first, and then give the details of the randomized construction in this section. The details of the deterministic construction are deferred to the full paper.

Let us first recall the $\Omega(k)$ deterministic lower bound for unweighted caching without predictions. Suppose the cache has size k and the set of distinct pages is $\{a_0, a_1, \ldots, a_k\}$. At each step, the adversary requests the page a_ℓ not contained in the cache of the algorithm ALG. Then ALG incurs a miss at every step, while OPT, upon a miss, evicts the page whose next request is furthest in the future. Therefore, ALG misses at least k more times before OPT misses again.

Ideally, we would like to imitate this construction. But, the adversary cannot simply request the missing page a_ℓ because that could violate the predictions made on previous requests. Our first idea is to replace this single request for a_ℓ with a "block" of requests of pages *containing* a_ℓ in a manner that all the previous predictions are met, but ALG still incurs the cost of page a_ℓ in serving this block of requests.

But, how do we guarantee that OPT only misses requests once for every k blocks? Indeed, it is not possible to provide such a guarantee. Instead, as a surrogate for OPT, we use an array of k algorithms ALG_i for $1 \leq i \leq k$, where each ALG_i follows a fixed strategy: maintain all pages except a_0 and a_i permanently in the cache, and swap a_0 and a_i as required to serve their requests. Our goal is to show that the sum of costs of all these algorithms is a lower bound (up to constants) on the cost of ALG; this would clearly imply an $\Omega(k)$ lower bound.

This is where the weights of pages come handy. We set the weight $w(a_i)$ of page a_i in the following manner: $w(a_i) = c^i$ for some constant $c \geq 2$. Now, imagine that a block requested for a missing page a_ℓ only contains pages $a_0, a_1, \ldots, a_\ell$ (we call this an ℓ-block). The algorithms ALG_i for $i \leq \ell$ suffer a cache miss on page a_i in this block, while the remaining algorithms ALG_i for $i > \ell$ do not suffer a cache miss in this block. Moreover, the sum of costs of all the algorithms ALG_i for $i \leq \ell$ in this block is at most a constant times that of the cost of ALG alone, because of the geometric nature of the cost function.

The only difficulty is that by constructing blocks that do not contain pages a_i for $i > \ell$, we might be violating the previous predictions for these pages. To overcome this, we create an invariant where for every i, an $(i+1)$-block must be introduced after a fixed number of i-blocks. Because of this invariant, we are sometimes forced to introduce a larger block than that demanded by the missing page in ALG. To distinguish between these two types of blocks, we call the ones that exactly correspond to the missing page a *regular* block, and the ones that are larger *irregular* blocks. Irregular blocks help preserve the correctness of all previous predictions, but the sum of costs of ALG_i's on an irregular block can no longer be bounded against that of ALG. Nevertheless, we can show that the number of irregular blocks is small enough that this extra cost incurred by ALG_i's in irregular blocks can be charged off to the regular blocks, thereby proving the deterministic lower bound:

▶ **Theorem 7.** *For weighted paging with PRP, any deterministic algorithm is $\Omega(k)$-competitive.*

A formal proof of this theorem is deferred to the full paper. Instead, we focus on proving the lower bound for randomized algorithms.

2.1 Randomized Lower Bound

This subsection is devoted to proving the following theorem:

▶ **Theorem 8.** *For weighted paging with PRP, any randomized algorithm is $\Omega(\log k)$-competitive.*

Here, we still use the same idea of request blocks, but now the input is derived from a fixed distribution and is not aware of the state of ALG. The main idea is to design a distribution over block sizes in a manner that still causes any fixed deterministic algorithm ALG to suffer a large cost *in expectation*, and then invoke Yao's minimax principle to translate this to a randomized lower bound.

Let $H_k = 1 + 1/2 + \cdots + 1/k \approx \ln k$ denote the k-th harmonic number. The input is defined as follows:

```
1. For $0 \leq i \leq k$, set $u_i = (2ckH_k + 2)^i$ and let $y_i = 0$ for $i < k$.
2. Repeat the following:
   a. Select a value of $\ell$ according to the following probability distribution: $\Pr[\ell = j] = \frac{c-1}{c^{j+1}}$
      for $j \in \{0, 1, \ldots, k-1\}$ and $\Pr[\ell = k] = \frac{1}{c^k}$.
   b. Increase $\ell$ until $\ell = k$ or $y_\ell < 2ckH_k$.
   c. For $j$ from 0 to $\ell$,
      i. Set all requests from time $t + 1$ through $u_j - 1$ as $a_{j-1}$. (Note: If $j = 0$, then
         $u_j = t + 1$, so this step is empty.)
      ii. Set the request at time $u_j$ as $a_j$.
      iii. Let $t = u_j$.
```

d. For $0 \leq j \leq \ell$, let $u_j = t + (2ckH_k + 2)^j$.
e. For $0 \leq j < \ell$, let $y_j = 0$. If $\ell < k$, increase y_ℓ by one.

Note that if ℓ is not increased in Step 2b, then this block is *regular*; otherwise, it is *irregular*. Let v_i denote the number of regular i-blocks, and let v'_i denote the number of irregular i-blocks. A j-block is an *i-plus* block if and only if $j \geq i$. We first lower bound the cost of ALG by the number of blocks.

▶ **Lemma 9.** *Every requested block increases $\mathbb{E}\left[\mathsf{cost}(\mathsf{ALG})\right]$ by at least a constant.*

Proof. At every time step, the cache of ALG is missing some page a_j. The probability that a_j is requested in the next block is at least $\Pr[\ell = j] \geq \frac{1}{2c^j}$, so the expected cost of serving this block is at least $c^j \cdot \Pr[\ell = j] = \Omega(1)$. ◀

For the rest of the proof, we upper bound the cost of OPT. We first upper bound the number of regular blocks, and then we use this to bound the number of irregular blocks.

▶ **Lemma 10.** *For every $i \in \{0, 1, \ldots, k\}$, we have $\mathbb{E}\left[v_i\right] \leq 2c^{-i}m$.*

Proof. Consider the potential function $\phi(y) = \sum_{i=0}^{k-1} y_i \geq 0$. The initial value of $\phi(y)$ is 0. Notice that whenever a regular block is generated, $\phi(y)$ increases by at most 1, and whenever an irregular block is generated, $\phi(y)$ decreases by at least $2ckH_k$. Thus, the number of irregular blocks is at most the number of regular blocks, so the total number of blocks is at most $2m$. The lemma follows by noting that the probability that a block is a regular i-block is at most c^{-i}. ◀

▶ **Lemma 11.** *For every $i \in \{0, 1, \ldots, k\}$, we have $\mathbb{E}\left[v'_i\right] \leq \frac{2m}{c^i kH_k}$.*

Proof. Observe that $v'_i \leq \frac{1}{2ckH_k}(v'_{i-1} + v_{i-1})$ and $v'_1 \leq \frac{1}{2ckH_k} v_0$. Repeatedly applying this inequality yields

$$\mathbb{E}\left[v'_i\right] \leq \sum_{j=0}^{i-1} \frac{\mathbb{E}\left[v_j\right]}{(2ckH_k)^{i-j}} \leq \sum_{j=0}^{i-1} \frac{2c^{-j}m}{(2ckH_k)^{i-j}} = \frac{2m}{c^i} \sum_{j=0}^{i-1} \frac{1}{(2kH_k)^{i-j}} \leq \frac{2m}{c^i kH_k},$$

where the second inequality holds due to Lemma 10. ◀

Now let A denote the entire sequence of requests, B the subsequence of A comprising all regular blocks, and m the number of blocks in B. We bound $\mathsf{OPT} = \mathsf{OPT}(A)$ in terms of the optimal cost on B and the number of irregular blocks.

▶ **Lemma 12.** *Let $\mathsf{OPT}(A)$ and $\mathsf{OPT}(B)$ denote the optimal offline algorithm on request sequences A and B respectively. Then $\mathsf{cost}(\mathsf{OPT}(A)) \leq \mathsf{cost}(\mathsf{OPT}(B)) + 4c \sum_{i=0}^{k} v'_i c^i$.*

Proof. Consider the following algorithm ALG_A on request sequence A:

1. For requests in regular blocks, imitate $\mathsf{OPT}(B)$. That is, copy the cache contents when $\mathsf{OPT}(B)$ serves this block.
2. Upon the arrival of an irregular i-block, let a_ℓ denote the page not in the cache.
   a. If $\ell > i$, then the cost of serving this block is 0.
   b. If $1 \leq \ell \leq i$, evict a_0 when a_ℓ is requested. Then evict a_ℓ and fetch a_0 at the end of this block; the cost of this is $2(c^i + 1)$.
   c. If $\ell = 0$, we evict a_1 and fetch a_0 when a_0 is requested. Then we evict a_0 and fetch a_1 when a_1 is requested or at the end of this block (if a_1 is not requested in this block). The cost is $2(c + 1)$.

For each irregular block, notice that the cache of ALG_A is the same at the beginning and the end of the block. So Step 2 does not influence the imitation in Step 1. The cost of serving an irregular i-block is at most $4c^{i+1}$. Combining these facts proves the lemma. ◀

To bound $\mathsf{OPT}(B)$, we divide the sequence B into phases. Each phase is a contiguous sequence of blocks. Phases are defined recursively, starting with 0-phases all the way through to k-phases. A 0-phase is defined as a single request. For $i \geq 1$, let M_i denote the first time that an i-plus-block is requested and let Q_i denote the first time that c $(i-1)$-phases have appeared. An i-phase ends immediately after M_i and Q_i have both occurred. In other words, an i-phase is a minimal contiguous subsequence that contains c $(i-1)$-phases and an i-plus block. (Notice that for a fixed i, the set of i-phases partition the input sequence.)

For any k-phase, we upper bound OPT by considering an algorithm ALG_B^k that is optimal for B subject to the additional restriction that a_0 is not in the cache at the beginning or end of any k-phase. We bound the cost of ALG_B^k in any k-phase using a more general lemma.

▶ **Lemma 13.** *For any i, let ALG_B^i be an optimal algorithm on B subject to the following: a_0 is not in the cache at the beginning or the end of any i-phase. Then the cost of ALG_B^i within an i-phase is at most $4c^{i+1}$. In particular, in each k-phase, the algorithm ALG_B^k incurs cost at most $4c^{k+1}$.*

Proof. We shall prove this by induction on i. If $i = 0$, then the phase under consideration is one step. To serve one step, we can evict a_1 to serve a_0, and then evict a_0 if necessary for a total cost of $4c$. Now assume that the lemma holds for all values in $\{0, \dots, i-1\}$. Let s_i denote the first i-plus block; there are two possible cases for the structure of an i-phase:

1. s_i appears after the c $(i-1)$-phases: In this case, the i-phase ends after this block. Thus, one strategy to serve the phase is to evict a_i at the beginning and evict a_0 when a_i is requested within s_i. These two evictions cost at most $4c^{i+1}$.
2. s_i appears within the first c $(i-1)$-phases: By the inductive hypothesis, the algorithm can serve these c $(i-1)$-phases with total cost at most $c \cdot 4c^i = 4c^{i+1}$. ◀

Finally, we lower bound the expected number of blocks in an i-phase. Since the total number of blocks is fixed, this allows us to upper bound the number of k-phases in the entire sequence. The next proposition forms the technical core of the lower bound:

▶ **Proposition 14.** *For $i \geq 1$, the expected number of blocks in an i-phase is at least $c^i H_i/4$.*

We defer the proof of Proposition 14 to the end of this section; first, we use it to prove Theorem 8.

Proof of Theorem 8. Let $\mathsf{OPT}(A)$ denote the cost of an optimal algorithm on the request sequence A, and let $\mathsf{OPT}(B)$ denote the cost of an optimal algorithm on the regular blocks B. Then we have the following:

$$\begin{aligned}
\mathbb{E}\left[\mathsf{cost}(\mathsf{OPT}(A))\right] &\leq \mathbb{E}\left[\mathsf{cost}(\mathsf{OPT}(B))\right] + 4c\sum_{i=0}^{k} c^i \cdot \mathbb{E}\left[v_i'\right] && \text{(Lemma 12)} \\
&\leq \mathbb{E}\left[\mathsf{cost}(\mathsf{ALG}_B^k)\right] + 4c\sum_{i=0}^{k} c^i \cdot \frac{2m}{c^i k H_k} && \text{(Lemma 11)} \\
&\leq 4c^{k+1} \cdot \mathbb{E}\left[N_k(B)\right] + \frac{16cm}{H_k}, && \text{(Lemma 13)}
\end{aligned}$$

where $N_k(B)$ denotes the number of k-phases in B. According to Proposition 14, the expected number of blocks in a k-phase is at least $c^k H_k/4$, which implies $\mathbb{E}[N_k(B)] \leq \frac{4m}{c^k H_k}$. Combining this with the above, we get

$$\mathbb{E}[\mathsf{cost}(\mathsf{OPT}(A))] \leq \frac{16cm}{H_k} + \frac{16cm}{H_k} = O\left(\frac{m}{H_k}\right).$$

Since any algorithm incurs at least some constant cost in every block by Lemma 9, its cost is $\Omega(m)$, which concludes the proof. ◀

Proof of Proposition 14

Let z_i be a random variable denoting the number of i-plus blocks in a fixed i-phase. We will first prove a sequence of three lemmas to yield a lower bound on $\mathbb{E}[z_i]$.

▶ **Lemma 15.** *For any $i \geq 1$, we have $\mathbb{E}[z_i] = \mathbb{E}[z_{i-1}] + \Pr\{M_i > Q_i\}$.*

Proof. Recall that an i-phase ends once it contains c $(i-1)$-phases and an i-plus block. In each of the $(i-1)$-phases, the expected number of $(i-1)$-plus blocks is $\mathbb{E}[z_{i-1}]$, so the total expected number of $(i-1)$-plus blocks in the first c $(i-1)$-phases of an i-phase is $c \cdot \mathbb{E}[z_{i-1}]$.

An elementary calculation shows that an $(i-1)$-plus block is an i-plus block with probability $1/c$. Thus, in expectation, the first c $(i-1)$-phases of this i-phase contain $\mathbb{E}[z_{i-1}]$ i-plus blocks.

If there are no i-plus blocks in the first c $(i-1)$-phases, then the i-phase ends as soon as an i-plus block appears. In this case, we have $z_i = 1$, and this happens with probability exactly $\Pr\{M_i > Q_i\}$. Otherwise, the i-phase ends immediately after the c $(i-1)$-phases, in which case no additional term is added. ◀

▶ **Lemma 16.** *For any $i \geq 1$, we have $\Pr\{M_i > Q_i\} \geq e^{-2\mathbb{E}[z_{i-1}]}$.*

Proof. We let $v_1, \ldots, v_c$ denote the number of i-plus blocks in the first c $(i-1)$-phases and let $V = \sum_{i=1}^{c} v_i$. As we saw in the proof of Lemma 15, an $(i-1)$-plus block is an i-plus block with probability $1/c$, so the probability that an $(i-1)$-plus block is an $(i-1)$-block is $1 - 1/c$. Thus, we have

$$\Pr\{M_i > Q_i\} = \mathbb{E}_{v_1, v_2, \ldots, v_c}\left[\left(1 - \frac{1}{c}\right)^V\right] \geq \left(1 - \frac{1}{c}\right)^{\mathbb{E}[V]} = \left(1 - \frac{1}{c}\right)^{c \cdot \mathbb{E}[z_{i-1}]}$$

where the inequality follows from convexity and the second equality holds due to linearity of expectation. The lemma follows from this and the fact that $c \geq 2$. ◀

▶ **Lemma 17.** *For any $i \geq 0$, we have $\mathbb{E}[z_i] \geq \frac{1}{4}H_i$.*

Proof. When $i \leq 4$, we have $\mathbb{E}[z_i] \geq 1 \geq \frac{1}{4}H_i$. Now for induction, assume the statement holds for $j < i$, and consider the two possible cases:

1. If $\mathbb{E}[z_{i-1}] \geq \frac{1}{2}H_{i-1}$, then Lemma 15 implies $\mathbb{E}[z_i] \geq \mathbb{E}[z_{i-1}] \geq \frac{1}{4}H_i$.
2. If $\mathbb{E}[z_{i-1}] < \frac{1}{2}H_{i-1} < \frac{1}{2}(1 + \ln(i-1))$, then $\mathbb{E}[z_i] = \mathbb{E}[z_{i-1}] + \Pr\{M_i > Q_i\} \geq \frac{1}{4}H_{i-1} + e^{-2\cdot\mathbb{E}[z_{i-1}]}$, where the equality follows from Lemma 15 and the inequality holds by the induction hypothesis and Lemma 16. Thus, $\mathbb{E}[z_i] \geq \frac{1}{4}H_{i-1} + \frac{1}{e} \cdot \frac{1}{i-1} \geq \frac{1}{4}H_i$. ◀

Now let L_i denote the number of blocks in an i-phase; recall that our goal is to lower bound its expectation by $c^i H_i/4$. The following lemma relates L_i to z_i.

▶ **Lemma 18.** *For any $i \geq 0$, we have $\mathbb{E}[L_i] = c^i \cdot \mathbb{E}[z_i]$.*

Proof. When $i = 0$, the lemma holds because $E[L_0] = E[z_0] = 1$, so now we assume $i \geq 1$. Recall that an i-phase contains at least c $(i-1)$-phases, so the expected total number of blocks in the first c $(i-1)$-phases of this i-phase is $c \cdot \mathbb{E}[L_{i-1}]$.

If there are no i-plus-blocks in these c $(i-1)$-phases, we need to wait for an i-plus block to appear in order for the i-phase to end. This is a geometric random variable with expectation c^i. Thus, we have: $\mathbb{E}[L_i] = c \cdot \mathbb{E}[L_{i-1}] + c^i \cdot \Pr\{M_i > Q_i\}$. Applying this recursively,

$$\mathbb{E}[L_i] = c^i \left(\sum_{j=1}^{i} \Pr\{M_j > Q_j\} + \mathbb{E}[L_0] \right) = c^i \left(\sum_{j=1}^{i} \Pr\{M_j > Q_j\} + 1 \right)$$

Furthermore, from Lemma 15, we have

$$\mathbb{E}[z_i] = \mathbb{E}[z_{i-1}] + \Pr\{M_i > Q_i\} = \mathbb{E}[z_0] + \sum_{j=1}^{i} \Pr\{M_j > Q_j\} = 1 + \sum_{j=1}^{i} \Pr\{M_j > Q_j\}.$$

Combining the two equalities yields the lemma. ◀

We conclude by proving Proposition 14. Fix some $i \geq 1$. Using Lemma 18 and Lemma 17, we get $\mathbb{E}[L_i] = c^i \cdot \mathbb{E}[z_i] \geq \frac{c^i H_i}{4}$.

3 The ℓ-Strong Lookahead Model

Now we consider the following prediction model: at each time t, the algorithm can see request p_t as well as $L(t)$, which is the set of all requests through the ℓ-th distinct request. In other words, the algorithm can always see the next contiguous subsequence of ℓ distinct pages (excluding p_t) for a fixed value of ℓ. This model was introduced by Albers [1], who (among other things) proved the following lower bounds on algorithms with ℓ-strong lookahead.

▶ **Lemma 19** ([1]). *For unweighted paging with ℓ-strong lookahead where $\ell \leq k-2$, any deterministic algorithm is $\Omega(k-\ell)$-competitive. For randomized algorithms, the bound is $\Omega(\log(k-\ell))$.*

Notice that Lemma 19 implies that for small values of ℓ, ℓ-strong lookahead provides no asymptotic improvement to the competitive ratio of any algorithm. The proof proceeds by constructing a particular sequence of requests and analyzing the performance of any algorithm on this sequence. By slightly modifying the sequence, we can prove a similar result for the weighted paging problem.

▶ **Theorem 20.** *For weighted paging with ℓ-strong lookahead where $n-k+1 \leq \ell \leq n-1$, any deterministic algorithm is $\Omega(n-\ell)$-competitive, and any randomized algorithm is $\Omega(\log(n-\ell))$-competitive.*

Proof. We modify the adversarial input in Lemma 19 as follows: insert $n-k-1$ distinct pages with very low weight between every two pages. This causes the lookahead to have effective size $\ell' = \ell - (n-k-1)$, because at any point $L(t)$ contains at most ℓ' pages with normal weight. Note that if $\ell \leq n-k$, then $\ell' \leq 1$, and from Lemma 19, a lookahead of size 1 provides no asymptotic benefit to any algorithm.

If $\ell \leq n-3$, then $\ell' \leq k-2$. Thus, we can apply Lemma 19 to conclude that for any deterministic algorithm, the competitive ratio is $\Omega(k-\ell') = \Omega(n-\ell-1)$, and for any randomized algorithm, the competitive ratio is $\Omega(\log(n-\ell-1))$. Otherwise, if $\ell \geq n-2$, then the lower bounds continue to hold because when $\ell = n-3$, they are $\Omega(1)$. ◀

4 The Strong Per-Request Prediction Model (SPRP)

In this section, we define a simple algorithm called STATIC that is 2-competitive when the SPRP predictions are always correct. At any time step t, let $L(t)$ denote the set of pages in the current prediction. The STATIC algorithm runs on "batches" of requests. The first batch starts at $t = 1$ and comprises all requests in $L(1)$. The next batch starts once the first batch ends, i.e. at $|L(1)| + 1$, and comprises all predicted requests at that time, and so on. Within each batch, the STATIC algorithm runs the optimal offline strategy, computed at the beginning of the batch on the entire set of requests in the batch.

▶ **Theorem 21.** *The STATIC algorithm is 2-competitive when the predictions from SPRP are entirely correct.*

Proof. In this proof, we assume w.l.o.g. that evicting page p costs $w(p)$, and fetches can be performed for free.

Suppose the algorithm runs a total of m batches $B_1, \dots, B_m$. Consider a page p in some batch B_i where $i < m$. If p appears again after B_i, then upon seeing the last request for p in B_i, SPRP will include p in the next batch B_{i+1}. (If p does not appear again, then the next batch must be the last batch.) Therefore, the batches satisfy $B_1 \subseteq B_2 \subseteq \cdots \subseteq B_{m-1}$.

Now let OPT denote a fixed optimal offline algorithm for the entire sequence, and let OPT_i denote the cost of OPT incurred in B_i. Similarly, let S denote the total cost of STATIC, and let S_i denote the cost that STATIC incurs in B_i. So we have $\mathsf{OPT} = \sum_{i=1}^{m} \mathsf{OPT}_i$ and $S = \sum_{i=1}^{m} S_i$.

Fix a batch index $j \in \{2, 3, \dots, m\}$ and let $C(\mathsf{OPT}_{j-1})$ and $C(S_{j-1})$ denote the cache states of OPT and STATIC immediately before batch B_j. We know that STATIC runs an optimal offline algorithm on B_j. One feasible solution is to immediately change the cache state to $C(\mathsf{OPT}_{j-1})$, and then imitate what OPT does to serve B_j. Since we charge for evictions, we have

$$S_j \leq \mathsf{OPT}_j + \sum_{p \in C(S_{j-1}) \setminus C(\mathsf{OPT}_{j-1})} w(p), \text{ for every } j \in \{2, 3, \dots, m\}.$$

Consider some $p \in C(S_{j-1}) \setminus C(\mathsf{OPT}_{j-1})$: since $p \in C(S_{j-1})$, we know p must have appeared before the start of B_j (because STATIC does not fetch pages that have never been requested). Since B_{j-1} contains all pages that appeared before, in particular, p must be in B_{j-1}. Furthermore, since $p \notin C(\mathsf{OPT}_{j-1})$, then at some point while serving B_{j-1}, OPT must have evicted p. Thus, $S_j \leq \mathsf{OPT}_j + \mathsf{OPT}_{j-1}$. Summing over all $j \geq 2$ and $S_1 \leq \mathsf{OPT}_1$ proves the theorem. ◀

5 The SPRP Model with Prediction Errors

In this section, we consider the SPRP prediction model with the possibility of prediction errors. We first define three measurements of error and then prove lower and upper bounds on algorithms with imperfect SPRP, in terms of these error measurements.

Let A denote a prediction sequence of length m, and let B denote an input sequence of length n. For any time t, let A_t and B_t denote the t-th element of A and B, respectively. We also define the following for any time step t:

- $\mathsf{prev}(t)$: The largest $i < t$ such that $B_i = B_t$ (or 0 if no such if no such i exists).
- $\mathsf{next}(t)$: The smallest $i > t$ such that $B_i = B_t$ (or $n + 1$ if no such i exists).
- $\mathsf{pnext}(t)$: The smallest $i > t$ such that $A_i = B_t$ (or $m + 1$ if no such i exists).
- We say two requests $A_i = B_j = p$ can be *matched* only if $\mathsf{pnext}(\mathsf{prev}(j)) = i$. In other words, A_i must be the earliest occurrence of p in A after the time of the last p in B before B_j. Furthermore, no edges in a matching are allowed to cross.

First, we define a variant of edit distance between the two sequences.

▶ **Definition 22.** *The* edit distance ℓ_{ed} *between A and B is the total minimum weight of unmatched elements of A and B.*

Next, we define an error measure based on the metric 1-norm distance between corresponding requests on the standard weighted star metric denoting the weighted paging problem.

▶ **Definition 23.** *The* 1-norm distance ℓ_1 *between A and B is defined as follows:*

$$\ell_1 = \sum_{\substack{i=1 \\ A_i \neq B_i}}^{n} \left(w(A_i) + w(B_i) \right). \tag{1-norm}$$

Third, we define an error measure inspired by the PRP model that was also used in [9].

▶ **Definition 24.** *The* prediction distance ℓ_{pd} *between A and B is defined as follows:*

$$\ell_{pd} = \sum_{i=1}^{n} w(B_i) \cdot |\mathsf{next}(i) - \mathsf{pnext}(i)|.$$

5.1 Lower Bounds

In this section, we give an overview of the lower bounds stated in Theorems 4, 5, and 6. We focus on the ℓ_{ed} (i.e., Theorem 6) error measurement; the proofs for ℓ_1 and ℓ_{pd} follow similarly. We defer some of the proofs to the full paper.

Our high-level argument proceeds as follows: recall that in Section 2, we showed a lower bound of $\Omega(k)$ on the competitive ratio of deterministic PRP-based algorithms. Given an SPRP algorithm ALG, we design a PRP algorithm ALG' specifically for the input generated by the procedure described in Section 2. (Recall that this input is a sequence of blocks, where a *block* is a string of a_0's, a_1's, and so on, ending with a single page a_ℓ for some ℓ.)

We show that if ALG has cost $o(k) \cdot \mathsf{OPT} + o(\ell_{ed})$ (where OPT is the optimal cost of the SPRP instance), then ALG' will have cost $o(k) \cdot \mathsf{OPT}'$ (where OPT' is the optimal cost of the PRP instance), which contradicts our PRP lower bound of $\Omega(k)$ on this input. For the randomized lower bound, we use the same line of reasoning, but replace $\Omega(k)$ with $\Omega(\log k)$.

Let k' denote the cache size of ALG'. Recall that the set of possible page requests received by ALG' is $A = \{a_0, a_1, \ldots, a_{k'}\}$ where $w(a_i) = c^i$ for some constant $c \geq 2$. The oracle ALG, maintained by ALG', has cache size $k = k' + 1$. The set of possible requests received by ALG is $A \cup \{b\}$ where $w(b) = 1/v$ for some sufficiently large value of v. (Thus, the instance for ALG has $k + 1$ distinct pages.) Our PRP algorithm ALG' must define a prediction and an input sequence for ALG.

The prediction sequence for ALG. For any strings X and Y, let $X + Y$ denote the concatenation of X and Y and let $\lambda \cdot X$ denote the concatenation of λ copies of X. Let $L = 2ck'H_{k'} + 1$, and consider the series of strings: $S_0 = 2 \cdot a_0$, and $S_i = L \cdot S_{i-1} + a_i$ for $i \in \{1, \ldots, k'\}$. We fix $S := M \cdot S_{k'}$, for some sufficiently large M, as the prediction sequence for the SPRP algorithm. (Observe that S only contains k distinct pages, and the oracle ALG has cache size k.)

ALG' and the request sequence for ALG. Our PRP algorithm ALG' will simultaneously construct input for ALG while serving its own requests. Since randomized and fractional algorithms are equivalent up to constants (see Bansal et al. [2]), we view the SPRP algorithm ALG from a fractional perspective. Let $q_i \in [0,1]$ denote the fraction of page a_i not in the cache of ALG. Notice that the vector $q = (q_0, q_1, \ldots, q_{k'})$ satisfies $\sum_{i=0}^{k'} q_i \geq 1$. (A deterministic algorithm is the special case where every $q_i \in \{0,1\}$.) Similarly, let $q' = (q'_0, q'_1, \ldots, q'_{k'})$, where q'_i denotes the amount of request for a_i that is not in the cache in ALG'.

When a block ending with a_i is requested, ALG' scans S for the next appearance of a_i. It then feeds the scanned portion to ALG, followed by a single request for page b. In this case, the prediction error only occurs due to the requests for this page b. After serving this request b, the cache of ALG contains at most k' pages in A. This enables ALG' to mimic the behavior of ALG upon serving the current block. This process continues for every block: ALG' modifies the input by inserting an extra request b into the input for ALG, and mimics the resulting cache state of ALG. The details of our algorithm ALG' are given below:

1. Initially, let S be the input for ALG and $t = 0$. (We will modify S as time passes.)
2. For all $0 \leq i \leq k'$, let $q'_i = 1$. (Note that the initial value of every q_i is also 1.)
3. On PRP request block $s_i = (a_0, a_1, \ldots, a_i)$ (for some unknown i):
   a. Let $q' = (q'_0, q'_1, \ldots, q'_{k'})$ denote the current cache state.
   b. Set $q' = (0, \min\{1, q'_0 + q'_1\}, q'_2, q'_3, \ldots, q'_{k'})$ to serve a_0. Note that after we serve a_0, the PRP prediction tells us the value of i.
   c. Find the first time t' after t when S requests a_i and set $t = t' + 2$.
   d. Change the request at time t into b. (Note that the original request is a_0.)
   e. Run ALG until this b is served to obtain a vector $q = (q_0, q_1, \ldots, q_{k'})$.
   f. If $i \geq 1$, set $q' = (\min\{1, \sum_{j=0}^{i} q'_j\}, 0, 0, \ldots, 0, q'_{i+1}, q'_{i+2}, \ldots, q'_{k'})$; this serves the requests $(a_1, a_2, \ldots, a_i)$.
   g. Set $q' = (q_0, q_1, \ldots, q_{k'})$.

Bounding the costs. The main idea in the analysis is the following: since the input sequences to ALG and ALG' are closely related, and they maintain similar cache states, we can show that they are coupled both in terms of the algorithm's cost and the optimal cost. Therefore, the ratio of $\Omega(k)$ for ALG' (from Theorem 7) translates to a ratio of $\Omega(k)$ for ALG. Furthermore, since the only prediction errors are due to the additional requests for page b, and this page has a very small weight, the cost of ALG is at least the value of ℓ_{ed}. (The same line of reasoning is used for randomized algorithms, but $\Omega(k)$ is replaced by $\Omega(\log k)$.)

We now formalize the above line of reasoning with the following lemmas.

▶ **Lemma 25.** *Using any SPRP algorithm* ALG *as a black box, the PRP algorithm* ALG' *satisfies the following:* $\mathsf{cost}(\mathsf{ALG}') \leq 2(c+1) \cdot \mathsf{cost}(\mathsf{ALG})$.

Proof. Note that $q = q'$ at the beginning and end of Step 3. For convenience, let q' denote the vector at the beginning of Step 3, and let q denote the vector at the end of Step 3. Let $\mathsf{cost}_{\mathsf{ALG}}$ and $\mathsf{cost}_{\mathsf{ALG}'}$ denote the cost of ALG and ALG' respectively incurred in a fixed Step 3.

Each time ALG' enters Step 3, the cost incurred is at most:

Step 3b: $q'_0 \cdot (1+c)$,

Step 3f: $(q'_0 + q'_1) \cdot (1+c) + \sum_{j=2}^{i} q'_j \cdot (1+c^j)$,

Step 3g: $\left(\sum_{j=1}^{i} q_j \cdot (1+c^j)\right) + \left(\sum_{j=i+1}^{k} \left|q'_j - q_j\right| \cdot (1+c^j)\right)$.

Summing the above yields the following:

$$\mathsf{cost}_{\mathsf{ALG}'} \leq 2(c+1) \cdot \left(\left(\sum_{j=0}^{i} c^j \cdot \left(q_j + q'_j\right)\right) + \left(\sum_{j=i+1}^{k} c^j \cdot \left|q_j - q'_j\right|\right)\right).$$

Now we consider ALG. For each j, at the beginning of Step 3, there is q'_j amount of a_j not in the cache, and at the end of Step 3, there is q_j amount of a_j not in the cache.

If $j > i$, the cost incurred due to a_j is at least $c^j \cdot \left|q_j - q'_j\right|$. If $j \leq i$, ALG' must serve a_j at some point in Step 3e, so the incurred cost due to a_j is at least $c^j \cdot \left(q_j + q'_j\right)$. Summing the above yields the following:

$$\mathsf{cost}_{\mathsf{ALG}} \geq \left(\sum_{j=0}^{i} c^j \cdot \left(q_j + q'_j\right)\right) + \left(\sum_{j=i+1}^{k} c^j \cdot \left|q_j - q'_j\right|\right).$$

Combining the two inequalities above proves the lemma. ◀

Now let OPT denote the optimal SPRP algorithm for the input sequence served by ALG, and let OPT' denote the optimal PRP algorithm for the input sequence served by ALG'. We can similarly prove the following lemma (proof in full paper):

▶ **Lemma 26.** *The algorithms* OPT *and* OPT' *satisfy* $\mathsf{cost}(\mathsf{OPT}) \leq 2 \cdot \mathsf{cost}(\mathsf{OPT}')$.

We are now ready to bound the cost of any algorithm with SPRP (proof in full paper):

▶ **Theorem 27.** *For weighted paging with SPRP, there is no deterministic algorithm whose cost is* $o(k) \cdot \mathsf{OPT} + o(\ell_{ed})$*, and there is no randomized algorithm whose cost is* $o(\log k) \cdot \mathsf{OPT} + o(\ell_{ed})$.

Proof (Sketch). From Theorem 7, we know $\mathsf{ALG}' = \Omega(k) \cdot \mathsf{OPT}'$. Thus, applying Lemmas 25 and 26, we have $\mathsf{ALG} = \Omega(k) \cdot \mathsf{OPT}$. Furthermore (as we saw in Section 2), each PRP block increases ALG by at least a constant. At the same time, for each block, we can show that ℓ_{ed} increases by at most 2. As a result, we can conclude that $\mathsf{ALG} = \Omega(\ell_1)$. The theorem follows by combining these facts. For randomized algorithms, the same line of reasoning holds, but with $\Omega(\log k)$ instead of $\Omega(k)$. ◀

5.2 Upper Bounds

In this section, we give algorithms whose performance degrades with the value of the SPRP error. In particular, we first prove the upper bound in Theorem 6 for the ℓ_{ed} measurement, and then analyze the FOLLOW algorithm, which proves the upper bound in Theorem 5.

Now we present an algorithm that uses a cache of size $k+1$ whose cost scales linearly with $\mathsf{OPT} + \ell_{ed}$. Following our previous terminology, let A denote a prediction sequence of length m, and let B denote an input sequence of length n.

Our algorithm, which we call Learn, relies on an algorithm that we call Idle. At a high level, Idle resembles Static (see Section 4): it partitions the prediction sequence A into batches and runs an optimal offline algorithm on each batch. The Learn algorithm tracks the cost of imitating Idle: if the cost is sufficiently low, then it will imitate Idle on k of its cache slots; otherwise, it will simply evict the page in the extra cache slot.

Before formally defining Idle, we consider a modified version of caching. Our cache has $k+1$ slots, where one slot is *memoryless*: it always immediately evicts the page it just fetched. In other words, this slot can serve any request, but it cannot store any pages. Let OPT^{+1} denote the optimal algorithm that uses a memoryless cache slot.

▶ **Lemma 28.** *For any sequences A and B, $\mathsf{cost}(\mathsf{OPT}^{+1}(A)) \leq \mathsf{cost}(\mathsf{OPT}(B)) + 2\ell_{ed}$, where ℓ_{ed} is the edit distance between A and B.*

Proof. Let M denote the optimal matching between A and B (for ℓ_{ed}). One algorithm for $\mathsf{OPT}^{+1}(A)$ is the following: imitate what $\mathsf{OPT}(B)$ does for requests matched by M, and use the memoryless slot for unmatched requests. The cost of this algorithm is $\mathsf{OPT}(B)+2\ell_{ed}$. ◀

Recall that the Static algorithm requires the use of an optimal offline algorithm. Similarly, for our new problem with a memoryless cache slot, we require a constant-approximation offline algorithm on A. This can be obtained from the following lemma (proof in full paper):

▶ **Lemma 29.** *Given a prediction sequence A, there is a randomized offline algorithm whose cost is at most a constant times the cost of $\mathsf{OPT}^{+1}(A)$.*

The Idle algorithm

Assume that our cache has size $k+1$ and the extra slot is memoryless (as defined above). For any time step t, let $L(t)$ denote the set of pages predicted to arrive starting at time $t+1$. At time step 1 (i.e., initially), Idle runs the offline algorithm from Lemma 29 on $L(1)$, ignoring future requests. After the requests in $L(1)$ have been served, i.e., at time $|L(1)|+1$, Idle then consults the predictor and runs the offline algorithm on the next "batch". The algorithm proceeds in this batch-by-batch manner until the end. We can show that the competitive ratio of this algorithm is at most a constant (see full paper).

▶ **Lemma 30.** *On the prediction sequence A, we have $\mathsf{cost}(\text{Idle}) = O(1) \cdot \mathsf{cost}(\mathsf{OPT}^{+1}(A))$.*

The Learn algorithm

Before defining the algorithm, we introduce another measurement of error that closely approximates ℓ_{ed}. Recall that A denotes a prediction sequence of length m and B denotes an input sequence of length n. In defining ℓ_{ed}, two elements $A_i = B_j$ can be matched only if $\mathsf{pnext}(\mathsf{prev}(j)) = i$, and no matching edges are permitted to cross.

▶ **Definition 31.** *The* constrained edit distance *ℓ'_{ed} is the minimum weight of unmatched elements of A and B, with the following additional constraint: if $|P(A_i)| \geq 2$, then A_i can only be matched with the latest-arriving element in $P(A_i)$.*

We note that ℓ'_{ed} is a constant approximation of ℓ_{ed} (proof in full paper):

▶ **Lemma 32.** *For any sequences A, B, we have $\ell_{ed} \leq \ell'_{ed} \leq 3\ell_{ed}$.*

Now we are ready to define the Learn algorithm. For any $i \leq j$, we let $A(i,j)$ denote the subsequence $(A_i, A_{i+1}, \ldots, A_j)$. For any set (or multiset) of pages S, we let $w(S)$ denote the total cost of pages in S. The algorithm is the following:

1. Let $s = 0$; the variable s always denotes that we have imitated the IDLE algorithm through the first s requests of the prediction.
2. Let $S = \emptyset$ be an empty queue.
3. On the arrival of request p, add p to S.
   a. If there is a t (in $[s+1, L]$ where L is the end of the current prediction) such that

   $$\ell'_{ed}(A(s+1,t), S) < \frac{1}{3}(w(A(s+1,t)) + w(S)), \tag{1}$$

   then imitate IDLE through position t, empty S and let $s = t$. (If more than one t satisfies the above, select the minimum.)

   b. Otherwise, evict the page in the final slot.

We first observe that the algorithm is indeed feasible (proof in full paper).

▶ **Lemma 33.** *In the LEARN algorithm, Step 3a is feasible, i.e., if t satisfies* (1)*, then* $A_t = p$.

Now we arrive at the heart of the analysis: we upper bound the cost of LEARN against the cost of IDLE (i.e., a surrogate for $\mathsf{OPT}(B)$) and the constrained edit distance ℓ'_{ed}. In particular, we sketch a proof of the following lemma and defer the full proof to the full paper.

▶ **Lemma 34.** *The algorithms LEARN and IDLE satisfy* $\mathsf{cost}(\text{LEARN}) \leq \mathsf{cost}(\text{IDLE}) + 12\ell'_{ed}$.

Proof (Sketch). Let cost_1 denote the total cost of Step 3a, and cost_2 denote the total cost of Step 3b so that $\mathsf{cost}(\text{LEARN}) = \mathsf{cost}_1 + \mathsf{cost}_2$. From the algorithm, we see that $\mathsf{cost}_1 \leq \mathsf{cost}(\text{IDLE})$, so now we must prove $\mathsf{cost}_2 \leq 12\ell'_{ed}$.

Now we establish some notation. Let $\ell'_{ed}((a,b)(c,d)) = \ell'_{ed}(A(a,b), B(c,d))$, and let $w_A(a,b) = w(A(a,b))$ and $w_B(a,b) = w(B(a,b))$.

We proceed by induction on the number of times we went Step 3a. Consider the first time we enter Step 3a; suppose we have read the input $B(1,b)$ and we now imitated IDLE through $A(1,a)$ for some values a, b. Since the matched edges for ℓ'_{ed} do not cross, there exists some c such that $\ell'_{ed} = \ell'_{ed}(A, B)$ satisfies

$$\ell'_{ed} = \ell'_{ed}((1,a),(1,c)) + \ell'_{ed}((a+1,m),(c+1,n)).$$

We consider the case where $c < b$; the other cases follow similarly. Let $\mathsf{cost}(x,y)$ denote the cost incurred by the algorithm when serving $B(x,y)$ and notice that

$$\mathsf{cost}_2 \leq \mathsf{cost}(1,c) + \mathsf{cost}(c+1,b) + \mathsf{cost}(b+1,n).$$

The cost of serving $B(1,c)$ is at most the weight of the requested pages, so $\mathsf{cost}(1,c) \leq w_B(1,c)$. Furthermore, we can upper bound $\mathsf{cost}(c+1,b)$ by a constant times $w_A(1,a)$ by analyzing a particular matching for $\ell'_{ed}((1,a)(1,c))$. Combining this together, we have

$$\mathsf{cost}(1,c) + \mathsf{cost}(c+1,b) \leq 4(w_B(1,c) + w_A(1,a)) \leq 12 \cdot \ell'_{ed}((1,a),(1,c)),$$

where the second inequality follows from that we did not enter Step 3a when c arrived. Finally, applying the inductive hypothesis to $B(b+1,n)$ and substituting the definition of c yields the lemma. ◀

The proof of Theorem 6 follows from Lemmas 28, 30, and 34.

The Follow algorithm

Now we show that the $\Omega(\ell_1)$ lower bound in Theorem 5 is tight, that is, we will give an SPRP algorithm Follow that has cost $O(1) \cdot (\mathsf{OPT} + \ell_1)$. Recall the Static algorithm from Theorem 21. The algorithm Follow ignores its input: it simply runs Static on the prediction sequence A and imitates its fetches/evictions on the input sequence B.

▶ **Theorem 35.** *The Follow algorithm has cost $O(1) \cdot (\mathsf{OPT} + \ell_1)$.*

Proof. Recall from Theorem 21 that $\mathsf{cost}(\text{Static}) \leq O(1) \cdot \mathsf{OPT}(A)$. Furthermore, we claim $\mathsf{OPT}(A) \leq \mathsf{OPT}(B) + 2\ell_1$. This is because on A, there exists an algorithm that imitates the movements of B: say at time t, $\mathsf{OPT}(B)$ evicts some element b that had appeared in B at time $v(t)$. Then $\mathsf{OPT}(A)$ can also evict whatever element appeared at time $v(t)$ in A, and if this is not b, then this cost can be charged to the $v(t)$ term of ℓ_1. Each term of ℓ_1 is charged at most twice because a specific request can be evicted and fetched at most once respectively.

By the same argument, we have $\mathsf{cost}(\text{Follow}) \leq \mathsf{cost}(\text{Static}) + 2\ell_1$. Combining these inequalities proves the theorem. ◀

6 Conclusion

In this paper, we initiated the study of weighted paging with predictions. This continues the recent line of work in online algorithms with predictions, particularly that of Lykouris and Vassilvitski [9] on unweighted paging with predictions. We showed that unlike in unweighted paging, neither a fixed lookahead not knowledge of the next request for every page is sufficient information for an algorithm to overcome existing lower bounds in weighted paging. However, a combination of the two, which we called the strong per request prediction (SPRP) model, suffices to give a constant approximation. We also explored the question of gracefully degrading algorithms with increasing prediction error, and gave both upper and lower bounds for a set of natural measures of prediction error. The reader may note that the SPRP model is rather optimistic and requires substantial information about the future. A natural question arises: can we obtain constant competitive algorithms for weighted paging with fewer predictions? While we refuted this for the PRP and fixed lookahead models, being natural choices because they suffice for unweighted paging, it is possible that an entirely different parameterization of predictions can also yield positive results for weighted paging. We leave this as an intriguing direction for future work.

References

1. Susanne Albers. The influence of lookahead in competitive paging algorithms. In *European Symposium on Algorithms*, pages 1–12. Springer, 1993.
2. Nikhil Bansal, Niv Buchbinder, and Joseph Seffi Naor. A primal-dual randomized algorithm for weighted paging. *Journal of the ACM (JACM)*, 59(4):19, 2012.
3. Laszlo A. Belady. A study of replacement algorithms for a virtual-storage computer. *IBM Systems journal*, 5(2):78–101, 1966.
4. Marek Chrobak, H Karloof, Tom Payne, and S Vishwnathan. New results on server problems. *SIAM Journal on Discrete Mathematics*, 4(2):172–181, 1991.
5. Amos Fiat, Richard M Karp, Michael Luby, Lyle A McGeoch, Daniel D Sleator, and Neal E Young. Competitive paging algorithms. *Journal of Algorithms*, 12(4):685–699, 1991.
6. Sreenivas Gollapudi and Debmalya Panigrahi. Online algorithms for rent-or-buy with expert advice. In *International Conference on Machine Learning*, pages 2319–2327, 2019.
7. Chen-Yu Hsu, Piotr Indyk, Dina Katabi, and Ali Vakilian. Learning-based frequency estimation algorithms. In *International Conference on Learning Representations*, 2019.

8 Silvio Lattanzi, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Online scheduling via learned weights. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1859–1877, 2020.

9 Thodoris Lykouris and Sergei Vassilvtiskii. Competitive caching with machine learned advice. In *International Conference on Machine Learning*, pages 3302–3311, 2018.

10 Michael Mitzenmacher. A model for learned bloom filters and optimizing by sandwiching. In *Advances in Neural Information Processing Systems*, pages 464–473, 2018.

11 Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms.* Cambridge University Press, New York, NY, USA, 1995.

12 Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ml predictions. In *Advances in Neural Information Processing Systems*, pages 9661–9670, 2018.

13 Dhruv Rohatgi. Near-optimal bounds for online caching with machine learned advice. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1834–1845. SIAM, 2020.

14 Daniel D Sleator and Robert E Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

15 Neal Young. On-line caching as cache size varies. In *Proceedings of the Second Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '91, pages 241–250, 1991.

16 Neal E Young. On-line file caching. *Algorithmica*, 33(3):371–383, 2002.

Popular Matchings with One-Sided Bias

Telikepalli Kavitha
Tata Institute of Fundamental Research, Mumbai, India
kavitha@tifr.res.in

Abstract

Let $G = (A \cup B, E)$ be a bipartite graph where A consists of agents or *main players* and B consists of jobs or *secondary players*. Every vertex has a strict ranking of its neighbors. A matching M is *popular* if for any matching N, the number of vertices that prefer M to N is at least the number that prefer N to M. Popular matchings always exist in G since every stable matching is popular.

A matching M is A-*popular* if for any matching N, the number of *agents* (i.e., vertices in A) that prefer M to N is at least the number of agents that prefer N to M. Unlike popular matchings, A-popular matchings need not exist in a given instance G and there is a simple linear time algorithm to decide if G admits an A-popular matching and compute one, if so.

We consider the problem of deciding if G admits a matching that is both popular and A-popular and finding one, if so. We call such matchings *fully popular*. A fully popular matching is useful when A is the more important side – so along with overall popularity, we would like to maintain "popularity within the set A". A fully popular matching is not necessarily a min-size/max-size popular matching and all known polynomial time algorithms for popular matching problems compute either min-size or max-size popular matchings. Here we show a linear time algorithm for the fully popular matching problem, thus our result shows a new tractable subclass of popular matchings.

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

Keywords and phrases Bipartite graphs, Stable matchings, Gale-Shapley algorithm, LP-duality

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.70

Category Track A: Algorithms, Complexity and Games

Funding Supported by the DAE, Government of India, under project no. 12-R&D-TFR-5.01-0500.

Acknowledgements Work done at MPI for Informatics, Saarland Informatics Campus, Germany. Thanks to Yuri Faenza for discussions that led to this problem and his helpful comments on the manuscript. Thanks to the reviewers for their valuable suggestions on improving the presentation.

1 Introduction

Let $G = (A \cup B, E)$ be a bipartite graph where vertices in A are called *agents* and those in B are called *jobs*. Every vertex has a strict ranking of its neighbors. Such a graph, also called a *marriage* instance, is a very well-studied model in two-sided matching markets. A matching M in G is *stable* if there is no *blocking pair* with respect to M, i.e., no pair (a, b) such that a and b prefer each other to their respective assignments in M. Gale and Shapley [10] in 1962 showed that stable matchings always exist in G and can be efficiently computed.

Stable matching algorithms have applications in several real-world problems. For instance, stable matchings have been extensively used to match students to schools and colleges [1, 3] and one of the oldest applications here is to match medical residents to hospitals [4, 21]. It is known that all stable matchings in G have the same size [11] and this may only be half the size of a max-size matching in G. Consider the following instance on 4 vertices a_0, a_1, b_0, b_1.

$$a_0 : b_1 \qquad a_1 : b_1 \succ b_0 \qquad b_0 : a_1 \qquad b_1 : a_1 \succ a_0.$$

Here a_1 and b_1 are each other's top choices. There is no edge between a_0, b_0. Note that $M_{\max} = \{(a_0, b_1), (a_1, b_0)\}$ has size 2 while the only stable matching $S = \{(a_1, b_1)\}$ has size 1.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 70; pp. 70:1–70:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Hence forbidding blocking edges constrains the size of the resulting matching. Rather than empower every edge with a "veto power" to block matchings (this is the notion of stability), we would like to relax stability so that only a *strict majority vote* from the entire vertex set has the power to block matchings. The motivation is to obtain a larger pool of feasible matchings so as to allow better matchings with respect to our objective.

The notion of *popularity* is a natural relaxation of stability that captures the notion of "majority preference". Preferences of a vertex over its neighbors extend naturally to preferences over matchings: consider an election between 2 matchings M and N where vertices are voters. In this M versus N election, every vertex (say, u) votes for the matching in $\{M, N\}$ that it prefers, i.e., where it gets a better assignment (being unmatched is its worst choice), and u abstains from voting if it has the same assignment in both M and N. Let $\phi(M, N)$ (resp., $\phi(N, M)$) be the number of votes for M (resp., N) in this election.

▶ **Definition 1.** *A matching M is* popular *if $\phi(M, N) \geq \phi(N, M)$ $\forall$ matchings N in G.*

So a popular matching never loses a head-to-head election against any matching, i.e., it is a weak *Condorcet winner* [5, 6] in the voting instance where matchings are candidates and vertices are voters. The notion of popularity was introduced by Gärdenfors [12] who showed that every stable matching is popular. So popular matchings always exist in any marriage instance. In fact, every stable matching is a *min-size* popular matching [14] and there are efficient algorithms to compute a *max-size* popular matching [14, 16].

Popular matchings are suitable for applications such as matching students to projects (where students and project advisers have strict preferences) – by relaxing stability to popularity, we can obtain better matchings in terms of size or our desired objective. We consider a natural and relevant objective here: observe that the two sides of $G = (A \cup B, E)$ are *asymmetric* in this application - students are *doers* of the projects, i.e., they are the main or more active players while project advisers are the secondary or more passive players. So along with overall popularity, we would like to maintain "popularity within the set A".

That is, we would like the popular matching that we compute to be popular even when we *only* count the votes of vertices in A, i.e., there should be no matching that is preferred by more vertices in A. Popularity within the set A is the notion of popularity with *one-sided* preferences and we will refer to this as *A-popularity* here. In the M versus N election, let $\phi_A(M, N)$ (resp., $\phi_A(N, M)$) be the number of vertices in A that vote for M (resp., N).

▶ **Definition 2.** *A matching M is* A-popular *if $\phi_A(M, N) \geq \phi_A(N, M)$ $\forall$ matchings N in G.*

A-popular matchings have been well-studied and are relevant in applications such as assigning training posts to applicants [2] and housing allocation schemes [19] where vertices on only one side of the graph have preferences over their neighbors. An A-popular matching need not necessarily exist in a given instance as shown below.

$$a_1 : b_1 \succ b_2 \succ b_3 \qquad a_2 : b_1 \succ b_2 \succ b_3 \qquad a_3 : b_1 \succ b_2 \succ b_3.$$

There are 3 agents here and they have identical preferences. It is easy to check that none of the matchings in this instance is A-popular. Let $M_0 = \{(a_1, b_1), (a_2, b_2), (a_3, b_3)\}$ and $M_1 = \{(a_1, b_3), (a_2, b_1), (a_3, b_2)\}$, we have $\phi_A(M_1, M_0) = 2 > 1 = \phi_A(M_0, M_1)$ since a_2, a_3 prefer M_1 to M_0 while a_1 prefers M_0 to M_1. Here we are interested in matchings that are both popular *and* A-popular.

▶ **Definition 3.** *A popular matching M is* fully popular *if M is also A-popular. So for any matching N in G, we have: $\phi(M, N) \geq \phi(N, M)$ and $\phi_A(M, N) \geq \phi_A(N, M)$.*

There may be exponentially many popular matchings in $G = (A \cup B, E)$. So when A is the more important/active side, say it consists of those doing their projects/internships/jobs, it is natural to seek a popular matching that is A-popular as well, i.e., a *fully popular* matching. Thus we seek a matching M such that (1) a majority of the vertices weakly prefer M to any matching, i.e., $\phi(M, N) \geq \phi(N, M)$ for all matchings N, and moreover, (2) a majority of the agents (those in A) weakly prefer M to any matching, i.e., $\phi_A(M, N) \geq \phi_A(N, M)$ for all N.

We show the following result here.

▶ **Theorem 4.** *There is a linear time algorithm to decide if a marriage instance $G = (A \cup B, E)$ with strict preferences admits a fully popular matching or not. If so, our algorithm returns a max-size fully popular matching.*

Another model for popularity in matchings with main players in A and secondary players in B is to scale the votes of those in A by a suitable factor $c \geq 1$ and count the weighted sum of votes in favor of M versus the weighted sum of votes in favor of N in the M versus N election. That is, M is *weighted popular* if $c \cdot \phi_A(M, N) + \phi_B(M, N) \geq c \cdot \phi_A(N, M) + \phi_B(N, M)$ for every matching N, where c is the scaling factor and $\phi_B(\cdot, \cdot)$ is analogous to $\phi_A(\cdot, \cdot)$. No results are currently known for the weighted popular matching problem with $c > 1$ in a marriage instance. Observe that a fully popular matching is a weighted popular matching for every scaling factor $c \geq 1$.

1.1 Background and Related results

The notion of popularity was proposed by Gärdenfors [12] in 1975. Algorithms in the domain of popular matchings were first studied in 2005 for *one-sided* preferences or the A-popular matching problem. Efficient algorithms were given in [2] to decide if a given instance (with ties permitted in preference lists) admits an A-popular matching or not; in particular, a linear time algorithm was given for the case with strict preference lists. An efficient algorithm for the weighted A-popular matching problem, where each agent's vote is scaled by its weight (these weights are a part of the input), was given in [20].

Algorithms for popular matchings in a marriage instance $G = (A \cup B, E)$ or *two-sided* preferences have been well-studied in the last decade. The max-size popular matching algorithms in [14, 16] compute special popular matchings called *dominant* matchings. A linear time algorithm for finding a popular matching with a given edge e was given in [7] (such an edge is called a popular edge). It was shown in [7] that if e is a popular edge then there is either a stable matching or a dominant matching with e. Popular half-integral matchings in $G = (A \cup B, E)$ were characterized in [17] as stable matchings in a larger graph related to G. The popular fractional matching polytope was analyzed in [15] where the half-integrality of this polytope was shown. Other than algorithms for min-size/max-size popular matchings and for popular edge, no other polynomial time algorithms are currently known for finding popular matchings with special properties.

To complete the picture, it was shown in [9] that it is NP-hard to decide if G admits a popular matching that is neither a min-size nor a max-size popular matching. A host of hardness results in [9] painted a bleak picture for efficient algorithms for popular matching problems (other than what is already known). For instance, it is NP-hard to find a popular matching in G with a given pair of edges. Thus finding a max-weight (similarly, min-cost) popular matching is NP-hard when there are weights (resp., costs) on edges.

1.2 Our Result and Techniques

It may be the case that no min-size/max-size popular matching in G is A-popular, however G admits a fully popular matching: Section 2 has such an example. As there are instances where it is NP-hard to decide if there is a popular matching that is neither a min-size nor a max-size popular matching [9], a first guess may be that the fully popular matching problem is NP-hard.

Though an A-popular matching is constrained to use only some special edges in G (see Theorem 5), this does not seem very helpful since it is NP-hard to solve the popular matching problem with forced/forbidden edges [9]. Note that a rival matching (wrt popularity) is free to use any edge in G. It was not known if there was any tractable subclass of popular matchings other than the classes of *stable matchings* [10] and *dominant matchings* [7, 14, 16].

We show the set of fully popular matchings is a new tractable subclass of popular matchings: unlike the classes of stable matchings and dominant matchings which are always non-empty, there need not be a fully popular matching in G. Our algorithm for finding a fully popular matching is based on the classical Gale-Shapley algorithm and works in a new graph H; this graph is essentially two copies of G and is a variant of the graph seen in [17] to study popular *half-integral* matchings. There is a natural map from stable matchings in H to popular half-integral matchings in G. Our goal is to compute a stable matching with sufficient symmetry in H so that we can obtain a popular *integral* matching in G.

We achieve this symmetry by using properties of both popular and A-popular matchings. These properties allow us to identify certain edges that have to be excluded from our matching. If there is no stable matching in H without these edges then we use the lattice structure on stable matchings [13] to show that G has no fully popular matching. Else we obtain a matching M in G from this "partially symmetric" stable matching in H. The most technical part of our analysis is to prove M's popularity in G.

2 Preliminaries

Our input is a bipartite graph $G = (A \cup B, E)$ where every vertex has a strict preference list ranking its neighbors. Let us augment G with self-loops, i.e., each vertex is assumed to be at the bottom of its own preference list.

We will first present the characterization of A-popular matchings in G – note that preferences of vertices in B play no role here. For each $a \in A$, define the vertex $f(a)$ to be a's top choice neighbor and let $s(a)$ be a's most preferred neighbor that is nobody's top choice neighbor. We assume every $a \in A$ has at least one neighbor other than itself, so $f(a) \in B$, however it may be the case that $s(a) = a$. Let $E' = E \cup \{(u,u) : u \in A \cup B\}$.

▶ **Theorem 5** ([2]). *A matching M in $G = (A \cup B, E')$ is A-popular if and only if:*
1. $M \subseteq \{(a, f(a)), (a, s(a)) : a \in A\}$.
2. *M matches all in A and all in $\{f(a) : a \in A\}$.*

Popular matchings. We will use an LP-based characterization of popular matchings [17, 18] in a marriage instance $G = (A \cup B, E')$. Observe that any matching in G can be regarded as a perfect matching by including self-loops for all vertices left unmatched. Let M be any perfect matching in G. For any pair of adjacent vertices u, v, let u's vote for v (vs its partner $M(u)$) be 1 if u prefers v to $M(u)$, it is -1 if u prefers $M(u)$ to v, else 0 (i.e., $M(u) = v$). In order to check if M is popular or not in G, the following edge weight function wt_M will be useful. Here $\mathsf{wt}_M(a, b)$ is the sum of votes of a and b for each other.

$$\text{Let } \mathsf{wt}_M(a,b) = \begin{cases} 2 & \text{if } (a,b) \text{ is a blocking edge to } M; \\ -2 & \text{if both } a \text{ and } b \text{ prefer their partners in } M \text{ to each other;} \\ 0 & \text{otherwise.} \end{cases}$$

Thus $\mathsf{wt}_M(a,b) = 0$ for every $(a,b) \in M$. We need to define wt_M for self-loops as well. For any $u \in A \cup B$, let $\mathsf{wt}_M(u,u) = 0$ if $(u,u) \in M$, else $\mathsf{wt}_M(u,u) = -1$. For any perfect matching N in G, observe that $\mathsf{wt}_M(N) = \sum_{e \in N} \mathsf{wt}_M(e) = \phi(N,M) - \phi(M,N)$. Thus M is popular if and only if $\mathsf{wt}_M(N) \le 0$ for every perfect matching N in G.

Consider the max-weight perfect matching LP in G with the edge weight function wt_M. This linear program is (LP1) given below and (LP2) is the dual of (LP1). The variables x_e for $e \in E'$ are primal variables and the variables y_u for $u \in A \cup B$ are dual variables. Here $\delta'(u) = \delta(u) \cup \{(u,u)\}$.

$$\begin{aligned} \max \sum_{e \in E'} \mathsf{wt}_M(e) \cdot x_e & & \text{(LP1)} \\ \text{s.t.} \quad \sum_{e \in \delta'(u)} x_e = 1 & \quad \forall\, u \in A \cup B \\ x_e \ge 0 & \quad \forall\, e \in E'. \end{aligned}$$

$$\begin{aligned} \min \sum_{u \in A \cup B} y_u & & \text{(LP2)} \\ \text{s.t.} \quad y_a + y_b \ge \mathsf{wt}_M(a,b) & \quad \forall\, (a,b) \in E \\ y_u \ge \mathsf{wt}_M(u,u) & \quad \forall\, u \in A \cup B. \end{aligned}$$

M is popular if and only if the optimal value of (LP1) is at most 0. In fact, the optimal value is exactly 0 since M is a perfect matching in G and $\mathsf{wt}_M(M) = 0$. Thus M is popular if and only if the optimal value of (LP2) is 0 (by LP-duality).

▶ **Theorem 6** ([17, 18])**.** *A matching M in $G = (A \cup B, E)$ is popular if and only if there exists $\vec{\alpha} \in \{0, \pm 1\}^n$ (where $|A \cup B| = n$) such that $\sum_{u \in A \cup B} \alpha_u = 0$ along with*

$$\alpha_a + \alpha_b \ge \mathsf{wt}_M(a,b) \quad \forall (a,b) \in E \qquad \textit{and} \qquad \alpha_u \ge \mathsf{wt}_M(u,u) \quad \forall u \in A \cup B.$$

Proof. Since E is the edge set of a bipartite graph, the constraint matrix of (LP2) is totally unimodular. So (LP2) admits an optimal solution that is integral. The vector $\vec{\alpha}$ is an integral optimal solution of (LP2). We have $\alpha_u \ge \mathsf{wt}_M(u,u) \ge -1$ for all u.

We now claim $\alpha_u \in \{0, \pm 1\}$ for all vertices u. Since M is an optimal solution to (LP1), complementary slackness implies $\alpha_u + \alpha_v = \mathsf{wt}_M(u,v) = 0$ for each edge $(u,v) \in M$. Thus $\alpha_u = -\alpha_v \le 1$ for every vertex u matched to a non-trivial neighbor v in M. Regarding any vertex u such that $(u,u) \in M$, we have $\alpha_u = \mathsf{wt}_M(u,u) = 0$ (by complementary slackness). Hence $\vec{\alpha} \in \{0, \pm 1\}^n$. ◀

For any popular matching M, a vector $\vec{\alpha}$ as given in Theorem 6 will be called a *witness* of M's popularity. Note that a popular matching may have several witnesses. A stable matching S has 0^n as a witness since $\mathsf{wt}_S(e) \le 0$ for all $e \in E'$.

Recall that our problem is to compute a *fully popular* matching, i.e., a popular matching that is also A-popular. It is easy to construct instances that admit A-popular matchings but admit no fully popular matching. It could also be the case that no min-size or max-size popular matching in $G = (A \cup B, E)$ is A-popular, however G has a fully popular matching. Consider the instance G given in Fig. 1. Vertex preferences are indicated on edges: 1 denotes top choice, and so on.

We list the vertices $f(u)$ and $s(u)$ for each $u \in A$ in this instance. The vertex b' is *not* $s(a)$ since a prefers q' to b' and $q' \ne f(u)$ for any $u \in A$.

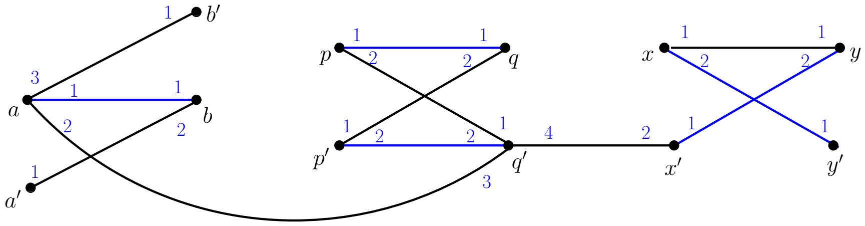


Figure 1 An instance on $A = \{a, a', p, p', x, x'\}$ and $B = \{b, b', q, q', y, y'\}$ where no min-size/max-size popular matching is A-popular. There is a fully popular matching (on blue edges) here.

- We have $f(a) = f(a') = b$, $f(p) = f(p') = q$, and $f(x) = f(x') = y$.
- We have $s(a) = s(p) = s(p') = s(x') = q'$, $s(x) = y'$, and $s(a') = a'$.

Since $s(u) \neq u$ for $u \in \{a, p, p', x, x'\}$, any A-popular matching M has to match these 5 vertices to neighbors in B (by Theorem 5). So $M(a) = f(a) = b$ which implies $M(a') = s(a') = a'$, i.e., after pruning self-loops from M, the vertex a' has to be left unmatched in M. So M has size 5. It is easy to check that a stable matching (thus any min-size popular matching) in G has size 4; also any max-size popular matching has size 6. Thus no min-size or max-size popular matching in G can be fully popular. Interestingly, this instance admits a fully popular matching: $M = \{(a, b), (p, q), (p', q'), (x, y'), (x', y)\}$ is fully popular.

3 Fully Popular Matchings

The input is a marriage instance $G = (A \cup B, E)$. Our algorithm will work in a bipartite graph H which is essentially 2 copies of the graph G as shown in Fig. 2. The vertex set of H is $A_L \cup B_L$ on the left and $B_R \cup A_R$ on the right. Here $A_L = \{a_\ell : a \in A\}$ and $A_R = \{a_r : a \in A\}$. Similarly, $B_L = \{b_\ell : b \in B\}$ and $B_R = \{b_r : b \in B\}$.

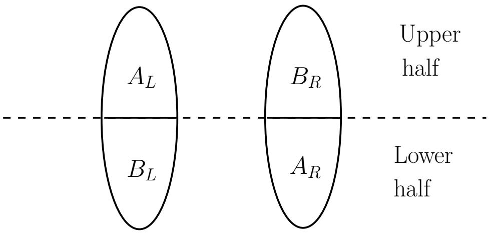


Figure 2 The bipartite graph H consists of 2 copies of the graph $G = (A \cup B, E)$.

The *upper half* of H consists of the set A_L of agents on the left and the set B_R of jobs on the right while the *lower half* of H consists of the set B_L of jobs on the left and the set A_R of agents on the right. Thus every vertex $u \in A \cup B$ has two copies in H: one as u_ℓ on the left of H and another as u_r on the right of H.

For every edge in E, there will be four edges in H: a pair of parallel edges in the upper half and a pair of parallel edges in the lower half. In order to distinguish two parallel edges with the same endpoints, we use superscripts $+$ and $-$ on the endpoints. For $(a, b) \in E$:

- in the upper half, we have two parallel edges (a_ℓ^+, b_r^-) and (a_ℓ^-, b_r^+) between a_ℓ and b_r;
- in the lower half, we have two parallel edges (b_ℓ^+, a_r^-) and (b_ℓ^-, a_r^+) between b_ℓ and a_r.

Corresponding to every vertex $u \in A \cup B$, there will be a *single* edge (u_ℓ^-, u_r^+) in H: this edge corresponds to the self-loop (u, u) and it will be convenient to use $+/-$ superscripts on the endpoints of this edge also. These edges (u_ℓ^-, u_r^+) for all u are the only edges in H that go across the two halves of H.

Vertices in H have preferences on their incident *edges* rather than their neighbors. However it would be more convenient to say u prefers v^- to w^+ rather than say u prefers (u^+, v^-) to (u^-, w^+). In fact, H is equivalent to a conventional graph H^* (with preferences on neighbors) that was used to study popular *half-integral* matchings in [17]: there were 4 vertices in H^* for each $u \in A \cup B$. The graph H is a sparser version of H^* with only 2 vertices u_ℓ and u_r for each $u \in A \cup B$ and a pair of parallel edges between every pair of adjacent vertices. We now describe the preferences of vertices in H.

Every vertex prefers superscript $-$ neighbors to superscript $+$ neighbors: among superscript $-$ neighbors (similarly, superscript $+$ neighbors), it will be its original preference order. Consider any vertex $u \in A \cup B$. Suppose u's preference list in G is $v \succ v' \succ \cdots \succ v''$, i.e., v is u's top choice, next comes v', and so on. In H, the preference list of u_ℓ is as follows:

$$\underbrace{v_r^- \succ v_r'^- \succ \cdots \succ v_r''^-}_{\text{superscript } - \text{ neighbors}} \succ \underbrace{v_r^+ \succ v_r'^+ \succ \cdots \succ v_r''^+ \succ u_r^+}_{\text{superscript } + \text{ neighbors}},$$

where $v_r, v'_r, \ldots$ correspond to the copies of $v, v', \ldots$ on the right side of H.

The vertex u_r^+ is the last choice of u_ℓ. In H, the preference list of u_r is as follows:

$$\underbrace{v_\ell^- \succ v_\ell'^- \succ \cdots \succ v_\ell''^- \succ u_\ell^-}_{\text{superscript } - \text{ neighbors}} \succ \underbrace{v_\ell^+ \succ v_\ell'^+ \succ \cdots \succ v_\ell''^+}_{\text{superscript } + \text{ neighbors}},$$

where $v_\ell, v'_\ell, \ldots$ correspond to the copies of $v, v', \ldots$ on the left side of H. This is analogous to u_ℓ's preference list: the main difference is in the position of its "twin" - the vertex u_r prefers u_ℓ^- to all its superscript $+$ neighbors.

Blocking edges. Let $u_\ell \in A_L \cup B_L$ and $v_r \in A_R \cup B_R$. For any matching M in H, we say an edge (u_ℓ^+, v_r^-) *blocks* M if (i) u_ℓ prefers v_r^- to its assignment in M and (ii) v_r prefers u_ℓ^+ to its assignment in M. Similarly, we say edge (u_ℓ^-, v_r^+) *blocks* M if (i) u_ℓ prefers v_r^+ to its assignment in M and (ii) v_r prefers u_ℓ^- to its assignment in M.

▶ **Definition 7.** *A matching M in H is* stable *if no edge in H blocks M.*

▷ Claim 8. Let S be a stable matching in $G = (A \cup B, E)$. The matching $S' = \{(a_\ell^-, b_r^+), (b_\ell^-, a_r^+) : (a, b) \in S\} \cup \{(u_\ell^-, u_r^+) : u \text{ is unmatched in } S\}$ is stable in H.

Proof. We need to show that no edge in H blocks S'. Note that S' is a perfect matching in H. Consider any edge (c_ℓ^+, d_r^-) in H where $c \in A$ and $d \in B$. Since every vertex prefers superscript $-$ neighbors to superscript $+$ neighbors, the vertex d_r prefers its partner in S' to c_ℓ^+. So consider any edge (c_ℓ^-, d_r^+) in H. If $(c, d) \in S$ then $(c_\ell^-, d_r^+) \in S'$ and so it does not block S'. If $(c, d) \notin S$ then it follows from the stability of S in G that either (1) c is matched to a neighbor b preferred to d or (2) d is matched to a neighbor a preferred to c. In case (1), c_ℓ is matched in S' to a neighbor b_r^+ preferred to d_r^+; in case (2), d_r is matched in S' to a neighbor a_ℓ^- preferred to c_ℓ^-. Thus (c_ℓ^-, d_r^+) does not block S'.

It can analogously be shown that neither (d_ℓ^-, c_r^+) nor (d_ℓ^+, c_r^-) blocks S'. Also (u_ℓ^-, u_r^+) for any $u \in A \cup B$ does not block S' since u_r^+ is u_ℓ's least preferred neighbor in H. Hence S' is a stable matching in H. ◁

Thus the graph H admits a perfect stable matching. Since all stable matchings in H have the same size [11], every stable matching in H has to be perfect. We seek to compute a "special" stable matching in H: one that has no edge that is *forbidden*. The edges marked forbidden are those that no fully popular matching uses. The definition of *valid* edges below is as given in Theorem 5: any A-popular matching in G can contain only these edges/self-loops.

▶ **Definition 9.** *Edges/self-loops in* $\{(a, f(a)), (a, s(a)) : a \in A\}$ *are* valid. *So are self-loops in* $\{(b, b) : b \neq f(a)$ for any $a \in A\}$.

All other edges and self-loops are *invalid*. Thus every $a \in A$ has exactly 2 valid edges incident to it: one of these may be the self-loop (a, a).

An edge e in G is *popular* if there exists a popular matching M in G such that $e \in M$. Call a vertex *stable* if it is matched in some (equivalently, every [11]) stable matching. It is known that every popular matching in G has to match all stable vertices to non-trivial neighbors [14]. So the self-loop (u, u) is popular if and only if u is unstable.

▶ **Definition 10.** *Call an edge e in $E \cup \{(u, u) : u \in A \cup B\}$* legal *if e is valid and popular.*

Forbidden edges. A fully popular matching, by definition, has to contain only legal edges. So if (a, b) is not legal then (a^+_ℓ, b^-_r), (a^-_ℓ, b^+_r), (b^+_ℓ, a^-_r), and (b^-_ℓ, a^+_r) are *forbidden* edges in the stable matching that we seek to compute in H. Similarly, for any $u \in A \cup B$, if (u, u) is not legal then (u^-_ℓ, u^+_r) is a forbidden edge in our algorithm.

▶ **Definition 11.** *A matching M in H is* legal *if M has no forbidden edge.*

Call a matching M in H *symmetric* if for each edge (a, b) in E, we have both (a_ℓ, b_r) and (b_ℓ, a_r) in M or neither (for convenience, we are ignoring $+/-$ superscripts on a_ℓ, a_r, b_ℓ, b_r), i.e., loosely speaking, M has the same edges in the upper and lower halves of H. A symmetric matching M in H will be called a *realization* of $\tilde{M} = \{(a, b) : (a_\ell, b_r)$ and (b_ℓ, a_r) are in $M\}$. Note that $\tilde{M}$ is a matching in G.

▶ **Lemma 12.** *Every fully popular matching in G has a realization as a legal stable matching in the instance H.*

Proof. Let N be a fully popular matching in G and let $\vec{\alpha} \in \{0, \pm 1\}^n$ be a witness of N's popularity (see Theorem 6). For $i \in \{0, \pm 1\}$, let A_i be the set of vertices $a \in A$ with $\alpha_a = i$ and let B_i be the set of vertices $b \in B$ with $\alpha_b = i$. Thus we have $A = A_0 \cup A_1 \cup A_{-1}$ and $B = B_0 \cup B_1 \cup B_{-1}$. Note that $\alpha_a + \alpha_b = \mathsf{wt}_N(a, b) = 0$ for each edge $(a, b) \in N$: this is by complementary slackness on (LP2) corresponding to N. So the matching $N \subseteq (A_0 \times B_0) \cup (A_{-1} \times B_1) \cup (A_1 \times B_{-1})$ (see Fig. 3).

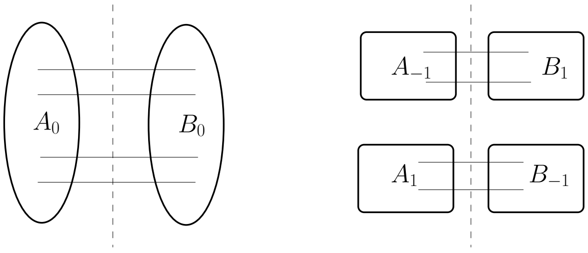


Figure 3 The partition $A_0 \cup A_1 \cup A_{-1}$ of A and $B_0 \cup B_{-1} \cup B_1$ of B.

We need to show a realization of N in H that is stable. We will use N's witness $\vec{\alpha}$ in G to define the following matching N^*_α in H: this is similar to how popular half-integral matchings were realized as stable matchings in a larger graph H^* in [17].

- For all $(a, b) \in N \cap (A_{-1} \times B_1)$ do: add edges (a^-_ℓ, b^+_r) and (b^+_ℓ, a^-_r) to N^*_α.
- For all $(a, b) \in N \cap (A_1 \times B_{-1})$ do: add edges (a^+_ℓ, b^-_r) and (b^-_ℓ, a^+_r) to N^*_α.
- For all $(a, b) \in N \cap (A_0 \times B_0)$ do: add edges (a^-_ℓ, b^+_r) and (b^-_ℓ, a^+_r) to N^*_α.

For each u such that $(u, u) \in N$, add (u_ℓ^-, u_r^+) to N^*_α. Using the constraints that $\vec{\alpha}$ has to satisfy, it is easy to argue that N^*_α is a stable matching in H. Moreover, the fact that N is a fully popular matching in G implies that N^*_α is a *legal* matching in H: since N is A-popular (resp., popular) in G, every edge used in N is valid (resp., popular). So N^*_α has no forbidden edge. Thus N^*_α is a legal stable matching in H. ◀

A variant of Gale-Shapley algorithm. A stable matching that avoids all forbidden edges (if such a matching exists) can be computed in linear time by running a variant of Gale-Shapley algorithm in H where proposals made along forbidden edges are rejected. Once a proposal made along a forbidden edge is rejected by a vertex, all further proposals made on *worse* edges also have to rejected by this vertex. If some vertex is left unmatched at the end of this algorithm, then there is no stable matching in H that avoids all forbidden edges; else we have a desired stable matching in H. We refer to [13] for details on this variant of Gale-Shapley algorithm.

Thus it can be efficiently checked if H admits a legal stable matching or not. If such a matching does not exist in H then there is no fully popular matching in G (by Lemma 12). So we will assume henceforth that there exists a legal stable matching in H. However the fact that such a stable matching exists in H does not imply that G admits a fully popular matching. This is because any matching M^* in H can only be mapped to a *half-integral* matching in G.

In order to claim the resulting matching in G is integral, we need M^* to be symmetric, i.e., have the same edges in both halves of H. We will not construct such a symmetric stable matching in H. The matching we compute will have a certain amount of symmetry and this will be enough to obtain a fully popular matching in G. If H does not admit such a "partially symmetric" stable matching, then we show that G has *no* fully popular matching.

3.1 Two partitions of the vertex set

We run Gale-Shapley algorithm that avoids all forbidden edges [13] in H. In this algorithm, vertices on the left of H propose and vertices on the right of H dispose. When $u_\ell \in A_L \cup B_L$ proposes to v_r^-, this proposal is made along (u_ℓ^+, v_r^-): so v_r sees this as u_ℓ^+'s proposal; when u_ℓ proposes to v_r^+, this proposal is made along (u_ℓ^-, v_r^+): so v_r sees this as u_ℓ^-'s proposal. If u_ℓ proposes to a neighbor v_r along (u_ℓ^+, v_r^-) or (u_ℓ^-, v_r^+), then v_r accepts u_ℓ's proposal only if the edge (u, v) is legal; otherwise v_r rejects u_ℓ's proposal since this is a forbidden edge. Edges ranked worse than $(u_\ell^+, v_r^-)/(u_\ell^-, v_r^+)$ (as the case may be) will be deleted from the current instance on whose edges proposals are made – this ensures that once v_r receives a proposal along a certain edge, whether this proposal is accepted or not, v_r cannot accept proposals made along *worse* edges. Let S_0 be the legal stable matching in H that is obtained.

- Let $U_A \subseteq A$ be the set of agents a such that $(a_\ell^-, a_r^+) \in S_0$ and let $U_B \subseteq B$ be the set of jobs b such that $(b_\ell^-, b_r^+) \in S_0$.

We know that H is made up of two halves: the upper half and the lower half. Since S_0 is stable and thus perfect (recall that any stable matching in H is perfect), the set of agents matched to *genuine* neighbors – not to their twins – in each half of H is $A \setminus U_A$ and similarly, the set of jobs matched to genuine neighbors in each half of H is $B \setminus U_B$. We now form sets $A_+, A_-, B_+, B_-, A'_+, A'_-, B'_+, B'_-$ corresponding to S_0: initially they are empty.

- For every $(a^+_\ell, b^-_r) \in S_0$ where $a \in A$ and $b \in B$: add a to A_+ and b to B_-.
- For every $(a^-_\ell, b^+_r) \in S_0$ where $a \in A$ and $b \in B$: add a to A_- and b to B_+.
- For every $(b^+_\ell, a^-_r) \in S_0$ where $a \in A$ and $b \in B$: add b to B'_+ and a to A'_-.
- For every $(b^-_\ell, a^+_r) \in S_0$ where $a \in A$ and $b \in B$: add b to B'_- and a to A'_+.

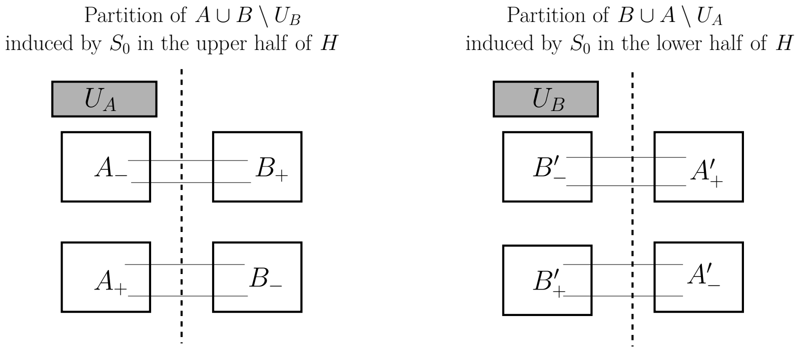

Partition of $A \cup B \setminus U_B$
induced by S_0 in the upper half of H
Partition of $B \cup A \setminus U_A$
induced by S_0 in the lower half of H
U_A
A_-
B_+
A_+
B_-
U_B
B'_-
A'_+
B'_+
A'_-

Figure 4 Two partitions of the set $(A \cup B) \setminus (U_A \cup U_B)$ induced by the matching S_0.

We have $A \setminus U_A = A_+ \cup A_- = A'_+ \cup A'_-$ and $B \setminus U_B = B_+ \cup B_- = B'_+ \cup B'_-$. Fig. 4 denotes these partitions of $A \setminus U_A$ and $B \setminus U_B$ induced by the matching S_0 in the upper and lower halves of H. In this figure, the set U_A has been included in the upper half and the set U_B in the lower half.

We will use $(a^-_\ell, *)$ to denote any edge in the set $\{(a^-_\ell, b^+_r) : b \in B\} \cup \{(a^-_\ell, a^+_r)\}$. Similarly $(*, a^+_r)$ denotes any edge in the set $\{(b^-_\ell, a^+_r) : b \in B\} \cup \{(a^-_\ell, a^+_r)\}$. Similarly for $(b^-_\ell, *)$ and $(*, b^+_r)$. Recall that every popular matching has a *witness* $\vec{\alpha} \in \{0, \pm 1\}^n$ (see Theorem 6).

▶ **Lemma 13.** *Let N be a fully popular matching in G and let $\vec{\alpha}$ be any witness of N. If $a \in A_- \cap A'_+$ then $\alpha_a = 0$.*

Proof. We have vertex $a \in A_- \cap A'_+$, where the sets A_- and A'_+ are defined above. Let $\mathcal{D}_0$ be the set of legal stable matchings in H. The set $\mathcal{D}_0$ forms a sublattice of the lattice[1] of stable matchings in H and the matching S_0 is the $(A_L \cup B_L)$-*optimal* matching in $\mathcal{D}_0$ [13]. Since $a \in A_-$, we have $(a^-_\ell, c^+_r) \in S_0$ for some neighbor c^+_r of a_ℓ. Thus c^+_r is the most preferred partner for $a_\ell \in A_L$ in all matchings in $\mathcal{D}_0$. Recall that every vertex prefers superscript $-$ neighbors to superscript $+$ neighbors. Hence no matching in $\mathcal{D}_0$ matches a_ℓ to a superscript $-$ neighbor, i.e., every legal stable matching in H has to contain $(a^-_\ell, *)$.

S_0 is also the $(A_R \cup B_R)$-*pessimal* matching in $\mathcal{D}_0$ [13]. Since $a \in A'_+$, we have $(d^-_\ell, a^+_r) \in S_0$ for some neighbor d^-_ℓ of a_r. So every matching in $\mathcal{D}_0$ has to match $a_r \in A_R$ to a neighbor at least as good as d^-_ℓ, i.e., every legal stable matching in H has to contain $(*, a^+_r)$.

Suppose N is a fully popular matching with a witness $\vec{\alpha}$ such that $\alpha_a \in \{\pm 1\}$. If $\alpha_a = 1$, i.e., if $a \in A_1$ (see Fig. 3), then there is a legal stable matching N^*_α in H such that $(a^+_\ell, *) \in N^*_\alpha$ (see the proof of Lemma 12). This contradicts our claim above that every legal stable matching in H has to contain $(a^-_\ell, *)$. So $\alpha_a = -1$, i.e., $a \in A_{-1}$. Then there is a legal stable matching N^*_α in H such that $(*, a^-_r) \in N^*_\alpha$ (by the proof of Lemma 12). This again contradicts our claim above that every legal stable matching in H has to contain $(*, a^+_r)$. Thus $\alpha_a \notin \{\pm 1\}$, hence $\alpha_a = 0$. ◀

[1] The meet of 2 stable matchings M and M' is the stable matching where every u in $A_L \cup B_L$ (resp., $A_R \cup B_R$) is matched to its *more* (resp., *less*) preferred partner in $\{M(u), M'(u)\}$. The join of M and M' is the stable matching where every u in $A_L \cup B_L$ (resp., $A_R \cup B_R$) is matched to its *less* (resp., *more*) preferred partner in $\{M(u), M'(u)\}$.

The proof of Lemma 14 is analogous to the proof of Lemma 13.

▶ **Lemma 14.** *Let N be a fully popular matching in G and let $\vec{\alpha}$ be any witness of N. If $b \in B_+ \cap B'_-$ then $\alpha_b = 0$.*

We will use $G_0 = (A \cup B, E_0)$ to denote the *popular subgraph* of G. The edge set E_0 of G_0 is the set of popular edges, i.e., those present in some popular matching in G. The subgraph G_0 need not be connected and Lemma 15 will be useful to us.

▶ **Lemma 15** ([8]). *Let C be any connected component in the popular subgraph G_0. For any popular matching N in G and any witness $\vec{\alpha}$ of N: if $\alpha_v = 0$ for some $v \in C$ then $\alpha_u = 0$ for all $u \in C$.*

Proof. Consider any popular edge (a, b). So there is some popular matching M with the edge (a, b). The matching M is an optimal solution to the max-weight perfect matching LP with edge weight function wt_N since $\mathsf{wt}_N(M) = \phi(M, N) - \phi(N, M) = 0$: recall that M and N are popular matchings in G. We know that $\vec{\alpha}$ is an optimal solution to the dual LP. So it follows from complementary slackness conditions that $\alpha_a + \alpha_b = \mathsf{wt}_N(a, b)$. Since $\mathsf{wt}_N(a, b) \in \{\pm 2, 0\}$ (an even number), the integers α_a and α_b have the same parity.

Let u and v be any 2 vertices in the same connected component in the popular subgraph G_0. So there is a u-v path ρ in G such that every edge in ρ is a popular edge. We have just seen that the endpoints of each popular edge have the same parity in $\vec{\alpha}$. Hence α_u and α_v have the same parity. Thus $\alpha_v = 0$ implies $\alpha_u = 0$. ◀

3.2 Our algorithm

Lemmas 13-15 motivate our algorithm which is described as Algorithm 1. The main step of the algorithm is the *while* loop that takes any unmarked vertex v in $(A_- \cap A'_+) \cup (B_+ \cap B'_-)$. Initially all vertices are unmarked. Consider the first iteration of the algorithm: let $v \in A$.

Lemma 13 tells us that for any fully popular matching N and any witness $\vec{\alpha}$ of N, we have $\alpha_v = 0$. Lemma 15 tells us that $\alpha_u = 0$ for every vertex u in the component C, where C is v's connected component in G_0. The proof of Lemma 12 shows N has a realization N^*_α in H such that N^*_α contains $(a^-_\ell, *)$ and $(*, a^+_r)$ for every agent $a \in C$.

Thus we are interested in those legal stable matchings in H that contain $(a^-_\ell, *)$ and $(*, a^+_r)$ for every agent $a \in C$. Hence our algorithm *forbids* all edges $(a^+_\ell, *)$ and $(*, a^-_r)$ for every agent $a \in C$ in the stable matching that we compute here. This step is implemented by making every neighbor reject offers from a^+_ℓ (this may induce other rejections) and symmetrically, a_r rejects all offers from superscript + neighbors. Note that the resulting matching may contain (a^-_ℓ, a^+_r) for some of the agents a in C. All vertices in C get marked in this iteration.

Let $\mathcal{D}_1 \subseteq \mathcal{D}_0$ be the set of all legal stable matchings in H that contain $(a^-_\ell, *)$ and $(*, a^+_r)$ for every agent $a \in C$. Thus $\mathcal{D}_1$ is a sublattice of $\mathcal{D}_0$. We know from the proof of Lemma 12 that $N^*_\alpha \in \mathcal{D}_1$ where N is a fully popular matching in G and $\vec{\alpha}$ any witness of N. So if $\mathcal{D}_1$ is empty then we can conclude that G has *no* fully popular matching.

Algorithm 1 *Input: $G = (A \cup B, E)$ with strict preferences.*

```
1:  Compute a legal stable matching $S_0$ in $H$ by running Gale-Shapley algorithm with
    forbidden edges.
    {Vertices in $A_L \cup B_L$ propose and those in $B_R \cup A_R$ dispose.}
2:  Let $A_-, A'_+$ and $B_+, B'_-$ be as defined earlier (see the start of Section 3.1).
3:  Initially all vertices are unmarked and $i = 0$.
4:  while there exists an unmarked vertex $v \in (A_- \cap A'_+) \cup (B_+ \cap B'_-)$ do
5:      $i = i + 1$.
6:      Modify $S_{i-1}$ to $S_i$ so as to forbid all edges $(a^+_\ell, *)$ and $(*, a^-_r)$ for every agent $a$ in $v$'s
        component in the popular subgraph $G_0$.
        ($S_i$ is the $(A_L \cup B_L)$-optimal legal stable matching in $H$ that avoids all forbidden edges
        identified in the first $i$ iterations of the while-loop.)
7:      if there is no such legal stable matching $S_i$ in $H$ then
8:          Return "No fully popular matching in $G$".
9:      end if
10:     Update the sets $A_-, A'_+$ and $B_+, B'_-$: these correspond to $S_i$ now.
11:     Mark all vertices in $v$'s component in the popular subgraph $G_0$.
12: end while
13: Return $M = \{(a, b) \in E : (a^+_\ell, b^-_r)$ or $(a^-_\ell, b^+_r)$ is in $S_i\}$.
```

Let us assume we are now in the i-th iteration and let $\mathcal{D}_i$ be the set of legal stable matchings in H that avoid all edges forbidden by our algorithm in the first i iterations. In other words, $\mathcal{D}_i$ is the set of those matchings in $\mathcal{D}_{i-1}$ where no edge forbidden in the i-th iteration is present. We have $\mathcal{D}_0 \supseteq \mathcal{D}_1 \supseteq \cdots \supseteq \mathcal{D}_{i-1} \supseteq \mathcal{D}_i$. For all $0 \le j \le i$, the set $\mathcal{D}_j$ forms a sublattice of the lattice of all stable matchings in H [13].

▶ **Lemma 16.** *For every fully popular matching N in G and every witness $\vec{\alpha}$ of N, the realization N^*_α is an element of $\mathcal{D}_i$.*

Proof. We will prove the lemma by induction. We know from Lemma 12 that the base case is true, i.e., $N^*_\alpha \in \mathcal{D}_0$. By induction hypothesis, let us assume that for every fully popular matching N and every witness $\vec{\alpha}$, the realization N^*_α is an element of $\mathcal{D}_{i-1}$. Since the algorithm entered the i-th iteration of the while loop, there was an unmarked vertex x in $(A_- \cap A'_+) \cup (B_+ \cap B'_-)$ at the start of this iteration.

▷ Claim 17. For any fully popular matching N and any witness $\vec{\alpha}$ of N, we have $\alpha_x = 0$.

Proof. The matching S_{i-1} computed in Step 6 of the $(i-1)$-th iteration is the $(A_L \cup B_L)$-optimal matching in the lattice $\mathcal{D}_{i-1}$. Hence if $(x^-_\ell, *) \in S_{i-1}$ for some $x_\ell \in A_L \cup B_L$ then $(x^-_\ell, *)$ belongs to every matching in $\mathcal{D}_{i-1}$. The matching S_{i-1} is also the $(A_R \cup B_R)$-pessimal matching in the set $\mathcal{D}_{i-1}$. Hence if $(*, x^+_r) \in S_{i-1}$ for some $x_r \in A_R \cup B_R$ then $(*, x^+_r)$ belongs to every matching in $\mathcal{D}_{i-1}$.

If the above claim is false then there is a fully popular matching N and a witness $\vec{\alpha}$ of N with $\alpha_x \in \{\pm 1\}$. If $\alpha_x = 1$ then there is a legal stable matching N^*_α in H such that $(x^+_\ell, *) \in N^*_\alpha$. If $\alpha_x = -1$ then there is a legal stable matching N^*_α in H such that $(*, x^-_r) \in N^*_\alpha$. Since $N^*_\alpha \in \mathcal{D}_{i-1}$, both cases contradict our earlier observation that every matching in $\mathcal{D}_{i-1}$ has to contain $(x^-_\ell, *)$ and $(*, x^+_r)$. Thus for any fully popular matching N and any witness $\vec{\alpha}$ of N, we have $\alpha_x = 0$. ◁

Claim 17 along with Lemma 15 tells us that for all vertices u in x's component C' in G_0, we have $\alpha_u = 0$. The proof of Lemma 12 shows us that N^*_α contains $(a^-_\ell, *)$ and $(*, a^+_r)$ for every agent $a \in C'$. Since $N^*_\alpha \in \mathcal{D}_{i-1}$, it follows that N^*_α is an element in $\mathcal{D}_i$. This finishes the proof of this lemma. ◀

Hence if $\mathcal{D}_i = \emptyset$, i.e., if the algorithm says "no" in Step 8, then there is indeed no fully popular matching in G. This finishes one part of our proof of correctness. We now need to show that if our algorithm returns a matching M, then M is a fully popular matching in G.

4 Correctness of our algorithm

In this section we show that the matching returned by Algorithm 1 is a fully popular matching in G. Let S_i be the matching in H computed in the final iteration of Algorithm 1. Let M be the matching (in G) induced by S_i in the upper half of H: this is as defined in Step 13 of Algorithm 1.

Note that $M \subseteq (A_+ \times B_-) \cup (A_- \times B_+)$, where the sets A_+, B_-, A_-, B_+ are defined at the beginning of Section 3.1: the matching S_i replaces S_0 in these definitions now. Similarly, let L be the matching (in G) induced by S_i in the lower half of H. So

$$L = \{(a,b) \in E : (b^+_\ell, a^-_r) \text{ or } (b^-_\ell, a^+_r) \text{ is in } S_i\}.$$

Thus $L \subseteq (A'_+ \times B'_-) \cup (A'_- \times B'_+)$. Let U_A (resp., U_B) be the set of vertices u in A (resp., B) such that $(u^-_\ell, u^+_r) \in S_i$. The vertices in $U_A \cup U_B$ are unmatched in both M and L. Since S_i is a legal stable matching in H, it matches all vertices in H using valid edges. Thus by Theorem 5, M is A-popular.[2] We need to show that M is popular in G. Theorem 18 will be our starting point.

▶ **Theorem 18.** *The matching M is popular in the subgraph $G \setminus U_B$. Also, the matching L is popular in the subgraph $G \setminus U_A$.*

Proof. The popularity of L in $G \setminus U_A$ will be shown using the witness $\vec{\beta}$ defined below and the popularity of M in $G \setminus U_B$ will be shown using the witness $\vec{\gamma}$ defined below.

1. $\beta_u = 1$ for $u \in A'_+ \cup B'_+$, $\beta_u = -1$ for $u \in A'_- \cup B'_-$, and $\beta_u = 0$ for $u \in U_B$.
2. $\gamma_u = 1$ for $u \in A_+ \cup B_+$, $\gamma_u = -1$ for $u \in A_- \cup B_-$, and $\gamma_u = 0$ for $u \in U_A$.

Since $L \subseteq (A'_+ \times B'_-) \cup (A'_- \times B'_+)$, we have $\sum_{u \in (A \cup B) \setminus U_A} \beta_u = 0$. Note that $\mathsf{wt}_L(u,u) = 0$ for $u \in U_B$ and $\mathsf{wt}_L(u,u) = -1$ for all other u. Thus $\beta_u \geq \mathsf{wt}_L(u,u)$ for all $u \in (A \cup B) \setminus U_A$.

Similarly, $\sum_{u \in (A \cup B) \setminus U_B} \gamma_u = 0$. Also, $\gamma_u \geq \mathsf{wt}_M(u,u)$ for all $u \in (A \cup B) \setminus U_B$.

▷ Claim 19. $\beta_a + \beta_b \geq \mathsf{wt}_L(a,b)$ for all edges (a,b) where $a \in A \setminus U_A$ and $b \in B$.

▷ Claim 20. $\gamma_a + \gamma_b \geq \mathsf{wt}_M(a,b)$ for all edges (a,b) where $a \in A$ and $b \in B \setminus U_B$.

We will prove Claim 20 below. The proof of Claim 19 is analogous.

- Consider $a \in U_A$. We set $\gamma_a = 0$ and we know that $(a^-_\ell, a^+_r) \in S_i$. Recall that a^+_r is a_ℓ's least preferred neighbor, thus a_ℓ must have been rejected by all its more preferred neighbors in our variant of the Gale-Shapley algorithm in H. That is, every neighbor b^+_r of a_ℓ received a proposal from a^-_ℓ. Since b_r prefers superscript $-$ neighbors to superscript $+$

[2] In order to apply Theorem 5, we ought to say $M \cup \{(u,u) : u \in U_A \cup U_B\}$ is A-popular.

neighbors, this means $(d_\ell^-, b_r^+) \in S_i$ for some neighbor d_ℓ^- that b_r prefers to a_ℓ^-, i.e., b prefers d to a. Thus $b \in B_1$ (so $\gamma_b = 1$) and moreover, $\mathsf{wt}_M(a,b) = 0$. Hence we have $\gamma_a + \gamma_b = 1 > \mathsf{wt}_M(a,b)$.

- We will next show this constraint holds for all edges (a,b) incident to $a \in A_-$. There are 2 cases: (1) $b \in B_-$ and (2) $b \in B_+$. In case (1), we have (a_ℓ^-, c_r^+) and (d_ℓ^+, b_r^-) in S_i. Since every vertex prefers superscript $-$ neighbors to superscript $+$ neighbors, it means a_ℓ proposed to b_r^- and got rejected, i.e., b_r prefers its partner d_ℓ^+ to a_ℓ^+. We also claim a_ℓ prefers its partner c_r^+ to b_r^+. This is because b_r prefers a_ℓ^- to d_ℓ^+ (superscript $-$ neighbors over superscript $+$ neighbors): so if a_ℓ^- had proposed to b_r, then b_r would have rejected its partner d_ℓ^+. This means both a and b prefers their partners in M to each other. Thus $\mathsf{wt}_M(a,b) = -2 = \gamma_a + \gamma_b$.
  In case (2) above, either (i) $(a_\ell^-, b_r^+) \in S_i$ or (ii) (a_ℓ^-, c_r^+) and (d_ℓ^-, b_r^+) are in S_i: since S_i is stable, a_ℓ prefers c_r^+ to b_r^+ or b_r prefers d_ℓ^- to a_ℓ^-. Thus $\mathsf{wt}_M(a,b) \le 0 = \gamma_a + \gamma_b$.
- Finally, we will show this constraint holds for all edges (a,b) incident to $a \in A_+$. There are 2 cases: $b \in B_+$ and $b \in B_-$. In the first case, we have $\gamma_a + \gamma_b = 2$ and since $\mathsf{wt}_M(a,b) \le 2$, the constraint $\mathsf{wt}_M(a,b) \le \gamma_a + \gamma_b$ obviously holds.
  In the second case, either (i) $(a_\ell^+, b_r^-) \in S_i$ or (ii) (a_ℓ^+, c_r^-) and (d_ℓ^+, b_r^-) are in S_i: since S_i is stable, a_ℓ prefers c_r^- to b_r^- or b_r prefers d_ℓ^+ to a_ℓ^+. Thus $\mathsf{wt}_M(a,b) \le 0 = \gamma_a + \gamma_b$.

This finishes the proof of M's popularity in $G \setminus U_B$ (by Theorem 6). ◀

Thus the matching M is popular in the subgraph $G \setminus U_B$. However we need to prove the popularity of M in the *entire* graph G, i.e., we need to include vertices in U_B as well. Setting $\gamma_b = 0$ for $b \in U_B$ will not cover edges in $A_- \times U_B$. Now we will use the fact that L is popular in $G \setminus U_A$ and that M and L have several edges in common (as shown in Lemma 22).

Let Z be the set of all vertices outside $U_A \cup U_B$ that got marked in our algorithm. So these are the marked vertices that are matched in S_i to genuine neighbors (not to their twins). Since we marked entire connected components in the popular subgraph G_0 in Algorithm 1, both M and L match vertices in Z to each other.

Lemma 22 shows that the matching S_i has "partial symmetry" across the upper and lower halves of the graph H; more precisely, M and L are identical on the set Z. This will be key to showing M's popularity. The following lemma will be useful in proving Lemma 22.

▶ **Lemma 21.** *M and L are stable matchings when restricted to vertices in $Z \cup U_A \cup U_B$.*

Proof. Let $Z_A = Z \cap A$ and let $Z_B = Z \cap B$. It follows from our algorithm that $Z_A \subseteq A_- \cap A'_+$ and $Z_B \subseteq B_+ \cap B'_-$ (see Fig. 5).

We need to show that M (similarly, L) has no blocking edge in $(Z_A \cup U_A) \times (Z_B \cup U_B)$. Consider any edge $(a,b) \in Z_A \times Z_B$. We know from Claim 20 (in the proof of Theorem 18) that $\mathsf{wt}_M(a,b) \le \gamma_a + \gamma_b = -1 + 1 = 0$. Similarly, $\mathsf{wt}_L(a,b) \le \beta_a + \beta_b = 1 - 1 = 0$. Thus (a,b) is not a blocking edge to either M or L.

Thus neither M nor L has a blocking edge in $Z_A \times Z_B$. Moreover, G has no edge in $U_A \times U_B$. This is because each vertex $u \in U_A \cup U_B$ is unstable – otherwise (u_ℓ^-, u_r^+) is an unpopular edge and thus forbidden. Consider any edge $(a,b) \in U_A \times Z_B$. We have $\mathsf{wt}_M(a,b) \le \gamma_a + \gamma_b = 0 + 1 = 1$. Since $\mathsf{wt}_M(a,b)$ is an even number, this means $\mathsf{wt}_M(a,b) \le 0$. Thus (a,b) is not a blocking edge to M.

We will now show that (a,b) is not a blocking edge to L. Since $a \in U_A$ and $b \in Z_B \subseteq B'_-$, we have (a_ℓ^-, a_r^+) and (b_ℓ^-, c_r^+) in S_i, where c is some neighbor of b. Note that a_ℓ^- is a_r^+'s least preferred superscript $-$ neighbor in H. Thus a_r^+ did not receive any offer from b_ℓ^- in Algorithm 1. Because S_i is stable, it has to be the case that b_ℓ prefers c_r^+ to a_r^+. Since $(c,b) \in L$, we have $\mathsf{wt}_L(a,b) = 0$. Thus (a,b) is not a blocking edge to L.

It can similarly be shown that no edge in $Z_A \times U_B$ blocks either M or L. Thus M and L are stable matchings when restricted to vertices in $Z \cup U_A \cup U_B$. ◀

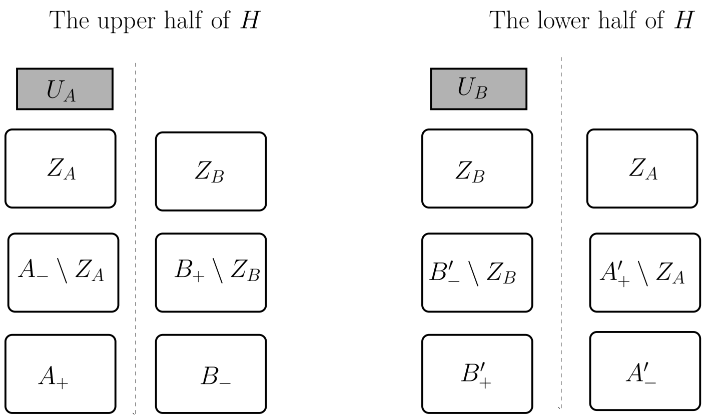


Figure 5 The final picture of the partitions created by M and L in the upper and lower halves of H, resp. The while-loop termination condition implies $(A_- \setminus Z_A) \subseteq A'_-$ and $(A'_+ \setminus Z_A) \subseteq A_+$, so on.

▶ **Lemma 22.** *The matching M restricted to vertices in Z is the same as the matching L restricted to vertices in Z.*

Proof. Consider any connected component C in the popular subgraph G_0. The component C splits into sub-components $C'_1, \ldots, C'_t$ when we restrict edges to only those marked "valid". We claim there is exactly *one* stable matching $T_{C'_j}$ in each such sub-component C'_j. Assume C'_j contains a job b that is a top choice neighbor for some agent.[3] Then b has to be matched in $T_{C'_j}$ to its most preferred neighbor a in C'_j, otherwise (a, b) would be a blocking edge to $T_{C'_j}$. Recall that every agent has exactly 2 valid edges incident to it. So fixing one edge (a, b) in the matching fixes $T_{C'_j}$.

In more detail, every agent $a' \neq a$ in C'_j such that $f(a') = b$ has to be matched in $T_{C'_j}$ to $s(a')$ (call it b'). Given that a' is matched to b', every agent $a'' \neq a'$ in C'_j such that $s(a'') = b'$ has to be matched in $T_{C'_j}$ to $f(a'')$ and so on. Thus the matching $T_{C'_j}$ gets fixed. The same happens with every sub-component in C and so the only stable matching in C is $T_C = \cup_{j=1}^{t} T_{C'_j}$.

Let $C_1, \ldots, C_r$ be the connected components of G_0 that contain vertices in Z. So all vertices in $\cup_{i=1}^{r} C_i$ are marked, thus $\cup_{i=1}^{r} C_i \subseteq Z \cup U_A \cup U_B$. We know from Lemma 21 that both M and L are stable matchings in each C_i, where $1 \leq i \leq r$. So M (similarly, L) restricted to $\cup_{i=1}^{r} C_i$ is $\cup_{i=1}^{r} T_{C_i}$. Thus M and L have the same edges on Z. ◀

Lemma 22 helps us in defining an appropriate witness $\vec{\alpha}$ to show M's popularity in G. Recall $\vec{\gamma}$ from Theorem 18: we will set $\alpha_u = 0$ for all $u \in Z \cup U_B$ and $\alpha_u = \gamma_u$ otherwise. Before we prove the popularity of M in Theorem 25, we need the following two lemmas.

▶ **Lemma 23.** *For every $a \in A_- \setminus Z_A$, a likes $M(a)$ at least as much as $L(a)$.*

Proof. Suppose not. Then $M(a) = s(a)$ while $L(a) = f(a)$. We claim $f(a) \in B_+$. Otherwise $f(a) \in B_-$, however for every edge $(x, y) \in A_- \times B_-$, we have $\mathsf{wt}_M(x, y) \leq \gamma_x + \gamma_y = -2$. But a prefers $f(a)$ to its partner in M, thus $\mathsf{wt}_M(a, f(a)) \geq 0$. Hence $f(a) \in B_+$. Since $\mathsf{wt}_M(x, y) \leq 0$ for every edge $(x, y) \in A_- \times B_+$, we can conclude that $\mathsf{wt}_M(a, f(a)) = 0$, i.e., $f(a)$ is matched in M to a neighbor $a' \in A_-$ that it prefers to a. Since S_i uses only valid edges, this means $f(a) = f(a')$, i.e., $f(a)$ is the top choice neighbor of a'.

[3] Otherwise C'_j consists of a single edge $(a, s(a))$ for some $a \in A$; if there was another agent u in C'_j then $s(u) = s(a)$ and so one of a, u would be left unmatched in S_i, a contradiction to S_i's stability in H.

We now move to the lower half of H: observe that both a and a' are in A'_-. This is because there is no unmarked vertex in $A_- \cap A'_+$ by the termination condition of our while-loop. Note that a is unmarked since $a \notin Z_A$. Thus a' is also unmarked since $(a, f(a))$ and $(a', f(a))$ are popular edges – hence a and a' are in the same connected component in G_0. Since $a \in A'_-$, $L(a) = f(a)$ is in B'_+. Consider the edge $(a', f(a)) \in A'_- \times B'_+$: both a' and $f(a)$ prefer each other to their respective partners in L. This means $\mathsf{wt}_L(a', f(a)) = 2$. However for each edge $(x, y) \in A'_- \times B'_+$, we have $\mathsf{wt}_L(x, y) \leq \beta_x + \beta_y = 0$, a contradiction. So any $a \in A_- \setminus Z_A$ likes $M(a)$ at least as much as $L(a)$. ◀

▶ **Lemma 24.** *For every $a \in A_+ \cap A'_+$, a likes $M(a)$ at least as much as $L(a)$.*

Proof. Suppose not. Then $M(a) = s(a)$ while $L(a) = f(a)$. Since $a \in A'_+$, $L(a) = f(a) \in B'_-$. This implies $f(a) \in B_-$ since there is no unmarked vertex in $B_+ \cap B'_-$ by the termination condition of our while-loop. We know $f(a)$ is unmarked since a (its partner in L) is unmarked and this is because $a \in A_+$. Since $a \in A_+$ and $f(a) \in B_-$, we have $\mathsf{wt}_M(a, f(a)) \leq \gamma_a + \gamma_b = 0$ and so $f(a)$ has to be matched in M to a more preferred neighbor $a' \in A_+$. As argued in the proof of Lemma 23, it follows from the legality of S_i that $f(a)$ is the top choice neighbor of a'.

Consider the matching L in the lower half of H. Since $L(a) = f(a)$, $\mathsf{wt}_L(a', f(a)) = 2$. That is, $(a', f(a))$ is a blocking edge to L. We need $\beta_{a'} = \beta_{f(a)} = 1$ to ensure $\beta_{a'} + \beta_{f(a)} \geq \mathsf{wt}_L(a', f(a)) = 2$. However $f(a) \in B'_-$ since $a \in A'_+$. This means $\beta_{f(a)} = -1$, a contradiction. Thus any $a \in A_+ \cap A'_+$ likes $M(a)$ at least as much as $L(a)$. ◀

▶ **Theorem 25.** *The matching M is popular in G.*

Proof. The popularity of M in G will be shown using $\vec{\alpha}$ defined below: (recall that $Z_A = Z \cap A$ and $Z_B = Z \cap B$)

- set $\alpha_u = 0\ \forall\, u \in Z \cup U_A \cup U_B$.
- set $\alpha_u = 1\ \forall\, u \in A_+ \cup (B_+ \setminus Z_B)$ and $\alpha_u = -1\ \forall\, u \in B_- \cup (A_- \setminus Z_A)$.

Since $M \subseteq (A_+ \times B_-) \cup (Z_A \times Z_B) \cup ((A_- \setminus Z_A) \times (B_+ \setminus Z_B))$ (see Fig. 5), we have $\sum_{u \in A \cup B} \alpha_u = 0$. Also $\alpha_u \geq \mathsf{wt}_M(u, u)$ for all vertices $u \in A \cup B$ since $\alpha_u = 0 = \mathsf{wt}_M(u, u)$ for $u \in U_A \cup U_B$ and $\alpha_u \geq -1 = \mathsf{wt}_M(u, u)$ for all other u. To use Theorem 6, we need to show $\alpha_a + \alpha_b \geq \mathsf{wt}_M(a, b)$ for all edges (a, b).

We will first show this constraint holds for edges incident to vertices in U_B. It is easy to show that the neighborhood of U_B is in A'_+ and also that each $a \in A'_+$ prefers its partner in L to $b \in U_B$. This is because $(b^-_\ell, b^+_r) \in S_i$ and b^+_r is b_ℓ's least preferred neighbor, thus b_ℓ must have been rejected by all its more preferred neighbors in our algorithm, i.e., every neighbor a^+_r of b_ℓ received a proposal from b^-_ℓ. Since a_r prefers superscript $-$ neighbors to superscript $+$ neighbors, this means $(c^-_\ell, a^+_r) \in S_i$ for some neighbor c^-_ℓ that a_r prefers to b^-_ℓ, i.e., a prefers c to b. Thus $a \in A'_+$.

We have $A'_+ = Z_A \cup (A'_+ \setminus Z_A)$ and $A'_+ \setminus Z_A \subseteq A_+$ (by the while-loop termination condition). Lemma 22 and Lemma 24 showed that for $a \in Z_A \cup (A_+ \cap A'_+)$, we have $M(a) \succeq_a L(a)$, i.e., a likes $M(a)$ at least as much as $L(a)$, and we showed above that each $a \in A'_+$ prefers $L(a)$ to b. Thus $\mathsf{wt}_M(a, b) = 0$. Since we set $\alpha_a = 0$ for $a \in Z_A$ and $\alpha_a = 1$ for $a \in A_+$, we have $\alpha_a + \alpha_b \geq 0 = \mathsf{wt}_M(a, b)$.

We now need to show $\alpha_a + \alpha_b \geq \mathsf{wt}_M(a, b)$ holds for all edges (a, b) in $G \setminus U_B$. Recall the witness $\vec{\gamma}$ defined in the proof of Theorem 18 to show the popularity of M in the subgraph $G \setminus U_B$. Observe that it is only for vertices u in Z that we have $\alpha_u \neq \gamma_u$. Moreover, $\alpha_a > \gamma_a$ for $a \in Z_A$. Thus we only have to worry about edges (a, b) in $G \setminus U_B$ where $b \in Z_B$ and check that $\mathsf{wt}_M(a, b) \leq \alpha_a + \alpha_b$. All other edges in $G \setminus U_B$ are covered by $\vec{\alpha}$ (since $\vec{\gamma}$ covers these edges). Let $b \in Z_B \subseteq B_+ \cap B'_-$.

1. Suppose $a \in U_A \cup Z_A$. For any $(a,b) \in (U_A \cup A_-) \times B_+$, we have $\mathsf{wt}_M(a,b) \le \gamma_a + \gamma_b \le 0+1$. Since $\mathsf{wt}_M(a,b)$ is an even number, this means $\mathsf{wt}_M(a,b) \le 0 = \alpha_a + \alpha_b$.
2. Suppose $a \in A_- \setminus Z_A$. Then $a \in A'_-$ by the termination condition of the while-loop in our algorithm. Since $\mathsf{wt}_L(x,y) \le \beta_x + \beta_y = -2$ for every edge $(x,y) \in A'_- \times B'_-$, it follows that $b \in Z_B \subseteq B'_-$ prefers $L(b)$ to a and similarly, $a \in A'_-$ prefers $L(a)$ to b.
   We know from Lemma 22 that $M(b) = L(b)$, so b prefers $M(b)$ to a. We know from Lemma 23 that $M(a) \succeq_a L(a)$ (i.e., a likes $M(a)$ at least as much as $L(a)$), so a prefers $M(a)$ to b. Thus $\mathsf{wt}_M(a,b) = -2 < \alpha_a + \alpha_b$ since $\alpha_a = -1$ and $\alpha_b = 0$.
3. Suppose $a \in A_+$. There are two sub-cases here: (i) $a \in A'_-$ and (ii) $a \in A'_+$. In sub-case (i), $\mathsf{wt}_L(a,b) \le \beta_a + \beta_b = -2$. Since $M(b) = L(b)$ (by Lemma 22), it means that b prefers $M(b)$ to a. Hence $\mathsf{wt}_M(a,b) \le 0 < \alpha_a + \alpha_b$ since $\alpha_a = 1$ and $\alpha_b = 0$ here.
   Consider sub-case (ii). We have $\mathsf{wt}_L(a,b) \le \beta_a + \beta_b = 0$. So either (1) b prefers $L(b)$ to a or (2) a prefers $L(a)$ to b. In case (1), we have $\mathsf{wt}_M(a,b) \le 0$ since $M(b) = L(b)$ (by Lemma 22). In case (2) also, we have $\mathsf{wt}_M(a,b) \le 0$ since $M(a) \succeq_a L(a)$ (by Lemma 24). So in both cases we have: $\mathsf{wt}_M(a,b) \le 0 < \alpha_a + \alpha_b$ since $\alpha_a = 1$ and $\alpha_b = 0$ here.

Thus $\vec{\alpha}$ is a witness of M's popularity (by Theorem 6). So M is popular in G. ◀

Since M is A-popular, Theorem 25 immediately implies that M is fully popular in G. Moreover, M is a *max-size* fully popular matching in G, as shown below.

▶ **Lemma 26.** *The matching M is a max-size fully popular matching in G.*

Proof. Observe that $U_A \cup U_B$ is the set of vertices left unmatched in the matching M. We claim the vertices in $U_A \cup U_B$ are left unmatched in any fully popular matching N. This claim holds because the matching S_i is the $(A_L \cup B_L)$-optimal matching in the lattice $\mathcal{D}_i$. Thus if $(a^-_\ell, a^+_r) \in S_i$, i.e., if a_ℓ is matched to its least preferred neighbor a^+_r in S_i then a_ℓ has to be matched to a^+_r in the realization N^*_α of N as well (for any witness $\vec{\alpha}$ of N), i.e., $(a^-_\ell, a^+_r) \in N^*_\alpha$; equivalently, a is left unmatched in N (after removing self-loops from N). Similarly, if $(b^-_\ell, b^+_r) \in S_i$ then b_ℓ has to be matched to its least preferred neighbor b^+_r in N^*_α as well, i.e., $(b^-_\ell, b^+_r) \in N^*_\alpha$; equivalently, b is left unmatched in N. ◀

Running time of the algorithm. The set of popular edges can be computed in linear time [7]. Similarly, the set of valid edges can be computed in linear time [2]. Gale-Shapley algorithm and in particular, the variant of Gale-Shapley algorithm used here – to compute a stable matching that avoids all forbidden edges – can be implemented to run in linear time [13]. Hence it can be shown that Algorithm 1 runs in linear time. Thus Theorem 4 stated in Section 1 follows.

References

1 A. Abdulkadiroğlu and T. Sönmez. School choice: A mechanism design approach. *American Economic Review*, 93(3):729–747, 2003.

2 D. J. Abraham, R. W. Irving, T. Kavitha, and K. Mehlhorn. Popular matchings. *SIAM Journal on Computing*, 37(4):1030–1045, 2007.

3 S. Baswana, P. P. Chakrabarti, S. Chandran, Y. Kanoria, and U. Patange. Centralized admissions for engineering colleges in india. *INFORMS Journal on Applied Analytics*, 49(5):338–354, 2019.

4 Canadian Resident Matching Service. How the matching algorithm works. `http://carms.ca/algorithm.htm`.

5 M.-J.-A.-N. de C. (Marquis de) Condorcet. *Essai sur l'application de l'analyse à la probabilité des décisions rendues à la pluralité des voix.* L'Imprimerie Royale, 1785.

6 Condorcet method. https://en.wikipedia.org/wiki/Condorcet_method.

7 Á. Cseh and T. Kavitha. Popular edges and dominant matchings. *Mathematical Programming*, 172(1):209–229, 2018.

8 Y. Faenza and T. Kavitha. Quasi-popular matchings, optimality, and extended formulations. In *Proceedings of the 31st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 325–344, 2020.

9 Y. Faenza, T. Kavitha, V. Powers, and X. Zhang. Popular matchings and limits to tractability. In *Proceedings of the 30th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2790–2809, 2019.

10 D. Gale and L.S. Shapley. College admissions and the stability of marriage. *American Mathematical Monthly*, 69(1):9–15, 1962.

11 D. Gale and M. Sotomayor. Some remarks on the stable matching problem. *Discrete Applied Mathematics*, 11:223–232, 1985.

12 P. Gärdenfors. Match making: assignments based on bilateral preferences. *Behavioural Science*, 20:166–173, 1975.

13 D. Gusfield and R. W. Irving. *The Stable Marriage Problem: Structure and Algorithms.* MIT Press, 1989.

14 C.-C. Huang and T. Kavitha. Popular matchings in the stable marriage problem. *Information and Computation*, 222:180–194, 2013.

15 C.-C. Huang and T. Kavitha. Popularity, mixed matchings, and self-duality. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2294–2310, 2017.

16 T. Kavitha. A size-popularity tradeoff in the stable marriage problem. *SIAM Journal on Computing*, 43:52–71, 2014.

17 T. Kavitha. Popular half-integral matchings. In *Proceedings of the 43rd International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 22:1–22:13, 2016.

18 T. Kavitha, J. Mestre, and M. Nasre. Popular mixed matchings. *Theoretical Computer Science*, 412:2679–2690, 2011.

19 D. F. Manlove and C. Sng. Popular matchings in the capacitated house allocation problem. In *Proceedings of the 14th Annual European Symposium on Algorithms (ESA)*, pages 492–503, 2006.

20 J. Mestre. Weighted popular matchings. *ACM Transactions on Algorithms*, 10(1):2:1–2:16, 2014.

21 National Resident Matching Program. Why the match? http://www.nrmp.org/whythematch.pdf.

Obviously Strategyproof Single-Minded Combinatorial Auctions

Bart de Keijzer
King's College London, UK
bart.de_keijzer@kcl.ac.uk

Maria Kyropoulou
University of Essex, UK
maria.kyropoulou@essex.ac.uk

Carmine Ventre
King's College London, UK
carmine.ventre@kcl.ac.uk

Abstract

We consider the setting of combinatorial auctions when the agents are single-minded and have no contingent reasoning skills. We are interested in mechanisms that provide the right incentives to these imperfectly rational agents, and therefore focus our attention to *obviously strategyproof (OSP)* mechanisms. These mechanisms require that at each point during the execution where an agent is queried to communicate information, it should be "obvious" for the agent what strategy to adopt in order to maximise her utility. In this paper we study the potential of OSP mechanisms with respect to the approximability of the optimal social welfare.

We consider two cases depending on whether the desired bundles of the agents are *known* or *unknown* to the mechanism. For the case of known-bundle single-minded agents we show that OSP can actually be as powerful as (plain) strategyproofness (SP). In particular, we show that we can implement the very same algorithm used for SP to achieve a $\sqrt{m}$-approximation of the optimal social welfare with an OSP mechanism, m being the total number of items. Restricting our attention to declaration domains with two values, we provide a 2-approximate OSP mechanism, and prove that this approximation bound is tight. We also present a randomised mechanism that is universally OSP and achieves a finite approximation of the optimal social welfare for the case of arbitrary size finite domains. This mechanism also provides a bounded approximation ratio when the valuations lie in a bounded interval (even if the declaration domain is infinitely large). For the case of unknown-bundle single-minded agents, we show how we can achieve an approximation ratio equal to the size of the largest desired set, in an OSP way. We remark this is the first known application of OSP to multi-dimensional settings, i.e., settings where agents have to declare more than one parameter.

Our results paint a rather positive picture regarding the power of OSP mechanisms in this context, particularly for known-bundle single-minded agents. All our results are constructive, and even though some known strategyproof algorithms are used, implementing them in an OSP way is a non-trivial task.

2012 ACM Subject Classification Theory of computation → Algorithmic game theory and mechanism design

Keywords and phrases OSP Mechanisms, Extensive-form Mechanisms, Single-minded Combinatorial Auctions, Greedy algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.71

Category Track A: Algorithms, Complexity and Games

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 71; pp. 71:1–71:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Algorithms might not have direct access to their inputs. This is by now a well-known issue, motivated by the internet and the self interests thereon. A body of work in computer science has been devoted to the design of algorithms that can faithfully elicit the input from their selfish sources. Typically, this is achieved through so-called mechanism design, a subdiscipline of game theory that studies the design of functions, or *mechanisms*, where the input is provided by multiple self-interested agents, and the output, or *outcome*, needs to satisfy a set of pre-specified desirable objectives. Each agent has a certain utility for each outcome. In its most general definition, a mechanism is given by a multi-round interaction protocol between a central authority and the agents, defining a game wherein the central authority wants to compute a certain function of the agents' inputs and the agents choose a strategy leading to the outcome that maximises their own utility. The main design requirements for such mechanisms are:

Strategyproofness (SP). There is a strategy for each agent that is guaranteed to result in a better outcome than any other strategy that the agent may adopt, irrespective of the other agents' strategies.

(Approximate) economic efficiency. The outcome of the mechanism must have a quality that is close to a theoretical optimum; this is often measured in terms of *social welfare*, i.e., the sum of all the agents' utilities.

Combinatorial auctions (CAs) have emerged as the paradigmatic problem in the area, exemplifying the tension between these two desiderata and the polynomial-time computation of the outcome. In CAs, we are given a set of items that need to be sold among a set of agents who are interested in buying the items. These agents express valuations for each of the items, or certain bundles of items, or even each possible bundle of items, depending on the level of complexity of the utility model that is assumed. A mechanism in this setting must determine how the items are allocated to the agents, and how much each agent is going to be charged.

Whilst it is not known to what extent it is possible to guarantee the properties above with polynomial-time algorithms, the design of mechanisms that satisfy these objectives is reasonably well-understood for some sufficiently simple auction settings. This is the case for the optimization problem of interest to this study, *single-minded combinatorial auctions*. In such a setting, there are multiple agents and multiple items to be allocated. An agent's utility function has a simple form: It is determined by a *valuation* of the set of items that the agent gets allocated, from which a potential payment charged by the mechanism is subtracted. The valuation function has the following structure: For an agent, say i, there is a number v_i and a subset of items R_i such that her valuation is v_i whenever the allocated bundle contains R_i, and 0 otherwise. The utility of each agent is therefore determined by a single pair (v_i, R_i). This simple setting has been the central object of study of the celebrated paper [17], where the authors design a simple greedy mechanism that is strategyproof and approximates the optimal social welfare to within a factor of $\sqrt{m}$, where m is the number of items. It is well-known that this combinatorial optimisation problem is NP-hard and inapproximable within a factor of $\sqrt{m}$ unless $\mathsf{NP} = \mathsf{ZPP}$ [14].

In many important cases, however, the implementation of strategyproof mechanisms can be too complex, unintelligible, unintuitive, or cognitively too demanding due to the limited ability of typical agents, such as, the capability to carry out *contingent reasoning*. Even for the simple setting of one-item auctions, a special case of single-minded agents, it is known how implementation details matter, see, for example, [2], for a discussion on the differences between sealed bid versus ascending bid auctions, and [15] for a related experimental study. We refer furthermore to [6] and [4] for further reading on the issue of contingent reasoning. Hence, the theoretically strong mechanisms that have been proposed in past mechanism design literature may be too difficult to understand and to use for agents with imperfect rationality.

This problem motivates the design of mechanisms with a reduced *cognitive burden* for agents who participate in the mechanism: Mechanisms should be very easy to understand, and transparent to participate in. While it is somewhat of a challenge to satisfactorily define formally what a "low cognitive burden" comprises, the concept of *obvious strategyproofness* (OSP) which has been introduced in [18] provides a strong and reasonably satisfactory notion in the context of agents unable to reason contingently. Informally, OSP requires that at each point during the execution of a mechanism where an agent is queried to reveal information (i.e., point where an agent is asked to make a decision about how the execution of the mechanism should proceed), it should be "obvious" for the agent what strategy to adopt in order to maximise her utility: For the agent, there must be a single choice for which it holds that all outcomes that can result from that choice are better for her than any other outcome that can result from an alternative choice. OSP therefore strengthens the classical notion of strategyproofness.

Our contributions. We study the extent to which OSP mechanisms can return good approximations to the optimal social welfare in the setting of single-minded combinatorial auctions. We measure the quality of mechanisms in terms of a relative approximation ratio. As in much of the literature on OSP, we assume that the set of possible types that the agents can have (which we refer to as the *declaration domain*) is publicly known; this is either finite or contained in a closed interval. As for the agents, we prove results depending on whether they are *known-bundle single-minded* – whereby the valuation v_i for agent i's desired bundle R_i is not known but R_i is – or, *unknown-bundle single-minded*, for which neither v_i nor R_i is known and must be elicited in an OSP way.

For the case of known-bundle single-minded agents, we provide the following results.

- If there are only two possible valuations, i.e., a low valuation L and a high valuation H, we express the characterisation of OSP mechanisms in [7] conveniently and design a deterministic OSP mechanism with an approximation ratio of 2, which we show to be the best possible. We give explicit payment functions for this mechanism and we can prove that truthtelling agents always have non-negative utility (a property known as individual rationality (IR)). We furthermore show that if the OSP mechanism were to use a fixed ordering of agents that does not depend on the instance, then the approximation guarantee of the mechanism is unbounded.
- If the declaration domain is an arbitrarily large finite set, we derive an OSP mechanism that achieves an approximation ratio of $\sqrt{m}$ to the optimal social welfare. This mechanism can be regarded as an obviously strategy-proof implementation of the mechanism in [17]. The payments here are implicitly given through the *cycle monotonicity* technique developed in [7]. Our mechanism makes use of the fact that the domain is finite and that the desired bundles are known.
- We further provide a randomised OSP and IR mechanism that achieves an approximation ratio strictly less than d, where d is the cardinality of the declaration domain. In particular, for arbitrary size finite domains of d valuations $V_1 < \cdots < V_d$, our mechanism achieves a $(d - V_1/V_2 - \cdots - V_{d-1}/V_d)$-approximation of the optimal social welfare. The idea behind this mechanism is to simply draw at random (with a carefully chosen probability

distribution) one value from the domain and ask agents if their valuation is at least as big; the mechanism thus stays OSP even if agents had access to the randomness used (i.e., they are "universally OSP"). We note that this result improves the aforementioned bound of 2 for two-value domains, and approaches 2 as $V_1/V_2 \to 0$. We also generalise this result for (uncountable) valuations contained in an interval $[a, b]$ and derive the continuous limit of the above mechanism, which results in an OSP and IR mechanism for this setting that achieves an approximation ratio of $1 + \ln(b/a)$ to the optimal social welfare. This means that the approximation ratio only grows logarithmically in the relative size of the interval.

The results above paint a rather positive picture regarding the power of OSP mechanisms for CAs with known-bundle single-minded agents. Firstly, our upper bound of $\sqrt{m}$, for finite valuation domains, matches what is known for strategyproofness – this is, to best of our knowledge, the first case in which OSP has been proved to be *as powerful as* SP. ([16] proves an asymptotic equivalence for randomised OSP without money for a variant of a scheduling problem.) The only additional time we need is used to sweep through the declaration domain of the agents, a seemingly unavoidable step to guarantee an OSP implementation of direct-revelation mechanisms. Secondly, this result shows that Deferred Acceptance (DA) auctions, whilst being OSP, are not the right algorithmic approach to optimise the approximation guarantee of OSP mechanisms. We in fact beat the lower bounds proved in [5] regarding the approximation guarantee of DA auctions in this setting. This observation reinforces the findings in [7, 9] concerning the power of DAs vs OSP, in the context of scheduling related machines. Thirdly, our tight bounds for two-value domains show (i) how the graph-theoretic approach to OSP in [7] can be made operational; and, (ii) that the order in which the mechanism queries the set of agents is of crucial importance in the design of the mechanism. Finally, our randomised OSP mechanisms show how it is possible to leverage "internal" chance nodes [18] and beat deterministic mechanisms, whilst agents do not need to compute expectations. This is to our knowledge the first known application of randomisation over OSP mechanisms.

OSP has so far only been considered for single-parameter settings, i.e., where the agents' private information is a single number, as no explicit technique is known to implement OSP mechanisms for higher dimensional settings. However, we complete this paper by giving the *first* OSP mechanism for agents with richer declaration domains. We in fact provide an OSP mechanism for the case of unknown-bundle single-minded agents, that returns an approximation of the optimal social welfare equal to the maximum size of a desired set R_i. This is an OSP implementation of the well-known Greedy-by-valuation algorithm for CAs. To obtain this result, we leverage the cycle monotonicity characterisation of OSP in [7] and an approach recently used in [10] to deal with long negative cycles. The idea is to give a structural property of the negative-weight cycles that have more than two vertices, for mechanisms that satisfy a natural notion of monotonicity between two instances. We then prove that any such mechanism that additionally queries the agents monotonically (that is, from an extreme of the domain to the other, irrespective of the desired set) cannot have any long negative-weight cycle. This is only proved to be a sufficient condition; the extent to which this is also necessary (proved to be true for single-parameter settings in [10]) will say whether our bound can be improved or not. This is the main open problem left by our work.

Further Related Work. The notion of OSP introduced in [18] has spawned numerous subsequent studies in both the computer science and economics community.

In [1], the authors study the OSP concept in the context of stable matchings and provide a suitable mechanism under an acyclicity assumption, as well as an impossibility result for a more general setting. The paper [3] further studies this concept for housing markets, single-peaked domains, and a general quasi-linear mechanism design setting. In [20] the authors study OSP mechanisms in general design domains where monetary transfers are not possible, and they provide a useful characterisation of OSP mechanisms in this setting.

OSP was investigated in machine scheduling domains and set system problems in [7], published in [9, 8], and furthermore a study was done in [11] where the authors prove that a restriction on the agents' declaration called *monitoring* can help obtain OSP mechanisms with a good approximation ratio in various mechanism design domains. The paper [16] builds forth on this by studying machine scheduling in the absence of monetary payments. For machine scheduling, a generalisation of OSP is furthermore studied in [13] where the restriction on the ability of agents to reason contingently is weakened, and the authors show that a large amount of "look-ahead"-ability is required for the agents in any mechanism that achieves a good approximation ratio in the considered scheduling setting. Another study that considers OSP under a restriction on the agents' behavior is [12], where the authors assume that non-truthful behaviour can be detected and penalised with a certain probability.

In [19] a revelation principle is presented that states that every social choice function implementable through an OSP mechanism can be implemented using a certain structured OSP protocol where agents take turns making announcements about their valuations.

2 Preliminaries

In a combinatorial auction we have a set U of m items and a set N of n agents. Each agent i has a *private* valuation function v_i and, in the general case, is interested in obtaining only one set in a *private* collection $\mathcal{S}_i$ of subsets of U, also called bundles. Thus, the valuation function maps subsets of items to nonnegative real numbers ($v_i(\emptyset)$ is normalised to be 0). The agents' valuations are monotone: for $S \supseteq T$ we have $v_i(S) \geq v_i(T)$. In single minded combinatorial auctions, $|\mathcal{S}_i| = 1$, each agent i is interested in obtaining only one particular subset of U; we denote i's desired bundle by R_i. This implies that agent i's valuation is the same for all supersets of R_i, while it is 0 otherwise. Formally, consider agent i and let $\mathcal{S}_i = \{R_i\}$. The valuation function of agent i for a given set S is

$$v_i(S) = \begin{cases} v_i & \text{if } S \supseteq R_i, \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

where (with a slight notation overloading) $v_i \in \mathbf{R}_{\geq 0}$ is a non-negative real number. Note that the valuation function of an agent i is fully represented by her desired set R_i and her valuation for that set v_i. The goal is to find a partition $(S_1, \ldots, S_n)$ of U such that $\sum_{i=1}^{n} v_i(S_i)$ –the *social welfare (SW)*– is maximised. We denote the optimal (maximum possible) social welfare by SW^* (we omit the dependence on the instance as it will be clear from the context).

We consider two versions of single-minded combinatorial auctions. In the case of *unknown-bundle single-minded agents*, we assume that the desired sets R_i and the valuations v_i are private knowledge of the agents, while in the *known-bundle single-minded agents* case, we assume that the desired sets R_i are known and only the valuations v_i are private knowledge of the agents. In each case, we refer to the private information of an agent as her *type*. Agents are typically asked to declare their types. We use D_i, the *declaration domain* of agent i, to denote the set of all possible types of agent i. In this paper we will consider declaration domains that are identical for all agents i, i.e., $D_i = D$. We use $\mathbf{b} = (b_1, \ldots, b_n)$ to denote declaration profiles, so that $b_i \in D$ stands for an agent i's demanded set R_i along with the valuation v_i that she has for it.

We want to design an auction mechanism that interacts with the agents in any sequence, and outputs (based on this interaction) an allocation A_i of items to each agent i, along with a payment $P_i > 0$ that is charged to each agent i. The *allocation function* satisfies that each item is allocated to exactly one agent. We refer to such a pair $(A = (A_1, \dots, A_n), P = (P_1, \dots P_n))$ as an *outcome.* We assume that agents have quasi-linear utility functions, that is, agent i with type b_i has a utility of

$$u_i(b_i, \mathbf{b}_{-i}) = v_i(A_i) - P_i$$

for outcome (A, P). Each agent will interact strategically with the mechanism, so as to make it output an outcome that maximises her utility.

When designing the mechanism, we want to define allocations and payments in such a way that they induce a certain type of behaviour of the strategically acting agents, while simultaneously ensuring that the outcome maximises or approximately maximises the social welfare. In particular, we aim to design *obviously strategyproof (OSP)* mechanisms. To define this notion, we view a mechanism in the form of an *implementation tree* $\mathcal{T}$ that captures the way in which the mechanism interacts with the agents [18, 7].

We now introduce some notation around $\mathcal{T}$ before we formally define the OSP notion. Each internal node u of $\mathcal{T}$ is labeled with an agent $\mathcal{Q}(u)$, called the *divergent agent* at u, and the outgoing edges from u are labeled with sets of types in the declaration domain of $\mathcal{Q}(u)$. At node u, the agent $\mathcal{Q}(u)$ is queried and asked to choose an action, that corresponds to selecting one of u's outgoing edges. The labels of the outgoing edges of u form a partition of the *current domain* of i, denoted as $D_i(u)$. The current domain $D_i(u)$ is equal to the label of the last edge e in the path from the root to u that i chose as an action. When player $\mathcal{Q}(u)$ chooses an outgoing edge at node u, we say that the chosen action *signals* that the type of $\mathcal{Q}(u)$ is in the set of types labeling the chosen edge. If a pair of types in the current domain of agent i occur in the labels of two distinct outgoing edges of u, we say that i is asked to *separate* the two possible types at u. The leaves of the tree will thus be linked to (a set of) type profiles; and at each leaf the mechanism will return an outcome (A, P) accordingly; in other words, each leaf corresponds to an outcome of the mechanism. (Observe that this means that the domain of A and P is effectively given by the leaves of $\mathcal{T}$.)

A mechanism (viewed in the form of an implementation tree as described above) is said to be OSP if at each node where an agent is asked to diverge, she *always* maximises her utility by choosing the edge (u, u') with the label containing her own type, in every node. That is, if we call the latter strategy s, then a mechanism is said to be OSP if the worst possible outcome after signaling her true type (taken over all the reachable outcomes in the leaves of the subtree rooted at u', with respect to s) gives her a utility *at least as good* as when she would get the best possible outcome after choosing any other edge (u, u'') at that particular point in the implementation tree (taken over all the outcomes in the leaves of the subtree rooted at u''). The corresponding utility-maximising strategies played by the players are called the *OSP strategies.* In cases where we discuss any particular OSP mechanism, we use $(A(\mathbf{b}), P(\mathbf{b}))$ to denote the outcome (i.e., allocation and payment vector) that results from agents playing their OSP strategies, i.e., strategy profiles where each agent at every node in the mechanism follows the edge containing her type.

We call a type profile $\mathbf{b}$ *compatible with* $u \in \mathcal{T}$ iff for each edge (u', u'') on the path from the root to u, the label of (u', u'') contains $b_{\mathcal{Q}(u')}$. We furthermore say that two type profiles $(t, \mathbf{b}_{-i})$ and $(b, \mathbf{b}'_{-i})$ *diverge* at u if $i = \mathcal{Q}(u)$ and t and b are labels of different edges outgoing from u (we sometimes will abuse this terminology and we also say that t and b diverge at u). For every agent i and types $t, b \in D_i$, we let $u^i_{t,b}$ denote a vertex u in the implementation tree $\mathcal{T}$, such that $(t, \mathbf{b}_{-i})$ and $(b, \mathbf{b}'_{-i})$ are compatible with u, but diverge at u for some $\mathbf{b}_{-i}, \mathbf{b}'_{-i} \in D_{-i}(u) = \times_{j \neq i} D_j(u)$. Note that such a vertex might not be unique as agent i will be asked to separate t from b in different paths from the root (but only once for every such path). We call these vertices of $\mathcal{T}$ *tb-separating* for agent i. For example, the node r in the tree in Figure 1 is an LH-separating node for agent 1; while v and w are two LH-separating nodes for agent 2.

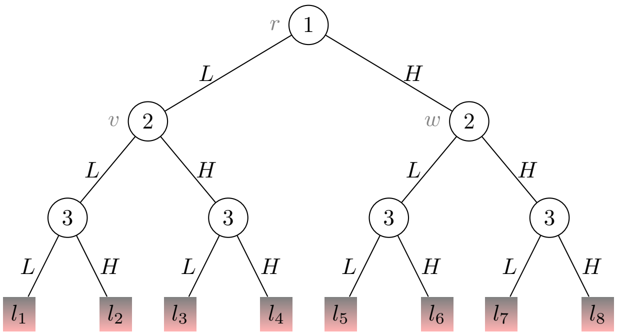


Figure 1 An implementation tree with three agents with two-value domains $\{L, H\}$; each agent separates the domain types upon playing; at each leaf l_i the mechanism computes $A(\mathbf{b})$ and $P(\mathbf{b})$, $\mathbf{b}$ being the declaration vector at l_i.

We also consider randomised mechanisms that are *universally OSP*, in Section 5. Such mechanisms are probability distributions on OSP mechanisms.

Besides the OSP requirement, we would like our mechanisms to satisfy *(ex-post) individual rationality (IR)*, i.e., when an agent plays an OSP strategy, the agent is guaranteed an outcome that gives her a non-negative utility. We measure the quality of mechanisms in terms of a relative *approximation ratio*, i.e., an upper bound on the ratio of the optimal social welfare and the social welfare of the outcome of the mechanism.

3 Deterministic mechanisms for known-bundle single-minded agents

We now focus on the case of known-bundle single-minded agents, i.e. the desired sets of the agents are known by the mechanism, and only the valuations of the agents for their corresponding desired set is private information. We let $v_i, i \in N$ denote the valuation of agent i for her desired bundle, and slightly abusing notation, we use $SW(I) = \sum_{i \in I} v_i$ where I is a set of agents who can be allocated their desired bundles at the same time.

3.1 Domains of size 2

We first restrict attention to the case where the declaration domain of each agent has two possible values and begin with a simple and elegant characterisation of individually rational OSP mechanisms for the case of known-bundle single-minded agents with two-value domains $D_i = \{L, H\}$, for $i \in N$.

▶ **Definition 1** ($\mathcal{C}$ mechanisms). *We define a class $\mathcal{C}$ of mechanisms, that only use the following types of queries:*

- L-query: *The divergent agent is asked to separate L and H. If she signals L then she will not get her desired bundle (regardless of the future).*
- H-query: *The divergent agent is asked to separate L and H. If she signals H then she is guaranteed to get her desired bundle.*

▶ **Theorem 2.** *The class $\mathcal{C}$ of mechanisms characterises deterministic IR and OSP mechanisms for the case where the declaration domain is $\{L, H\}$ and the desired bundles are known, in the following sense:*

- *There exist payments such that every mechanism in $\mathcal{C}$ is deterministic IR and OSP.*
- *Every deterministic IR OSP mechanism is equivalent to a mechanism in $\mathcal{C}$ with respect to their allocations, i.e. for every possible valuation profile of the agents, the allocations resulting from both mechanisms are identical.*

Proof. Regarding the first claim, fix the payment of an agent who is allocated her bundle to be L, and the payment to be 0 otherwise. It should be straightforward to see why pairing mechanisms in $\mathcal{C}$ with these payment yields a deterministic IR mechanism. Moreover, observe that regardless of the query, if an agent has valuation L, then her utility will be 0 regardless of her signals and allocation (even if she is allocated her bundle, then she will be asked to pay L). If an agent has valuation H and she is asked an L-query, she can only get positive utility for signaling her true valuation H, while if she is asked an H-query, she can guarantee herself the maximum possible utility under the mechanism $(H - L)$ again by reporting her true valuation H. This demonstrates that whenever an agent is asked to diverge, she is not worse off by acting according to her true type in *any* possible future scenario, hence OSP is satisfied.

Regarding the second claim, consider any OSP mechanism and its associated implementation tree $\mathcal{T}$. Clearly, since we are dealing with the case of two-type domains, if $\mathcal{T}$ has a node with more than two children, then this node can be pruned to yield an OSP mechanism where the node has exactly two children. Furthermore, nodes that have only one child are trivial and can be removed from the tree. So, at every node in $\mathcal{T}$ an agent is asked to separate L and H, and at any path from the root to a leaf in $\mathcal{T}$ each agent is only asked to diverge at most once.

Next, fix a node $u \in \mathcal{T}$ and let agent $i = \mathcal{Q}(u)$ be the divergent agent at u. Suppose for a contradiction that if i signals L when queried at u, then it is possible that i gets her desired bundle, while if i signals H then it is possible that i is not allocated her desired bundle. In other words, the corresponding subtrees of $\mathcal{T}$ have outcomes o_L where i gets her desired bundle when signaling L, and o_H where i doesn't get her desired bundle when signaling H. In this case the OSP condition would be violated at node u when $v_i = H$, as i's utility under o_H is lower than under o_L (by IR the payment under o_H is 0, while the payment under o_L is at most L). We can conclude that at any node $w \in \mathcal{T}$, either all possible outcomes when the divergent agent $j = \mathcal{Q}(w)$ signals L do not allocate j's desired bundle to j, or all outcomes when j signals H allocate j's desired bundle to j. These two alternatives correspond to the L-queries and H-queries in the definition of $\mathcal{Q}$. ◀

Next, we use the above characterisation theorem[1] to provide a tight bound on the approximability of the optimal social welfare under OSP mechanisms.

▶ **Theorem 3.** *The IS mechanism described in Algorithm 1 is an OSP mechanism that achieves an approximation ratio $\rho \leq 2$ for the case of known-bundle single-minded agents and domains of size 2.*

Proof. Consider any instance of the problem, let $I \subseteq N$ be the allocation output by IS, and let $I^* \subseteq [n]$ be the SW-maximising allocation according to the true valuations. Also fix $\mathcal{N}$ to its value at the last iteration of IS. At the last iteration of IS it holds that every agent in

[1] Note that Theorem 2 essentially expresses the cycle monotonicity property of [7] in a convenient way.

Algorithm 1 The IS mechanism.

```
1  $\mathcal{N} \leftarrow N$ ($\mathcal{N}$ is the set of agents currently under consideration) $A_q \leftarrow \emptyset$ ($A_q$ is the set of agents who have already been queried) $I \leftarrow \emptyset$ ($I$ is the set of agents currently allocated their desired bundle) $v_i = L$, for $i \in N$ (Our mechanism originally assumes that all agents have valuation $L$)
2  Compute the SW-maximising feasible allocation and update set $I$ accordingly
3  while there exists an agent $i \in \mathcal{N} \setminus A_q$ who is not in $I$ do
4      Perform an $L$-query to $i$ (break ties arbitrarily)
5      if $i$ signals $L$ then
6          $\mathcal{N} = \mathcal{N} \setminus \{i\}$
7      else
8          $v_i = H$
9      Compute the SW-maximising feasible allocation of items to agents in $\mathcal{N}$ and update $I$
10 Return $I$
11 If an agent $i$ is allocated her desired bundle, i.e. $i \in I$, charge her $L$, otherwise charge 0.
```

$N \setminus I$ has been queried. Mechanism IS belongs to the class $\mathcal{C}$ of mechanisms (Definition 1), and hence by Theorem 2 is IR and OSP. So, we can assume that the agents who have been queried have signaled their true valuation. Regarding the agents that have not been queried, the mechanism assumes valuation L when computing the SW-maximising allocation, while their true valuation could potentially be higher; all these agents have been allocated their desired bundle. This leads to the conclusion that I is the SW-maximising allocation of $\mathcal{N}$ with respect to the true valuations, which in turn implies that $SW(I) \geq SW(I^* \cap \mathcal{N})$.

We now claim that $SW(I) \geq SW(I^* \setminus \mathcal{N})$. Observe that any agent who does not belong to $\mathcal{N}$ has valuation L. All these agents were considered in line 2 of Algorithm 1 as having valuation L indeed; denote SW_1 the SW computed in line 2 of Algorithm 1. So, it holds that $SW(I^* \setminus \mathcal{N}) \leq SW_1 \leq SW(I)$, where the second inequality holds because the SW computed in Algorithm 1 can only increase between rounds.

Combining the above, we derive that $SW(I^*) = SW(I^* \cap \mathcal{N}) + SW(I^* \setminus \mathcal{N}) \leq 2SW(I)$, as desired. ◀

Next, we show that the above bound is tight by designing a family of instances where no mechanism can achieve an approximation ratio better than 2. This results shows a separation between obviously strategy proof mechanisms and strategy proof mechanisms, in terms of social welfare.

▶ **Theorem 4.** *For the setting that the domain is of size* 2 *and desired bundles are known, no OSP and IR mechanism can achieve an approximation ratio better than* 2.

Proof. By Theorem 2 we may restrict our analysis to mechanisms in class $\mathcal{C}$ as defined in Definition 1. Consider any such mechanism $M \in \mathcal{C}$.

Consider an instance with m items and a set N of $n = 2\sqrt{m}$ agents. For $i \in \{1, \ldots, \sqrt{m}\}$, define sets $S_i = \{(i-1)\sqrt{m}, (i-1)\sqrt{m}+1, \ldots, (i-1)\sqrt{m} + (\sqrt{m}-1)\}$ and $T_i = \{i-1, (i-1) + \sqrt{m}, (i-1) + 2\sqrt{m}, \ldots, i + (\sqrt{m}-1)\sqrt{m}\}$. Let agent $i \in \{1, \ldots, \sqrt{m}\}$ desire bundle S_i, and let agent $i \in \{\sqrt{m}+1, \ldots, 2\sqrt{m}\}$ desire bundle T_i. Note that the agents are intuitively split in two equal-sized groups, A and B, such that the desired bundle of an agent in A

overlaps with the desired bundle of all agents in B and vice versa. Consider such a split and let $G(i) \in \{A, B\}$ denote the group of agents that i belongs to, and $\overline{G(i)} = N \setminus G(i)$. The valuations of the agents in our instance depend on the sequence of queries made in the implementation tree $\mathcal{T}$ of mechanism M. We let $H = n/2$ and $L = 1$, and we define the valuations of the divergent agents in turn, starting with the first divergent agent and considering divergent agents in a sequence on the path of $\mathcal{T}$, where previous agents have signaled their true valuation. Let i refer to the agent whose valuation we define at the current step:

(i) If agent i is asked an L-query, no agent has been asked an H-query so far, and at least 3 agents in $\overline{G(i)}$ have not been queried at this point: Set $v_i = L$.

(ii) If i is the first agent that is asked an H-query, and there are at least 3 agents in $\overline{G(i)}$ that have not been queried at this point: Set $v_i = H$, set $v_j = L$ for each agent $j \in G(i)$ that has not been queried at this point, and set $v_j = H$ for each $j \in \overline{G(i)}$ that has not been queried at this point.

(iii) If no agent has been asked an H-query so far, and exactly 2 agents in $G(i) \setminus \{i\}$ have not been queried at this point: Set $v_i = H$, and set $v_j = L$ for every agent j that has not been queried at this point.

Observe that the above three points yield a complete and consistent specification of the valuations of all agents.

Clearly, in any mechanism with bounded approximation ratio, one of cases (ii) or (iii) in the definition of valuations above is realised, since otherwise, M will not allocate any desired bundle to any agent. Assume first that case (ii) is realised, and let i' be the agent who is H-queried at that point. i' will signal her true valuation H, and will be allocated her desired bundle by the definition of an H-query. Hence, no agent in $\overline{G(i')}$ can be allocated her desired bundle and there are at least 3 such agents with valuation H. Therefore the optimal social welfare in this case is at least $3H + (n/2 - 3)L$, but the mechanism can only allocate to a subset of $G(i)$, resulting in a social welfare of at most $H + (n/2 - 1)L$. Thus, the approximation ratio of M in this case is at least

$$\frac{3H + (n/2 - 3)L}{H + (n/2 - 1)L} = \frac{2n - 3}{n - 1} \tag{2}$$

on this instance.

Suppose now that case (iii) in the definition of valuations above is realised, and let i^* be the agent who is queried at that point. By definition of the instance, the optimal social welfare now is $H + (n/2 - 1)L$. In case M gives the desired bundles to a subset of agents in $G(i^*)$, then this subset may contain only agent i^* and the remaining two agents of $G(i^*)$ who have not been queried yet with valuation L, yielding a social welfare of $H + 2L$. Recall that everyone else in $G(i^*)$ has signaled L to an L-query, hence can not be allocated her desired bundle. In case M gives the desired bundles to a subset of agents in $\overline{G(i^*)}$, then the social welfare of M is at most $(n/2)L$. Thus, the approximation ratio of M in this case is at least $\frac{H+(n/2-1)L}{\max\{H+2L,(n/2)L\}}$. By setting $H = n/2$ and $L = 1$ we get

$$\frac{H + (n/2 - 1)L}{\max\{H + 2L, (n/2)L\}} = \frac{n - 1}{n/2 + 2} = \frac{2n - 2}{n + 4}. \tag{3}$$

The limit of both bounds (2) and (3) as $n \to \infty$ is 2, and this proves the claim. ◀

We now provide a stronger inapproximability bound for the case where the mechanism is restricted to be *instance-independent*, i.e., the mechanism's implementation tree is not dependent on the demanded bundles of the agents. That is, there is a mapping from nodes in the implementation tree to queried players and respective query types, and this mapping is *fixed*, i.e., it is *not* a function of the demanded bundles of the agents. Note that e.g. the mechanism of Theorem 3 is *not* instance-independent, because the order in which the players are queried and the types of queries that the mechanism asks, do depend on the demanded bundles of the players.

▶ **Theorem 5.** *No instance-independent mechanism has a bounded approximation ratio.*

Proof. Consider any instance-independent OSP and IR mechanism M. We define the instance (valuations and desired bundles) that yields the upper bound by considering the sequence of queries made in the implementation tree $\mathcal{T}$ of M. We start with the first divergent agent and consider divergent agents in a sequence on the path of $\mathcal{T}$, where previous agents have signaled their true valuation. Let i refer to the ith queried agent, whose valuation is v_i and whose desired bundle is R_i. Let $\overline{Q(i)}$ denote the set of agents who have not been queried yet at the time that i is queried by the mechanism. As $\mathcal{T}$ is instance-independent, we may construct an instance with a bad approximation ratio for this mechanism by letting the agents' valuation and demanded bundles depend on the sequence of queries that the mechanism asks. We do this as follows:

(i) If agent i is asked an L-query: Set $v_i = L$ and let R_i comprise a single item that does not belong to the desired bundle of any other agent.

(ii) If agent i is asked an H-query: Set $v_i = H$ and $v_j = H$ for each agent $j \in \overline{Q(i)}$. Let R_j comprise a single item, distinct for each $j \in \overline{Q(i)}$ and not being desired by any previously considered agent. Also, let $R_i = \cup_{j \in \overline{Q(i)}} R_j$.

In any mechanism with a finite approximation ratio, case (ii) in the definition of the instance above is realised, since otherwise, M will not allocate any desired bundle to any agent. Suppose ℓ L-queries are asked before the first H-query on the path of $\mathcal{T}$ discussed in the definition of the instance. The optimal social welfare is at least $\ell L + (n - \ell - 1)H$, while M can only obtain social welfare equal to H by allocating to the agent who got the H-query. Thus, the approximation ratio ρ of M is at least

$$\rho \geq \frac{\ell L + (n - \ell - 1)H}{H} = \frac{(n-1)H - \ell(H - L)}{H} \geq \frac{(n-1)L}{H}, \tag{4}$$

since $\ell \leq n - 1$. The ratio can be made arbitrarily high, for suitable values of L and H. ◀

3.2 Large domains

In this section we prove an upper bound of $\sqrt{m}$ for the approximation ratio of OSP mechanisms for known-bundle single minded agents and arbitrary domains. We begin with definitions of classes of mechanisms that are of interest. First, we define extremal mechanisms. Informally, the queries of an extremal mechanism always separate an extreme of the current domain of the queried agent from the rest of her current domain (the same extreme is chosen consistently). Formally,

▶ **Definition 6** (Extremal mechanism). *A mechanism with implementation tree $\mathcal{T}$ is an extremal mechanism if for each internal node $u \in \mathcal{T}$, and divergent agent $i = \mathcal{Q}(u)$ at u, agent i's current domain $D_i(u)$ is partitioned by the query at u into a singleton, containing the maximum element (or minimum element, consistently), and the remaining elements of $D_i(u)$.*

Algorithm 2 An extensive-form implementation of the Greedy algorithm in [17] for known-bundle single-minded agents.

```
1  Define function $\Phi_i$ as $\Phi_i(x) = x/\sqrt{|R_i|}$.
2  $\mathcal{P} \leftarrow \emptyset$ ($\mathcal{P}$ is the set of bundles that have already been allocated) $\mathcal{N} \leftarrow N$ ($\mathcal{N}$ is the
     set of agents currently under consideration) $\mathcal{D}_i \leftarrow D_i$ for all $i \in N$ ($\mathcal{D}_i$ is the set of
     values in $i$'s domain currently under consideration)
3  while $\mathcal{N} \neq \emptyset$ do
4      Let $j = \arg\max_{k \in \mathcal{N}} \Phi_k(\max \mathcal{D}_k)$
5      if there is $S$ in $\mathcal{P}$ such that $R_j \cap S \neq \emptyset$ then
6          $\mathcal{N} = \mathcal{N} \setminus \{j\}$
7      else
8          Ask $j$ if her valuation is $\max \mathcal{D}_j$
9          if yes then
10             $\mathcal{P} \leftarrow \mathcal{P} \cup \{R_j\}$
11             $\mathcal{N} = \mathcal{N} \setminus \{j\}$
12         else
13             $\mathcal{D}_j = \mathcal{D}_j \setminus \{\max \mathcal{D}_j\}$
14 Return $\mathcal{P}$
```

▶ **Definition 7** ($\mathcal{C}_d$ mechanisms)**.** *We define a class $\mathcal{C}_d$ of mechanisms, whose implementation tree $\mathcal{T}$ satisfies:*

- *Consider a divergent agent i who is asked to separate between valuations v_i and v_i' with $v_i < v_i'$ at some internal node $u \in \mathcal{T}$. Consider any outcome o_{v_i} in the subtree rooted at u consistent with signaling valuation v_i. Then if i is allocated her desired bundle under o_{v_i}, she also gets her desired bundle in all possible outcomes consistent with signaling valuation v_i' at u.*

▶ **Theorem 8.** *Every extremal mechanism that belongs to class $\mathcal{C}_d$ is OSP.*

Theorem 8 is a special case of Theorem 12 which we present in Section 4. We note that Theorem 8 is one direction of the characterisation in [10].

▶ **Theorem 9.** *Algorithm 2 is an OSP mechanism that achieves an approximation ratio $\rho \leq \sqrt{m}$, for the case of known-bundle single-minded agents.*

Proof. The approximation guarantee of the algorithm is well known in the literature, cf. [17]. By Theorem 8, it remains to prove that Algorithm 2 is extremal and belongs to class $\mathcal{C}_d$. Indeed, observe that queries only take place in line 8 of the algorithm, where the corresponding divergent agent is asked to separate the maximum value in her current domain from all other possible values.

We will now prove that the algorithm belongs to class $\mathcal{C}_d$ (see Definition 7). Indeed, when an agent is queried in line 8 then she is allocated her set when replying yes (as feasibility has been previously guaranteed). Since we only query for the maximum in the current domain of each agent, this implies that the agent would still be allocated her desired bundle had her true valuation (and signals) been higher. ◀

4 Deterministic mechanisms for unknown-bundle single-minded agents

In this section we prove an upper bound to the approximability of the optimal social welfare of single-minded combinatorial auctions by OSP mechanisms. It is now assumed that both the desired bundle of the agent and her valuation for it belong to her type, hence are private information. We first define a general class of mechanisms which we prove are OSP. We then provide an implementation of the known Greedy-by-valuation mechanism and prove that it belongs to this class.

Let us define the class of valuation-extremal mechanisms. Informally, the queries of a valuation-extremal mechanism at every node separates extreme valuations in the current domain of the queried agent from the rest of her current domain (where the same extreme is chosen consistently). Formally,

▶ **Definition 10** (Valuation-extremal mechanism). *A mechanism with implementation tree $\mathcal{T}$ is a valuation-extremal mechanism if for each internal node $u \in \mathcal{T}$, and divergent agent $i = \mathcal{Q}(u)$ at u, it holds that either agent i's current domain, $D_i(u)$, comprises a single valuation, or $D_i(u)$ is partitioned by the query at u into a set containing all the pairs $(v, S) \in D_i(u)$, where v is the maximum valuation (or minimum valuation, consistently) in $D_i(u)$, and the remaining elements of $D_i(u)$.*

▶ **Definition 11** ($\mathcal{C}^u_d$ mechanisms). *We define a class $\mathcal{C}^u_d$ of mechanisms, whose implementation tree $\mathcal{T}$ satisfies:*

- *Consider a divergent agent i who is asked to separate between $(v_i, S_i) \neq (v'_i, S'_i)$ with $v'_i \geq v_i$ and $S' \subseteq S$, at some internal node $u \in \mathcal{T}$. Consider any outcome $o_{(v_i,S_i)}$ in the subtree rooted at u consistent with signalling type (v_i, S_i). Then if i is allocated her desired bundle under $o_{(v_i,S_i)}$, she also gets her desired bundle in all possible outcomes consistent with signalling type (v'_i, S'_i) at u.*

▶ **Theorem 12.** *Every valuation-extremal mechanism that belongs to class $\mathcal{C}^u_d$ is OSP.*

Theorem 12 extends the result in [10] to the more general case of multi-dimensional agents. Due to lack of space, the proof is omitted.

Algorithm 3 presents an implementation of the known Greedy-by-valuation mechanism for the case of unknown-bundle single minded agents.

▶ **Theorem 13.** *Algorithm 3 is an OSP mechanism that achieves an approximation ratio $\rho \leq \delta$ for the case of unknown-bundle single-minded agents, where δ is the size of the largest desired set.*

Proof. The approximation guarantee of the algorithm is folklore. By Theorem 12, it remains to prove that Algorithm 3 is valuation-extremal and belongs to class $\mathcal{C}^u_d$. Indeed, note that when an agent is queried in line 4 then she is separating all types with valuation v', which is the maximum compatible valuation at this point of the execution, with all types consistent to smaller values v (regardless of the desired bundles).

We will now prove that the algorithm belongs to class $\mathcal{C}^u_d$ (see Definition 11). Let u be the node of the implementation tree in which (v, S) is separated from (v', S') with $v' \geq v$ and $S' \subseteq S$; clearly there is nothing to separate if $v = v'$ and $S = S'$. Consider first the case that $v' > v$; u corresponds to a query at line 4. Assume that there exists $\mathbf{b}_{-i} \in D_{-i}(u)$ for which S is won by agent i when playing according to (v, S). This means that S is feasible

Algorithm 3 An extensive-form implementation of the Greedy-by-valuation algorithm.

```
1  $\mathcal{P} \leftarrow \emptyset$ ($\mathcal{P}$ is the set of bundles that have already been allocated) $\mathcal{N} \leftarrow N$ ($\mathcal{N}$ is the
   set of agents currently under consideration) $\mathcal{D}_i \leftarrow D_i$ for all $i \in N$ ($\mathcal{D}_i$ is the set of
   values in $i$'s domain currently under consideration)
2  while $\mathcal{N} \neq \emptyset$ do
3      Let $j = \arg\max_{k \in \mathcal{N}} \max \mathcal{D}_k$
4      Ask $j$ if her valuation is $\max \mathcal{D}_j$
5      if yes then
6          Ask agent $j$ to reveal her desired set $R_j$
7          if there is $S$ in $\mathcal{P}$ such that $R_j \cap S \neq \emptyset$ then
8              $\mathcal{N} = \mathcal{N} \setminus \{j\}$
9          else
10             $\mathcal{P} \leftarrow \mathcal{P} \cup \{R_j\}$
11             $\mathcal{N} = \mathcal{N} \setminus \{j\}$
12     else
13         $\mathcal{D}_j = \mathcal{D}_j \setminus \{\max \mathcal{D}_j\}$
14 Return $\mathcal{P}$
```

at u; since $S' \subseteq S$ then S' is also feasible at that point. Therefore, the feasibility check in line 7 is successful and S' is won by agent i playing according to (v', S') irrevocably at that point, that is for any $\mathbf{b}_{-i} \in D_{-i}(u)$, as requested. Consider now the case in which $v' = v$ and $S' \subset S$; u corresponds to a query at line 6. Again, if S is feasible at this point of the execution, so is S'. Therefore, S' is allocated if S is, and the proof is complete. ◀

5 Randomised mechanisms for known-bundle single-minded agents

In this section we consider randomised mechanisms that are universally OSP. We note that the bounded rationality assumption that motivates OSP does not prevent agents from using and understanding such a mechanism, as they do not need to compute expected utilities to determine obvious dominance. We start by presenting a class of randomised mechanisms that are universally OSP.

▶ **Definition 14** ($\mathcal{M}_R$ mechanisms)**.** *We define a class $\mathcal{M}_R$ of randomised mechanisms, that work as follows:*

- *Fix some probability distribution F on the domain D of valuations of all agents. A mechanism $M \in \mathcal{M}_R$ selects one of the values v in D according to F and asks every agent if her valuation is at least equal to v. M selects the maximum number of agents whose requests can be satisfied simultaneously, allocates them their desired bundles and charges each of them v.*

▶ **Lemma 15.** *Any mechanism $M \in \mathcal{M}_R$ is universally OSP.*

Proof. We claim that any mechanism that belongs to class $\mathcal{M}_R$ is a randomisation among different deterministic OSP mechanisms. Indeed, fix a value $v \in D$ and consider the mechanism that queries all agents if their valuation is at least equal to v and outputs a feasible solution that allocates their desired bundles to the maximum number of agents who reply *yes*. It should be easy to see that if, when queried, an agent signals *yes* when her true valuation is smaller than v, then the best she can achieve is 0 utility, which is what she would achieve by signalling truthfully. On the other hand, if, when queried, an agent signals *no* when her true valuation is at least equal to v, then the best she can achieve is 0 utility, which is at least what she would achieve by signalling truthfully. ◀

▶ **Theorem 16.** *Consider mechanism $M_F \in \mathcal{M}_R$ (Definition 14), where F is a probability distribution assigning probability p_j to value $V_j \in D$, with $V_j > V_{j-1}$, for $j \in \{1, \dots, d\}$ (set $V_0 = 0$), as follows:*

$$p_1 = \left(d - \sum_{j \le d} \frac{V_{j-1}}{V_j} \right)^{-1} \quad \textit{and} \quad p_j = p_1 \left(1 - \frac{V_{j-1}}{V_j} \right), \textit{ for } j \in \{2, \dots, d\}.$$

Then M_F is a universally OSP randomised mechanism that achieves a $\min \left\{ d, 1 + \ln \left(\frac{V_d}{V_1} \right) \right\}$-approximation of the optimal SW for known-bundle single-minded agents, where $d = |D|$.

Proof. Lemma 15 straightforwardly implies that M_F is universally OSP. Denote by T_j the set of agents who are allocated their desired bundles after M_F selects V_j with probability p_j. Also, let T_j^* denote the set of agents i such that $v_i = V_j$ who are allocated their desired bundle in the optimal allocation.

For the optimal social welfare, SW^*, it holds that

$$SW^* = \sum_{j \le d} V_j \cdot |T_j^*|$$

We bound the social welfare SW of M_F as follows.

$$\begin{aligned} SW &= \sum_{j \le d} p_j V_j \cdot |T_j| \ge \sum_{j \le d} p_j V_j \left(\sum_{j \le \ell \le d} |T_\ell^*| \right) \\ &= \sum_{j \le d} |T_j^*| \cdot \sum_{\ell \le j} p_\ell V_\ell = \sum_{j \le d} |T_j^*| \sum_{\ell \le j} p_1 \left(1 - \frac{V_{\ell-1}}{V_\ell} \right) V_\ell \\ &= p_1 \sum_{j \le d} |T_j^*| \sum_{\ell \le j} (V_\ell - V_{\ell-1}) = p_1 \sum_{j \le d} |T_j^*| \cdot V_j \\ &= p_1 \cdot SW^*, \end{aligned}$$

where the inequality holds because by definition of mechanism M_F, T_j is the maximum cardinality set of agents with valuations at least V_j that can be satisfied simultaneously, and the third equality uses the definition $V_0 = 0$.

Thus, the approximation ratio ρ of this mechanism satisfies

$$\rho \le \frac{1}{p_1} = d - \sum_{2 \le j \le d} \frac{V_{j-1}}{V_j} \tag{5}$$

So, the right-hand-side of (5) is strictly less than d. It remains to prove that $\rho \le 1 + \ln(\frac{V_d}{V_1})$. The expression in (5) is maximised if $\sum_{2 \le j \le d} V_{j-1}/V_j$ is minimised. Letting $X_1, \dots, X_{d-1}$ denote the ratios $V_1/V_2, \dots, V_{d-1}/V_d$, and letting c denote the ratio V_1/V_d, this amounts to minimising $\sum_{i \in [d-1]} X_i$ subject to $\prod_{i \in [d-1]} X_i = c$. By the inequality of arithmetic and geometric means it holds that $\sum_{i \in [d-1]} X_i \ge n \cdot c^{1/d-1}$ for any solution to this minimisation problem, and furthermore this holds with equality for the solution where $X_i = c^{1/d-1}$ for all

$i \in [d-1]$. We conclude from this that the expression in (5)) is maximised if all the ratios V_{j-1}/V_j in the summation are equal (for a fixed choice of V_d/V_1), and this is achieved by setting $V_j = V_d^{\frac{j-1}{d-1}}$ so that $V_{j-1}/V_j = (V_d/V_1)^{-1/(d-1)}$, for all $2 \leq j \leq d$. Inequality (5) then yields

$$\rho \leq d - (d-1)\left(\frac{V_d}{V_1}\right)^{\frac{-1}{d-1}} \leq 1 + \ln\left(\frac{V_d}{V_1}\right), \tag{6}$$

where the last inequality holds because $1 + \ln(V_d/V_1)$ is the limit of the left hand side, and the left hand side is increasing in d. This completes the proof. ◀

We note that a similar bound of $O(\ln(r))$ can be achieved in settings where the valuation space is uncountable and contained in an interval $[a, b]$ with $b/a = r$. One can define a continuous version of M_F by deriving its limit running on D_ϵ as ϵ approaches 0. The mechanism would then work as follows:

(i) Draw a valuation $x \in [a, b]$ according to the cumulative distribution function F, with $F(x) = \frac{\ln(x/a)+1}{\ln(r)+1}$.

(ii) Ask each agent whether her valuation exceeds x.

(iii) Compute the maximum cardinality feasible solution T_x among all yes-responding agents.

A similar analysis to that in the proof of Theorem 16 yields the desired bound.

References

1 Itai Ashlagi and Yannai A. Gonczarowski. Stable matching mechanisms are not obviously strategy-proof. *Journal of Economic Theory*, 177:405–425, 2018.

2 Lawrence M Ausubel. An efficient ascending-bid auction for multiple objects. *American Economic Review*, 94(5):1452–1475, 2004.

3 Sophie Bade and Yannai A. Gonczarowski. Gibbard-satterthwaite success stories and obvious strategyproofness. In *Proceedings of the 2017 ACM Conference on Economics and Computation*, EC '17, page 565, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3033274.3085104.

4 Gary Charness and Dan Levin. The origin of the winner's curse: A laboratory study. *American Economic Journal: Microeconomics*, 1(1):207–36, February 2009. doi:10.1257/mic.1.1.207.

5 Paul Dütting, Vasilis Gkatzelis, and Tim Roughgarden. The performance of deferred-acceptance auctions. *Math. Oper. Res.*, 42(4):897–914, 2017.

6 Ignacio Esponda and Emanuel Vespa. Hypothetical thinking and information extraction in the laboratory. *American Economic Journal: Microeconomics*, 6(4):180–202, November 2014. doi:10.1257/mic.6.4.180.

7 Diodato Ferraioli, Adrian Meier, Paolo Penna, and Carmine Ventre. On the approximation guarantee of obviously strategyproof mechanisms. *arXiv preprint*, 2018. arXiv:1805.04190.

8 Diodato Ferraioli, Adrian Meier, Paolo Penna, and Carmine Ventre. Automated optimal OSP mechanisms for set systems - the case of small domains. In *Web and Internet Economics - 15th International Conference, WINE 2019, New York, NY, USA, December 10-12, 2019, Proceedings*, pages 171–185, 2019.

9 Diodato Ferraioli, Adrian Meier, Paolo Penna, and Carmine Ventre. Obviously strategyproof mechanisms for machine scheduling. In *27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany*, pages 46:1–46:15, 2019.

10 Diodato Ferraioli, Paolo Penna, and Carmine Ventre. Two-way greedy: Algorithms for imperfect rationality. Submitted for publication, 2020.

11 Diodato Ferraioli and Carmine Ventre. Obvious strategyproofness needs monitoring for good approximations. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

12 Diodato Ferraioli and Carmine Ventre. Probabilistic verification for obviously strategyproof mechanisms. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 240–246. International Joint Conferences on Artificial Intelligence Organization, July 2018.

13 Diodato Ferraioli and Carmine Ventre. Obvious strategyproofness, bounded rationality and approximation. In Dimitris Fotakis and Evangelos Markakis, editors, *Algorithmic Game Theory*, pages 77–91. Springer, 2019.

14 Magnús M. Halldórsson. Approximations of weighted independent set and hereditary subset problems. In *Computing and Combinatorics*, pages 261–270, Berlin, Heidelberg, 1999. Springer.

15 John H. Kagel, Ronald M. Harstad, and Dan Levin. Information impact and allocation rules in auctions with affiliated private values: A laboratory study. *Econometrica*, 55(6):1275–1304, 1987. URL: http://www.jstor.org/stable/1913557.

16 Maria Kyropoulou and Carmine Ventre. Obviously strategyproof mechanisms without money for scheduling. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1574–1581. International Foundation for Autonomous Agents and Multiagent Systems, 2019.

17 Daniel Lehmann, Liadan Ita Oundefinedallaghan, and Yoav Shoham. Truth revelation in approximately efficient combinatorial auctions. *J. ACM*, 49(5):577–602, September 2002. doi:10.1145/585265.585266.

18 Shengwu Li. Obviously strategy-proof mechanisms. *American Economic Review*, 107(11):3257–87, November 2017. doi:10.1257/aer.20160425.

19 Andrew Mackenzie. A revelation principle for obviously strategy-proof implementation. Research Memorandum 014, Maastricht University, Graduate School of Business and Economics (GSBE), May 2018.

20 Marek Pycia and Peter Troyan. Obvious dominance and random priority. In *Proceedings of the 2019 ACM Conference on Economics and Computation*, EC '19, page 1, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3328526.3329613.

Knapsack Secretary with Bursty Adversary

Thomas Kesselheim
University of Bonn, Germany
thomas.kesselheim@uni-bonn.de

Marco Molinaro
PUC-Rio, Rio de Janeiro, Brazil
mmolinaro@inf.puc-rio.br

Abstract

The random-order or *secretary* model is one of the most popular beyond-worst case model for online algorithms. While this model avoids the pessimism of the traditional adversarial model, in practice we cannot expect the input to be presented in perfectly random order. This has motivated research on *best of both worlds* (algorithms with good performance on both purely stochastic and purely adversarial inputs), or even better, on inputs that are a *mix* of both stochastic and adversarial parts. Unfortunately the latter seems much harder to achieve and very few results of this type are known.

Towards advancing our understanding of designing such robust algorithms, we propose a random-order model with *bursts* of adversarial time steps. The assumption of burstiness of unexpected patterns is reasonable in many contexts, since changes (e.g. spike in a demand for a good) are often triggered by a common external event. We then consider the *Knapsack Secretary* problem in this model: there is a knapsack of size k (e.g., available quantity of a good), and in each of the n time steps an item comes with its value and size in $[0,1]$ and the algorithm needs to make an irrevocable decision whether to accept or reject the item.

We design an algorithm that gives an approximation of $1-\tilde{O}(\Gamma/k)$ when the adversarial time steps can be covered by $\Gamma \geq \sqrt{k}$ intervals of size $\tilde{O}(\frac{n}{k})$. In particular, setting $\Gamma = \sqrt{k}$ gives a $(1-O(\frac{\ln^2 k}{\sqrt{k}}))$-approximation that is resistant to up to a $\frac{\ln k}{\sqrt{k}}$-fraction of the items being adversarial, which is almost optimal even in the absence of adversarial items. Also, setting $\Gamma = \tilde{\Omega}(k)$ gives a constant approximation that is resistant to up to a constant fraction of items being adversarial. While the algorithm is a simple "primal" one it does not possess the crucial symmetry properties exploited in the traditional analyses. The strategy of our analysis is more robust and significantly different from previous ones, and we hope it can be useful for other beyond-worst-case models.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases Beyond worst-case, secretary problem, random order, online algorithms, knapsack

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.72

Category Track A: Algorithms, Complexity and Games

Related Version The full version is available at `https://arxiv.org/abs/2006.11607`.

Funding This work was done in part while the authors were visiting the Simons Institute for the Theory of Computing.
Marco Molinaro: MM was supported by CNPq grants Universal #431480/2016-8 and Bolsa de Produtividade em Pesquisa #310516/2017-0, FAPERJ grant Jovem Cientista do Nosso Estado, and a Microsoft Research Fellowship at the Simons Institute for the Theory of Computing.

1 Introduction

In standard competitive analysis of online algorithms, one assumes that an adversary completely defines the input. While this is a useful model for designing algorithms for many problems, for many others this model is too pessimistic and no algorithm can outperform the trivial ones. One classical example is the *Secretary Problem* and its generalizations. In this problem, one is presented a sequence of n items of values $v_1, \ldots, v_n$. Upon each arrival,

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 72; pp. 72:1–72:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

one has to decide *irrevocably* if one accepts or rejects the item, without knowing the value of future items in the sequence. The goal is to select a single item in order to maximize the value obtained. It is easy to see that in the adversarial model the best guarantee possible is to obtain expected value that is a $\frac{1}{n}$-fraction of the offline optimum, and this is achieved by the trivial algorithm that chooses one of the n time steps at random and blindly accepts the item in this time step.

In order to avoid the pessimism of this model and allow for the design of non-trivial algorithms with *hopefully* better performance in practice, there has been a push to consider *beyond worst-case* models. One of the most prominent such models is the *random-order model*, where the adversary can choose the set of items in the instance, but they are presented in uniformly random order. This model has been studied since at least the 60s and has seen a lot of developments in the past decade, and several problems are now well-understood under this model, such as Knapsack and more generally Packing LPs [21, 3, 2, 28, 18, 16, 1], assignment problems [8, 12, 18], matroid optimization [4, 6, 23, 13, 14], and many more. For example, for the Secretary Problem in the random-order model one can obtain a $\frac{1}{e}$-fraction of the offline optimal value (as $n \to \infty$) with the following classical threshold-based algorithm: reject the first $\frac{1}{e}$-fraction of items but note their maximum value, then select the next element which exceeds this value if such an element appears.

However, in practice we cannot expect the sequence to arrive exactly in random order. This has motivated research on *best of both worlds*, namely algorithms with good performance on both purely stochastic and purely adversarial inputs [25, 26, 22, 27]. Even more interesting are algorithms that work well on inputs that are a *mix* of both stochastic and adversarial parts. But this seems to be much harder to achieve: in online algorithms we are only aware of the results of [11] on budgeted allocation (see Section 1.3 for a description of their model and assumptions), while in online learning results of this type have only been obtained very recently for multi-armed bandits [29, 24, 30, 15]. We note that all these results are for settings in which non-trivial guarantees can be achieved for pure adversarial inputs.

Towards advancing our understanding of designing such robust algorithms, we introduce a model that mixes random-order and adversarial time steps, assuming that the latter comes in *bursts*. The random-order times represent when the environment is in a "stationary" or "predictable" state, while the adversarial times represent "unexpected" patterns. The assumption of burstiness of unexpected patterns is reasonable in many contexts, since changes are often triggered by an external common event, e.g., the surge in gun sales after news of possible changes in gun control regulations. See [19, 20, 9, 7] for examples of the different ways in which burstiness can be modeled and areas of applications.

1.1 The *Bursty Adversary plus Random Order* (BARO) model

We describe more formally the general version of the proposed model BARO. Consider an online problem where decisions are made sequentially and irrevocably at times $1, 2, \ldots, n$. In our model, the adversary first chooses some the time steps $Adv \subseteq [n]$ to be "adversarial" and leaves the others $RO = [n] \setminus Adv$ as "random-order" times. In order to capture the burstiness of the adversarial time steps in a clean way, let $\mathbb{W}$ be the partition of $[n]$ into disjoint intervals of length ℓ. We then assume that the adversarial times Adv are covered by at most Γ intervals in $\mathbb{W}$. Notice that this allows various patterns in the adversarial part of the input, including individual (non-bursty) adversarial times as well as bursts of size much larger than ℓ, for a total of up to $\Gamma\ell$ adversarial times. As in the standard random-order model, the items/inputs on the random-order times RO are arbitrary but presented in uniformly random order. The *sequence* items/inputs on the adversarial times Adv is fully adversarial that can be adaptively generated based on an algorithm's behavior and may even depend on the *order* of the items in RO.

It is important to highlight that the algorithm does not know which time steps are adversarial or random-order, and that in each time step only one item arrives (i.e., the adversarial items do not come in batches).

Note that in many problems this adversary can make an instance completely adversarial by sending "dummy" random-order items. For example, in the Secretary Problem the adversary can set the value of all random-order items to be 0; so again no non-trivial guarantees is possible in this case. In order to obtain meaningful guarantees, we compare the algorithm's performance only to the optimum over the random-order times RO, which we denote by OPT_{RO}. Thus, in a maximization problem we say that an algorithm is *α-competitive* in the BARO model if the expected value of the algorithm is at least $\alpha \mathrm{OPT}_{RO}$.

1.2 Our Results

In this paper we use the BARO model to obtain a more robust algorithm for the *Knapsack Secretary* problem, a well-studied generalization of the Secretary Problem. The offline version of the problem is the standard Knapsack Problem: there are n items, each with a value $v_i \geq 0$ and size $w_i \in [0,1]$, and we have a knapsack of size k; the goal is to select a subset of items with total size at most k, and with total value as large as possible.

Our main result is an algorithm for the Knapsack Problem in the BARO model that is resistant to a fraction of items being adversarial.

▶ **Theorem 1.** *There is a* $\big(1 - O\big(\frac{\Gamma\ell}{n} \ln \frac{n}{\Gamma\ell}\big)\big) = \big(1 - O\big(\frac{\Gamma \ln k}{k} \ln \frac{k}{\Gamma \ln k}\big)\big)$*-competitive algorithm for the Knapsack Problem in the BARO model where the adversarial times can be covered by* $\Gamma \geq \sqrt{k}$ *windows of size* $\ell = \frac{n \ln k}{k}$.

Notice that the term $\frac{\Gamma\ell}{n}$ in the guarantee is precisely the fraction of adversarial items that the algorithm can cope with. For example, setting $\Gamma = \sqrt{k}$, our algorithm obtains a $(1 - O(\frac{\ln^2 k}{\sqrt{k}}))$-approximation in the presence of up to a $O(\frac{\ln k}{\sqrt{k}})$-fraction of items being adversarial. For large k this approximation is almost optimal: even in the absence of adversarial items (and even when all items are unit-sized) the best approximation possible is $1 - \Omega(\frac{1}{\sqrt{k}})$ [21] (and this is achieved for example by [28, 18, 1, 16]). Note that these competitive ratios go to 1 as the budget $k \to \infty$ (recall the normalization of sizes being at most 1). Moreover, with $\Gamma = \Omega(\frac{n}{\ell})$ the algorithm achieves a constant approximation in the presence of a constant fraction of adversarial items.

Primal Algorithm with Time-Based Constraints. Our starting point is the *primal* strategy for the random-order model, whose high-level idea is the following: At time t, one solves a knapsack LP with the items seen so far but with budget proportionally scaled to be $\lceil \frac{t}{n} k \rceil$, and pick (a fraction of) the item at time t exactly as prescribed by the optimal LP solution, if there is space available in the full budget of k.

While this strategy obtains the optimal guarantee in the random-order model [18], it fails in the presence of adversarial items. One way in which it fails is by picking "too many items": Suppose that the first k items are adversarial, have size 1, and they all have infinitesimal values but sorted in increasing order, and the random-order items have all value and size equal to 1; it is easy to see that the primal algorithm will pick all the adversarial items, filling up the budget with items of infinitesimal value. (Similar examples exist where the adversarial items are not in the beginning of the sequence.) To counter this, in our algorithm we add additional restrictions, outside of the LP, that the algorithm can only pick a constant mass of items in each window of size $\ell \approx \frac{n}{k}$, which is roughly the behavior of the optimal solution if the n items were in random order.

However, the algorithm may now fail by picking "too few" items: consider the same example as before but now all the adversarial items have value $1+\varepsilon$, thus slightly more valuable than the random-order items. The algorithm will then only pick 1 of these adversarial items (by the new restriction added) and will not pick any of the random-order items, since the LP will always fill up its budget with the better adversarial items; so the algorithm obtains value $1+\varepsilon$, while the $\mathrm{OPT}_{RO} = k$. To avoid this, we also add additional constraints *to the LP* that its solution can select at most a constant number of items in each window of size $\ell \approx \frac{n}{k}$ (note there are $\frac{t}{n}k$ disjoint such windows in $[t]$ and the LP selects total size $\approx \frac{t}{n}k$, again on average 1 per window).

The main difficulty is analyzing the algorithm in the presence of the additional restrictions/constraints. Previous analyses of primal-style algorithms crucially relied on the fact the LP (and its optimal solution) was invariant to the permutation of items/coordinates. This brings about some crucial independence properties: Decisions at time t are independent of the *order* of the arrivals at times $1,\ldots,t-1$ and therefore of the respective decisions. This property allows for the direct use of known concentration inequalities to control the total occupation incurred by the algorithm.

Since our new restrictions/constraints are not permutation invariant, we need to use a different type of analysis. The main handle is what we call the *weighted rank* of an item: the sum of the weights of items with higher value density $\frac{v_i}{w_i}$ than this item, divided by the knapsack capacity. That is, it is by how much one would have to scale the knapsack capacity before the offline optimum would start picking this item. The very high-level idea of the analysis is intuitive: The higher the weighted rank of an item, the smaller its probability of being picked by the LP, even with the new constraints. In addition, while there are complicated dependencies between the events "the algorithms picks the item at time t", the weighted ranks of the items in the random-order times are almost independent: they are just sampled without replacement. We leverage this to obtain custom concentration inequalities that control the algorithm's occupation of the different restrictions/constraints.

1.3 Related Work

As already pointed out above, many algorithms have been proposed for online optimization problems with random arrival order. However, these algorithms usually break when moving to the BARO model. For concreteness, let us illustrate the effect on Kleinberg's algorithm [21] for the multiple-choice secretary problem, a special case of our problem. The algorithm is allowed up to k selections. Throughout the sequence, it never picks items which are not among the best k so far. Therefore, we can construct the following counterexample. Consider a sequence starting with an adversarial burst of k items of very high value, followed by a random-order sequence with items of smaller values. On this sequence, the algorithm will not pick any random-order items at all. If $n \gg k$, then with high probability (over the randomness of the algorithm) none of the adversarial items are picked either (the threshold-based algorithm for the secretary problem is applied to the first $\approx \frac{n}{k}$ items w.h.p., in which case the first $\approx \frac{1}{e}\frac{n}{k} \gg k$ items are rejected). This argument transfers immediately to other algorithms, such as [2, 18]. Other algorithms such as the one by [1] or by [4] use the beginning of the sequence to estimate the optimal value, which also fails in this sequence.

There is only surprisingly little work when it comes to non-uniform random order model. Recently, [17] introduced models where the order of the items is "much less random" than the uniform random order. Among other results, they show that it is possible to obtain constant-competitive algorithms for the Multiple-choice Secretary Problem under these weaker assumptions, and quantify the minimum entropy of the distribution over orders that admits constant-competitive algorithms for the Secretary Problem. We remark that these models do not explicitly contain adversarial items.

Closer in spirit to our model, [11] consider online budgeted allocation in an online model that mixes both stochastic and adversarial inputs. They provide algorithms that are optimal when the input is totally adversarial, and whose performance improves when the instance becomes "more stochastic". There are two crucial differences between our proposed model and Esfandiari et al.'s model: in the latter, while the adversarial items may appear at any point in the sequence (i.e., no burstiness assumption), it is assumed that the algorithm *knows* the distribution of the items in the non-adversarial times, unlike in our model. Also, unlike the Knapsack Problem studied here, the budgeted allocation problem has constant-competitive algorithms even in the adversarial model. Thus, while to some extent an algorithm does not need to worry about "losing everything" if it is fooled by the adversarial part of the instance, its design and analysis have to be delicate enough to obtain fine control over the constants in the competitive-ratio in order to yield interesting results.

In a very recent paper, Bradac et al. [5] present several results for robust secretary problems in a mixed model very similar to ours, which was inspired by a discussion about a preliminary version of this present paper. In contrast to our model, there is no assumption on the number or burstiness of adversarial rounds, making the results incomparable. Our focus is to understand situations in which we are close to the optimal guarantee without adversarial rounds. Since their adversary is more powerful, the guarantees are worse in two ways: (i) Their benchmark is weakened by leaving out the best item. (ii) The guarantees depend on the overall number of rounds n, whereas ours only depend on k. The techniques are also quite different.

2 BARO Knapsack: model and algorithm

Model. We consider an online knapsack problem. The algorithm knows upfront the knapsack size k and the number of items n, and the items are presented online, one-by-one. In the t-th time step, the current item's value V_t and size W_t are revealed, and the algorithm needs to irrevocably decide what fraction $X_t^{alg} \in [0,1]$ of this item to select. Our algorithm's selection is always integral, i.e., $X_t^{alg} \in \{0,1\}$, but our point of comparison is the best fractional solution. The selections made by the algorithm need to fit the knapsack, namely $\sum_{t\in[n]} W_t X_t^{alg} \leq k$ with probability 1, and it tries to maximize the total value of its selections: $\sum_{t\in[n]} V_t X_t^{alg}$. Importantly, the choice in the t-th step has to be made only knowing $V_1, \dots V_t$ and $W_1, \dots, W_t$ (as well as k and n).

The sequences $V_1, \dots, V_n \geq 0$ and $W_1, \dots, W_n \in [0,1]$ are generated by the following *Bursty Adversary plus Random Order* (BARO) model. Let us fix a window size ℓ, and let $\mathbb{W}$ denote the collection of disjoint windows of size ℓ that partitions the time steps $[n]$, that is, $\mathbb{W} = \{\{1,2,\dots,\ell\}, \{\ell+1,\dots,2\ell\}, \dots\}$. For concreteness we will use window size $\ell := \frac{n \ln k}{k}$. The adversary first partitions the n times steps into sets *Adv* (adversarial) and *RO* (random-order) with the property that *Adv* can be covered by Γ windows in $\mathbb{W}$; we use $\mathbb{W}^{adv} \subseteq \mathbb{W}$ to denote one such cover, fixed throughout. The adversary also fixes the items for the random-order times, namely the value/size pairs $(v_1, w_1), (v_2, w_2), \dots, (v_{|RO|}, w_{|RO|})$, with $w_i \in [0,1]$ for all i. Moreover, for each random-order time $t \in RO$, nature samples *without replacement* an index I_t from $\{1, 2 \dots, |RO|\}$, i.e., randomly chooses which random-order item will appear at that time. Then, for each time step t the adversary outputs an item with value V_t and size $W_t \in [0,1]$ as follows:

- (Adversarial) If $t \in Adv$, the adversary outputs an item with arbitrary value V_t and size $W_t \in [0,1]$; this may depend on an algorithm's behavior and on the I_t's.
- (Random-order) If $t \in RO$, the adversary outputs the item indexed by I_t, namely that with value $V_t := v_{I_t}$ and size $W_t := w_{I_t}$.

Note that there is a subtle difference between capital and small letters here. By V_t and W_t, we refer to the value and weight of the item *arriving in the t-th step.* By v_i and w_i we refer to the i-th random-order item specified by the adversary *before the random permutation is applied.* Consequently, V_t and W_t are random variables whereas v_i and w_i are not. Furthermore, since the I_t's are sampled without replacement, the items $((V_t, W_t))_{t \in RO}$ in the random-order times are precisely the items $(v_1, w_1), (v_2, w_2), \ldots, (v_{|RO|}, w_{|RO|})$ randomly permuted.

Again we highlight that the algorithm does not know which time steps are adversarial and which are random-order, and that the adversarial items do not come in batches. As mentioned before, the benchmark for comparison is the offline optimum for the problem on the random-order items alone, namely $\mathrm{OPT}_{RO} := \max\{\sum_i v_i x_i : \sum_i w_i x_i \leq k,\ x \in [0,1]^{|RO|}\}$.

Algorithm. The algorithm we propose is a modification of the primal method of [18] and can be described as follows. Let $\mathbb{W}_t$ be the collection of windows $\mathbb{W}$ truncated to the prefix $[t]$, namely $\{1, \ldots, \ell\}, \{\ell+1, \ldots, 2\ell\}, \ldots, \{\lfloor \frac{t}{\ell} \rfloor \ell + 1, \ldots, t\}$. At time t, in order to compute its selection $X^{alg}_t \in \{0,1\}$ of the current item, the algorithm first finds an optimal solution X^t to the following (random) linear program LP_t:

$$\begin{aligned} &\max \sum_{t' \leq t} V_{t'} X_{t'} \\ &s.t. \sum_{t' \leq t} W_{t'} X_{t'} \leq c_t \frac{t}{n} k && \text{(main inner budget)} \\ &\qquad \sum_{t' \in B} W_{t'} X_{t'} \leq a_1 \frac{\ell}{n} k, \quad \forall B \in \mathbb{W}_t && \text{(inner constraints)} \\ &\qquad X \in [0,1]^t, \end{aligned}$$

where we introduce the slight budget scaling $c_t := (1 - \frac{4\Gamma\ell}{t})$, and set the constant $a_1 := 601$. If $X^t_t > 0$, we say that the algorithm *tentatively picks* the item at time t. The algorithm checks if it can permanently pick this item by verifying whether its past selections $X^{alg}_1, \ldots, X^{alg}_{t-1}$ satisfy the following constraints:

$$\begin{aligned} &\sum_{t' < t} W_{t'} X_{t'} \leq k - 1 && \text{(main budget)} \\ &\sum_{t' \in B_{\text{last}}} W_{t'} X_{t'} \leq a_4 \frac{\ell}{n} k - 1, && \text{(outer constraint)} \end{aligned}$$

where B_{last} denotes the last window in $\mathbb{W}_{t-1}$, and a_4 is a sufficiently large constant (set in Lemma 13). If so, the algorithm fully picks the item, namely it sets $X^{alg}_t = 1$; otherwise we say that it is *blocked* and it does not pick the item at all, setting $X^{alg}_t = 0$.

To get some intuition why the algorithm is reasonable, let us observe how the "offline optimum" OPT_{RO} builds up over time. We can define random variables X^*_t indicating what fraction of the item arriving at time t is packed in OPT_{RO}. Because the permutation is uniformly random, these random variables are identically distributed for all $t \in RO$. More specifically, we have $\mathbb{E}[V_t X^*_t] = \mathrm{OPT}_{RO}/|RO| \approx \mathrm{OPT}_{RO}/n$ and $\mathbb{E}[W_t X^*_t] \leq k/|RO| \approx k/n$. So, *in expectation*, slightly scaled versions of the random variables fulfill all constraints stated above. Our algorithm, of course, does not know X^*_t but tries to mimic this process. Particularly, the goal of (inner constraints) and (outer constraint) is to spread out the choices made by the algorithm over time so that the consequences of adversarial bursts are mitigated.

Notice that by construction the solution X^{alg} returned by the algorithm is always feasible, namely $\sum_{t \le n} W_t X^{alg}_t \le k$. Thus, we only need to argue that it obtains enough value.

▶ **Theorem 2** (Total value). *The expected value of the solution X^{alg} returned by the algorithm satisfies*

$$\mathbb{E}\left[\sum_{t \in RO} V_t X^{alg}_t\right] \ge \left(1 - O\left(\frac{\Gamma \ell}{n} \ln \frac{n}{\Gamma \ell}\right)\right) OPT_{RO}.$$

Roadmap of the analysis. In Section 3 we upper bound for each random-order time t the probability that the algorithm tentatively selects that item. Next, we boost this per-time upper bound into concentration inequalities for the volume of the selections made up to a given point, and use it to upper bound the probability that the algorithm is blocked by constraint (main budget) or (outer constraint), in which case it would not be able to make permanent its tentative selection (Section 4). Using this, we lower bound the value obtained by the algorithm in each (free) random-order time step (Section 5), and add over all such time steps to show that the algorithm obtains the desired value (Section 6). Due to space constraints, the proofs of several lemmas are deferred to the full version of the paper.

Without loss of generality we assume that the random-order times are sorted in decreasing order of value density, namely $\frac{v_1}{w_1} \ge \frac{v_2}{w_2} \ge \ldots \ge \frac{v_{|RO|}}{w_{|RO|}}$. Also, we say that an item is **better** than another if it has higher value density. For simplicity, we also assume that no item has value or weight equal to 0 (else automatically exclude/include in the solution), and that the sum of all item sizes is at least the knapsack size k. We also assume that there are no ties in the value densities $\frac{v_i}{w_i}$; this can be accomplished by infinitesimal perturbations to the values, for example. We also assume $\frac{n}{2} \ge k \ge 80$ and that $\frac{\Gamma \ell}{n} \le \frac{1}{2}$, so at most half of the windows can have adversarial items. With overload of notation, we use I_t to denote the actual item (pair (V_t, W_t)) at time t, even when t is an adversarial time.

3 Controlling tentative selections via weighted rank

We use $T_t := \mathbf{1}(X^t_t > 0)$ to denote the indicator of *tentative* selection by the algorithm at time t. Our goal in this section is to argue that the algorithm does not tentatively select too many items. As mentioned before, the main handle for making this formal is the notion of *weighted rank*. The weighted rank of the random-order item i is a $\frac{1}{k}$ scaling of the sum of the weights of random-order items better than it (recall these items are sorted in decreasing order of value density $\frac{v_i}{w_i}$).

▶ **Definition 3** (Weighted rank). *The* weighted rank *of the random-order item i is $r_i := \frac{1}{k} \sum_{i' < i} w_{i'}$ (we also define $r_{|RO|+1} = \frac{1}{k} \sum_i w_i$ for convenience). For a random-order time t, we use $R_t := r_{I_t}$ to denote the total weighted rank of the item I_t at this time.*

As before, one interpretation of the weighted rank r_i is the following: considering the offline problem with only random-order items, r_i is by how much we need to scale the knapsack of size k before the optimal fractional solution wants to pick a strictly positive fraction of item i. Thus, the higher the rank the worse the item is.

The main result of this section says that the worse the item at time t is, the less likely the algorithm is to tentatively pick it (the extra conditioning on items $(I_{t'})_{t' \in S}$ will be technically useful later and may be ignored throughout at a first read).

▶ **Lemma 4** (UB tentative selection). *Consider a random-order time $t \geq 8\ell(\Gamma+1)$, and a set S of random-order times with $|S| \leq \frac{\ln k}{4}$. Then*

$$\Pr\left(T_t = 1 \;\middle|\; R_t,\, (I_{t'})_{t' \in S}\right) \leq \psi(R_t), \qquad \textit{where } \psi(\gamma) = \begin{cases} 1, & \textit{if } \gamma < 1 \\ \frac{2}{k}, & \textit{if } \gamma \in [1, 50] \\ 4ke^{-\frac{\gamma}{20}\ln k}, & \textit{if } \gamma > 50. \end{cases}$$

For the rest of the section we prove this result. At its heart is the following deterministic monotonicity property of the LP: Fix a scenario (so the LP is deterministic); if there *is* a solution for the LP with only items better than I_t that saturates the main budget, then I_t is not included at all in the *optimal* LP solution. This is clear if we did not have the inner constraints: The optimal LP solution is obtained by the greedy procedure, and if we can saturate the budget with only better items the greedy will stop before reaching I_t. While this does not necessarily hold in the presence of general side constraints, we show it still does under the simple inner constraints. The proof is deferred to the full version of the paper.

▶ **Lemma 5.** *Consider a time $t \in [n]$, and fix a scenario $I_1, I_2, \ldots, I_n$. Suppose that there is a feasible solution $\bar{X}$ of LP_t with $\sum_{t' \leq t} W_{t'} \bar{X}_{t'} = c_t \frac{tk}{n}$ and whose support only includes times with items strictly better than I_t (i.e., $\bar{X}_{t'} > 0$ implies that $I_{t'}$ is strictly better than I_t, for all $t' \in [t]$). Then in any optimal solution X^* of LP_t we have $X^*_t = 0$. (Thus, I_t is not tentatively selected by our algorithm.)*

Our next lemma will leverage this result to show that if there are many items in random-order-only windows better than I_t, then the probability of tentatively selecting the latter is small. Before that, we need to introduce the definition of *free time*, the ones we will focus on for most of the analyses.

▶ **Definition 6** ($\textsc{Free}_t$ and RO_t). *A time is* free *if it does not belong to one of the adversarial windows $\mathbb{W}^{adv}$. We use $\textsc{Free}_t$ to denote the collection of free times in $[t]$. Furthermore, $\mathbb{W}^{free}_t$ denotes the windows from $\mathbb{W}_t$ that only contain free times.*

We also use $RO_t := RO \cap [t]$ to denote all the random-order times (free or otherwise) in $[t]$. With slight abuse in notation, we also use RO_t to denote the cardinality *of RO_t.*

The following estimates follow directly from the assumption that there are at most Γ adversarial windows, each of size ℓ.

▶ **Observation 7.** *The following holds: (a) If $t \geq 2\Gamma\ell$ then $|\textsc{Free}_t| \geq \frac{t}{2}$; (b) $\frac{1}{RO_n} \leq \frac{1}{|\textsc{Free}_n|} \leq \frac{1}{n}(1 + \frac{2\Gamma\ell}{n})$.*

▶ **Lemma 8.** *Consider a random-order time $t \geq 2(\Gamma\ell + 1)$. For a value $\gamma \geq 0$, let G_γ be the event that the sum of the sizes of the items in the times $\textsc{Free}_t$ that are better than I_t equals $\gamma c_t \frac{tk}{n}$ (i.e., $\sum_{t' \in \textsc{Free}_t : I_{t'} < I_t} W_{t'} = \gamma c_t \frac{tk}{n}$). Then for any set of random-order times $S \subseteq RO$ with $|S| \leq \frac{\ln k}{4}$, we have*

$$\Pr(T_t = 1 \mid G_\gamma, I_t, (I_{t'})_{t' \in S}) \leq \frac{1}{2}\psi(\gamma) = \begin{cases} \frac{1}{k}, & \textit{if } \gamma \in [1, 50] \\ 2ke^{-\frac{\gamma}{20}\ln k}, & \textit{if } \gamma > 50. \end{cases}$$

Proof. Condition on I_t, $(I_{t'})_{t' \in S}$, and on the *set* of items $\{I_{t'}\}_{t' \in \textsc{Free}_{t-1}}$ in the free times in a way that the event G_γ holds; let ω denote this conditioning. If suffices to show the upper bound $\Pr(T_t = 1 \mid \omega) \leq \frac{1}{2}\psi(\gamma)$, and the lemma follows by taking expectation with respect to multiple of these ω's. Also notice that this conditioning does not fix the *relative order* of the items in $\textsc{Free}_{t-1} \setminus S$, thus: ($\star$) The items at times $\textsc{Free}_{t-1} \setminus S$ are in random order even when conditioning on ω.

Let E be the event that there is a feasible solution X for LP_t whose support only has items better than I_t and that saturates the main budget, i.e., $\sum_{t' \leq t} W_{t'} X_{t'} = c_t \frac{tk}{n}$. From Lemma 5, whenever E holds I_t is *not* tentatively selected, so it suffices to *lower bound* the probability $\Pr(E \mid \omega)$.

Case 1: $\gamma \in [\mathbf{1}, \mathbf{50}]$**.** If for each of the free windows $\mathbb{W}^{\mathit{free}}_{t-1}$ the total size of items better than I_t in the window is at most $a_1 \ell \frac{k}{n}$ (not "too many good items" in any free window), then any (fractional) selection of these items of total size $c_t \frac{tk}{n}$ gives a feasible solution for LP_t saturating the main budget, so E holds; notice that it is possible to select this much size because we are in the case $\gamma \geq 1$. The intuition is that since the total size of these good items is $\gamma c_t \frac{tk}{n} \leq 50 \frac{tk}{n}$, each window should have about $\frac{\ell}{t} \cdot 50 \frac{tk}{n} = 50 \ell \frac{k}{n}$ of their size in it, so with high probability no window has more than $a_1 \ell \frac{k}{n}$ of their size (recall $a_1 \gg 50$). More formally, consider a free window $B \in \mathbb{W}^{\mathit{free}}_{t-1}$. Let $Z_{B \setminus S} = \sum_{t' \in B \setminus S} \mathbf{1}(I_{t'} < I_t) \cdot W_{t'}$ be the sum of sizes of items in $B \setminus S$ better than I_t, and let $Z = \sum_{t' \in \textsc{Free}_{t-1} \setminus S} \mathbf{1}(I_{t'} < I_t) \cdot W_{t'}$. Notice that under the conditioning ω, Z is a fixed number satisfying $Z \leq \gamma c_t \frac{tk}{n} \leq \gamma \frac{tk}{n}$, and that $Z_{B \setminus S}$ is a sum of terms sampled without replacement from the terms in Z (because of observation ($\star$)). Thus, we have

$$\begin{aligned} \mathbb{E}\left[Z_{B \setminus S} \mid \omega\right] &= \frac{|B \setminus S|}{|\textsc{Free}_{t-1} \setminus S|} \mathbb{E}[Z \mid \omega] \leq \frac{\ell}{|\textsc{Free}_{t-1}| - \ln k} \cdot \frac{\gamma t k}{n} \\ &= \frac{t}{|\textsc{Free}_{t-1}| - \ln k} \cdot \gamma \ell \frac{k}{n} \leq 3 \gamma \ell \frac{k}{n}, \end{aligned}$$

where the last inequality uses the fact that $t \geq 2\Gamma\ell + 1$, Observation 7, and the assumptions $\Gamma \geq \sqrt{k}$ and $k \geq 80$. Moreover, we can apply the Bernstein's Inequality for sampling without replacement (Corollary 2.3 of [16]) conditionally to the sum $Z_{B \setminus S}$ to obtain

$$\Pr\left[Z_{B \setminus S} \geq 600 \ell \frac{k}{n} \;\middle|\; \omega \right] \leq 2 \exp\left(-\frac{9}{7} 3 \gamma \ell \frac{k}{n} \right) \leq 2 \frac{1}{k^3} \leq \frac{1}{k^2},$$

where in the first inequality we also used that $\tau \geq 3 \cdot 3 \gamma \ell \frac{k}{n}$ because $\gamma \leq 50$, and in the last inequality that $k \geq 80$. Since $|S| \leq \frac{\ln k}{4}$ and each item has size at most 1, the items in $B \cap S$ have total size less than $\ell \frac{k}{n}$. Thus, the conditional probability is at most $\frac{1}{k^2}$ that the total size of items in B better than I_t is at least $a_1 \ell \frac{k}{n}$ ("too many good items"). Since there are fewer than k windows, by taking a union bound over all free windows $B \in \mathbb{W}^{\mathit{free}}_t$ we see that with probability at least $1 - \frac{1}{k}$ none of these windows has too many good items. Thus, $\Pr(E \mid \omega) \geq 1 - \frac{1}{k}$.

Case 2: $\gamma > \mathbf{50}$**.** The number of windows in $\mathbb{W}^{\mathit{free}}_{t-1}$ of size ℓ (i.e., possibly excluding the last window) is at least $\mathit{num} := \frac{t}{\ell} - \Gamma - 1$. If in each such window the total size of items better than I_t is at least $2\ell \frac{k}{n}$ ("good items everywhere"), then one can (fractionally) select up to $2\ell \frac{k}{n}$-mass of them in each window and get a feasible solution for LP_t that saturates the main budget; this saturation is possible because this can give a total of size $(2\ell \frac{k}{n}) \cdot \mathit{num} \geq c_t \frac{t}{n} k$ of these better items, where the last inequality uses $t \geq 2\ell(\Gamma + 1)$. Since in this case event E holds, it suffices to lower bound the probability of having good items everywhere. The intuition again is that by assumption there is total mass $\gamma c_t \frac{tk}{n} \geq 12 \frac{tk}{n}$ of these better items, so each window should have about $\frac{\ell}{t} \cdot 12 \frac{tk}{n} = 12 \ell \frac{k}{n}$ size in it, and with high probability all of them should have at least $2\ell \frac{k}{n}$ size in it.

More precisely, considering any fixed window $B \in \mathbb{W}^{free}_{t-1}$ it is easy to obtain the lower estimate $\mathbb{E}\left[Z_{B\setminus S} \,\middle|\, \omega\right] \geq \frac{\gamma \ell k}{3n}$ and again applying Bernstein's Inequality we get $\Pr\left[Z_{B\setminus S} \leq 2\ell\frac{k}{n} \,\middle|\, \omega\right] \leq 2\exp\left(-\frac{\gamma}{20}\ln k\right)$. Taking a union bound over the at most k such windows, the probability that we have enough good items in each window in $\mathbb{W}^{free}_{t-1}$ of size ℓ is at least $1 - 2ke^{-\frac{\gamma}{20}\ln k}$. This concludes the proof. ◀

To conclude the proof of Lemma 4, we show that the item at time t having rank R_t implies with high probability that G_{R_t} holds (actually that the weight in $\textsc{Free}_t$ of items better than I_t is *at least* $R_t c_t \frac{tk}{n}$); this is a consequence of the definition of rank and concentration. With these final details, presented in the full version of the paper, we have the proof of Lemma 4.

4 Controlling the probability of being blocked

In this section we show that with good probability, when the algorithm tentatively selects an item, it also permanently selects it, i.e., it is not blocked by the constraints (main budget) and (outer constraint). More precisely, let $O_t := W_t X^{alg}_t$ be the actual occupation incurred by the the algorithm at time t. We use F_t to denote the indicator of the event that the algorithm is *not* blocked at time t, i.e., $F_t = 1$ if (again B_{last} is the last window in $\mathbb{W}_{t-1}$)

$$\sum_{t' \in B_{\text{last}}} O_{t'} \leq a_4 \frac{\ell}{n} k - 1 \qquad \text{and} \qquad \sum_{t'<t} O_{t'} \leq k - 1, \tag{4.1}$$

otherwise $F_t = 0$. The following is the main result of this section.

▶ **Lemma 9** (Probability of being blocked). *For all free times $t \geq 8\ell(\Gamma+2)$, the probability of being blocked is upper bounded as $\Pr(F_t = 0 \mid I_t) \leq \frac{O(1)}{k\left(1-\frac{t}{n}-a_5\frac{\Gamma \ln k}{k}\right)^2}$, for some constant a_5.*

To prove this lemma, we will upper bound the probability that either of the two parts of (4.1) is violated. For the first part, this will be Lemma 13. The bound for the second part is Lemma 14. Lemma 9 then follows by a union bound.

While the first part of (4.1) only concerns the occupation from free time steps, the second part also includes non-free ones. To control this second part, we will nonetheless focus on the occupation over the free windows; for non-free windows B the outer constraints guarantee $\sum_{t' \in B} O_{t'} \leq O(\frac{\ell k}{n}) = O(\ln k)$, and so all the Γ of these windows combined can consume only $O(\Gamma \ln k)$ of the budget (so, for example, in the important case $\Gamma = \sqrt{k}$ this is negligible).

For the free time steps, it suffices to upper bound the (permanent) occupation O_t by the *tentative occupation* $O'_t := W_t T_t$: For the algorithm to select the item at time t, it is necessary but not sufficient that $T_t = 1$. Therefore, we have $O_t \leq O'_t$ and we focus on controlling the O'_t's from now on.

As a start, we use Lemma 4 to show that in each free time step the expected tentative occupation $\mathbb{E}[O'_t]$ is at most $\approx \frac{k}{n}$; thus, essentially both (4.1) hold in expectation. While what we actually need is a generalization of this result, we present it to illustrate the techniques in a clearer way.

▶ **Lemma 10** (UB tentative occupation). *For all free times $t \geq 8\ell(\Gamma+1)$, we have $\mathbb{E}\left[O'_t\right] \leq \frac{k}{RO_n}\left(1 + O\left(\frac{1}{k}\right)\right)$.*

Proof. Since fixing I_t fixes W_t, using Lemma 4 we have

$$\begin{aligned} \mathbb{E}\, O'_t = \mathbb{E}\, W_t T_t &= \mathbb{E}_{I_t}\left[W_t \cdot \mathbb{E}[T_t \mid I_t]\right] \\ &= \mathbb{E}_{I_t}\left[W_t \cdot \Pr(T_t = 1 \mid I_t)\right] \overset{L.4}{\leq} \mathbb{E}_{I_t}\left[W_t \cdot \psi(R_t)\right] = \frac{1}{RO_n}\sum_i w_i\, \psi(r_i). \end{aligned}$$

Since by definition of rank $r_j = \frac{1}{k}\sum_{j'<j} w_{j'}$, we have $r_{i+1} - r_i = \frac{w_i}{k}$, and thus $w_i = k \cdot \int_{r_i}^{r_{i+1}} 1 \,\mathrm{d}x$. Applying this to the last displayed inequality we get

$$\mathbb{E}\, O'_t \leq \frac{k}{RO_n} \sum_i \int_{r_i}^{r_{i+1}} \psi(r_i)\,\mathrm{d}x. \tag{4.2}$$

Since the item sizes are at most 1, we have $r_{i+1} \leq r_i + \frac{1}{k}$ and so $x - \frac{1}{k} \leq r_i$ for all $x \in [r_i, r_{i+1}]$. Thus, as the function ψ is nonincreasing, the right-hand side of (4.2) is at most

$$\frac{k}{RO_n} \sum_i \int_{r_i}^{r_{i+1}} \psi(x - 1/k)\,\mathrm{d}x \leq \frac{k}{RO_n} \int_0^{\infty} \psi(x - 1/k)\,\mathrm{d}x.$$

Finally, inspecting $\psi(x)$ we see that it takes value 1 for $x < 1$, takes value $\frac{2}{k}$ for $x \in [1, 50]$, and has exponential decay $\leq \frac{e^{-x}}{k}$ after that. Thus, it is easy to see that the integral on the right-hand side is at most $1 + O(\frac{1}{k})$. This concludes the proof. ◀

However, what we actually need is to show that (4.1) (with O'_t's) holds with good probability; for that we need concentration inequalities for the sums of the tentative occupations O'_t's. The biggest problem is that the tentative selections induced by the LP are correlated in a non-trivial way. In particular, it is not clear whether they are negatively associated: for example, if the items up to time $t-1$ are all "very good" the algorithm will not tentatively select at times t, $t+1$, etc., indicating possibility of positive correlations on these times. Thus, the O'_t's are also correlated and it is not clear how to apply standard concentrations inequalities.

4.1 Concentration I: controlling the outer constraint

However, as the example above illustrates, we still have hopes of obtaining good *upper bounds* on the probability of multiple tentative selections. In fact, the probability of multiple selection of items $I_{t_1}, \ldots, I_{t_m}$ is at most the probability that the "worst" of these items is selected; more precisely:

▶ **Lemma 11.** *Consider $m \leq \frac{\ln k}{4}$ random-order times $t_1, \ldots, t_m \geq 8\ell(\Gamma + 1)$. Then*

$$\Pr\left(T_{t_1} = \ldots = T_{t_m} = 1 \;\middle|\; R_{t_1}, \ldots, R_{t_m}\right) \leq \psi\left(\max_i R_{t_i}\right).$$

Proof. The inequality follows from the fact $\Pr(X_1 = \ldots = X_m = 1 \mid E) \leq \min_i \Pr(X_i = 1 \mid E)$, Lemma 4, and $\min_i \psi(R_{t_i}) = \psi(\max_i R_{t_i})$ (by the monotonicity of ψ). ◀

The main advantage of this bound is that the ranks R_{t_i} are "almost" independent (they would be independent if the input sequence was generated by sampling items *with* replacement).

Moreover, this lemma allows us to upper bound products of tentative occupation $\prod_i O'_{t_i}$: for this product to be strictly positive, all these items have to be tentatively selected. In fact, one can prove such upper bound using a similar strategy as in Lemma 10, with a main new element: a simple but general comparison for the expectation of a *non-negative* function under sampling with and without replacement, that allow us to work with a decoupled (independent) version $R'_{t_1}, \ldots, R'_{t_m}$ of the ranks. Formally we have the following, which is deferred to the full version of the paper.

▶ **Lemma 12** (Control of products). *Fix a random-order time t. Consider a set of $m \leq \frac{\ln k}{4}$ distinct RO times $t_1, \dots, t_m$, all of which are at least $8\ell(\Gamma+1)$ and less than t. Then there are constants $a_2, a_3 > 1$ such that $\mathbb{E}\left[\prod_{i\in[m]} O'_{t_i} \,\middle|\, I_t\right] \leq \left(1 + \frac{a_2^m}{k}\right)\left(1 + \frac{4m^2}{RO_n}\right)\left(\frac{k}{RO_n}\right)^m \leq \left(a_3 \frac{k}{n}\right)^m$. In particular, choosing $a_2 = 500$ and $a_3 = 8a_2$ is sufficient.*

Finally, these product estimates can be converted into raw moments/tail inequalities using reasonably standard estimates (e.g., Section 3.4 of [10]), giving the desired control of the outer constraint's occupation (the proof is deferred to the full version of the paper).

▶ **Lemma 13** (Control of outer constraints). *Consider a free time $t \geq 8\ell(\Gamma+1)$, and let B be the last window in $\mathbb{W}_{t-1}$. Then $\Pr\left(\sum_{t'\in B} O'_{t'} > a_4 \ell \frac{k}{n} \,\middle|\, I_t\right) \leq \frac{1}{k}$, where $a_4 \geq 2e^6 a_3$, and a_3 is the constant from Lemma 12.*

4.2 Concentration II: control of main budget

In order to obtain Lemma 9 we need to show that the second part of (4.1) holds with reasonable probability *even when $t \approx n$*; but since $\mathbb{E}O'_t \approx \frac{k}{n}$, the expected cumulative occupation by the end of the game $\mathbb{E}[\sum_{t'=1}^{t} O'_{t'}]$ is $\approx k$ for $t \approx n$, so we do not have much room. So unlike the previous section, we are interested in "'medium deviations", where the variance is the right quantity to look at. While Lemma 12 directly gives that the cumulative variance until time t is $\lesssim (\frac{tk}{n})^2$, we actually need an upper bound of order $O(\frac{tk}{n})$, which is what one would expect from independent Bernoulli's with success probability $\frac{k}{n}$. Since $\text{Var}(Z) = \mathbb{E}Z^2 - (\mathbb{E}Z)^2$, to obtain variance upper bounds we will obtain an upper bound on the second raw moment and a *lower bound on the expectation.*

For that we actually prove concentration for a modified occupation that upper bounds O'_t in every scenario: $\bar{O}'_t = W_t \bar{T}_t$, where $\bar{T}_t := \max\{T_t, \mathbf{1}[R_t \leq 1]\}$, that is $\bar{T}_t$ equals 1 if either $T_t = 1$ or the weighted rank R_t is at most 1. While Lemma 12 still holds for these variables, we can now get almost matching lower bounds on $(\mathbb{E}\sum_{t'=1}^{t} \bar{O}'_{t'})^2$, giving the desired variance bound $\text{Var}[\sum_{t'=1}^{t} \bar{O}'_{t'}] \leq O(\frac{tk}{n})$. Essentially using the bound in expectation from Lemma 10 and Chebychev's inequality with this variance control gives the following (see the full version of the paper).

▶ **Lemma 14** (Control of main budget). *For every random-order time t, the probability we are blocked by the main budget can be upper bounded as $\Pr\left[\sum_{t'<t} O'_t > k - 1 \,\middle|\, I_t\right] \leq \frac{O(1)}{k\left(1 - \frac{t}{n} - O\left(\frac{\Gamma \ln k}{k}\right)\right)^2}$.*

Taking a union bound over Lemma 13 and Lemma 14 proves Lemma 9.

5 Lower bounding the value obtained

Recall that $X_t^{alg} = T_t F_t$, i.e., the item is permanently selected exactly when it is tentatively selected and it fits the budgets, and that V_t is the value of the item at time t. The following is then our main lower bound on the value obtained by the algorithm.

▶ **Lemma 15** (Value lower bound). *Consider a free time $t \geq 1{,}212\Gamma\ell$. Then $\mathbb{E}[V_t T_t F_t] \geq \left(c_t - \varepsilon_t - p_t - \frac{2}{k}\right) \frac{OPT_{RO}}{RO_n}$, where p_t is the bound from Lemma 9 and $\varepsilon_t = (a_1 + 3)\frac{\Gamma\ell}{t} + \sqrt{10 \ln k}\sqrt{\frac{2n}{tk}}$.*

To prove this, the first step is to obtain the following “dual” to Lemma 4, which says that if the item has low (i.e. good) rank then it is fully tentatively selected with good probability.

▶ **Lemma 16.** *For any free time* $t \geq 1{,}212\Gamma\ell$ *we have that the probability of fully tentatively selecting item* I_t *satisfies:* $\Pr\left(X_t^t = 1 \;\middle|\; R_t \leq c_t - \varepsilon_t\right) \geq 1 - \frac{1}{k}$.

This is proved via a “dual” of Lemma 5 plus concentration. Given that good items are tentatively selected with good probability and it is likely to fit the budget (Lemma 9), final difficulty in proving Lemma 15 is the correlation between these quantities and the value V_t. This is why most previous bounds actually work even conditioned on the item I_t; notice that fixing I_t, fixes V_t as a deterministic constant. We sketch the argument.

Proof sketch of Lemma 15. Using the non-negativity of V_t, T_t, and F_t, we have

$$\begin{aligned}\mathbb{E}[V_t T_t F_t] = \mathbb{E}_{I_t}\left[V_t\, \mathbb{E}\left[T_t F_t \mid I_t\right]\right] &\geq \mathbb{E}_{I_t}\big(V_t\, \mathbb{E}\left[T_t F_t \mid I_t\right] \;\big|\; R_t \leq c_t - \varepsilon_t\big) \Pr(R_t \leq c_t - \varepsilon_t)\\ &\geq \left(1 - p_t - \frac{1}{k}\right) \mathbb{E}_{I_t}\big(V_t \;\big|\; R_t \leq c_t - \varepsilon_t\big) \Pr(R_t \leq c_t - \varepsilon_t),\end{aligned}$$

where the second inequality uses Lemmas 16 and 9 (via the definition of p_t). Since $c_t - \varepsilon_t \approx 1$ and essentially OPT_{RO} gets value exactly from items of rank at most 1, the last two factors in the inequality together give roughly the expected value OPT_{RO} gets in a time step. ◀

6 Wrapping up: finishing the proof of Theorem 2

To finish the proof we just need to add Lemma 15 over all free time steps except the ones very early or very late in the sequence. More precisely, let $t_0 = 1{,}212\Gamma\ell$ and $\gamma = 1 - (a_5 + 1)\frac{\Gamma\ell}{n}$, and define $T = \{t \in \textsc{Free}_n : t_0 \leq t \leq \gamma n\}$. For $t \notin T$, we use the trivial bound $V_t \geq 0$. For the other time steps we use Lemma 15. With the fact $RO_n \leq n$ we have

$$\mathbb{E}\left[\sum_{t \in RO} V_t X_t^{alg}\right] \geq \sum_{t \in T} \mathbb{E}[V_t T_t F_t] \geq \left(\sum_{t \in T} c_t - \sum_{t \in T} \varepsilon_t - \sum_{t \in T} p_t - \frac{2n}{k}\right) \frac{\mathrm{OPT}_{RO}}{n}. \tag{6.3}$$

We bound each of the remaining sums:

- $\sum_{t \in T} c_t = |T| - \sum_{t \in T} \frac{4\Gamma\ell}{t} \geq (n - O(\Gamma\ell)) - \int_{t_0 - 1}^{n} \frac{4\Gamma\ell}{t}\,\mathrm{d}t = n - O(\Gamma\ell \ln \frac{n}{\Gamma\ell})$.
- $\sum_{t \in T} p_t \leq O\left(\frac{1}{k}\right) \cdot \int_0^{\gamma n} \frac{1}{\left(1 - \frac{t}{n} - a_5 \frac{\Gamma \ln k}{k}\right)^2}\,\mathrm{d}t = O\left(\frac{n}{k}\right) \int_0^{\gamma} \frac{1}{(a - x)^2}\,\mathrm{d}x$, where in the last step we set $a = 1 - a_5 \frac{\Gamma \ln k}{k}$ and use change of variables $x = \frac{t}{n}$. The remaining integral equals $\frac{1}{a - x}\Big|_0^{\gamma} \leq \frac{1}{a - \gamma}$. By our setting of γ we have $a - \gamma = \frac{\Gamma\ell}{n}$, so we obtain $\sum_{t \in T} p_t \leq O\left(\frac{n}{\Gamma}\right)$.
- $\sum_{t \in T} \varepsilon_t \leq \int_{t_0 - 1}^{n} (a_1 + 3) \frac{\Gamma\ell}{t}\,\mathrm{d}t + \int_{t_0 - 1}^{n} \sqrt{\frac{20 n \ln k}{kt}}\,\mathrm{d}t \leq O(\Gamma\ell \ln \frac{n}{\Gamma\ell}) + O(\frac{n\sqrt{\ln k}}{\sqrt{k}})$.

Using these bounds on (6.3) and the assumption $\Gamma \geq \sqrt{k}$ concludes the proof of the theorem.

7 Conclusions

A natural follow-up question is how our results could generalize to other settings. In particular, it would be interesting to extend our algorithm and analysis to packing LPs. The difficulty in using our technique is that there is no natural notion similar to the weighted rank for this setting.

It would also be interesting to better understand the limitations and trade-offs in this and similar models. For example, what regimes of parameter allow constant-competitive or $(1 - \varepsilon)$-competitive algorithms?

References

1 Shipra Agrawal and Nikhil R. Devanur. Fast algorithms for online stochastic convex programming. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 1405–1424, 2015. doi:10.1137/1.9781611973730.93.

2 Shipra Agrawal, Zizhuo Wang, and Yinyu Ye. A dynamic near-optimal algorithm for online linear programming. *Operations Research*, 62(4):876–890, 2014. doi:10.1287/opre.2014.1289.

3 Moshe Babaioff, Nicole Immorlica, David Kempe, and Robert Kleinberg. A knapsack secretary problem with applications. In *APPROX-RANDOM*, 2007.

4 Moshe Babaioff, Nicole Immorlica, and Robert Kleinberg. Matroids, secretary problems, and online mechanisms. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 434–443, 2007.

5 Domagoj Bradac, Anupam Gupta, Sahil Singla, and Goran Zuzic. Robust algorithms for the secretary problem. In Thomas Vidick, editor, *11th Innovations in Theoretical Computer Science Conference, ITCS 2020, January 12-14, 2020, Seattle, Washington, USA*, volume 151 of *LIPIcs*, pages 32:1–32:26. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ITCS.2020.32.

6 Sourav Chakraborty and Oded Lachish. Improved competitive ratio for the matroid secretary problem. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1702–1712, 2012. doi:10.1137/1.9781611973099.135.

7 G.R. Dattatreya. *Performance Analysis of Queuing and Computer Networks (Chapman & Hall/Crc Computer & Information Science Series)*. Chapman & Hall/CRC, 2008.

8 Nikhil R. Devanur and Thomas P. Hayes. The adwords problem: online keyword matching with budgeted bidders under random permutations. In *EC*, 2009.

9 Qiming Diao, Jing Jiang, Feida Zhu, and Ee-Peng Lim. Finding bursty topics from microblogs. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers - Volume 1*, ACL '12, pages 536–544, 2012.

10 Devdatt Dubhashi and Alessandro Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, New York, NY, USA, 1st edition, 2009.

11 Hossein Esfandiari, Nitish Korula, and Vahab Mirrokni. Online allocation with traffic spikes: Mixing adversarial and stochastic models. In *Proceedings of the Sixteenth ACM Conference on Economics and Computation*, EC '15, pages 169–186, New York, NY, USA, 2015. ACM. doi:10.1145/2764468.2764536.

12 Jon Feldman, Monika Henzinger, Nitish Korula, Vahab S. Mirrokni, and Clifford Stein. Online stochastic packing applied to display ad allocation. In *ESA*, 2010.

13 Moran Feldman, Ola Svensson, and Rico Zenklusen. A simple o(log log(rank))-competitive algorithm for the matroid secretary problem. In *Proceedings of the Twenty-sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '15, pages 1189–1201, 2015.

14 Moran Feldman, Ola Svensson, and Rico Zenklusen. A framework for the secretary problem on the intersection of matroids. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '18, pages 735–752, 2018.

15 Anupam Gupta, Tomer Koren, and Kunal Talwar. Better algorithms for stochastic bandits with adversarial corruptions. In *Conference on Learning Theory, COLT 2019, 25-28 June 2019, Phoenix, AZ, USA*, pages 1562–1578, 2019. URL: http://proceedings.mlr.press/v99/gupta19a.html.

16 Anupam Gupta and Marco Molinaro. How the experts algorithm can help solve lps online. *Mathematics of Operations Research*, 41(4):1404–1431, 2016. doi:10.1287/moor.2016.0782.

17 Thomas Kesselheim, Robert D. Kleinberg, and Rad Niazadeh. Secretary problems with non-uniform arrival order. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 879–888, 2015. doi:10.1145/2746539.2746602.

18 Thomas Kesselheim, Andreas Tönnis, Klaus Radke, and Berthold Vöcking. Primal beats dual on online packing lps in the random-order model. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, STOC '14, pages 303–312, New York, NY, USA, 2014. ACM. doi:10.1145/2591796.2591810.

19 Jon Kleinberg. Bursty and hierarchical structure in streams. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02, pages 91–101, New York, NY, USA, 2002. ACM. doi:10.1145/775047.775061.

20 Jon Kleinberg, Yuval Rabani, and Éva Tardos. Allocating bandwidth for bursty connections. In *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing*, STOC '97, pages 664–673, New York, NY, USA, 1997. ACM. doi:10.1145/258533.258661.

21 Robert Kleinberg. A multiple-choice secretary algorithm with applications to online auctions. In *SODA*, 2005.

22 Nitish Korula, Vahab Mirrokni, and Morteza Zadimoghaddam. Online submodular welfare maximization: Greedy beats 1/2 in random order. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*, STOC '15, pages 889–898, 2015. doi:10.1145/2746539.2746626.

23 O. Lachish. O(log log rank) competitive ratio for the matroid secretary problem. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 326–335, October 2014. doi:10.1109/FOCS.2014.42.

24 Thodoris Lykouris, Vahab Mirrokni, and Renato Paes Leme. Stochastic bandits robust to adversarial corruptions. In *STOC 2018*, 2018.

25 A. Meyerson. Online facility location. In *Proceedings of the 42Nd IEEE Symposium on Foundations of Computer Science*, FOCS '01, pages 426–431, Washington, DC, USA, 2001. IEEE Computer Society. URL: http://dl.acm.org/citation.cfm?id=874063.875567.

26 Vahab S. Mirrokni, Shayan Oveis Gharan, and Morteza Zadimoghaddam. Simultaneous approximations for adversarial and stochastic online budgeted allocation. In *Proceedings of the Twenty-third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '12, pages 1690–1701, 2012.

27 Marco Molinaro. Online and random-order load balancing simultaneously. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '17, pages 1638–1650, 2017.

28 Marco Molinaro and R. Ravi. Geometry of online packing linear programs. In Artur Czumaj, Kurt Mehlhorn, Andrew Pitts, and Roger Wattenhofer, editors, *Automata, Languages, and Programming*, pages 701–713, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

29 Yevgeny Seldin and Aleksandrs Slivkins. One practical algorithm for both stochastic and adversarial bandits. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, pages 1287–1295, 2014. URL: http://proceedings.mlr.press/v32/seldinb14.html.

30 Julian Zimmert and Yevgeny Seldin. An optimal algorithm for stochastic and adversarial bandits. In *The 22nd International Conference on Artificial Intelligence and Statistics, AISTATS 2019, 16-18 April 2019, Naha, Okinawa, Japan*, pages 467–475, 2019. URL: http://proceedings.mlr.press/v89/zimmert19a.html.

The Iteration Number of Colour Refinement

Sandra Kiefer
RWTH Aachen University, Germany
kiefer@cs.rwth-aachen.de

Brendan D. McKay
Australian National University, Canberra, Australia
brendan.mckay@anu.edu.au

Abstract

The Colour Refinement procedure and its generalisation to higher dimensions, the Weisfeiler-Leman algorithm, are central subroutines in approaches to the graph isomorphism problem. In an iterative fashion, Colour Refinement computes a colouring of the vertices of its input graph.

A trivial upper bound on the iteration number of Colour Refinement on graphs of order n is $n-1$. We show that this bound is tight. More precisely, we prove via explicit constructions that there are infinitely many graphs G on which Colour Refinement takes $|G|-1$ iterations to stabilise. Modifying the infinite families that we present, we show that for every natural number $n \geq 10$, there are graphs on n vertices on which Colour Refinement requires at least $n-2$ iterations to reach stabilisation.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Mathematics of computing → Combinatorial algorithms; Mathematics of computing → Graph theory

Keywords and phrases Colour Refinement, iteration number, Weisfeiler-Leman algorithm, quantifier depth

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.73

Category Track A: Algorithms, Complexity and Games

Related Version A full version of the paper is available at [21], `http://arxiv.org/abs/2005.10182`.

1 Introduction

Colour Refinement, which is also known as Naïve Vertex Classification or the 1-dimensional Weisfeiler-Leman algorithm (1-WL), is an important combinatorial algorithm in theoretical and practical approaches to the graph isomorphism problem. In an iterative fashion, it refines an isomorphism-invariant partition of the vertex set of the input graph. This process stabilises at some point and the final partition can often be used to distinguish non-isomorphic graphs [3]. Colour Refinement can be implemented to run in time $O((m+n)\log n)$, where n is the order of the input graph and m is its number of edges [6, 29]. Most notably, its efficient implementations are used in all competitive graph isomorphism solvers (such as `Nauty` and `Traces` [30], `Bliss` [19] and `saucy` [7]).

Colour Refinement has been rediscovered many times, one of its first occurences being in a paper on chemical information systems from the 1960s [31]. The procedure is applied in plenty of other fields, for example, it can be modified to reduce the dimension of linear programs significantly [14]. Other applications are in the context of graph kernels [34] or static program analysis [27]. A recently discovered connection to deep learning shows that the expressive power of Colour Refinement is captured by graph neural networks [32].

As described above, Colour Refinement computes a stable colouring of its input graph. It is known that two given graphs result in equal colourings, i.e. are not distinguished by Colour Refinement, if and only if there is a fractional isomorphism between them [12, 33, 35]. Moreover, the graphs which Colour Refinement identifies up to isomorphism (i.e. distinguishes from all non-isomorphic ones) have been completely characterised [2, 24].

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 73; pp. 73:1–73:19
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

To obtain its final colouring, the algorithm proceeds in iterations. In this paper, we investigate how many iterations it takes for the algorithm to terminate. More specifically, for $n \in \mathbb{N}$, we are interested in $\mathrm{WL}_1(n)$, the maximum number of iterations required to reach stabilisation of Colour Refinement among all graphs of order n.

While not directly linked to the running time on a sequential machine, the iteration number corresponds to the parallel running time of Colour Refinement (on a standard PRAM model) [17, 25]. Furthermore, via a connection to counting logics, a bound on the iteration number for graphs of a fixed size directly translates into a bound on the descriptive complexity of the difference between the two graphs, namely into a bound on the quantifier depth of a distinguishing formula in the extension of the 2-variable fragment of first-order logic by counting quantifiers [5, 18]. Moreover, the iteration number of 1-WL equals the depth of a graph neural network that outputs the stable vertex colouring of the underlying graph with respect to Colour Refinement [32].

Considering paths, one quickly determines that $\mathrm{WL}_1(n) \geq \frac{n}{2} - 1$ holds for every $n \in \mathbb{N}$. By contrast, on random graphs, the iteration number is asymptotically almost surely 2 [3]. The best published lower bound on the iteration number of Colour Refinement on n-vertex graphs is $n - O(\sqrt{n})$ [26]. Concerning the upper bound, the trivial inequality $\mathrm{WL}_1(n) \leq n-1$ holds for every repeated partitioning of a set of size n and it does not take into account any further properties of the input graph or of the algorithm used to execute the partitioning. Still, no improvement over this upper bound has been established.

Our first main result reads as follows.

▶ **Theorem 1.** *For every $n \in \mathbb{N}_{\geq 10}$ with $n = 12$ or $n \bmod 18 \notin \{6, 12\}$, it holds that* $\mathrm{WL}_1(n) = n - 1$.

Thus, there are infinitely many $n \in \mathbb{N}$ with $\mathrm{WL}_1(n) = n - 1$. We can even determine the iteration number up to an additive constant of 1 for all $n \in \mathbb{N}$ (where the precise numbers for $n \leq 9$ can easily be determined computationally), as stated in our second main result.

▶ **Theorem 2.** *For every $n \in \mathbb{N}_{\geq 10}$, it holds that* $\mathrm{WL}_1(n) \in \{n-2, n-1\}$.

We obtain our bounds via an empirical approach. More precisely, we have designed a procedure that enables us to systematically generate for all $n \leq 64$ graphs of order n that obey certain constraints (to render the procedure tractable) and on which Colour Refinement takes $n - 1$ iterations to stabilise. Analysing the graphs, we have determined the connections between colour classes during the execution of the algorithm in detail. If the vertex degrees that are present in the graph are low, then the connections between colour classes of size 2 are restricted. This allows us to develop an elegant graphical visualisation and a compact string representation of the graphs with low vertex degrees that take $n - 1$ iterations to stabilise. Using these encodings, we are able to provide infinite families with $n - 1$ Colour Refinement iterations until stabilisation.

Our analysis enables a deep understanding of the families that we present. Via slight modifications of the graph families, we can then cover a large portion of graph sizes and, allowing to go from connected graphs to general graphs, we can construct the graphs that yield Theorem 2.

Due to space limits, we omit some of the proof details here and defer the reader to the full version for them [21].

Related work

Colour Refinement is the 1-dimensional version of the so-called Weisfeiler-Leman algorithm. For every $k \in \mathbb{N}$, there exists a generalisation of it (k-WL), which colours vertex k-tuples in the input graph instead of single vertices only. See [20] for an in-depth study of the main parameters of Colour Refinement and k-WL.

Similarly as for Colour Refinement, one can consider the number $\mathrm{WL}_k(n)$ of iterations of k-WL on graphs of order n. Notably, contrasting our results for Colour Refinement, in [22], it was first proved that the trivial upper bound of $\mathrm{WL}_2(n) \leq n^2 - 1$ is not even asymptotically tight (see also the journal version [23]). This foundation fostered further work, leading to an astonishingly good new upper bound of $O(n \log n)$ on the iteration number of 2-WL [28].

For fixed $k > 1$, it is already non-trivial to show linear lower bounds on $\mathrm{WL}_k(n)$. Modifying a construction of Cai, Fürer, and Immerman [5], this was achieved by Fürer [9], who showed that $\mathrm{WL}_k(n) \in \Omega(n)$, remaining to date the best known lower bound when the input is supposed to be a graph. Only when considering structures with relations of higher arity than 2 as input, better lower bounds on the iteration number of k-WL have been proved [4].

For $k > 2$, regarding upper bounds on the iteration number of k-WL, without further knowledge about the input graph, no significant improvements over the trivial $n^k - 1$ are known.[1] Still, when the input graph has bounded treewidth or is a 3-connected planar graph, polylogarithmic bounds on the iteration number of k-WL needed to identify the graph have been published [17, 36].

Although for every k, there are non-isomorphic graphs that are not distinguished by k-WL [5], it is known that for every graph class with a forbidden minor, a sufficiently high-dimensional Weisfeiler-Leman algorithm correctly decides isomorphism [13]. Recent results give new upper bounds on the dimension needed for certain interesting graph classes [15, 16]. A closely-related direction of research investigates what properties the Weisfeiler-Leman algorithm can detect in graphs [1, 8, 10].

2 Preliminaries

By $\mathbb{N}$, we denote the set of natural numbers, i.e. $\{1, 2, \dots\}$. We set $\mathbb{N}_0 := \mathbb{N} \cup \{0\}$ and, for $k, \ell \in \mathbb{N}_0$, we define $[k, \ell] := \{n \in \mathbb{N}_0 \mid k \leq n \leq \ell\}$ and $[k] := [1, k]$. For a set S, a *partition* of S is a set Π of non-empty sets such that $\bigcup_{M \in \Pi} M = S$ and for all $M, M' \in \Pi$ with $M \neq M'$, it holds that $M \cap M' = \emptyset$. For two partitions Π and Π' of the same set S, we say that Π' is *finer* than Π (or Π' *refines* Π) if every element of Π' is a (not necessarily proper) subset of an element of Π. We write $\Pi \succeq \Pi'$ (and equivalently $\Pi' \preceq \Pi$) to express that Π' is finer than Π. Concurrently, we say that Π is *coarser* than Π'. Note that if both $\Pi \succeq \Pi'$ and $\Pi' \succeq \Pi$ hold, then $\Pi = \Pi'$.

For $S \neq \emptyset$, the partition $\{S\}$ is the *unit* partition of S. The partition $\big\{\{s\} \mid s \in S\big\}$ is called the *discrete partition* of S. A set of cardinality 1 is a *singleton*.

All graphs that we consider in this paper are finite and simple, i.e. undirected without self-loops at vertices. For a graph with vertex set $V(G)$ and edge set $E(G)$, its *order* is $|G| := |V(G)|$. For a vertex $v \in V(G)$, we denote by $N(v)$ the neighbourhood of v in G, i.e. the set $\{w \mid \{v, w\} \in E(G)\}$. Similarly, for a vertex set W, we set $N(W) := \big\{v \mid v \notin W, \exists w \in W \colon v \in N(w)\big\}$. The *degree* of a vertex v is $\deg(v) := |N(v)|$. (Since the graph G

[1] The bound $n^k - 1$ is not tight, since the initial partition of the k-tuples already has multiple classes, for example, one consisting of all tuples of the form $(v, v, \dots, v)$.

will be clear from the context, we do not need to include it in our notation.) We also set $\deg(G) \coloneqq \{\deg(v) \mid v \in V(G)\}$. If there is a $d \in \mathbb{N}_0$ such that $\deg(G) = \{d\}$, the graph G is d-regular. A *regular* graph is a graph that is d-regular for some $d \in \mathbb{N}_0$. By a *matching*, we mean a 1-regular graph.

Let G be a graph with at least two vertices. If there are sets $\emptyset \neq P, Q \subseteq V(G)$ such that $V(G) = P \cup Q$ and $P \cap Q = \emptyset$ and $E(G) \cap \{\{v,w\} \mid v,w \in P\} = \emptyset = E(G) \cap \{\{v,w\} \mid v,w \in Q\}$, then G is *bipartite (on bipartition* (P,Q)*)*. If, additionally, $\big\{\{v,w\} \mid v \in P, w \in Q\big\} = E(G)$, the graph G is *complete bipartite*.

For $k, \ell \in \mathbb{N}_0$, a *(k,ℓ)-biregular graph (on bipartition (P,Q))* is a bipartite graph on bipartition (P,Q) such that for every $v \in P$, it holds that $|N(v)| = k$, and for every $w \in Q$, it holds that $|N(w)| = \ell$. A *biregular* graph is a graph G for which there are $P, Q \subseteq V(G)$ and $k, \ell \in \mathbb{N}_0$ such that G is (k,ℓ)-biregular on bipartition (P,Q).

For a graph G and a set $V' \subseteq V(G)$, we let $G[V']$ be the *induced subgraph of G on V'*, i.e. the subgraph of G with vertex set V' and edge set $E(G) \cap \{\{v,w\} \mid v,w \in V'\}$. We define $G - V' \coloneqq G[V(G) \setminus V']$. Furthermore, for vertex sets $V_1, V_2 \subseteq V(G)$, we denote by $G[V_1, V_2]$ the graph with vertex set $V_1 \cup V_2$ and edge set $E(G) \cap \{\{v_1, v_2\} \mid v_1 \in V_1, v_2 \in V_2\}$.

A *coloured graph* is a tuple (G, λ), where G is a graph and $\lambda\colon V(G) \to \mathcal{C}$ is a function that assigns colours (i.e. elements from a particular set $\mathcal{C}$) to the vertices. We interpret all graphs treated in this paper as coloured graphs. If the colouring is not specified, we assume a monochromatic colouring, i.e. all vertices have the same colour.

For a coloured graph G with colouring λ, a *(vertex) colour class* of G is a maximal set of vertices that all have the same λ-colour. Every graph colouring λ induces a partition $\pi(\lambda)$ of $V(G)$ into the vertex colour classes with respect to λ.

3 Colour Refinement

Colour Refinement proceeds by iteratively refining a partition of the vertex set of its input graph until the partition is stable with respect to the refinement criterion.

▶ **Definition 3** (Colour Refinement). *Let $\lambda\colon V(G) \to \mathcal{C}$ be a colouring of the vertices of a graph G, where $\mathcal{C}$ is some set of colours. The colouring computed by Colour Refinement on input (G,λ) is defined recursively: we set $\chi_G^0 \coloneqq \lambda$, i.e. the initial colouring is λ. For $i \in \mathbb{N}$, the colouring χ_G^i computed by Colour Refinement after i iterations on G is defined as $\chi_G^i(v) \coloneqq \big(\chi_G^{i-1}(v), \{\!\{\chi_G^{i-1}(w) \mid w \in N(v)\}\!\}\big)$.*

That is, $\chi_G^i(v)$ consists of the colour of v from the previous iteration as well as the multiset of colours of neighbours of v from the previous iteration. It is not difficult to see that $\pi(\chi_G^{i-1}) \succeq \pi(\chi_G^i)$ holds for every graph G and every $i \in \mathbb{N}$. Therefore, there is a unique minimal integer j such that $\pi(\chi_G^j) = \pi(\chi_G^{j+1})$. For this value j, we define the *output* of Colour Refinement on input G to be $\chi_G \coloneqq \chi_G^j$ and call χ_G and $\pi(\chi_G)$ *the stable colouring* and *the stable partition*, respectively, of G. Accordingly, executing i *Colour Refinement iterations* on G means computing the colouring χ_G^i. We call a graph G with initial colouring λ and the induced partition $\pi(\lambda)$ *stable* if $\pi(\lambda) = \pi(\chi_G)$. Note that if λ is stable, then for all $P, Q \in \pi(\lambda)$ with $P \neq Q$, the graph $G[P]$ is regular and the graph $G[P,Q]$ is biregular.

Colour Refinement can be used to check whether two given graphs G and G' are non-isomorphic by computing the stable colouring on the disjoint union of the two. If there is a colour C such that, in the stable colouring, the numbers of vertices of colour C differ in G and G', they are non-isomorphic. However, even if they agree in every colour class size in the stable colouring, the graphs might not be isomorphic. It is non-trivial to describe for which graphs this isomorphism test is always successful (see [2, 24]).

▶ **Notation 4.** *We write* $\mathrm{WL}_1(G)$ *for the number of iterations of Colour Refinement on input* G*, that is,* $\mathrm{WL}_1(G) = j$*, where* j *is the minimal integer for which* $\pi(\chi_G^j) = \pi(\chi_G^{j+1})$*. Similarly, for* $n \in \mathbb{N}$*, we write* $\mathrm{WL}_1(n)$ *to denote the maximum number of iterations that Colour Refinement needs to reach stabilisation on an* n*-vertex graph.*

We call every graph G with $\mathrm{WL}_1(G) = |G| - 1$ a *long-refinement graph.*

▶ **Fact 5.** *Let* G *be an uncoloured path with* n *vertices. Then* $\mathrm{WL}_1(G) = \lfloor \frac{n-1}{2} \rfloor$.

Proof sketch. In the first iteration, the two end vertices are distinguished from all others because they are the only ones with degree 1. Then in each iteration, the information of being adjacent to a "special" vertex, i.e. the information about the distance to a vertex of degree 1, is propagated one step closer to the vertices in the centre of the path. This procedure takes $\lfloor \frac{n-1}{2} \rfloor$ iterations. ◀

In 2015, Krebs and Verbitsky improved on the explicit linear lower bound for graphs of order n given by Fact 5 by constructing a family of pairs of graphs whose members of order n can only be distinguished after $n - 8\sqrt{n}$ Colour Refinement iterations (see [26, Theorem 4.6]). Hence, since for a set $\{v_1, \dots, v_n\} =: S$ and every sequence $\pi_1, \dots, \pi_\ell$ of partitions of S that satisfy

$$\pi_1 \gneqq \pi_2 \gneqq \dots \gneqq \{\{v_1\}, \dots \{v_n\}\} = \pi_\ell,$$

it holds that $\ell \leq n - 1$, we obtain the following corollary.

▶ **Corollary 6.** *For every* $n \in \mathbb{N}$*, it holds that* $n - 8\sqrt{n} \leq \mathrm{WL}_1(n) \leq n - 1$.

It has remained open whether any of the two bounds is tight. In preliminary research conducted together with Gödicke and Schweitzer, towards improving the lower bound, the first author took up an approach to reverse-engineer the splitting of colour classes. Gödicke's implementation of those split procedures led to the following result.

▶ **Theorem 7** ([11])**.** *For every* $n \in \{1, 10, 11, 12\}$*, it holds that* $\mathrm{WL}_1(n) = n - 1$*. For* $n \in [2, 9]$*, it holds that* $\mathrm{WL}_1(n) < n - 1$.

Unfortunately, due to computational exhaustion, it was not possible to test for larger graph sizes. Also, the obtained graphs do not exhibit any structural properties that would lend themselves for a generalisation to larger orders. Using a fast implementation of Colour Refinement, we could verify that there are exactly 16 long-refinement graphs of order 10, 24 long-refinement graphs of order 11, 32 of order 12, and 36 of order 13. However, again, with simple brute-force approaches, we could not go beyond this number exhaustively.

4 Compact Representations of Long-Refinement Graphs

In the light of the previous section, the question whether the lower bound obtained by Krebs and Verbitsky is asymptotically tight has remained open. With the brute-force approach, it becomes infeasible to test all graphs of orders much larger than 10 exhaustively for their number of Colour Refinement iterations until stabilisation. Still, knowing that there exist long-refinement graphs, it is natural to ask whether the ones presented in [11] are exceptions or whether there are infinitely many such graphs. We show that the latter is the case.

When the input is a coloured graph with at least two vertex colours, the initial partition already has two elements. Hence, all long-refinement graphs are monochromatic. Therefore, in the following, all initial input graphs are considered to be monochromatic.

▶ **Proposition 8.** *Let G be a graph and let $n := |G|$. If there exists an $i \in \mathbb{N}_0$ such that $|\{\chi_G^{i+1}(v) \mid v \in V(G)\}| - |\{\chi_G^{i}(v) \mid v \in V(G)\}| \geq 2$ holds, then G is not a long-refinement graph.*

Proof. Every pair of partitions π, π' with $\pi \supsetneq \pi'$ satisfies $|\pi'| \geq |\pi| + 1$. Thus, every sequence of partitions of the form

$$\pi_1 := \{\{v_1, \dots, v_n\}\} \supsetneq \pi_2 \supsetneq \dots \supsetneq \{\{v_1\}, \dots \{v_n\}\} =: \pi_n$$

must satisfy $|\pi_i| = |\pi_{i-1}| + 1$ for all $i \in [2, n]$. ◀

The proposition implies that, in order to find long-refinement graphs, we have to look for graphs in which, in every Colour Refinement iteration, only one additional colour class appears. That is, in each iteration, only one colour class is split and the splitting creates exactly two new colour classes.

▶ **Corollary 9.** *Let G be a long-refinement graph with at least two vertices. Then there exist $d_1, d_2 \in \mathbb{N}_0$ with $d_1 \neq d_2$ and such that $\deg(G) = \{d_1, d_2\}$.*

Proof. This is a direct consequence of Proposition 8: every (monochromatic) regular graph G satisfies $\mathrm{WL}_1(G) = 0$ and if there were more than two vertex degrees present in G, we would have $|\{\chi_G^1(v) \mid v \in V(G)\}| - |\{\chi_G^0(v) \mid v \in V(G)\}| \geq 3 - 1 \geq 2$. ◀

We can thus restrict ourselves to graphs with exactly two vertex degrees.

▶ **Notation 10.** *For a graph G and $i \in \mathbb{N}_0$, we let π_G^i denote the partition induced by χ_G^i on $V(G)$, i.e. after i Colour Refinement iterations on G. If G is clear from the context, we omit it in the expression.*

As a result of the regularity conditions that must hold for the bipartite graph $G[V_1, V_2]$, we make the following observation. It implies that, in a long-refinement graph, to determine the class C that is split in iteration i, it suffices to consider the neighbourhood of an arbitrary class obtained in the preceding iteration.

▶ **Lemma 11.** *Let G be a graph. Suppose there are $i \in \mathbb{N}$ and C_1, C_2, C' with $\pi^i \setminus \pi^{i-1} = \{C_1, C_2\}$ and $C' \in \pi^i \setminus \pi^{i+1}$. Then there are vertices $v_1', v_2' \in C'$ such that $|N(v_1') \cap C_1| \neq |N(v_2') \cap C_1|$.*

Proof. Note that there must be a $C \in \pi^{i-1} \setminus \pi^i$ with $C_1 \cup C_2 = C$. Since $C' \in \pi^i$ and $C \in \pi^{i-1}$, there is a $d \in \mathbb{N}_0$ such that for every $v \in C'$, it holds that $d = |N(v) \cap C|$. Since $\{C_1, C_2\} = \pi^i \setminus \pi^{i-1}$ and $C' \notin \pi^{i+1}$, there are vertices $v_1', v_2' \in C'$ such that $|N(v_1') \cap C_1| \neq |N(v_2') \cap C_1|$ or $|N(v_1') \cap C_2| \neq |N(v_2') \cap C_2|$. In the first case, we are done. In the second case, we obtain $|N(v_1') \cap C_1| = d - |N(v_1') \cap C_2| \neq d - |N(v_2') \cap C_2| = |N(v_2') \cap C_1|$. ◀

Note that the validity of the lemma depends on the assumption $\{C_1, C_2\} = \pi^i \setminus \pi^{i-1}$, which by Proposition 8 is always fulfilled in long-refinement graphs as long as $\pi^{i-1} \neq \pi^i$.

▶ **Corollary 12.** *No graph with more than one connected component is a long-refinement graph.*

Proof. By Corollary 9, the graph cannot have isolated vertices, since the subgraph formed by the other connected components would have to be d-regular for some $d \geq 1$. Therefore, the stable partition would not be discrete.

Suppose G is a long-refinement graph in which every connected component has size at least 2. Then there is an iteration i for which there is a unique connected component with vertex set V' such that the current partition is discrete on V' but not discrete on the other connected components. Then the vertex sets C_1, C_2 from Lemma 11 must be subsets of V'. However, the lemma implies that no further classes are split because none is adjacent to C_1 or C_2. ◀

We can therefore restrict ourselves to connected graphs. The only connected graphs G with $\deg(G) = \{1, 2\}$ are paths and, by Fact 5, they are not long-refinement graphs. Thus, the smallest degree pairs for a search for candidates are $\{1, 3\}$ and $\{2, 3\}$.

▶ **Lemma 13.** *Let G be a long-refinement graph. Then $|\{v \in V(G) \mid \deg(v) = 1\}| \leq 2$.*

Table 1 displays the adjacency lists of two long-refinement graphs on 12 and 14 vertices, respectively, which each have exactly one vertex of degree 1.

Table 1 Adjacency lists of long-refinement graphs G with $\deg(G) = \{1, 5\}$ (left) and $\deg(G) = \{1, 3\}$ (right), respectively.

v	$N(v)$
0	1
1	0,2,3,4,5
2	1,3,5,7,10
3	1,2,4,6,10
4	1,3,5,9,11
5	1,2,4,8,11
6	3,7,8,9,11
7	2,6,8,9,10
8	5,6,7,10,11
9	4,6,7,10,11
10	2,3,7,8,9
11	4,5,6,8,9

v	$N(v)$
0	1
1	0,2,3
2	1,11,13
3	1,10,12
4	5,7,10
5	4,6,10
6	5,9,11
7	4,8,11
8	7,9,13
9	6,8,12
10	3,4,5
11	2,6,7
12	3,9,13
13	2,8,12

The lemma allows us to reduce the decision problem whether there are infinitely many long-refinement graph with degrees in $\{1, 2, 3\}$ to the question whether there are such families with degrees in $\{2, 3\}$. The proof of the lemma as well as this reduction can be found in [21]. With the help of the tool `Nauty` [29], our quest for long-refinement graphs with degrees 2 and 3 was successful. Exploiting the degree restrictions and some other conditions that we imposed to render the search tractable, it was possible to test for graphs up to order 64. We found graphs G with $n - 1$ Colour Refinement iterations, where $n = |G|$, for all even $n \in [10, 64] \setminus \{24, 30, 42, 48, 60\}$ and for all odd $n \in [11, 63] \setminus \{21, 27, 39, 45, 57, 63\}$.

In the following, in order to generalise the results, we analyse the obtained graphs. Among our computational results, the even-order graphs G have the following property in common: there is an iteration j such that for every $c \in \chi_G^j(V(G))$, it holds that $|\{v \in V(G) \mid \chi_G^j(v) = c\}| = 2$. That is, with respect to their assigned colours, the vertices remain in pairs until there are no larger colour classes left. Then the first such pair is split into singletons, which must induce a splitting of another pair, and so on, until the discrete partition is obtained. (Similar statements hold for the odd-order long-refinement graphs, but are more technical.) In the following, a *pair* is a set of two vertices which occurs as a colour class during the execution of Colour Refinement. That is, vertices v, v' form a pair if and only if $\{v, v'\}$ is an element of π^i for some $i \in \mathbb{N}_0$.

As just argued, there is a splitting order on the pairs, i.e., a linear order $\prec$ induced by the order in which pairs are split into singletons. We now examine the possible connections between pairs.

From now on, we make the following assumption.

▶ **Assumption 14.** *G is a long-refinement graph with $\deg(G) = \{2, 3\}$ and such that there is an $i \in \mathbb{N}_0$ for which π^i contains only pairs. Let $\prec$ be the splitting order of these pairs.*

We call pairs $P_1, P_2 \subseteq V(G)$ *successive* if P_2 is the successor of P_1 with respect to $\prec$. Note that for successive pairs P_1, P_2, in the graph $G[P_1, P_2]$, every $v_2 \in P_2$ must have the same number of neighbours in P_1, otherwise it would hold that $P_2 \prec P_1$. By a simple case analysis, together with an application of Lemma 11, this rules out all connections but matchings for successive pairs.

▶ **Corollary 15.** *Let P_1 and P_2 be successive pairs. Then $G[P_1, P_2]$ is a matching.*

Towards a compact representation of the graphs, we further examine the connections between pairs P_1 and P_2 with $S(P_1) \prec P_2$, where $S(P_1)$ is the successor of P_1 with respect to $\prec$.

▶ **Lemma 16.** *Let P_1 be a pair. Then exactly one of the following holds.*

- *$P_1 \neq \min(\prec)$ and for every pair P_2 with $S(P_1) \prec P_2$, it holds that $E(G[P_1, P_2]) = \emptyset$.*
- *$P_1 = \min(\prec)$ and there are exactly two choices P_2, P_2' for a pair P' with $S(P_1) \prec P'$ such that $E(G[P_1, P']) \neq \emptyset$. Furthermore, there is a vertex $v_1 \in P_1$ such that $G[\{v_1\}, P_2]$ and $G[P_1 \setminus \{v_1\}, P_2']$ are complete bipartite and $E(G[\{v_1\}, P_2']) = E(G[P_1 \setminus \{v_1\}, P_2]) = \emptyset$.*

Proof. Suppose $P_1 \neq \min(\prec)$. If $P_1 = \max(\prec)$, the statement trivially holds. Otherwise, by Corollary 15, every vertex $v_1 \in P_1$ has exactly one neighbour in $S(P_1)$ and exactly one neighbour in the predecessor of P_1, i.e. in the unique pair $A(P_1)$ such that $P_1 = S(A(P_1))$. Thus, due to the degree restrictions, v_1 can have at most one additional neighbour in a pair P' with $P_1 \prec P'$ and $P' \neq S(P_1)$. However, if v_1 had a neighbour in such a P', the graph $G[\{v_1\}, P']$ would not be biregular, implying that $P' = S(P_1)$, a contradiction. Therefore, $N(v_1) \subseteq A(P_1) \cup P_1 \cup S(P_1)$ and thus, $N(P_1) \subseteq A(P_1) \cup S(P_1)$. In particular, for every pair P_2 with $P_1 \prec P_2$ and $P_2 \neq S(P_1)$, it holds that $E(G[P_1, P_2]) = \emptyset$.

Now suppose that $P_1 = \min(\prec)$. Since the splitting of P_1 must be induced by a splitting of a union of two pairs and $G[P_1, S(P_1)]$ is biregular and $G[P_1]$ is regular, we cannot have $N(P_1) \subseteq S(P_1)$. Thus, there is a pair P_2 with $S(P_1) \prec P_2$ and such that $E(G[P_1, P_2]) \neq \emptyset$. Let $v_1 \in P_1$ be a vertex with $N(v_1) \cap P_2 \neq \emptyset$. Then $P_2 \subseteq N(v_1)$, otherwise $P_2 = S(P_1)$. Thus, $G[\{v_1\}, P_2]$ is complete bipartite. Therefore and due to the degree restrictions, v_1 has exactly three neighbours: one in $S(P_1)$ and two in P_2. In particular, for every pair P_2' with $P_2 \neq P_2' \neq S(P_1)$, it holds that $E(G[\{v_1\}, P_2']) = \emptyset$.

Let $v_1' \neq v_1$ be the second vertex in P_1. Since the splitting of P_1 induces the splitting of $S(P_1)$, by Proposition 8, for every pair P' with $P_1 \neq P' \neq S(P_1)$, the graph $G[\{v_1'\}, P']$ must be biregular, i.e. either empty or complete bipartite.

Moreover, since $\deg(v_1) = 3$, also $\deg(v_1') = 3$. By Corollary 15, $|N(v_1') \cap S(P_1)| = 1$. Therefore, there is exactly one pair P_2' such that $G[\{v_1'\}, P_2']$ is complete bipartite and for all other pairs P' with $P_1 \neq P' \neq S(P_1)$, the graph $G[\{v_1'\}, P']$ is empty.

Suppose $P_2' = P_2$. Choose i such that $\pi^i \setminus \pi^{i+1} = \{P_1\}$. Then the unique element in $\pi^{i-1} \setminus \pi^i$ is a union of two pairs, whose splitting induces the splitting of P_1. However, $N(P_1) = S(P_1) \cup P_2$ and both graphs $G[P_1, S(P_1)]$ and $G[P_1, P_2]$ are biregular.

Thus, $P_2' \neq P_2$, which concludes the proof. ◀

Corollary 15 and Lemma 16 characterise $G[P_1, P_2]$ for all pairs $P_1 \neq P_2$. Thus, all additional edges must be between vertices from the same pair. Hence, we can use the following compact graphical representation to fully describe the graphs of order at least 12 that we found. As the set of nodes, we take the pairs. We order them according to $\prec$ and connect successive pairs with an edge representing the matching. If the two vertices of a pair are adjacent, we indicate this with a loop at the corresponding node. The only other type of connection between pairs is constituted by the edges from $\min(\prec)$ to two other pairs which form the last colour class of size 4, i.e. a colour class of size 4 in the partition π^i for which $\pi^{i+1} \setminus \pi^{i+2} = \{\min(\prec)\}$. We indicate this type of edge with a dotted curve.

An example graph as well as the evolution of the colour classes computed by Colour Refinement on the graph is depicted in Figure 1.

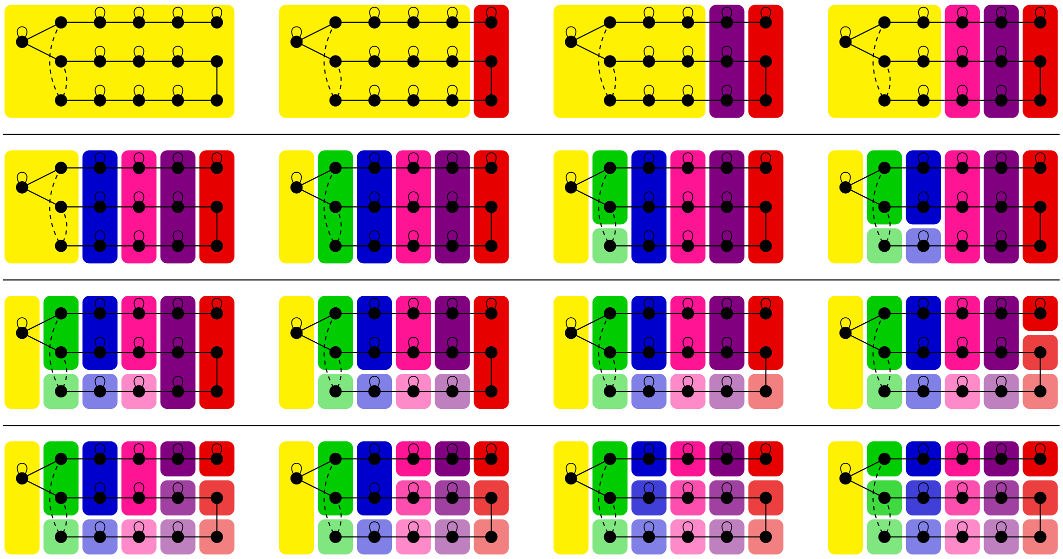


Figure 1 Top left: A long-refinement graph G on 32 vertices. The subsequent pictures show the partitions of $V(G)$ after the first 15 Colour Refinement iterations. There are 16 further iterations not depicted here, which consist in the splitting of the pairs into singletons.

In fact, there is an even more compact representation for our even-order long-refinement graphs.

▶ **Notation 17.** *Since $\prec$ is a linear order, we can also use a string representation to fully describe the graphs. For this, we introduce the following notation, letting $A(P)$ and $S(P)$ be the predecessor and successor of P, respectively, with respect to $\prec$.*

- *0 represents a pair of vertices of degree 2.*
- *1 represents a pair P of vertices of degree 3 that is not the minimum of $\prec$ and for which $N(P) \subseteq A(P) \cup S(P)$. (This implies that $P \in E(G)$.)*
- *X represents a pair P of vertices of degree 3 that is not the minimum of $\prec$ and for which $N(P) \nsubseteq A(P) \cup S(P)$.*
- *S represents the minimum of $\prec$.*

Thus, by Lemma 16, there are exactly two pairs of type X, namely P_2 and P_2' from the lemma. Now we can use the alphabet $\Sigma = \{0, 1, \mathrm{S}, \mathrm{X}\}$ and the order $\prec$ to encode the graphs in strings. The i-th letter of a string is the i-th element of $\prec$. Note that S is always a pair of non-adjacent vertices of degree 3 due to the degree restrictions. For example, the string representation for the graph in Figure 1 is S11100111X1X1110.

Formally, for every $\ell \geq 3$ and every string $\Xi\colon [\ell] \to \{0, 1, \mathrm{S}, \mathrm{X}\}$ with $\Xi(1) = \mathrm{S}$ and $\Xi^{-1}(\mathrm{X}) = \{r, r'\}$ for some $r, r' \in [\ell]$ with $r < r'$, we define the corresponding graph $G := G(\Xi)$ with $V(G) = \{v_{i,j} \mid i \in [\ell], j \in [2]\}$ and

$$\begin{aligned} E(G) = {} & \bigl\{\{v_{i,1}, v_{i,2}\} \bigm| i \in [\ell], \Xi(i) = 1\bigr\} \cup \\ & \bigl\{\{v_{i,j}, v_{i+1,j}\} \bigm| i \in [\ell - 1], j \in [2]\bigr\} \cup \\ & \bigl\{\{v_{r,j}, v_{1,1}\} \bigm| j \in [2]\bigr\} \cup \\ & \bigl\{\{v_{r',j}, v_{1,2}\} \bigm| j \in [2]\bigr\}. \end{aligned}$$

We use this encoding in the next section, which contains our main results.

5 Infinite Families of Long-Refinement Graphs

In this section, we present infinite families of long-refinement graphs. We adapt them further to deduce that $\mathrm{WL}_1(n) \geq n - 2$ holds for all $n \in \mathbb{N}_{\geq 10}$.

For $w \in \{0, 1\}^*$, the notation $(w)^k$ abbreviates the k-fold concatenation of w. We let $1^k := (1)^k$.

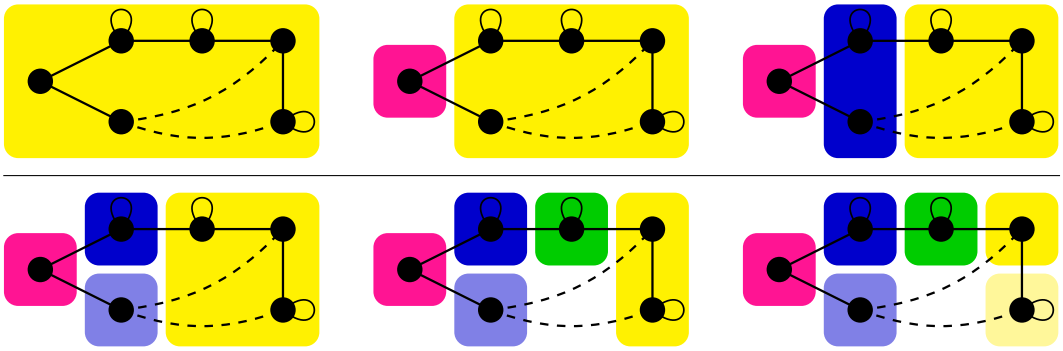


Figure 2 A visualisation of the graph with string representation S011XX and the evolution of the colour classes in the first 5 Colour Refinement iterations on the graph.

Theorem 18. *For every string Ξ contained in the following sets, the graph $G(\Xi)$ is a long-refinement graph.*

- $\{\mathrm{S011XX}\}$
- $\{\mathrm{S1}^k\mathrm{001}^k\mathrm{X1X1}^k\mathrm{0} \mid k \in \mathbb{N}_0\}$
- $\{\mathrm{S1}^k\mathrm{11001}^k\mathrm{XX1}^k\mathrm{0} \mid k \in \mathbb{N}_0\}$
- $\{\mathrm{S1}^k\mathrm{0011}^k\mathrm{XX1}^k\mathrm{10} \mid k \in \mathbb{N}_0\}$
- $\{\mathrm{S011(011)}^k\mathrm{00(110)}^k\mathrm{XX(011)}^k\mathrm{0} \mid k \in \mathbb{N}_0\}$
- $\{\mathrm{S(011)}^k\mathrm{00(110)}^k\mathrm{1X0X1(011)}^k\mathrm{0} \mid k \in \mathbb{N}_0\}$

We only present the parts of the proof that are most essential for an intuition why the graphs are indeed long-refinement graphs. The full proof can be found in [21].

Proof. Let $G := G(\mathrm{S011XX})$ (cf. Figure 2). The vertices $v_{2,1}$ and $v_{2,2}$ are the only ones of degree 2. Thus,

$$\begin{aligned} \pi^1 &= \bigl\{\{v_{2,1}, v_{2,2}\}, V(G) \setminus \{v_{2,1}, v_{2,2}\}\bigr\}, \\ \pi^2 &= \bigl\{\{v_{2,1}, v_{2,2}\}, \{v_{i,j} \mid i \in \{1, 3\}, j \in [2]\}, \{v_{i,j} \mid i \in [4, 6], j \in [2]\}\bigr\}. \end{aligned}$$

Then

$$\pi^3 = \bigl\{\{v_{1,1}, v_{1,2}\}, \{v_{2,1}, v_{2,2}\}, \{v_{3,1}, v_{3,2}\}, \{v_{i,j} \mid i \in [4,6], j \in [2]\}\bigr\},$$

since the vertices in the S-pair have no neighbours in $\{v_{i,j} \mid i \in \{1,3\}, j \in [2]\}$. Similarly,

$$\pi^4 = \bigl\{\{v_{1,1}, v_{1,2}\}, \{v_{2,1}, v_{2,2}\}, \{v_{3,1}, v_{3,2}\}, \{v_{4,1}, v_{4,2}\}, \{v_{i,j} \mid i \in [5,6], j \in [2]\}\bigr\},$$
$$\pi^5 = \bigl\{\{v_{i,j} \mid j \in [2]\} \mid i \in [6]\bigr\}.$$

Now the splitting of the last colour class of size 4 into two X-pairs induces the splitting of the S-pair into singletons, which is propagated linearly according to $\prec$, adding 6 further iterations, thus summing up to 11 iterations.

We now consider the various infinite families of graphs. The proofs for them work similarly by induction over k. Therefore, we only present the full detailed proof for the family $\{\mathrm{S}1^k001^k\mathrm{X}1\mathrm{X}1^k0 \mid k \in \mathbb{N}_0\}$, which includes the graph from Figure 1.

For $k = 0$, the graph $G_0 := G(\mathrm{S00X1X0})$ has 14 vertices. It is easy to verify that it indeed takes 13 Colour Refinement iterations to stabilise. We sketch how Colour Refinement processes the graph: for this, for $i \in \mathbb{N}_0$, we let π_0^i denote the partition of $V(G_0)$ induced by $\chi_{G_0}^i$, i.e. after i iterations of Colour Refinement on G_0. First, vertices are assigned colours indicating their degrees. That is,

$$\pi_0^1 = \bigl\{\{v_{i,j} \mid i \in \{2,3,7\}, j \in [2]\}, \{v_{i,j} \mid i \in \{1,4,5,6\}, j \in [2]\}\bigr\}.$$

Now

$$\pi_0^2 = \bigl\{\{v_{i,j} \mid i \in \{2,3,7\}, j \in [2]\}, \{v_{i,j} \mid i \in \{1,4,6\}, j \in [2]\}, \{v_{5,i} \mid i \in [2]\}\bigr\},$$

since the vertices contained in the 1-pair are not adjacent to vertices from 0-pairs. Since no vertex contained in the S-pair is adjacent to any vertex from the 1-pair, we obtain

$$\pi_0^3 = \bigl\{\{v_{i,j} \mid i \in \{2,3,7\}, j \in [2]\}, \{v_{i,j} \mid i \in \{4,6\}, j \in [2]\}, \{v_{5,i} \mid j \in [2]\},$$
$$\{v_{1,j} \mid j \in [2]\}\bigr\}.$$

Furthermore,

$$\pi_0^4 = \bigl\{\{v_{i,j} \mid i \in \{3,7\}, j \in [2]\}, \{v_{i,j} \mid i \in \{4,6\}, j \in [2]\}, \{v_{5,i} \mid j \in [2]\},$$
$$\{v_{1,j} \mid j \in [2]\}, \{v_{2,j} \mid j \in [2]\}\bigr\},$$
$$\pi_0^5 = \bigl\{\{v_{7,j} \mid j \in [2]\}, \{v_{i,j} \mid i \in \{4,6\}, j \in [2]\}, \{v_{5,i} \mid j \in [2]\},$$
$$\{v_{1,j} \mid j \in [2]\}, \{v_{2,j} \mid j \in [2]\}, \{v_{3,j} \mid j \in [2]\}\bigr\},$$
$$\pi_0^6 = \bigl\{\{v_{i,j} \mid j \in [2]\} \mid i \in [7]\bigr\},$$

i.e. with respect to the order $\prec$ induced by the string representation, the first 0-pair, the second 0-pair, and the first X-pair are separated from the others. Once the two X-pairs form separate colour classes, this induces the splitting of S into two singletons, which is propagated linearly through the entire string, adding 7 further iterations, thus summing up to 13 iterations.

For general $k \geq 1$, let $G_k := G(\mathrm{S}1^k001^k\mathrm{X}1\mathrm{X}1^k0)$. To count the iterations of Colour Refinement, we introduce some vocabulary for the pairs in G_k (see also Figure 1). We let $V := \{v_{i,j} \mid i \in [2, k{+}1] \cup [k{+}4, 2k{+}3] \cup [2k{+}7, 3k{+}6], j \in [2]\}$. Note that V is the set of vertices contained in the subgraphs corresponding to the substrings 1^k in the string representation.

Furthermore, for all $i \in [k+2]$, we call the set $\{v_{i',j} \mid i' \in \{i, 2k+5-i, 2k+5+i\}, j \in [2]\}$ the *i-th column* and denote it by V_i. The *0-th column* is the set $\{v_{2k+5,j} \mid j \in [2]\}$. Thus,

$$V = \bigcup_{2 \le i \le k+1} V_i.$$

For every $j \in [2]$, the sets $\{v_{i,j} \mid i \in [1, k+2]\}$, $\{v_{i,j} \mid i \in [k+3, 2k+4]\}$, and $\{v_{i,j} \mid i \in [2k+6, 3k+7]\}$ are called *rows*. In accordance with Figure 1, we fix an ordering on the rows: the *first row* is $\{v_{i,1} \mid i \in [2k+6, 3k+7]\}$, the *second row* is $\{v_{i,2} \mid i \in [2k+6, 3k+7]\}$, $\dots$, the *sixth row* is $\{v_{i,2} \mid i \in [1, k+2]\}$. To be able to refer to the vertices in V and the adjacent columns more easily, we relabel them: for $i \in [k+2], j \in [6]$, the vertex $w_{i,j}$ is defined to be the unique vertex in the i-th column and the j-th row.

The following observation is the crucial insight for counting the iterations of Colour Refinement on G_k. We will use it to show that, informally stated, the subgraph $G_k[V]$ delays the propagation of the splitting of the colour classes in the remainder of the graph by k iterations whenever the splitting of a colour class contained in V_1 or V_{k+2} initiates a splitting of a colour class contained in V.

▷ Claim 19. Consider a colouring λ of G_k and its induced partition π_k of $V(G_k)$. For $t \in \mathbb{N}_0$, let π_k^t be the partition induced by $\chi_{G_k}^t$ on input (G_k, λ). Suppose G_k, λ, π_k satisfy the following conditions.

1. There exist $\ell \in [6]$ and $I_1, \dots, I_\ell \subseteq [6]$ such that $\bigcup_{i \in [\ell]} I_i = [6]$ and $I_i \cap I_j = \emptyset$ for $1 \le i < j \le \ell$ and for every $i \in [\ell]$, it holds that $\{w_{k+2,j'} \mid j' \in I_i\} \in \pi_k^0$. That is, V_{k+2} is a union of colour classes with respect to λ.
2. $\pi_k^1 = \big\{\{w_{k+1,j'} \mid j' \in I_i\} \,\big|\, i \in [\ell]\big\} \cup \big\{C \setminus V_{k+1} \,\big|\, C \in \pi_k, C \setminus V_{k+1} \neq \emptyset\big\}$.
3. For all $C, C' \subseteq V_{k+1}$ with $C, C' \in \pi_k^1$, the graph $G_k[C]$ is regular and $G_k[C, C']$ is biregular.

Then for every $t \in [k]$, it holds that

$$\pi_k^t = \bigcup_{i' \in [t]} \big\{\{w_{k+2-i',j'} \mid j' \in I_i\} \,\big|\, i \in [\ell]\big\} \cup \left\{C \setminus \Big(\bigcup_{i' \in [t]} V_{k+2-i'}\Big) \,\middle|\, C \in \pi_k^0, C \setminus \Big(\bigcup_{i' \in [t]} V_{k+2-i'}\Big) \neq \emptyset\right\}.$$

The proof of the claim follows by induction on t. For $t = 1$, the statement is exactly the second item from the assumptions. The induction step is quite technical and we defer the reader to the full version for the details [21]. Informally speaking, the idea is to show that, within one iteration of Colour Refinement, if there is a a column V_i with $i \in [2, k+1]$ whose partition into row indices is strictly coarser than the partition of the column V_{i+1}, then V_i copies the partition from V_{i+1} and all other colour classes remain intact.

We call the property described in the claim *path propagation from right to left.* Modifying the indices, a similar statement yields *path propagation from left to right*, as formulated in the following claim.

▷ Claim 20. Consider a colouring λ of G_k and its induced partition π_k of $V(G_k)$. For $t \in \mathbb{N}_0$, let π_k^t be the partition induced by $\chi_{G_k}^t$ on input (G_k, λ). Suppose G_k, λ, π_k satisfy the following conditions.

1. There exist $\ell \in [6]$ and $I_1, \dots, I_\ell \subseteq [6]$ such that $\bigcup_{i \in [\ell]} I_i = [6]$ and $I_i \cap I_j = \emptyset$ for $1 \le i < j \le \ell$ and for every $i \in [\ell]$, it holds that $\{w_{1,j'} \mid j' \in I_i\} \in \pi_k$. That is, the first column is a union of colour classes with respect to λ.

2. $\pi_k^1 = \{\{w_{2,j'} \mid j' \in I_i\} \,|\, i \in [\ell]\} \cup \{C \setminus V_2 \,|\, C \in \pi_k, C \setminus V_2 \neq \emptyset\}$.
3. For all $C, C' \subseteq V_2$ with $C, C' \in \pi_k^1$, the graph $G_k[C]$ is regular and $G_k[C, C']$ is biregular.

Then for every $t \in [k]$, it holds that

$$\pi_k^t = \bigcup_{i' \in [t]} \{\{w_{i'+1,j'} \mid j' \in I_i\} \,|\, i \in [\ell]\} \cup \left\{C \setminus \Big(\bigcup_{i' \in [t]} V_{i'+1}\Big) \,\middle|\, C \in \pi_k^0, C \setminus \Big(\bigcup_{i' \in [t]} V_{i'+1}\Big) \neq \emptyset\right\}.$$

Again, we defer the reader to the full version for the proof details for the path propagation. We are now ready to analyse the run of Colour Refinement on input G_k. Recall that π_0^t denotes the partition induced by $\chi_{G_0}^t$ on $V(G_0) \subseteq V(G_k)$. For the following arguments, see also Figure 1.

In π_k^1, the vertices are distinguished according to their degrees. We can then use path propagation from right to left to deduce that

$$\pi_k^{k+1} = \pi_0^1 \cup \{V_i \,|\, i \in [2, k+1]\} \quad \text{and thus} \quad \pi_k^{k+2} = \pi_0^2 \cup \{V_i \,|\, i \in [2, k+1]\},$$
$$\pi_k^{k+3} = \pi_0^3 \cup \{V_i \,|\, i \in [2, k+1]\} \quad \text{and also} \quad \pi_k^{k+4} = \pi_0^4 \cup \{V_i \,|\, i \in [2, k+1]\}.$$

Now path propagation from left to right yields that

$$\pi_k^{2k+4} = \pi_0^4 \cup \{\{w_{i,j} \mid j \in [4]\} \,|\, i \in [2, k+1]\} \cup \{\{w_{i,j} \mid j \in \{5,6\}\} \,|\, i \in [2, k+1]\}$$

and $\pi_k^{2k+5} = \pi_0^5 \cup (\pi_k^{2k+4} \setminus \pi_0^4)$. With three more combinations of path propagation from right to left and path propagation from left to right, we obtain

$$\pi_k^{6k+13} = \pi_0^{13} \cup \{\{w_{i,j}\} \,|\, i \in [2, k+1], j \in [6]\},$$

which is the discrete partition by the induction assumption for $k = 0$.

This implies that on input G_k, Colour Refinement takes $6k + 13$ iterations to stabilise and, since $|G_k| = 6k + 14$, it holds that $\mathrm{WL}_1(G_k) = n - 1$, where $n = |G_k|$. ◀

▶ **Corollary 21.** *There are infinitely many $n \in \mathbb{N}$ with $\mathrm{WL}_1(n) = n - 1$.*

▶ **Corollary 22.** *For every even $n \in \mathbb{N}_{\geq 12}$ such that $n = 12$ or $n \bmod 18 \notin \{6, 12\}$, there is a long-refinement graph G with $|G| = n$. The graph G can be chosen to satisfy $\deg(G) = \{2, 3\}$.*

Proof. The string representation S011XX covers $n = 12$. The first infinite family from Theorem 18 covers all even $n \in \mathbb{N}_{\geq 14}$ with $n = 2 \mod 6$, i.e. with $n \bmod 18 \in \{2, 8, 14\}$. The second and the third infinite family both cover all even $n \in \mathbb{N}_{\geq 16}$ with $n = 4 \mod 6$, i.e. with $n \bmod 18 \in \{4, 10, 16\}$. The fourth and the fifth infinite family cover all even $n \in \mathbb{N}_{\geq 18}$ with $n = 0 \mod 18$. Thus, among the even graph orders larger than 12, only the ones with $n \bmod 18 \in \{6, 12\}$ remain not covered. ◀

We now turn to the long-refinement graphs G of odd order with vertex degrees in $\{2, 3\}$. If the graph has odd order, we cannot represent it just with pairs. For this, we relax Assumption 14 as follows.

▶ **Assumption 23.** *G is a long-refinement graph with $\deg(G) = \{2, 3\}$ and such that there is an $i \in \mathbb{N}_0$ for which π^i contains only pairs and at most one singleton.*

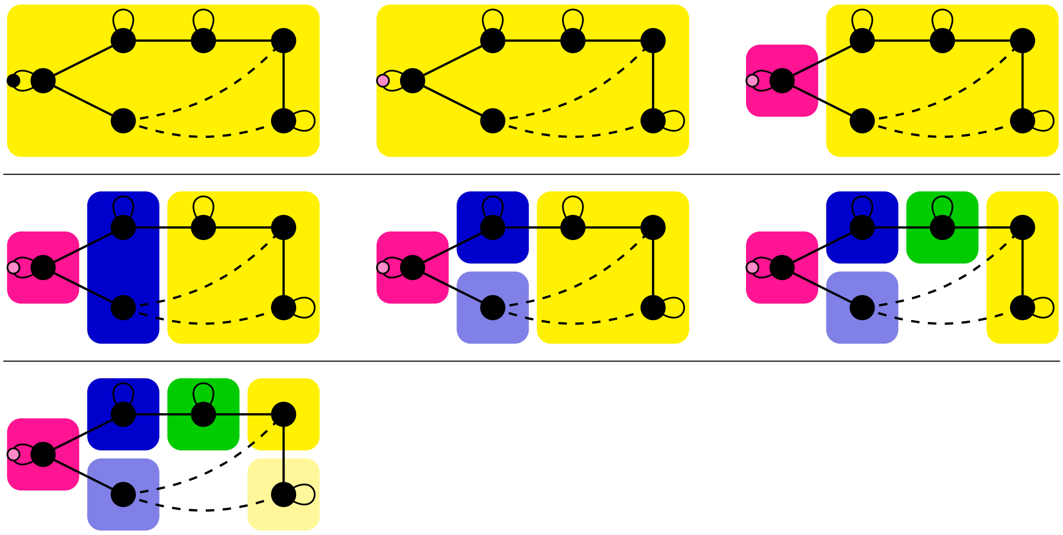


Figure 3 A visualisation of the graph with string representation S$\hat{1}$11XX and the evolution of the colour classes in the first 6 Colour Refinement iterations on the graph.

We maintain the vocabulary and notation from the long-refinement graphs of even order, i.e. 0, 1, S, X will be used in the same way as before. However, in order to fully describe the odd-order graphs via strings, we have to extend the string alphabet by fresh letters $\hat{1}$ and $\hat{\mathrm{X}}$, which represent particular pairs with attached vertices as follows. For a string $\hat{\Xi}\colon [\ell] \to \{0, 1, \mathrm{S}, \mathrm{X}, \hat{1}, \hat{\mathrm{X}}\}$, we define the *base string* Ξ as the string obtained by removing hats. More precisely, we replace every $\hat{\mathrm{X}}$ with an X, and we replace every $\hat{1}$ with a 1 if it is non-terminal (i.e. if it is not at position ℓ) and with a 0 otherwise. Let $I(\hat{\Xi}) \subseteq [\ell]$ be the set of positions i with $\hat{\Xi}(i) \in \{\hat{1}, \hat{X}\}$. If, in the base graph $G(\Xi)$, every vertex pair corresponding to a position in $I(\hat{\Xi})$ (a *hat vertex pair*) is adjacent, we call $\hat{\Xi}$ a *hat string*.

Similarly as for the even-order long-refinement graphs, to every hat string $\hat{\Xi}$, we assign a graph $G(\hat{\Xi})$. We obtain the graph $G(\hat{\Xi})$ by subdividing in $G(\Xi)$ each edge connecting a hat vertex pair with a new fresh vertex, which we call a *hat*. Additionally, if the vertices of the hat vertex pair have degree 2, we now insert another edge between them. (This can only happen if the hat vertex pair corresponds to a terminal 0.) For a hat $\hat{v}$, we call the neighbourhood $N(\hat{v}) \subseteq V\big(G(\hat{\Xi})\big)$ the *hat base* of $\hat{v}$. Note that every vertex in the hat base has degree 3 by construction. Also, a hat always has degree 2 and thus, with respect to $\chi^1_{G(\hat{\Xi})}$, it has a different colour than its hat base.

Graphically, we represent a hat by a loop attached to the corresponding hat vertex pair, which we subdivide by inserting a small vertex (see Figure 3). It is not difficult to see that every graph $G(\hat{\Xi})$ corresponding to a hat string $\hat{\Xi}$ has exactly one hat and that the hat is the first vertex forming a singleton colour class during the execution of Colour Refinement on $G(\hat{\Xi})$. Thus, $|G(\hat{\Xi})| = |G(\Xi)| + 1$.

▶ **Theorem 24.** *For every string $\hat{\Xi}$ contained in the following sets, the graph $G(\hat{\Xi})$ is a long-refinement graph.*

- $\{\mathrm{S}\hat{1}11\mathrm{XX}\}$
- $\{\mathrm{S0X1}\hat{\mathrm{X}}\} \cup \{\mathrm{S1}^k\mathrm{1011}^k\mathrm{X1X1}^k\hat{1} \mid k \in \mathbb{N}_0\}$
- $\{\mathrm{S110X}\hat{\mathrm{X}}\} \cup \{\mathrm{S111}^k\mathrm{1011}^k\mathrm{XX1}^k\hat{1} \mid k \in \mathbb{N}_0\}$
- $\{\mathrm{S1}^k\mathrm{01}^k\mathrm{1XX1}^k\hat{1} \mid k \in \mathbb{N}_0\}$
- $\{\mathrm{S(011)}^k\mathrm{00(110)}^k\mathrm{X}\hat{1}\mathrm{X(011)}^k\mathrm{0} \mid k \in \mathbb{N}_0\}$

Proof sketch. The proof techniques for the infinite families are very similar to the ones presented for Theorem 18. Therefore, we only sketch the proof on two concrete examples containing $\hat{1}$ and $\hat{\mathrm{X}}$, respectively. To be able to refer to vertices explicitly, recall the formal definition of $G := G(\Xi)$ from Section 4. Since for a hat string $\hat{\Xi}$, it holds that $|G(\hat{\Xi})| = |G(\Xi)| + 1$, we can use the same indexing of vertices, additionally letting $\hat{v}$ be the unique hat in $G(\hat{\Xi})$.

In the graph $G := G(\mathrm{S}\hat{1}11\mathrm{XX})$ (cf. Figure 3), the only vertex of degree 2 is $\hat{v}$. Thus, in π^1, it forms a singleton colour class. In π^2, the hat base $\{v_{2,1}, v_{2,2}\}$ forms a new colour class.

In general, for $i \in \mathbb{N}$, we have that $\pi^i_G = \pi^{i-1}_{G'} \cup \{\hat{v}\}$, where $G' = G(\mathrm{S}011\mathrm{XX})$ (cf. Theorem 18 and Figure 2).

Thus,

$$\begin{aligned}
\pi^1 &= \big\{\{\hat{v}\}, V(G) \setminus \{\hat{v}\}\big\},\\
\pi^2 &= \big\{\{\hat{v}\}, \{v_{2,1}, v_{2,2}\}, V(G) \setminus \{\hat{v}, v_{2,1}, v_{2,2}\}\big\},\\
\pi^3 &= \big\{\{\hat{v}\}, \{v_{2,1}, v_{2,2}\}, \{v_{i,j} \mid i \in \{1,3\}, j \in [2]\}, \{v_{i,j} \mid i \in [4,6], j \in [2]\}\big\},\\
\pi^4 &= \big\{\{\hat{v}\}, \{v_{1,1}, v_{1,2}\}, \{v_{2,1}, v_{2,2}\}, \{v_{3,1}, v_{3,2}\}, \{v_{i,j} \mid i \in [4,6], j \in [2]\}\big\},\\
\pi^5 &= \big\{\{\hat{v}\}, \{v_{1,1}, v_{1,2}\}, \{v_{2,1}, v_{2,2}\}, \{v_{3,1}, v_{3,2}\}, \{v_{4,1}, v_{4,2}\}, \{v_{i,j} \mid i \in [5,6], j \in [2]\}\big\},\\
\pi^6 &= \big\{\{\hat{v}\}\big\} \cup \big\{\{v_{i,j} \mid j \in [2]\} \mid i \in [6]\big\}.
\end{aligned}$$

Now in 6 further iterations, the splitting of the pairs is propagated linearly according to the order $\prec$.

Next, let G be the graph $G(\mathrm{S}0\mathrm{X}1\hat{\mathrm{X}})$, i.e. a member of the first infinite family from Theorem 24. It has three vertices of degree 2, namely $\hat{v}$, $v_{2,1}$, and $v_{2,2}$, which therefore form a colour class in π^1. Also, the vertices contained in the 1-pair are the only vertices of degree 3 that are not adjacent to any vertex of degree 2. Thus,

$$\pi^2 = \big\{\{\hat{v}, v_{2,1}, v_{2,2}\}, \{v_{4,1}, v_{4,2}\}, V(G) \setminus \{\hat{v}, v_{2,1}, v_{2,2}, v_{4,1}, v_{4,2}\}\big\},$$

and similarly,

$$\pi^3 = \big\{\{\hat{v}, v_{2,1}, v_{2,2}\}, \{v_{4,1}, v_{4,2}\}, \{v_{1,1}, v_{1,2}\}, \{v_{i,j} \mid i \in \{3,5\}, j \in [2]\}\big\}.$$

Now the hat forms a singleton since, in contrast to the vertices of the 0-pair, it is not adjacent to any vertex in the S-pair. We obtain:

$$\begin{aligned}
\pi^4 &= \big\{\{\hat{v}\}, \{v_{2,1}, v_{2,2}\}, \{v_{4,1}, v_{4,2}\}, \{v_{1,1}, v_{1,2}\}, \{v_{i,j} \mid i \in \{3,5\}, j \in [2]\}\big\},\\
\pi^5 &= \big\{\{\hat{v}\}\big\} \cup \big\{\{v_{i,j} \mid j \in [2]\} \mid i \in [5]\big\}.
\end{aligned}$$

Then in 5 further iterations, the splitting of the pairs is propagated linearly according to the order $\prec$. ◀

As an example, Figure 3 shows the evolution of the colour classes of the graph with string representation $\mathrm{S}\hat{1}11\mathrm{XX}$.

▶ **Corollary 25.** *For every odd $n \in \mathbb{N}_{\geq 11}$ with $n \bmod 18 \notin \{3, 9\}$, there is a long-refinement graph G with $|G| = n$. The graph G can be chosen to satisfy* $\deg(G) = \{2, 3\}$.

Proof. The string representation $\mathrm{S}\hat{1}11\mathrm{XX}$ covers $n = 13$. The first infinite family covers all odd $n \in \mathbb{N}_{\geq 11}$ with $n = 5 \mod 6$, i.e. with $n \bmod 18 \in \{5, 11, 17\}$. The second and the third infinite family both cover all all odd $n \in \mathbb{N}_{\geq 13}$ with $n = 1 \mod 6$, i.e. with $n \bmod 18 \in \{1, 7, 13\}$. The fourth infinite family covers all odd $n \in \mathbb{N}_{\geq 15}$ with $n = 15 \mod 18$. Thus, among the odd orders larger than 10, only the ones with $n \bmod 18 \in \{3, 9\}$ are skipped. ◀

We summarise the results from Corollaries 22 and 25.

▶ **Corollary 26.** *For every $n \in \mathbb{N}_{\geq 11}$ such that $n = 12$ or $n \bmod 18 \notin \{3, 6, 9, 12\}$, there is a long-refinement graph G with $|G| = n$. The graph G can be chosen to satisfy* $\deg(G) = \{2, 3\}$.

The following lemma allows to cover more graph sizes.

▶ **Lemma 27.** *Let $n \in \mathbb{N}$ be arbitrary. Suppose there is a long-refinement graph G such that $|G| = n$. If there is a $d \in \mathbb{N}$ with* $\deg(G) = \{d, d+1\}$ *such that* $|\{v \in V(G) \mid \deg(v) = d\}| \neq d + 1$, *then there is also a long-refinement graph G' with $|G'| = n + 1$.*

Proof. We can insert an isolated vertex w into G and insert edges from w to every vertex $v \in V(G)$ with $\deg(v) = d$. In the new graph G', the vertex w has a degree other than $d+1$, whereas all other vertices have degree $d+1$. Thus, the colour classes in $\pi^1_{G'}$ are $\{w\}$ and $V(G') \setminus \{w\}$. After the second iteration, the neighbours of w are distinguished from all other vertices, just like they are in π^1_G. Inductively, it is easy to see that for $i \in \mathbb{N}$, it holds that $\pi^i_{G'} = \pi^{i-1}_G \cup \big\{\{w\}\big\}$. Thus, Colour Refinement takes $n - 1 + 1 = n$ iterations to stabilise on G'. ◀

▶ **Corollary 28.** *For every odd $n \in \mathbb{N}_{\geq 11}$, there is a long-refinement graph G with $|G| = n$.*

Proof. By Corollary 25, it suffices to provide long-refinement graphs of order n for every odd $n \in \mathbb{N}_{\geq 11}$ with $n \bmod 18 \in \{3, 9\}$. We will accomplish this by applying Lemma 27 to suitable graphs of orders n' with $n' \bmod 18 \in \{2, 8\}$. Every graph G with a string representation contained in one of the infinite families from Theorem 18 has an even number of vertices of degree 2. In particular, it satisfies $|\{v \in V(G) \mid \deg(v) = 2\}| \neq 3$. Furthermore, every even graph order larger than 10 not covered by Corollary 22 is a multiple of 6. Hence, since $N := \{n \in \mathbb{N}_{\geq 18} \mid n \bmod 18 \in \{2, 8\}\}$ contains only even numbers and no multiples of 6, for every $n' \in N$, there is a graph of order n' that satisfies the prerequisites of Lemma 27 with $d = 2$. (Actually, we can cover all of these graph orders with the family $\{\mathrm{S}1^k 001^k \mathrm{X}1\mathrm{X}1^k 0 \mid k \in \mathbb{N}_0\}$.)

Thus, applying the lemma, we can construct for every $n \in \mathbb{N}_{\geq 18}$ with $n \bmod 18 \in \{3, 9\}$ a long-refinement graph G' of order n. ◀

Note that, since we apply Lemma 27 to close the gaps, we cannot guarantee anymore that the vertex degrees are 2 and 3, as we could in Corollary 26.

We are ready to prove Theorem 1.

Proof of Theorem 1. The theorem follows from combining Corollaries 22 and 28 with Theorem 7. ◀

Although the theorem leaves some gaps, we can show that, starting from $n = 10$, the worst-case number of Colour Refinement iterations until stabilisation on graphs of order n is always either $n - 2$ or $n - 1$.

Proof of Theorem 2. By Theorem 1, we only need to consider the numbers $n \geq 24$ for which $n \bmod 18 \in \{6, 12\}$. Since these numbers are all even, for every such n, we can use Corollary 28 to obtain a long-refinement graph G' with $|G'| = n - 1$. Let v be a fresh vertex not contained in $V(G')$. Now define $G := (V, E)$ with $V := V(G') \cup \{v\}$ and $E := E(G')$. Then $\pi^i_G = \pi^i_{G'} \cup \{\{v\}\}$ for every $i \in \mathbb{N}$. In particular, $\mathrm{WL}_1(G) = \mathrm{WL}_1(G') = n - 2$. ◀

6 Conclusion

With Theorem 2, it holds for all $n \in \mathbb{N}_{\geq 10}$ that $\mathrm{WL}_1(n) \geq n-2$. In particular, this proves that the trivial upper bound $\mathrm{WL}_1(n) = n-1$ is tight, up to an additive constant of 1.

For infinitely many graph orders, the graph G can even be chosen to have vertex degrees 2 and 3, as Theorems 18 and 24 show. We applied Lemma 27 to cover some of the remaining sizes. However, no order $n \in \mathbb{N}_{\geq 18}$ with $n \bmod 18 \in \{6, 12\}$ is covered by Theorem 18. Also, for $|G| \in \{n \in \mathbb{N}_{\geq 18} \mid n \bmod 18 \in \{5, 11\}\}$, all the long-refinement graphs G we have found satisfy $|\{v \in V(G) \mid \deg(v) = 2\}| = 3$ (see the first infinite family in Theorem 24). Thus, these graphs do not satisfy the prerequisites of Lemma 27. Note that we cannot apply the construction from the proof if $|\{v \in V(G) \mid \deg(v) = d\}| = d+1$, since the new graph G' would be $(d+1)$-regular and would thus satisfy $\mathrm{WL}_1(G') = 0$. Hence, it is not clear how to apply our techniques to construct a long-refinement graph of order 24. Altogether, the values $n \in \mathbb{N}_{\geq 24}$ with $n \bmod 18 \in \{6, 12\}$ are precisely the graph orders for which it remains open whether there is a graph G with $\mathrm{WL}_1(G) = |G| - 1$.

A related question is for which values $d_1 \neq d_2$ there are long-refinement graphs G with $\deg(G) = \{d_1, d_2\}$. It would be nice to know whether we have actually found all long-refinement graphs G with $\deg(G) = \{2, 3\}$. Similarly, we can ask for long-refinement graphs when fixing other parameters. For example, since all long-refinement graphs that we found have girth at most 4, it would be interesting to know whether there exists any long-refinement graph with larger girth, or infinite families with unbounded girth.

Also, in the light of the graph isomorphism problem, it is a natural follow-up task to find for each order n pairs of non-isomorphic graphs G, H for which it takes $n-1$ Colour Refinement iterations to distinguish the graphs from each other. A first step towards this goal is the search for pairs of long-refinement graphs of equal order. It is easy to see that for infinitely many n, Theorems 18 and 24 yield such pairs of graphs. Still, for example, when evaluating the colourings computed by Colour Refinement on the graphs with string representations S1100XX0 and S001XX10, they differ after less than $n-1$ iterations. To see this, observe that in G(S1100XX0), all vertices of degree 2 have paths of length 3 to a vertex of degree 2 whose inner vertices only have degrees other than 2. This is not the case for G(S001XX10) and this property is detected by Colour Refinement after at most 4 iterations. Thus, finding for $n \in \mathbb{N}_{\geq 10}$ two graphs of order n which Colour Refinement only distinguishes after $n-1$ iterations remains a challenge.

References

1 Vikraman Arvind, Frank Fuhlbrück, Johannes Köbler, and Oleg Verbitsky. On Weisfeiler-Leman invariance: Subgraph counts and related graph properties. In *Fundamentals of Computation Theory – 22nd International Symposium, FCT 2019, Copenhagen, Denmark, August 12–14, 2019, Proceedings*, pages 111–125, 2019. doi:10.1007/978-3-030-25027-0_8.

2 Vikraman Arvind, Johannes Köbler, Gaurav Rattan, and Oleg Verbitsky. Graph isomorphism, color refinement, and compactness. *Computational Complexity*, 26(3):627–685, 2017. doi:10.1007/s00037-016-0147-6.

3 László Babai, Paul Erdős, and Stanley M. Selkow. Random graph isomorphism. *SIAM Journal on Computing*, 9(3):628–635, 1980. doi:10.1137/0209047.

4 Christoph Berkholz and Jakob Nordström. Near-optimal lower bounds on quantifier depth and Weisfeiler-Leman refinement steps. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 267–276, 2016. doi:10.1145/2933575.2934560.

5 Jin-Yi Cai, Martin Fürer, and Neil Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12(4):389–410, 1992.

6 Alain Cardon and Maxime Crochemore. Partitioning a graph in $O(|A| \log |A|)$. *Theoretical Computer Science*, 19:85–98, 1982. doi:10.1016/0304-3975(82)90016-0.

7 Paul T. Darga, Mark H. Liffiton, Karem A. Sakallah, and Igor L. Markov. Exploiting structure in symmetry detection for CNF. In *Proceedings of the 41th Design Automation Conference, DAC 2004, San Diego, CA, USA, June 7-11, 2004*, pages 530–534. ACM, 2004. doi:10.1145/996566.996712.

8 Frank Fuhlbrück, Johannes Köbler, and Oleg Verbitsky. Local WL invariance and hidden shades of regularity. *CoRR*, abs/2002.04590, 2020. arXiv:2002.04590.

9 Martin Fürer. Weisfeiler-Lehman refinement requires at least a linear number of iterations. In *Automata, Languages and Programming, 28th International Colloquium, ICALP 2001, Crete, Greece, July 8-12, 2001, Proceedings*, volume 2076 of *Lecture Notes in Computer Science*, pages 322–333. Springer, 2001. doi:10.1007/3-540-48224-5_27.

10 Martin Fürer. On the combinatorial power of the Weisfeiler-Lehman algorithm. In Dimitris Fotakis, Aris Pagourtzis, and Vangelis Th. Paschos, editors, *Proceedings of the Tenth International Conference on Algorithms and Complexity*, volume 10236 of *Lecture Notes in Computer Science*, pages 260–271. Springer Verlag, 2017. doi:10.1007/978-3-319-57586-5_22.

11 Maximilian Gödicke. The iteration number of the Weisfeiler-Lehman-algorithm. Master's thesis, RWTH Aachen University, 2015.

12 Christopher D. Godsil. Compact graphs and equitable partitions. *Linear Algebra Appl.*, 255:259–266, 1997. doi:10.1016/S0024-3795(97)83595-1.

13 Martin Grohe. Fixed-point definability and polynomial time on graphs with excluded minors. *J. ACM*, 59(5):27, 2012. doi:10.1145/2371656.2371662.

14 Martin Grohe, Kristian Kersting, Martin Mladenov, and Erkal Selman. Dimension reduction via colour refinement. In *Algorithms - ESA 2014 - 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*, volume 8737 of *Lecture Notes in Computer Science*, pages 505–516. Springer, 2014. doi:10.1007/978-3-662-44777-2_42.

15 Martin Grohe and Sandra Kiefer. A linear upper bound on the Weisfeiler-Leman dimension of graphs of bounded genus. In *Proceedings of the Forty-Sixth International Colloquium on Automata, Languages, and Programming*, pages 117:1–117:15, July 2019. doi:10.4230/LIPIcs.ICALP.2019.117.

16 Martin Grohe and Daniel Neuen. Canonisation and definability for graphs of bounded rank width. In *Proceedings of the Thirty-Fourth Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 1–13, June 2019. doi:10.1109/LICS.2019.8785682.

17 Martin Grohe and Oleg Verbitsky. Testing graph isomorphism in parallel by playing a game. In *Automata, Languages and Programming, 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10–14, 2006, Proceedings, Part I*, pages 3–14, 2006. doi:10.1007/11786986_2.

18 Neil Immerman and Eric Lander. Describing graphs: A first-order approach to graph canonization. In Alan L. Selman, editor, *Complexity theory retrospective*, pages 59–81. Springer, 1990. doi:10.1007/978-1-4612-4478-3_5.

19 Tommi A. Junttila and Petteri Kaski. Engineering an efficient canonical labeling tool for large and sparse graphs. In *Proceedings of the Nine Workshop on Algorithm Engineering and Experiments, ALENEX 2007, New Orleans, Louisiana, USA, January 6, 2007.* SIAM, 2007. doi:10.1137/1.9781611972870.13.

20 Sandra Kiefer. *Power and Limits of the Weisfeiler-Leman Algorithm.* PhD thesis, RWTH Aachen University, Aachen, 2020. doi:10.18154/RWTH-2020-03508.

21 Sandra Kiefer and Brendan D. McKay. The iteration number of Colour Refinement. *CoRR*, 2020. arXiv:2005.10182.

22 Sandra Kiefer and Pascal Schweitzer. Upper bounds on the quantifier depth for graph differentiation in first order logic. In *Proceedings of the Thirty-First Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 287–296, July 2016. doi:10.1145/2933575.2933595.

23 Sandra Kiefer and Pascal Schweitzer. Upper bounds on the quantifier depth for graph differentiation in first-order logic. *Logical Methods in Computer Science*, 15(2), 2019. URL: https://lmcs.episciences.org/5522, doi:10.23638/LMCS-15(2:19)2019.

24 Sandra Kiefer, Pascal Schweitzer, and Erkal Selman. Graphs identified by logics with counting. In *Proceedings of the Fortieth International Symposium on Mathematical Foundations of Computer Science*, volume 9234 of *Lecture Notes in Computer Science*, pages 319–330. Springer, August 2015. doi:10.1007/978-3-662-48057-1_25.

25 Johannes Köbler and Oleg Verbitsky. From invariants to canonization in parallel. In *Computer Science - Theory and Applications, Third International Computer Science Symposium in Russia, CSR 2008, Moscow, Russia, June 7-12, 2008, Proceedings*, volume 5010 of *Lecture Notes in Computer Science*, pages 216–227. Springer, 2008. doi:10.1007/978-3-540-79709-8_23.

26 Andreas Krebs and Oleg Verbitsky. Universal covers, color refinement, and two-variable counting logic: Lower bounds for the depth. In *Proceedings of the Thirtieth Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 689–700, 2015. doi:10.1109/LICS.2015.69.

27 Wenchao Li, Hossein Saidi, Huascar Sanchez, Martin Schäf, and Pascal Schweitzer. Detecting similar programs via the Weisfeiler-Leman graph kernel. In *Software Reuse: Bridging with Social-Awareness - 15th International Conference, ICSR 2016, Limassol, Cyprus, June 5-7, 2016, Proceedings*, pages 315–330, 2016. doi:10.1007/978-3-319-35122-3_21.

28 Moritz Lichter, Ilia N. Ponomarenko, and Pascal Schweitzer. Walk refinement, walk logic, and the iteration number of the Weisfeiler-Leman algorithm. In *Proceedings of the Thirty-Fourth Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 1–13, June 2019. doi:10.1109/LICS.2019.8785694.

29 Brendan D. McKay. Practical graph isomorphism. *Congressus Numerantium*, 30:45–87, 1981.

30 Brendan D. McKay and Adolfo Piperno. Practical graph isomorphism, II. *Journal of Symbolic Computation*, 60:94–112, 2014. doi:10.1016/j.jsc.2013.09.003.

31 Harry L. Morgan. The generation of a unique machine description for chemical structures – a technique developed at chemical abstracts service. *Journal of Chemical Documentation*, 5(2):107–113, 1965. doi:10.1021/c160017a018.

32 Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan E. Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and Leman go neural: Higher-order graph neural networks. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence*, January 2019. doi:10.1609/aaai.v33i01.33014602.

33 Motakuri V. Ramana, Edward R. Scheinerman, and Daniel Ullman. Fractional isomorphism of graphs. *Discrete Mathematics*, 132(1–3):247–265, 1994. doi:10.1016/0012-365X(94)90241-0.

34 Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. Weisfeiler-Lehman graph kernels. *Journal of Machine Learning Research*, 12:2539–2561, 2011. URL: http://dl.acm.org/citation.cfm?id=2078187.

35 Gottfried Tinhofer. A note on compact graphs. *Discrete Applied Mathematics*, 30(2–3):253–264, 1991. doi:10.1016/0166-218X(91)90049-3.

36 Oleg Verbitsky. Planar graphs: Logical complexity and parallel isomorphism tests. In *STACS 2007, 24th Annual Symposium on Theoretical Aspects of Computer Science, Aachen, Germany, February 22–24, 2007, Proceedings*, pages 682–693, 2007. doi:10.1007/978-3-540-70918-3_58.

Towards Optimal Set-Disjointness and Set-Intersection Data Structures

Tsvi Kopelowitz
Department of Computer Science, Bar-Ilan University, Ramat Gan, Israel
kopelot@gmail.com

Virginia Vassilevska Williams
EECS Department, MIT, Cambridge, MA, USA
virgi@mit.edu

Abstract

In the online *set-disjointness* problem the goal is to preprocess a family of sets $\mathcal{F}$, so that given two sets $S, S' \in \mathcal{F}$, one can quickly establish whether the two sets are disjoint or not. If $N = \sum_{S\in\mathcal{F}} |S|$, then let N^p be the preprocessing time and let N^q be the query time. The most efficient known combinatorial algorithm is a generalization of an algorithm by Cohen and Porat [TCS'10] which has a tradeoff curve of $p + q = 2$. Kopelowitz, Pettie, and Porat [SODA'16] showed that, based on the 3SUM hypothesis, there is a conditional lower bound curve of $p + 2q \geq 2$. Thus, the current state-of-the-art exhibits a large gap.

The online *set-intersection* problem is the reporting version of the online set-disjointness problem, and given a query, the goal is to report all of the elements in the intersection. When considering algorithms with N^p preprocessing time and $N^q + O(op)$ query time, where op is the size of the output, the combinatorial algorithm for online set-disjointess can be extended to solve online set-intersection with a tradeoff curve of $p + q = 2$. Kopelowitz, Pettie, and Porat [SODA'16] showed that, assuming the 3SUM hypothesis, for $0 \leq q \leq 2/3$ this curve is tight. However, for $2/3 \leq q < 1$ there is no known lower bound.

In this paper we close both gaps by showing the following:

- For online set-disjointness we design an algorithm whose runtime, assuming $\omega = 2$ (where ω is the exponent in the fastest matrix multiplication algorithm), matches the lower bound curve of Kopelowitz et al., for $q \leq 1/3$. We then complement the new algorithm by a matching conditional lower bound for $q > 1/3$ which is based on a natural hypothesis on the time required to detect a triangle in an unbalanced tripartite graph. Remarkably, even if $\omega > 2$, the algorithm matches the lower bound curve of Kopelowitz et al. for $p \geq 1.73688$ and $q \leq 0.13156$.
- For set-intersection, we prove a conditional lower bound that matches the combinatorial upper bound curve for $q \geq 1/2$ which is based on a hypothesis on the time required to enumerate all triangles in an unbalanced tripartite graph.
- Finally, we design algorithms for detecting and enumerating triangles in unbalanced tripartite graphs which match the lower bounds of the corresponding hypotheses, assuming $\omega = 2$.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis

Keywords and phrases Set-disjointness data structures, Triangle detection, Triangle enumeration, Fine-grained complexity, Fast matrix multiplication

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.74

Category Track A: Algorithms, Complexity and Games

Funding *Tsvi Kopelowitz*: Tsvi Kopelowitz is supported by ISF grant no. 1926/19, and by BSF grant no. 2018364.
Virginia Vassilevska Williams: Virginia Vassilevska Williams is supported by an NSF CAREER Award, NSF Grants CCF-1528078, CCF-1514339 and CCF-1909429, a BSF Grant BSF:2012338, a Google Research Fellowship and a Sloan Research Fellowship.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 74; pp. 74:1–74:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

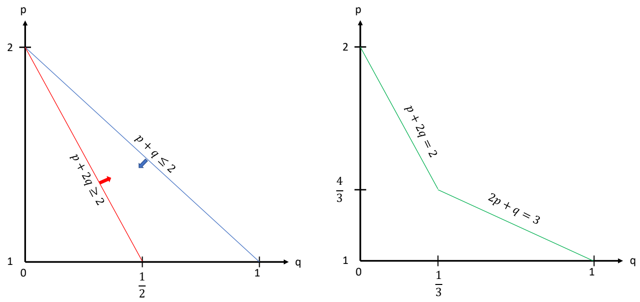


Figure 1 The left graph depicts the gap between the upper bound and lower bound curves for online SetDisjointness prior to this work. The lower bound tradeoff (red curve) is based on the 3SUM hypothesis and the upper bound tradeoff (blue curve) is a variation of the algorithm of Cohen and Porat [14]. The right graph depicts the optimal online SetDisjointness tradeoff (green curve) shown in this paper, assuming that $\omega = 2$.

1 Introduction

In the *online SetDisjointness* problem the goal is to preprocess a family F of subsets from universe U such that given a query pair $(S, S') \in \mathcal{F} \times \mathcal{F}$, one can quickly establish whether S and S' are disjoint or not. The *online SetIntersection* problem is the reporting version of the online SetDisjointness problem, where given a query pair $(S, S') \in \mathcal{F} \times \mathcal{F}$ one must enumerate all of the elements in $S \cap S'$.

Set-disjointness problems at large, including both online SetDisjointness and online SetIntersection, are fundamental algorithmic problems, and have many applications, for example, in information retrieval [14, 24, 20], graph related problems [28, 5, 29, 33, 34], and data structures [27, 16, 29]. Moreover, both problems have played a crucial role in obtaining conditional lower bounds (CLB) in fine-grained complexity. Specifically, many CLBs that are based on the 3SUM hypothesis[1] are reductions from 3SUM to versions of SetDisjointness or SetIntersection, which are further reduced to other algorithmic problems [32, 1, 29, 27, 5, 4, 22, 23]. The Boolean matrix multiplication (BMM) problem can be interpreted as online SetDisjointness with the requirement that the answers to all of the queries must be computed and stored during the preprocessing phase. Thus, the "combinatorial" BMM hypothesis[2] and the CLBs that follow [1, 37, 10] are closely related to online SetDisjointness. Another example is the orthogonal vectors (OV) hypothesis[3] [1, 37], which can be interpreted as asking whether a given family of sets contains two disjoint sets.

[1] The 3SUM hypothesis states that in the Word RAM model of computation with $O(\log n)$ bit words, determining whether a set of n integers contains three that sum to 0, requires $n^{2-o(1)}$ time.

[2] This hypothesis roughly states that algorithms for $n \times n$ BMM that are simple and do not use Strassen-like techniques must take $n^{3-o(1)}$ time in the Word RAM Model with $O(\log n)$ bit words.

[3] The OV hypothesis states that in the Word RAM model with $O(\log n)$ bit words, an algorithm that can decide whether a set of n binary vectors of dimension d contains two orthogonal vectors, must take $n^{2-o(1)}d^{O(1)}$ time.

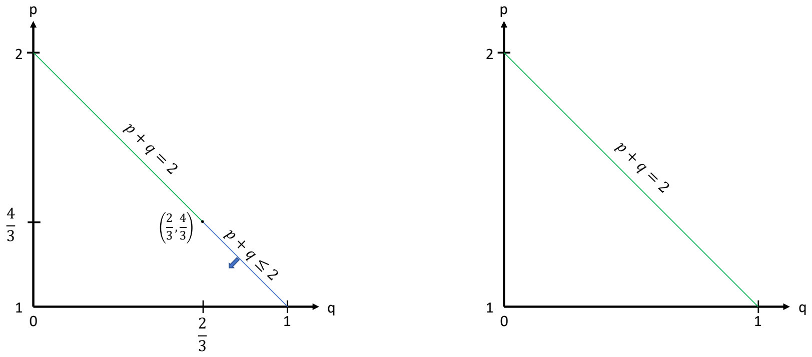


Figure 2 The left graph depicts the gap between the upper bound and lower bound curves for online SetIntersection prior to this work. The green curve (for $q \leq \frac{2}{3}$) is optimal, while the blue curve is an upper bound with no matching lower bound. The right graph depicts the optimal online SetIntersection tradeoff (green curve) shown in this paper, for all values of q.

Measuring efficiency. We measure the efficiency of algorithms for online SetDisjointness and online SetIntersection in terms of $N = \sum_{S\in\mathcal{F}} |S|$. For online SetDisjointness, let N^p and N^q be the preprocessing and query time, respectively. For online SetIntersection, let N^p and $N^q + O(op)$ be the preprocessing and query time, respectively, where op is the size of the output.

When discussing tradeoffs between p and q we make the standard assumption that the preprocessing phase must scan the input at least once, and so $p \geq 1$. Moreover, there is no advantage in allowing $p > 2$ since for $p = 2$ it is straightforward to obtain a constant query time. Thus we assume that $1 \leq p \leq 2$. Similarly, a trivial query algorithm is to scan the entire instance in $O(N)$ time, so we assume that $0 \leq q \leq 1$.

A brief history and the gaps. A variation of the algorithm of Cohen and Porat [14] for online SetDisjointness has $p + q = 2$ (see Section 2.1). A straightforward variation of this algorithm also solves online SetIntersection with $p+q = 2$ (see Section 2.1). To our knowledge, for any values of p and q, there is no published algorithm with a better upper-bound tradeoff.

Regarding lower bounds, assuming the 3SUM hypothesis, Pătraşcu [32] proved that for online SetDisjointness whenever $1 \leq p < 4/3$ we have $q \geq 1/3$. Pătraşcu's CLB is fairly limited with regard to the range of options for p and q. The CLB tradeoff was later improved by Kopelowitz, Pettie and Porat [29] to $p + 2q \geq 2$ for the full range of p and $0 \leq q \leq 1/2$. Kopelowitz et al. [29] also showed that, assuming the 3SUM hypothesis, for online SetIntersection, whenever $4/3 \leq p < 2$ and $0 \leq q \leq 2/3$ we have $p + q \geq 2$. Thus, the combinatorial algorithm is tight for $q \leq \frac{2}{3}$.

In both problems, a large gap remains; see Figures 1 and 2. The goal of this paper is to close the gaps for both problems.

1.1 Our Results

In this paper we take a step towards closing the gaps for both online SetDisjointness and online SetIntersection as follows.

New algorithm for online SetDisjointness. For online SetDisjointness we design an algorithm that utilizes fast matrix multiplication (FMM) (see [15, 36, 35, 31]) and, assuming $\omega = 2$ (where ω is the exponent in the fastest matrix multiplication algorithm), matches the lower bound curve of Kopelowitz et al. [29] for $q \leq 1/3$. The algorithm borrows some ideas from the fast sparse matrix multiplication algorithm of Yuster and Zwick [38], and is stated in the following theorem whose proof appears in Section 2.2.

▶ **Theorem 1.** *There exists an algorithm for the online* SetDisjointness *problem where*

$$\begin{array}{ll} p + \frac{2}{\omega-1}q = 2, & \text{for } 0 \leq q \leq \frac{\omega-1}{\omega+1} \\ \frac{2}{\omega-1}p + q = 1 + \frac{2}{\omega-1}, & \text{for } \frac{\omega-1}{\omega+1} \leq q \leq 1 \end{array}.$$

If $\omega > 2$, the time bounds of Theorem 1 can be improved using fast rectangular matrix multiplication (FRMM) (see [30]). In particular, if we denote by $\omega(1, 1, k)$ the exponent of n in the time required to multiply an $n \times n^k$ matrix by an $n^k \times n$ matrix, then the following corollary is straightforward from the proof of Theorem 1 (see Section 2.2).

▶ **Corollary 2.** *There exists an algorithm for the online* SetDisjointness *problem where*

$$p = (1 - q) \cdot \omega(1, 1, \frac{2-p}{1-q}).$$

We note that, since $\omega(1, 1, k) = 2$ for $k \leq 0.30298$ [30, 21], for the range of $p \geq \frac{4}{2.30298} = 1.73688$ and $q \leq \frac{0.30298}{2.30298} = 0.13156$, the tradeoff becomes $p + 2q = 2$, which is optimal by the 3SUM conjecture.

Unbalanced triangle detection. We complement our new algorithm with a matching CLB for the case of $q \geq 1/3$ which is based on the problem of detecting a triangle in an unbalanced tripartite graph.

▶ **Problem 3** (Unbalanced Triangle Detection). *In the* Unbalanced Triangle Detection *(UTD) problem the goal is to determine whether an undirected tripartite graph $G = (A \cup B \cup C, E)$ contains a triangle or not, where $m_1 = |E \cap (A \times B)|$, $m_2 = |E \cap (B \times C)|$, $m_3 = |E \cap (C \times A)|$, and $m_1 \leq m_2 \leq m_3$.*

▶ **Hypothesis 4** (UTD hypothesis). *Assuming $\omega = 2$, any algorithm for the UTD problem in the word RAM model with $O(\log m_3)$ bit words, for any $m_1 \leq m_2 \leq m_3$, has time cost*

$$\Omega\left((m_1 \cdot m_2)^{\frac{1}{3}} (m_3)^{\frac{2}{3} - o(1)}\right).$$

The UTD hypothesis is the natural extension of the popular Triangle Detection hypothesis [1, 11, 26, 32] which states that, assuming $\omega = 2$, the current best known running time $O(m^{4/3})$ for triangle detection in m-edge graphs is optimal, up to $m^{o(1)}$ factors . To see this, just set $m_1 = m_2 = m_3$ in the UTD hypothesis and the Triangle Detection Hypothesis is obtained by noting that when it comes to triangle problems, without loss of generality, the input graph is tripartite[4].

[4] The reduction works by creating 3 copies of the vertex set, each of which is an independent set, and placing copies of the original edges between copies of vertices (but each vertex copy is in a different copy of the vertex set). Each triangle in the original graph appears 6 times.

We remark an important subtlety in the UTD hypothesis: in order to disprove the hypothesis, it is enough to design an algorithm that beats the hypothesized lower bound for a single combination of m_1, m_2 and m_3. Nevertheless, the CLBs that we prove based on the UTD hypothesis hold even if we restrict the UTD hypothesis to be true for the restricted cases of $m_2 = m_3$.

UTD algorithm. In Section 3 We design a new algorithm for UTD which matches the lower bounds of the UTD hypothesis if $\omega = 2$. The algorithm is a natural (albeit not exactly straightforward) extension of the best known algorithms for triangle detection [3].

▶ **Theorem 5.** *There exists an algorithm for the UTD problem whose time cost is*

$$O\left(m_3 + (m_1 \cdot m_2)^{\frac{\omega-1}{\omega+1}}(m_3)^{\frac{2}{\omega+1}}\right).$$

CLB for online SetDisjointness. In Section 4 we prove a CLB for online SetDisjointness which is conditioned on the UTD hypothesis. The CLB, which matches the upper bound of Theorem 1 for $q \geq \frac{1}{3}$, assuming $\omega = 2$, is summarized in the following theorem.

▶ **Theorem 6.** *Assuming $\omega = 2$, any algorithm for online SetDisjointness that has $\frac{1}{3} \leq q < 1$ must obey $2p + q \geq 3$, unless the UTD hypothesis is false.*

Assuming that $\omega = 2$, Theorems 6 and 1 combined with the 3SUM CLB of Kopelowitz et al. [29] provide a (conditionally) optimal curve, as depicted in Figure 1.

Unbalanced triangle enumeration. For set-intersection, we prove a conditional lower bound that matches the combinatorial upper bound curve for $q \geq 1/2$ which is based on a hypothesis on the time required to enumerate all triangles in an unbalanced tripartite graph.

▶ **Problem 7** (Unbalanced Triangle Enumeration)**.** *In the* Unbalanced Triangle Enumeration *(UTE) problem the goal is to enumerate all triangles in a given undirected tripartite graph $G = (A \cup B \cup C, E)$, where $m_1 = |E \cap (A \times B)|$, $m_2 = |E \cap (B \times C)|$, $m_3 = |E \cap (C \times A)|$, and $m_1 \leq m_2 \leq m_3$.*

▶ **Hypothesis 8** (UTE hypothesis)**.** *Assuming $\omega = 2$, any algorithm for the UTE problem on a graph with t triangles must have time cost $t^{\frac{1}{3}}(m_1 m_2 m_3)^{\frac{1}{3}} m_3^{-o(1)}$ in the word RAM model with $O(\log m_3)$ bit words.*

Similar to the UTD hypothesis, the UTE hypothesis implies that, assuming $\omega = 2$, the current best known running time of $O(m + m^{\frac{4}{3}} + t^{\frac{1}{3}} m)$ for triangle enumeration in m-edge graphs by Bjørklund et al. [9] is optimal, up to $m^{o(1)}$ factors (just set $m_1 = m_2 = m_3$).

UTE algorithm. In Section 5 we design a new algorithm for UTE which, assuming $\omega = 2$, matches the lower bounds of the UTE hypothesis. The algorithm is a natural (albeit not exactly straightforward) extension of the best known algorithms for output-sensitive triangle enumeration [9].

▶ **Theorem 9.** *There exists an algorithm for UTE on graphs with at most t triangles whose time cost is*

$$\tilde{O}\left(m_3 + m_3^{\frac{2}{\omega+1}}(m_1 m_2)^{\frac{\omega-1}{\omega+1}} + t^{\frac{3-\omega}{\omega+1}}(m_1 m_2 m_3)^{\frac{\omega-1}{\omega+1}}\right).$$

CLB for online SetIntersection In Section 6, we prove the following CLB for SetIntersection based on the UTE hypothesis, which matches the algorithm of Section 2.1 for $q \geq 1/2$.

▶ **Theorem 10.** *Any algorithm for online SetIntersection that has $\frac{1}{2} \leq q < 1$ must obey $p + q \geq 2$, unless the UTE hypothesis is false.*

1.2 More Related Work

SetDisjointness and SetIntersection. Many existing set intersection data structures, e.g., [17, 7, 6], work in the comparison model in which sets are represented as sorted lists or arrays. The benchmark in this model is the minimum number of comparisons needed to answer a query. Bille, Pagh, and Pagh [8] used word-packing techniques to evaluate expressions of set intersections and unions. Their query algorithm finds the intersection of k sets with a total of n elements in $O(n/\frac{w}{\log^2 w} + k \cdot op)$ time, where op is the size of the output and w is the size of a machine word. Cohen and Porat [13] designed a *static* $O(N)$-space data structure for answering online SetIntersection queries in $O(\sqrt{N(1 + |S \cap S'|)})$ time. Kopelowitz, Porat and Pettie [28] designed an incremental algorithm for online SetDisjointness where both queries and element insertions into sets cost $O(\sqrt{\frac{N}{\log n/\log\log n}})$ time.

Kopelowitz, Porat and Pettie [28] also designed a fully dynamic algorithm for both online SetDisjointness and online SetIntersection which uses M words of space, each update costs $O(\sqrt{M}\log N)$ expected time, each SetIntersection query costs $O(\frac{N\sqrt{\log N}}{\sqrt{M}}\sqrt{op+1})$ expected time where op is the size of the output, and each online SetDisjointness query costs $O(\frac{N\sqrt{\log N}}{\sqrt{M}} + \log N)$ expected time. The relationship between the space usage and query time was also investigated by Afshani and Neilsen [2] and Goldstein et al. [23].

Triangle Enumeration. Itai and Rodeh [25] showed that all t triangles in a graph could be enumerated in $O(m^{3/2})$ time. Thirty years ago Chiba and Nishizeki [12] generalized [25] to show that $O(m\alpha)$ time suffices, where α is the *arboricity* of the graph. Kopelowitz, Pettie, and Porat [28] proved that enumerating t triangles takes $O(m\lceil\alpha/\frac{\log n}{\log\log n}\rceil + t)$ time. Eppstein et al. [19] designed an algorithm for the w-bit word RAM model running in $O(m\lceil\alpha/\frac{w}{\log w}\rceil + t)$ time.

The fastest algorithm for triangle enumeration in general m-edge, n-node graphs is the algorithm of Bjørklund, Pagh, Williams, and Zwick [9] which if $\omega = 2$, runs in $\tilde{O}(\min\{n^2 + nt^{2/3}, m^{4/3} + mt^{1/3}\})$ time.

Duraj et al. [18] showed that the related problem of establishing for each edge e in a graph the number of triangles that contain e is equivalent to several natural range query problems.

2 SetDisjointness Algorithms

Before we describe our new algorithm, we begin by presenting a simple algorithm for online SetDisjointness which is a variation of the algorithm of Cohen and Porat [14], and has a efficiency tradeoff of $p + q = 2$. This algorithm is a building block for our new algorithm.

2.1 Heavy-Light Decomposition

▶ **Lemma 11.** *There exists an algorithm for the online SetDisjointness problem with $p+q = 2$.*

Proof. Let $0 \leq \alpha \leq 1$. A set S is said to be *light* if $|S| \leq N^\alpha$, and *heavy* otherwise. Notice that the number of heavy sets is at most $N^{1-\alpha}$.

The algorithm stores each set via a lookup table, and precomputes the answers to all $O(N^{2-2\alpha})$ heavy pairs (pairs of heavy sets). Specifically, for a heavy pair S and S', the algorithm checks for each element $e \in S$ whether $e \in S'$. The sets S and S' are disjoint if and only if all of the tests fail. Notice that during the precomputation, an element in a heavy set is looked up at most $N^{1-\alpha}$ times, once for each heavy set. Since the number of elements in all heavy sets is at most N, the total preprocessing cost is $O(N^{2-\alpha})$ time and $p = 2 - \alpha$.

For the query, if both of the queries sets are heavy then the answer is obtained from the precomputed information, and if at least one query set is light then the algorithm scans the at most N^{α} elements in the light set to test whether any of these elements are in the other set (regardless of whether the other set is heavy or light). Thus, the query cost is $O(N^{\alpha})$ and $q = \alpha$. Finally, $p + q = 2 - \alpha + \alpha = 2$ as required. ◀

SetIntersection Algorithm. It is fairly straightforward to convert the algorithm of Lemma 11 to also solve online SetIntersection with $p + q = 2$: for each pair of heavy sets, instead of storing only an indication of whether the sets are disjoint or not the algorithm stores the entire intersection.

2.2 Improved algorithm

▶ **Theorem 1.** *There exists an algorithm for the online SetDisjointness problem where*

$$\begin{array}{ll} p + \frac{2}{\omega-1}q = 2, & \textit{for } 0 \le q \le \frac{\omega-1}{\omega+1} \\ \frac{2}{\omega-1}p + q = 1 + \frac{2}{\omega-1}, & \textit{for } \frac{\omega-1}{\omega+1} \le q \le 1 \end{array}.$$

Proof. The algorithm is similar to the algorithm in the proof of Lemma 11, but with a faster method for precomputing all of the answers for heavy pairs. Thus, assume without loss of generality that there are at most $N^{1-\alpha}$ sets, where α is taken from the proof of Lemma 11.

For every element $e \in U$, let $f_e = |\{S \in \mathcal{F} : e \in S\}|$ be the number of sets in $\mathcal{F}$ that contain e. For a parameter $0 \le \beta \le 1$, an element $e \in U$ is said to be *frequent* if $f_e > N^{\beta}$, and *rare* otherwise. Let F be the set of frequent elements and let R be the set of rare elements. Notice that $|F| \le N^{1-\beta}$ and $\sum_{e\in R} f_e \le N$.

For each rare element e, there are at most $O((f_e)^2) = O(N^{2\beta})$ heavy pairs that contain e in their intersection, and enumerating these pairs costs $O((f_e)^2)$ time. In order to efficiently enumerate these pairs for all rare elements, the algorithm computes for each element e a list of sets that contain e by scanning the entire instance in linear time. Given these lists, the algorithm enumerates all of the heavy pairs that have a rare element in their intersection in $O(\sum_{e\in R}(f_e)^2) = O(\sum_{e\in R} N^{\beta} f_e) = O(N^{\beta}\sum_{e\in R} f_e) = O(N^{1+\beta})$ time.

The algorithm is now left with the task of establishing which heavy pairs have at least one frequent element in their intersection. However, enumerating all heavy pairs that have a frequent element in their intersection is too expensive. In order to reduce the time cost, the algorithm constructs a Boolean matrix M such that the columns of M correspond to frequent elements and the rows of M correspond to characteristic vectors of the heavy sets after removing all of the rare elements. Thus, the size of M is $|H| \times |F|$ where H is the set of heavy sets. Let $P = M \cdot M^T$ where the product is a Boolean product. Notice that $p_{i,j} = 1$ if and only if there exists an element e such that the both $m_{i,e} = 1$ and $m^T_{e,j} = m_{j,e} = 1$. Thus, the non-zero entries of P exactly correspond to the heavy pairs that have a frequent element in their intersection. The time cost of computing P using FMM is

$$O\left(\frac{|H|^2|F|}{\min(|H|,|F|)^{3-\omega}}\right) = O\left(N^{\omega-2\alpha-\beta+(3-\omega)(\max(\alpha,\beta))}\right).$$

Thus, the total preprocessing time is $N^p = O\left(N^{1+\beta} + N^{\omega-2\alpha-\beta+(3-\omega)(\max(\alpha,\beta))}\right)$, which is minimized whenever $1+\beta = \omega - 2\alpha - \beta + (3-\omega)(\max(\alpha,\beta))$, and then $p = 1+\beta$. Recall that the query time is $O(N^\alpha)$ and so $q = \alpha$.

If $\alpha \leq \beta$, then $\beta = 1 - \frac{2\alpha}{\omega-1}$, the preprocessing time is given by $p = 1+\beta = 2 - \frac{2q}{\omega-1}$, and so $p + \frac{2}{\omega-1}q = 2$. Notice that in order for this case to hold, it must be that $\alpha \leq \beta = 1 - \frac{2\alpha}{\omega-1}$ implying that $q = \alpha \leq \frac{\omega-1}{\omega+1}$.

If $\alpha > \beta$, then $\alpha = 1 - \frac{2\beta}{\omega-1}$, the query time is given by $q = \alpha = 1 - \frac{2(p-1)}{\omega-1}$, and so $\frac{2}{\omega-1}p + q = 1 + \frac{2}{\omega-1}$. Notice that in order for this case to hold, it must be that $\alpha = 1 - \frac{2\beta}{\omega-1} \geq 1 - \frac{2\alpha}{\omega-1}$ implying that $q = \alpha \geq \frac{\omega-1}{\omega+1}$. ◀

▶ **Corollary 2.** *There exists an algorithm for the online* SetDisjointness *problem where*

$$p = (1-q) \cdot \omega(1, 1, \frac{2-p}{1-q}).$$

Proof. When using FRMM, the cost to compute P is $O\left(|H|^{\omega(1,1,\log_{|H|}\frac{|F|}{|H|})}\right) = O\left(N^{(1-\alpha)\cdot\omega(1,1,\frac{1-\beta}{1-\alpha})}\right)$ time. Thus, the total preprocessing time is $N^p = O\left(N^{1+\beta} + N^{(1-\alpha)\cdot\omega(1,1,\frac{1-\beta}{1-\alpha})}\right)$, which is minimized whenever $1+\beta = (1-\alpha)\cdot\omega(1,1,\frac{1-\beta}{1-\alpha})$, and so $p = (1-q)\cdot\omega(1,1,\frac{2-p}{1-q})$. ◀

3 Unbalanced Triangle Detection Algorithm

▶ **Theorem 5.** *There exists an algorithm for the UTD problem whose time cost is*

$$O\left(m_3 + (m_1 \cdot m_2)^{\frac{\omega-1}{\omega+1}}(m_3)^{\frac{2}{\omega+1}}\right).$$

Proof. The algorithm uses three positive integer parameters to be set later: τ_A, τ_B, and τ_C. A vertex $a \in A$ is said to be light if the number of edges $(a,b) \in E \cap (A \times B)$ is at most τ_A, and heavy otherwise. Thus, the number of heavy nodes in A is at most $\frac{m_1}{\tau_A}$. A vertex $b \in B$ is said to be light if the number of edges $(b,c) \in E \cap (B \times C)$ is at most τ_B, and heavy otherwise. Thus, the number of heavy nodes in B is at most $\frac{m_2}{\tau_B}$. A vertex $c \in C$ is said to be light if the number of edges $(c,a) \in E \cap (C \times A)$ is at most τ_C, and heavy otherwise. Thus, the number of heavy nodes in C is at most $\frac{m_3}{\tau_C}$.

Light vertices. For each light $a \in A$ the algorithm enumerates all pairs of edges touching a where one edge touches a vertex in B and the other edge touches a vertex in C, and for each such pair the algorithm tests (in constant time) whether the pair is part of a triangle. If there exists a triangle that contains a light $a \in A$ then one of the enumerated pairs must be two edges from this triangle and thus the algorithm will detect this triangle. Let $d_B(a)$ be the number of edges of a whose other endpoint is in B, and let $d_C(a)$ be the number of edges of a whose other endpoint is in C. Thus, the total number of pairs for a given $a \in A$ is $d_B(a) \cdot d_C(a)$. Notice that $\sum_{a\in A} d_C(a) = m_3$, and recall that since a is light then $d_B(a) \leq \tau_A$. Thus, the time cost for testing whether there exists a triangle with a light $a \in A$ vertex is

$$O(\sum_{a\in A} d_B(a) \cdot d_C(a)) \leq O(\tau_A \sum_{a \in A} d_C(a)) = O(\tau_A \cdot m_3).$$

Similarly, the algorithm checks whether there is exists a triangle with a light $b \in B$ or a light $c \in C$ in $O(\tau_B \cdot m_1 + \tau_C \cdot m_2)$. Thus, the total cost of detecting whether there exists a triangle with at least one light vertex is $O(m_3 + \tau_A \cdot m_3 + \tau_B \cdot m_1 + \tau_C \cdot m_2)$ time, where the first m_3 term comes for the necessity of scanning the entire graph.

Heavy vertices. If there is no triangle that contains at least one light vertex, then there can only be a triangle with three heavy vertices. Here the algorithm utilizes the upper bound on the number of heavy vertices in each part of the tripartite graph. Without loss of generality, let $a_1, a_2, \dots, a_{\frac{m_1}{\tau_A}}$ be the set of heavy vertices in A, let $b_1, b_2, \dots, b_{\frac{m_2}{\tau_B}}$ be the set of heavy vertices in B, and let $c_1, c_2, \dots, c_{\frac{m_3}{\tau_C}}$ be the set of heavy vertices in A. Let L be a $\frac{m_1}{\tau_A} \times \frac{m_2}{\tau_B}$ Boolean matrix where $q_{i,j} = 1$ if and only if $(a_i, b_j) \in E$. Similarly, let R be a $\frac{m_2}{\tau_B} \times \frac{m_3}{\tau_C}$ Boolean matrix where $r_{i,j} = 1$ if and only if $(b_i, c_j) \in E$, and let T be a $\frac{m_1}{\tau_A} \times \frac{m_3}{\tau_C}$ Boolean matrix where $t_{i,j} = 1$ if and only if $(a_i, c_j) \in E$.

The algorithm computes $Z = (L \cdot R) \bigwedge T$ where the first operator is a BMM and the second operator is an entry-wise AND.

▷ Claim 12. $Z \neq 0$ if and only if there exists a triangle in G whose vertices are all heavy.

Proof. Let $X = L \cdot R$. If there exists an entry $z_{i,j} = 1$ then $x_{i,j} = t_{i,j} = 1$. Since $t_{i,j} = 1$, then by definition $(a_i, c_j) \in E$. Since $x_{i,j} = 1$, then there must exist an integer $1 \le k \le \frac{m_2}{\tau_B}$ such that $l_{i,k} = 1$ and $r_{k,j} = 1$, and so $(a_i, b_k), (b_k, c_j) \in E$, implying that the triangle (a_i, b_k, c_j) is in G, and all of the vertices of this triangle are heavy.

For the other direction, suppose that the triangle (a_i, b_k, c_j) is in G, and all of the vertices of this triangle are heavy. Then in particular $l_{i,k} = r_{k,i} = t_{i,j} = 1$. Thus, it must be that $x_{i,j} = 1$ and so $z_{i,j} = 1$. ◁

The cost of computing Z is dominated by the cost of computing the BMM of L and R, which is

$$O\left(\frac{m_1 m_2 m_3}{\tau_A \cdot \tau_B \cdot \tau_C \cdot (\min(\frac{m_1}{\tau_A}, \frac{m_2}{\tau_B}, \frac{m_3}{\tau_C}))^{3-\omega}}\right).$$

Time cost. The total time cost is

$$O\left(m_3 + \tau_A \cdot m_3 + \tau_B \cdot m_1 + \tau_C \cdot m_2 + \frac{m_1 m_2 m_3}{\tau_A \cdot \tau_C \cdot (\min(\frac{m_1}{\tau_A}, \frac{m_2}{\tau_B}, \frac{m_3}{\tau_C}))^{3-\omega}}\right).$$

The time cost is minimized when the last four[5] terms in the summation are all equal:

$$\tau_A \cdot m_3 = \tau_B \cdot m_1 = \tau_C \cdot m_2 = \frac{m_1 m_2 m_3}{\tau_A \cdot \tau_B \cdot \tau_C \cdot (\min(\frac{m_1}{\tau_A}, \frac{m_2}{\tau_B}, \frac{m_3}{\tau_C}))^{3-\omega}}.$$

Since $m_1 \le m_2 \le m_3$ it must be that $\tau_A \le \tau_C \le \tau_B$, and so $\frac{m_2}{\tau_B} \le \frac{m_3}{\tau_C}$. Moreover, $m_2 \tau_A \le m_2 \tau_C = m_1 \tau_B$ and so $\frac{m_2}{\tau_B} \le \frac{m_1}{\tau_A}$. Thus, $\min(\frac{m_1}{\tau_A}, \frac{m_2}{\tau_B}, \frac{m_3}{\tau_C}) = \frac{m_2}{\tau_B}$. By plugging in $\tau_B = \frac{m_3}{m_1}\tau_A$ and $\tau_C = \frac{m_3}{m_2}\tau_A$, we have

$$\begin{aligned}\tau_A \cdot m_3 &= \frac{m_1 m_2 m_3}{\tau_A \cdot \tau_B \cdot \tau_C \cdot (\min(\frac{m_1}{\tau_A}, \frac{m_2}{\tau_B}, \frac{m_3}{\tau_C}))^{3-\omega}} \\ &= \frac{m_1 m_2 m_3}{\tau_A \cdot \frac{m_3}{m_1}\tau_A \cdot \frac{m_3}{m_2}\tau_A \cdot (\frac{m_2}{\frac{m_3}{m_1}\tau_A})^{3-\omega}} = \frac{(m_1 m_2)^{\omega-1}}{\tau_A^{\omega} \cdot (m_3)^{\omega-2}},\end{aligned}$$

and so $\tau_A = (\frac{m_1 m_2}{m_3})^{\frac{\omega-1}{\omega+1}}$. Finally, the time cost is

$$O(m_3 + \tau_A \cdot m_3) = O(m_3 + (m_1 m_2)^{\frac{\omega-1}{\omega+1}} (m_3)^{\frac{2}{\omega+1}}).$$

[5] The reason for focusing only on the last four terms and not on the first term is that the last four terms contain parameters which we can control, while the first term m_3 is always set.

Notice that with an $O(m_3)$ time preprocessing we can ensure that every vertex in A has at least one edge to B and to C (process the vertices in B and C similarly), by removing any vertex (and its incident edges) without this property. This procedure never removes any triangles, and ensures that $m_1 \geq \max\{|A|, |B|\}$, $m_2 \geq \max\{|C|, |B|\}$, and $m_3 \geq \max\{|A|, |C|\}$. As $m_3 \leq |A| \cdot |C| \leq m_1 \cdot m_2$, $\tau_A \geq 1$. Similarly, $\tau_B, \tau_C \geq 1$, so the thresholds used by the algorithm make sense. ◀

4 Optimal Conditional Lower Bound for SetDisjointness

▶ **Theorem 6.** *Assuming $\omega = 2$, any algorithm for online SetDisjointness that has $\frac{1}{3} \leq q < 1$ must obey $2p + q \geq 3$, unless the UTD hypothesis is false.*

Proof. To prove the theorem we first describe a reduction from the UTD problem to the online SetDisjointness problem by describing an algorithm for UTD that uses an algorithm for online SetDisjointness as a black box. We then show that if the online SetDisjointness algorithm obeys $2p + q = 3 - \epsilon$ for $\frac{1}{3} \leq q \leq 1$, for some constant $\epsilon > 0$, then there exists an algorithm contradicting the UTD Hypothesis.

Reduction from UTD to online SetDisjointness. Given an instance $G = (A \cup B \cup C, E)$ of UTD, for each $x \in A \cup B$ define the set S_x to be the set of vertices from C that are neighbors of x. All of the sets are given as input for the preprocessing phase of the online SetDisjointness algorithm. Notice that the sum of the sizes of the sets is exactly $N = m_2 + m_3 = \Theta(m_3)$, since each edge touching a vertex in C contributes exactly one element to exactly one of the sets. Next, for each of the m_1 edges $(a, b) \in E \cup (A \times B)$, the algorithm performs an online SetDisjointness query on S_a and S_b. If any of the queries returns a false (meaning that the intersection of the two sets is not empty) then the algorithm returns that there is a triangle in G, and otherwise, the algorithm returns that there is no triangle in G.

▷ Claim 13. There exists an edge $(a, b) \in E \cap (A \times B)$ such that $S_a \cap S_b \neq \emptyset$ if and only if G contains a triangle.

Proof. If there exists a triangle $(a, b, c) \in A \times B \times C$ in G, then both S_a and S_b contain c. Thus, $S_a \cap S_b \neq \emptyset$. For the other direction, if there exists an edge $(a, b) \in E \cap (A \times B)$ such that $S_a \cap S_b \neq \emptyset$ then there exists some $c \in S_a \cap S_b$ which implies that $(a, c), (b, c) \in E$, and so (a, b, c) is a triangle in G. ◁

Finally, the time cost of solving UTD is $O(N^p + m_1 \cdot N^q) = O((m_3)^p + m_1 \cdot (m_3)^q)$.

The lower bound. Suppose that there exists an online SetDisjointness algorithm with $\frac{1}{3} \leq q < 1$ and $2p + q = 3 - \epsilon$ for some positive $\epsilon > 0$. By rearranging,

$$3(p - 1) = \frac{3(1 - q)}{2} - \frac{3}{2}\epsilon.$$

Thus, there exist constant positive numbers x and $\epsilon_q = \frac{3}{4}\epsilon$ such that

$$3(p - 1) + \epsilon_q = x = \frac{3(1 - q)}{2} - \epsilon_q.$$

Notice that, since $q \geq \frac{1}{3}$, then $x < \frac{3(1-q)}{2} \leq 1$. Moreover, by rearranging, there exists a constant $\epsilon' = \frac{1}{3}\epsilon_q$ such that

$$p \leq \frac{x + 3}{3} - \epsilon'$$

and

$$q + x \leq \frac{x+3}{3} - \epsilon'.$$

Thus, since the UTD hypothesis holds for any combination of m_1, m_2 and m_3 (as long as $m_1 \leq m_2 \leq m_3$), we set $m_1 = (m_3)^x$ and $m_2 = m_3$ (recall that $x < 1$). The UTD hypothesis states that the time cost for solving UTD on this setting of m_1, m_2 and m_3 is $\Omega\left((m_3)^{\frac{x+3}{3}}\right)$, while the time cost of solving UTD using the reduction is

$$O((m_3)^p + m_1 \cdot (m_3)^q) = O((m_3)^p + (m_3)^{x+q}) = O\left((m_3)^{\frac{x+3}{3} - \epsilon'}\right),$$

thereby obtaining a contradiction. ◀

5 Unbalanced Triangle Enumeration

We begin with an algorithm which does not care about the number of triangles in the input.

▶ **Lemma 14.** *There exists an algorithm for the UTE problem whose time cost is*

$$O\left(m_3 + \sqrt{m_1 \cdot m_2 \cdot m_3}\right).$$

Proof. The algorithm uses the same definition and treatment of light vertices as in the algorithm in the proof of Theorem 5, but instead of stopping once a triangle is found, the algorithm continues until all triangles that contain at least one light vertex are enumerated. Recall that this process costs $O(m_3 + \tau_A \cdot m_3 + \tau_B \cdot m_1 + \tau_C \cdot m_2)$ time.

Heavy vertices. Since there can be at most $\frac{m_1}{\tau_A}$ heavy vertices in A, at most $\frac{m_2}{\tau_B}$ heavy vertices in B, and at most $\frac{m_3}{\tau_C}$ heavy vertices in C, there can be at most $\frac{m_1 \cdot m_2 \cdot m_3}{\tau_A \cdot \tau_B \cdot \tau_C}$ triangles whose vertices are *all* heavy. Thus, for each triplet of a heavy vertex $a \in A$, a heavy vertex $b \in B$, and a heavy vertex $c \in C$, the algorithm spends constant time looking up whether (a, b, c) is a triangle or not. The time cost of enumerating all such triplets is $O\left(\frac{m_1 \cdot m_2 \cdot m_3}{\tau_A \cdot \tau_B \cdot \tau_C}\right)$.

Time cost. The total time cost is

$$O(m_3 + \tau_A \cdot m_3 + \tau_B \cdot m_1 + \tau_C \cdot m_2 + \frac{m_1 \cdot m_2 \cdot m_3}{\tau_A \cdot \tau_B \cdot \tau_C}).$$

The time cost is minimized when the last four terms in the summation are all equal:

$$\tau_A \cdot m_3 = \tau_B \cdot m_1 = \tau_C \cdot m_2 = \frac{m_1 \cdot m_2 \cdot m_3}{\tau_A \cdot \tau_B \cdot \tau_C}.$$

By plugging in $\tau_B = \frac{m_3}{m_1}\tau_A$ and $\tau_C = \frac{m_3}{m_2}\tau_A$, we have

$$\begin{aligned} \tau_A \cdot m_3 &= \frac{m_1 \cdot m_2 \cdot m_3}{\tau_A \cdot \tau_B \cdot \tau_C} \\ &= \frac{m_1 \cdot m_2 \cdot m_3}{\tau_A \cdot \frac{m_3}{m_1}\tau_A \cdot \frac{m_3}{m_2}\tau_A} \\ &= \frac{(m_1 \cdot m_2)^2}{(\tau_A)^3 \cdot m_3}. \end{aligned}$$

Therefore, $\tau_A = \sqrt{\frac{m_1 \cdot m_2}{m_3}}$, and the total time cost is $O(m_3 + \tau_A \cdot m_3) = O(m_3 + \sqrt{m_1 \cdot m_2 \cdot m_3})$.

Similarly to the UTD algorithm, notice that with an $O(m_3)$ time preprocessing we can ensure that every vertex in A has at least one edge to B and to C (process the vertices in B and C similarly), by removing any vertex (and its incident edges) without this property. This procedure never removes any triangles, and ensures that $m_1 \geq \max\{|A|, |B|\}$, $m_2 \geq \max\{|C|, |B|\}$, and $m_3 \geq \max\{|A|, |C|\}$. As $m_3 \leq |A| \cdot |C| \leq m_1 \cdot m_2$, $\tau_A \geq 1$. Similarly, $\tau_B, \tau_C \geq 1$, so the thresholds the algorithm uses make sense. ◀

Now we give an algorithm for UTE, assuming that the input graph has t triangles. Notice that our algorithm for UTD can count the number of triangles, so we can assume that we know t.

▶ **Theorem 9.** *There exists an algorithm for UTE on graphs with at most t triangles whose time cost is*

$$\tilde{O}\left(m_3 + m_3^{\frac{2}{\omega+1}}(m_1 m_2)^{\frac{\omega-1}{\omega+1}} + t^{\frac{3-\omega}{\omega+1}}(m_1 m_2 m_3)^{\frac{\omega-1}{\omega+1}}\right).$$

Proof. Let L, D_1, D_2, D_3 be parameters to be chosen later; we will make sure that all of these parameters are at least 1. Recall that $m_1 = E \cap (A \times B), m_2 = E \cap (B \times C), m_3 = E \cap (A \times C)$.

For every $a \in A$ with at most D_1 neighbors in C, list all triangles through a by going through all pairs of neighbors of a. The total time over all low-degree $a \in A$ is $O(m_1 D_1)$ time. Similarly, in $O(m_2 D_2)$ time list all triangles through all $b \in B$ with at most D_2 neighbors in A and in $O(m_3 D_3)$ time list all triangles through all $c \in C$ with at most D_3 neighbors in B.

Now let us set $D_1 = \frac{m_3 D_3}{m_1}$ and $D_2 = \frac{m_3 D_3}{m_2}$. As $m_3 \geq m_1, m_2$, $D_1, D_2 \geq 1$. This makes the total time so far $O(m_3 D_3)$.

Any triangle (a, b, c) that has not been listed must have that a has at least D_1 neighbors in C, b has at least D_2 neighbors in A and c has at least D_3 neighbors in B. Thus we can restrict to a subset A' of A of size at most $n_A = m_1/D_2 = (m_1 m_2)/(m_3 D_3)$, a subset B' of B of size at most $n_B = m_2/D_3$ and a subset C' of C of size at most $n_C = m_3/D_1 = m_1/D_3$. Notice that

$$n_A = (m_1/D_3) \cdot (m_2/m_3) \leq (m_1/D_3) = n_C \leq (m_2/D_3) = n_B.$$

Bjørklund et al. [9] give an $\tilde{O}(L^{3-\omega} n^{\omega})$ time algorithm that given a tripartite graph G' with n nodes in each partition, every edge e of G' the algorithm lists L triangles that contain e, for some parameter $L \geq 1$, or all triangles containing e if e is in fewer than L triangles. Our algorithm reduces to this balanced case.

Since $n_A \leq n_C \leq n_B$, the algorithm splits the larger partitions into parts of size roughly n_A, thereby obtaining $(n_B n_C)/n_A^2$ instances of balanced graphs, where every partition has n_A vertices. On each one of these instances the algorithm executes the algorithm of [9] to list up to L triangles for every edge in the remaining graph. The running time of this step is

$$\tilde{O}\left(n_B n_C n_A^{\omega-2} L^{3-\omega}\right) = \tilde{O}\left(\frac{(m_1 m_2)^{\omega-1}}{D_3^{\omega} m_3^{\omega-2}} L^{3-\omega}\right).$$

To minimize the total runtime we set

$$m_3 D_3 = \frac{(m_1 m_2)^{\omega-1}}{D_3^{\omega} m_3^{\omega-2}} L^{3-\omega}.$$

This sets $D_3 = L^{\frac{3-\omega}{\omega+1}}(m_1 m_2/m_3)^{\frac{\omega-1}{\omega+1}}$. Notice that as long as $L \geq 1$, $D_3 \geq 1$, just as with the UTD algorithm the algorithm can execute an $O(m_3)$ time preprocessing phase to make sure that $m_3 \leq m_1 m_2$.

With this setting of D_3, the runtime of this step becomes

$$O\left(m_3^{\frac{2}{\omega+1}}(m_1m_2)^{\frac{\omega-1}{\omega+1}}L^{\frac{3-\omega}{\omega+1}}\right).$$

Now, set $L = \max\{1, 6t/m_3\}$. The total runtime of the algorithm so far becomes:

$$\tilde{O}\left(m_3 + m_3^{\frac{2}{\omega+1}}(m_1m_2)^{\frac{\omega-1}{\omega+1}} + (m_1m_2m_3)^{\frac{\omega-1}{\omega+1}}T^{\frac{3-\omega}{\omega+1}}\right).$$

The only triangles remaining are those through edges that are contained in more than L triangles. As the number of triangles is t and since each triangle has 3 edges, the total number of edges whose triangles the algorithm has not found is at most $3t/L$. Notice that from the earlier steps of the algorithm we know for every edge e how many triangles contain e. Thus, the algorithm also knows the $3t/L$ edges that are left.

If $L = 1$ and so $6t/m_3 \leq 1$, we must have $t \leq m_3/6$ and so $3t/L = 3t \leq m_3/2$. Otherwise, if $L = 6t/m_3$, then we also get $3t/L = m_3/2$. In both cases, the total number of remaining edges is at most $m_3/2$, and so the algorithm recurses, applying the same steps but on an unbalanced graph with at most m_1, m_2 and $m_3/2$ edges and still at most t triangles. When the number of edges becomes constant, the algorithm solves the problem via brute force.

Since in each recursive step the current largest edge set shrinks by a factor of 2, the number of recursive steps is $O(\log n)$ and we at most tack on a logarithmic factor to the runtime. ◀

6 Optimal Conditional Lower Bound for SetIntersection

▶ **Theorem 10.** *Any algorithm for online SetIntersection that has $\frac{1}{2} \leq q < 1$ must obey $p + q \geq 2$, unless the UTE hypothesis is false.*

Proof. To prove the theorem we first describe a reduction from the UTE problem to the online SetIntersection problem by describing an algorithm for UTE that uses an algorithm for online SetIntersection as a black box. We then show that if the online SetIntersection algorithm obeys $p + q \geq 2 - \epsilon$ for $\frac{1}{2} \leq q < 1$ and some constant $\epsilon > 0$, then there exists an algorithm contradicting the UTE Hypothesis.

Reduction from UTE to online SetIntersection. The reduction is the same as the reduction given in the proof of Theorem 6, except that instead of using online SetDisjointness, the reduction algorithm uses SetIntersection. Specifically, the reduction algorithm does not stop after it is established that two sets are not disjoint. Instead, for each of the m_1 edges $(a, b) \in E \cup (A \times B)$, the algorithm performs an online SetIntersection query, and for each c in the output the algorithm enumerates triangle (a, b, c). The correctness of the reduction follows from the following claim.

▷ Claim 15. For every edge $(a, b) \in E \cap (A \times B)$ there is a bijection between every $c \in S_a \cap S_b$ and every triangle in G containing (a, b).

Proof. If there exists a triangle $(a, b, c) \in A \times B \times C$ in G, then both S_a and S_b contain c, and so $c \in S_a \cap S_b$. For the other direction, for every edge $(a, b) \in E \cap (A \times B)$ and every $c \in S_a \cap S_b$, the edges (a, c) and (b, c) must be in E, and so (a, b, c) is a triangle in G. ◁

As in the proof of Theorem 6, the sum of the sizes of the sets in the online SetIntersection instance is exactly $N = m_2 + m_3 = \Theta(m_3)$, and the size of the output is $O(t)$. Finally, the time cost of solving UTE is $O(N^p + m_1 \cdot N^q + t) = O((m_3)^p + m_1 \cdot (m_3)^q + t)$.

The lower bound. Suppose that there exists an online SetIntersection algorithm with $\frac{1}{2} \leq q < 1$ and $p + q \leq 2 - \epsilon$ for some positive $\epsilon > 0$. Without loss of generality, assume that $\epsilon < 2 - 2q$, which is okay since $q < 1$.

By rearranging, $2p - 2 \leq 2 - 2q - 2\epsilon$. Thus, there exists a constant positive number x such that

$$2p - 2 + \epsilon \leq x \leq 2 - 2q - \epsilon.$$

Notice that since $q \geq \frac{1}{2}$ then $x < 2 - 2q \leq 1$. Moreover, by rearranging, there exists a constant $\epsilon' > 0$ such that $p \leq 1 + \frac{x}{2} - \epsilon'$ and $q + x \leq 1 + \frac{x}{2} - \epsilon'$.

Since the UTE hypothesis holds for any combination of m_1, m_2 and m_3 (as long as $m_1 \leq m_2 \leq m_3$), we set $m_3 = m_2$ and $m_1 = (m_3)^x$. Moreover, let $t = (m_3)^y$ where $y = 1 + \frac{x}{2} - \epsilon'$.

Notice that the maximum number of triangles in an unbalanced tripartite graph with edge set sizes m_1, m_2, m_3 is $\sqrt{m_1 m_2 m_3} = m_3^{1+x/2}$, so that the number of triangles we need to list here is just a bit smaller than this.

The UTE hypothesis states that the time cost for solving UTE on this setting of m_1, m_2 and m_3 is

$$\Omega\left(t^{\frac{1}{3}}(m_1 m_2 m_3)^{\frac{1}{3}}\right) = \Omega\left(m_3^{\frac{2+x+y}{3}}\right) = \Omega\left(m_3^{(1+\frac{x}{2})-\frac{1}{3}\epsilon'}\right).$$

However, the time cost of solving UTE using the reduction is

$$\begin{aligned} O((m_3)^p + m_1 \cdot (m_3)^q + t) &= O((m_3)^p + (m_3)^{x+q} + (m_3)^y) \\ &= O((m_3)^{1+\frac{x}{2}-\epsilon'}) << \Omega((m_3)^{1+\frac{x}{2}-\frac{\epsilon'}{3}}), \end{aligned}$$

where the last transition is due to $\epsilon' > 0$. Thus, we have obtained a contradiction. ◀

References

1 A. Abboud and V. Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 434–443, 2014.

2 P. Afshani and J. Sindahl Nielsen. Data structure lower bounds for document indexing problems. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP*, pages 93:1–93:15, 2016.

3 N. Alon, R. Yuster, and U. Zwick. Finding and counting given length cycles. *Algorithmica*, 17:209–223, 1997.

4 A. Amir, T. M. Chan, M. Lewenstein, and N. Lewenstein. On hardness of jumbled indexing. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP, Part I*, pages 114–125, 2014.

5 A. Amir, T. Kopelowitz, A. Levy, S. Pettie, E. Porat, and B. R. Shalom. Mind the gap: Essentially optimal algorithms for online dictionary matching with one gap. In *27th International Symposium on Algorithms and Computation, ISAAC*, pages 12:1–12:12, 2016.

6 R. A. Baeza-Yates. A fast set intersection algorithm for sorted sequences. In *Combinatorial Pattern Matching, 15th Annual Symposium, CPM*, pages 400–408, 2004. doi:10.1007/978-3-540-27801-6_30.

7 J. Barbay and C. Kenyon. Adaptive intersection and t-threshold problems. In *Proceedings 13th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 390–399, 2002.

8 Philip Bille, Anna Pagh, and Rasmus Pagh. Fast evaluation of union-intersection expressions. In *Algorithms and Computation, 18th International Symposium, ISAAC*, pages 739–750, 2007.

9 A. Bjorklund, R. Pagh, V. Vassilevska Williams, and U. Zwick. Listing triangles. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP, Part I*, pages 223–234, 2014.

10 T. M. Chan, S. Durocher, K. Green Larsen, J. Morrison, and B. T. Wilkinson. Linear-space data structures for range mode query in arrays. *Theory Comput. Syst.*, 55(4):719–741, 2014.

11 K. Chatterjee, W. Dvorák, M. Henzinger, and A. Svozil. Algorithms and conditional lower bounds for planning problems. In *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling, ICAPS*, pages 56–64. AAAI Press, 2018.

12 N. Chiba and T. Nishizeki. Arboricity and subgraph listing algorithms. *SIAM J. Comput.*, 14(1):210–223, 1985.

13 H. Cohen and E. Porat. Fast set intersection and two-patterns matching. *Theor. Comput. Sci.*, 411(40-42):3795–3800, 2010. doi:10.1016/j.tcs.2010.06.002.

14 H. Cohen and E. Porat. On the hardness of distance oracle for sparse graph. *CoRR*, abs/1006.1117, 2010. arXiv:1006.1117.

15 D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *J. Symbolic Computation*, 9(3):251–280, 1990.

16 P. Davoodi, M. H. M. Smid, and F. van Walderveen. Two-dimensional range diameter queries. In *LATIN 2012: Theoretical Informatics - 10th Latin American Symposium*, pages 219–230, 2012.

17 E. D. Demaine, A. López-Ortiz, and J. Ian Munro. Adaptive set intersections, unions, and differences. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 743–752, 2000. URL: http://dl.acm.org/citation.cfm?id=338219.338634.

18 Lech Duraj, Krzysztof Kleiner, Adam Polak, and Virginia Vassilevska Williams. Equivalences between triangle and range query problems. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 30–47. SIAM, 2020.

19 D. Eppstein, M. T. Goodrich, M. Mitzenmacher, and M. R. Torres. 2-3 Cuckoo filters for faster triangle listing and set intersection. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS)*, pages 247–260, 2017.

20 J. Fischer, T. Gagie, T. Kopelowitz, M. Lewenstein, V. Mäkinen, L. Salmela, and N. Välimäki. Forbidden patterns. In *LATIN 2012: Theoretical Informatics - 10th Latin American Symposium*, pages 327–337, 2012.

21 F. Le Gall and F. Urrutia. Improved rectangular matrix multiplication using powers of the coppersmith-winograd tensor. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1029–1046, 2018.

22 I. Goldstein, T. Kopelowitz, M. Lewenstein, and E. Porat. How hard is it to find (honest) witnesses? In *24th Annual European Symposium on Algorithms, ESA*, pages 45:1–45:16, 2016.

23 I. Goldstein, T. Kopelowitz, M. Lewenstein, and E. Porat. Conditional lower bounds for space/time tradeoffs. In *Algorithms and Data Structures - 15th International Symposium, WADS*, pages 421–436, 2017.

24 Wing-Kai Hon, Rahul Shah, Sharma V. Thankachan, and Jeffrey Scott Vitter. Space-efficient frameworks for top-k string retrieval. *J. ACM*, 61(2):9:1–9:36, 2014.

25 A. Itai and M. Rodeh. Finding a minimum circuit in a graph. *SIAM J. Comput.*, 7(4):413–423, 1978.

26 Z. Jafargholi and E. Viola. 3sum, 3xor, triangles. *Algorithmica*, 74(1):326–343, 2016.

27 T. Kopelowitz and R. Krauthgamer. Color-distance oracles and snippets. In *27th Annual Symposium on Combinatorial Pattern Matching, CPM*, pages 24:1–24:10, 2016.

28 T. Kopelowitz, S. Pettie, and E. Porat. Dynamic set intersection. In *Proceedings 14th Int'l Symposium on Algorithms and Data Structures (WADS)*, pages 470–481, 2015.

29 T. Kopelowitz, S. Pettie, and E. Porat. Higher lower bounds from the 3SUM conjecture. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1272–1287, 2016.

30 F. Le Gall. Faster algorithms for rectangular matrix multiplication. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS*, pages 514–523, 2012.

31 F. Le Gall. Powers of tensors and fast matrix multiplication. In *International Symposium on Symbolic and Algebraic Computation, ISSAC '14, Kobe, Japan, July 23-25, 2014*, pages 296–303, 2014.

32 M. Patrascu. Towards polynomial lower bounds for dynamic problems. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC*, pages 603–610, 2010.

33 M. Patrascu and L. Roditty. Distance oracles beyond the Thorup-Zwick bound. *SIAM J. Comput.*, 43(1):300–311, 2014.

34 M. Patrascu, L. Roditty, and M. Thorup. A new infinity of distance oracles for sparse graphs. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS*, pages 738–747, 2012.

35 A. Stothers. On the complexity of matrix multiplication. *Ph.D. Thesis, U. Edinburgh*, 2010.

36 V. Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC*, pages 887–898, 2012.

37 V. Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *Proceedings of the International Congress of Mathematicians*, pages 3431–3475, 2018.

38 R. Yuster and U. Zwick. Fast sparse matrix multiplication. *ACM Trans. on Algorithms*, 1(1):2–13, 2005.

Kinetic Geodesic Voronoi Diagrams in a Simple Polygon

Matias Korman
Department of Computer Science, Tufts University, Medford, MA, USA
matias.korman@tufts.edu

André van Renssen
School of Computer Science, University of Sydney, Australia
andre.vanrenssen@sydney.edu.au

Marcel Roeloffzen
Department of Mathematics and Computer Science,
Eindhoven University of Technology, The Netherlands
m.j.m.roeloffzen@tue.nl

Frank Staals
Department of Information and Computing Sciences, Utrecht University, The Netherlands
f.staals@uu.nl

Abstract

We study the geodesic Voronoi diagram of a set S of n linearly moving sites inside a static simple polygon P with m vertices. We identify all events where the structure of the Voronoi diagram changes, bound the number of such events, and then develop a kinetic data structure (KDS) that maintains the geodesic Voronoi diagram as the sites move. To this end, we first analyze how often a single bisector, defined by two sites, or a single Voronoi center, defined by three sites, can change. For both these structures we prove that the number of such changes is at most $O(m^3)$, and that this is tight in the worst case. Moreover, we develop compact, responsive, local, and efficient kinetic data structures for both structures. Our data structures use linear space and process a worst-case optimal number of events. Our bisector KDS handles each event in $O(\log m)$ time, and our Voronoi center handles each event in $O(\log^2 m)$ time. Both structures can be extended to efficiently support updating the movement of the sites as well. Using these data structures as building blocks we obtain a compact KDS for maintaining the full geodesic Voronoi diagram.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases kinetic data structure, simple polygon, geodesic voronoi diagram

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.75

Category Track A: Algorithms, Complexity and Games

Related Version A full version of the paper is available at `http://arxiv.org/abs/2002.05910`.

Funding *Matias Korman*: M.K. was partially supported by MEXT KAKENHI No. 17K12635 and the NSF award CCF-1422311.
André van Renssen: A.V.R. was supported by JST ERATO Grant Number JPMJER1201, Japan.
Marcel Roeloffzen: M.R. was supported by JST ERATO Grant Number JPMJER1201, Japan and the Netherlands Organisation for Scientific Research (NWO) under project no. 628.011.005.
Frank Staals: F.S. was supported by the Netherlands Organisation for Scientific Research under project no. 612.001.651.

Acknowledgements We would like to thank Man-Kwun Chiu and Yago Diez for interesting discussions during the initial stage of this research.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 75; pp. 75:1–75:17
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Polygons are one of the most fundamental objects in computational geometry. As such, they have been used for many different purposes in different contexts. Within the path planning community, polygons are often used to model different regions. A simple example is when we have a robot moving within a building: in such a case we model all possible locations that a robot can reach by a polygon (the walls or any obstacle in the way form the boundary of this polygon). Then, the goal is to find a path that connects the source point and the destination and that minimizes some objective function. There are countlessly many results that depend on the exact function used (distance traveled [10], time needed to reach [18], number of required turns [27], etc.) Paths that minimize distance are often called *geodesics*.

Two of the most fundamental problems in this setting are constructing *shortest path maps* and *augmented Voronoi diagrams*. A shortest path map (or SPM for short) is a partition of the space into regions so that points in the same region travel in the same way to the fixed source [10, 13]. The exact definition of "in the same way" depends on the exact problem setting, but it often means that paths are *combinatorially* the same, that is, they have the same internal vertices. Augmented Voronoi diagrams are a generalization of SPMs for the case in which we have more than one fixed source and we are interested in the topology of the path to the closest source [3]. See Fig. 1 for an illustration. These structures are of critical importance in obtaining efficient solutions to related problems such as finding center points, closest pairs, nearest neighbors, and constructing spanners [22, 23].

It often happens that while we are moving to our destination, that destination is also moving. For example, when two agents try to meet, one wants to evade the other, or one simply needs to meet up with a second one that is doing a different task [17]. Since it is very costly to recompute the solution after each infinitesimal movement, the aim is to somehow maintain some information from which we can easily obtain the solution, and update this information only when the solution has significant changes. A data structure that can handle such a setting is known as a *kinetic data structure* (or KDS for short) [6]. There is a wide range of problems that have been studied in this setting. We refer to the survey by Basch et al. [6] for an overview of these results.

Surprisingly, there is very little work that combines all three of the above concepts (polygons, shortest paths, and kinetic data structures). We are aware of only two results. Aronov et al. [5] present a KDS for maintaining the shortest path map of a single point

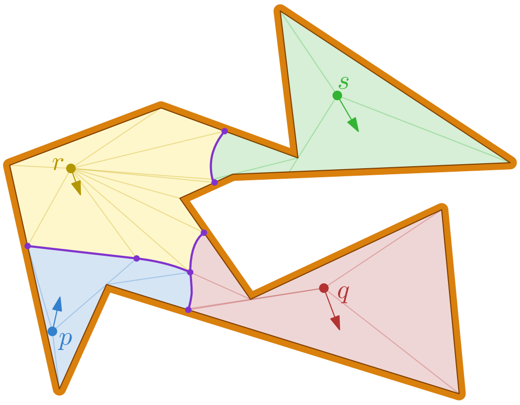


Figure 1 The (augmented) geodesic Voronoi diagram of four moving sites p, q, r, and s.

moving inside a simple polygon, and Karavelas and Guibas [14] give a KDS to maintain a *constrained Delaunay triangulation* of a set of moving points. This allows them to maintain nearest neighbors and the geodesic hull.

We present the first KDS to maintain the full (augmented) geodesic Voronoi diagram of a set of point sites moving inside a simple polygon, thus generalizing the above results. We carefully analyze when and how often it can change. To this end, we prove tight bounds on the number of combinatorial changes in a single bisector, and on the trajectory of a Voronoi center. Our results provide an important tool for maintaining related structures in which the agents (sites) move linearly within the simple polygon (e.g. minimum spanning trees, nearest-neighbors, closest pairs, etc.).

Related Work. Our data structures are based on the Kinetic Data Structures (KDS) framework introduced by Basch et al. [6]. In this framework motions are assumed to be known in advance. Each KDS maintains a set of *certificates* that together certify that the KDS currently correctly represents the target structure. Typically these certificates involve a few objects each and represent some simple geometric primitive. For example a certificate may indicate that three points form a clockwise oriented triangle. As the points move these certificates may become invalid, requiring the KDS to update. This requires repairing the target structure and creating new certificates. Such a certificate failure is called an (*internal*) *event*. An event is *external* if the target structure also changes. The performance of a KDS is measured according to four measures. A KDS is considered *compact* if it requires little space, generally close to linear, *responsive* if each event is processed quickly, generally polylogarithmic time, *local* if each site participates in few events, and *efficient* if the ratio between external and internal events is small, generally polylogarithmic. Note that for efficiency it is common to compare the worst-case number of events for either case.

Let S be a set of n point sites moving linearly in a space P, that is, each point moves with a fixed speed and direction. The *Voronoi diagram* $\mathrm{VD}_P(S)$ of S is a partition of P into n regions, one per site $s \in S$, such that for any point q in such a region $\mathcal{V}_s$ is closer to s than to any other site from S. Guibas et al. [11] studied maintaining the Voronoi diagram in case $P = \mathbb{R}^2$ and distance is measured by the Euclidean distance. They prove that $\mathrm{VD}_{\mathbb{R}^2}(S)$ may change $\Omega(n^2)$ times, and present an a KDS that handles at most $O(n^3\beta_4(n))$ events, each in $O(\log n)$ time. Here, $\beta_z(n) = \lambda_z(n)/n$ and $\lambda_z(n)$ is the maximum length of a Davenport-Schinzel sequence of n symbols of order z [25]. Their results actually extend to slightly more general types of movement. It is one of the long outstanding open problems if this bound can be improved [8, 9]. Only recently, Rubin [24] showed that if all sites move linearly and with the same speed, the number of changes is at most $O(n^{2+\varepsilon})$ for some arbitrarily small $\varepsilon > 0$. For arbitrary speeds, the best known bound is still $O(n^3\beta_4(n))$. When the distance function is specified by a convex k-gon the number of changes is $O(k^4n^2\beta_z(n))$ [2]. Here, and throughout the rest of the paper z denotes some small constant.

Let P be a simple polygon with m vertices, and let $\pi(s, q)$ be the shortest path between s and q that stays entirely inside P. We measure length of a path by the sum of the Euclidean edge lengths. Such a shortest path $\pi(s, q)$ is known as a geodesic, and its length as the *geodesic distance* between s and q. With some abuse of notation we use $\pi(s, q)$ to denote both the shortest path and its length.

Aronov was the first to study the geodesic Voronoi diagram [3]. He proved that when the sites in S are static, $\mathrm{VD}_P(S)$ has complexity $O(n + m)$. The same bound applies for the augmented geodesic Voronoi diagram. Moreover, he presented an $O((n+m)\log(n+m)\log n)$ time algorithm for constructing $\mathrm{VD}_P(S)$, which was improved to $O((n + m)\log(n + m))$ by

Table 1 The different types of events at which the geodesic Voronoi diagram changes, and their number. At an a, b-collapse event two vertices of $\mathrm{VD}_P(S)$ with degrees a and b collide and one disappears. Similarly, at an a, b-expand one such a vertex appears. At a vertex event a vertex of $\mathrm{VD}_P(S)$ collides with a vertex of P.

Event	Lower bound	Upper bound
$1, 2$-collapse/expand	$\Omega(m^2n)$	$O(m^2n^2)$
$1, 3$-collapse/expand	$\Omega(mn \min\{n, m\})$	$O(m^2n^2 \min\{m\beta_z(n), n\})$
$2, 2$-collapse/expand	$\Omega(m^3n)$	$O(m^3n\beta_4(n))$
$2, 3$-collapse/expand	$\Omega(mn^2 + m^3n)$	$O(m^3n^2\beta_4(n)\beta_z(n))$
$3, 3$-collapse/expand	$\Omega(mn^2 + m^2n)$	$O(m^3n^3\beta_z(n))$
vertex	$\Omega(m^2n)$	$O(m^2n\beta_4(n))$

Papadopoulou and Lee [23]. Recently, there have been several improved algorithms [16, 21] which ultimately lead to an optimal $O(m + n \log n)$ time algorithm by Oh [20]. Furthermore, Agarwal et al. [1] recently showed that finding the site in S closest to an arbitrary query point $q \in P$ – a key application of geodesic Voronoi diagrams – can be achieved efficiently even if sites may be added to, or removed from, S. Note however, that their result cannot be used directly to maintain a substructure of the Voronoi diagram (e.g. an MST).

There are no known results on maintaining an (augmented) geodesic Voronoi diagram when multiple sites S move continuously in a simple polygon P. In case there is only one site s, Aronov et al. [5] presented a KDS that maintains the shortest path map SPM_s of s. Their data structure uses $O(m)$ space, and processes a total of $O(m)$ events in $O(\log m)$ time each[1]. Karavelas and Guibas [14], provide a KDS to maintain a constrained Delaunay triangulation of S. This allows them to maintain the geodesic hull of S w.r.t. P, and the set of nearest neighbors in S (even in case P has holes). Their KDS processes $O((m+n)^3\beta_z(n+m))$ events in $O(\log(n + m))$ time each.

Organization and Results. We present a kinetic data structure to maintain the geodesic Voronoi diagram $\mathrm{VD}_P(S)$ of a set S of n sites moving linearly inside a simple polygon P with m vertices. To this end, we prove a tight $O(m^3)$ bound on the number of combinatorial changes in a single bisector, and develop a compact, efficient, and responsive KDS to maintain it (Section 3). Our KDS for the bisector uses $O(m)$ space and processes events in $O(\log m)$ time. We then show that the movement of the Voronoi center c_{pqs} – the point equidistant to three sites $p, q, s \in S$ – can also change $O(m^3)$ times (Section 4). We again show that this bound is tight, and develop a compact, efficient, and responsive KDS to maintain c_{pqs}. The space usage is linear, and handling an event takes $O(\log^2 m)$ time. Both our KDSs can be made local as well, and therefore efficiently support updates to the movement of the sites. Building on these results we then analyze the full Voronoi diagram $\mathrm{VD}_P(S)$ of n moving sites (Section 5). We identify the different types of events at which $\mathrm{VD}_P(S)$ changes, and bound their number. Table 1 gives an overview of our bounds. We then develop a compact KDS to maintain $\mathrm{VD}_P(S)$. Omitted proofs are in the full version of this paper [15].

[1] The original description by Aronov et al. [5] uses a dynamic convex hull data structure that supports $O(\log^2 m)$ time queries and updates. Instead, we can use the data structure by Brodal and Jacob [7] which supports these operations in $O(\log m)$ time.

2 Preliminaries

We first review some properties of geodesic Voronoi diagrams and shortest path maps that we will use. Let SPM_s be the shortest path map of s, hence for all points in a region of SPM_s the shortest path from s has the same internal vertices. Each such region R is star-shaped with respect to the last internal vertex v on the shortest path. Often it will be useful to refine R into triangles incident to v. We refer to the resulting subdivision of P as the *extended* shortest path map. With some abuse of notation we will use SPM_s to denote this subdivision as well. An edge in SPM_s that starts in a vertex v that is colinear with the last edge in $\pi(s, v)$ is called an *extension segment* $E_{vs} = E_v$.

Let $\mathbb{T} = \mathbb{R}$ denote the time domain. We consider each site $s \in S$ as a function from $\mathbb{T}$ to P. For functions we will not distinguish between the function itself and its graph. We say that a function is *simple* if it is continuous, i.e. if it has no break points.

Given two sites p and q, the bisector B_{pq} is the set of all points that are equidistant to p and q. If no vertex of P lies on the bisector, then B_{pq} is a piecewise curve connecting two points on ∂P. Each curve on B_{pq} is a subarc of a hyperbola [3, 19].

▶ **Lemma 1** (Aronov [3]). *$\mathrm{VD}_P(S)$ consists of $O(n)$ vertices with degree 1 or 3, and $O(m)$ vertices of degree 2. For each degree 2 vertex v there is are $p, q \in S$ so that v lies on the bisector B_{pq} and v lies on extension segment of SPM_p or SPM_q. All edges of $\mathrm{VD}_P(S)$ are hyperbolic arc segments. Every vertex v of P contributes at most one extension segment E_v.*

▶ **Lemma 2** (Aronov et al. [5]). *Let s be a point moving linearly inside a simple polygon P with m vertices. The extended shortest path map SPM_s changes at most $O(m)$ times.*

▶ **Lemma 3.** *Let v be a vertex of P, there are $O(mn\beta_4(n))$ time intervals in which v has a unique closest site $s \in S$, and the distance from v to s over time is a hyperbolic function.*

3 A Single Bisector

Fix a pair of sites p and q, and let $b_{pq}(t)$ and $b_{qp}(t)$ be the endpoints of the bisector B_{pq} defined so that p lies to the right of $B_{pq}(t)$ when following the bisector from $b_{pq}(t)$ to $b_{qp}(t)$. As p and q move, the structure of B_{pq} changes at discrete times, or events. We distinguish between the following types of events (see Fig. 2):

- *vertex* events, at which an endpoint of B_{pq} coincides with a vertex of P,
- $1, 2$-*collapse* events, at which a degree 2 vertex (an interior vertex) of B_{pq} disappears as it collides with a degree 1 vertex (an endpoint),
- $1, 2$-*expand* events, at which a new degree 2 vertex appears from a degree 1 vertex,

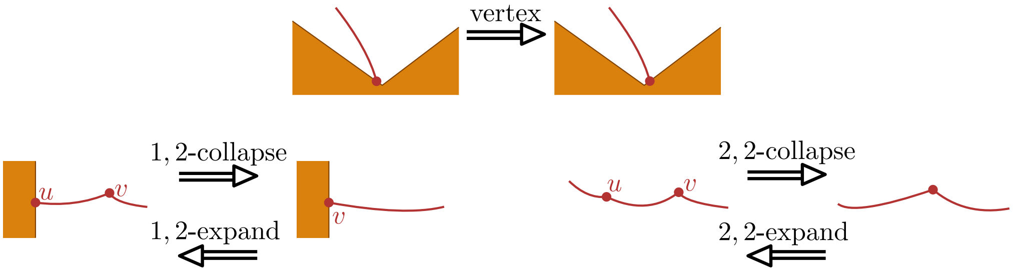


Figure 2 The types of events during which the structure of B_{pq} changes.

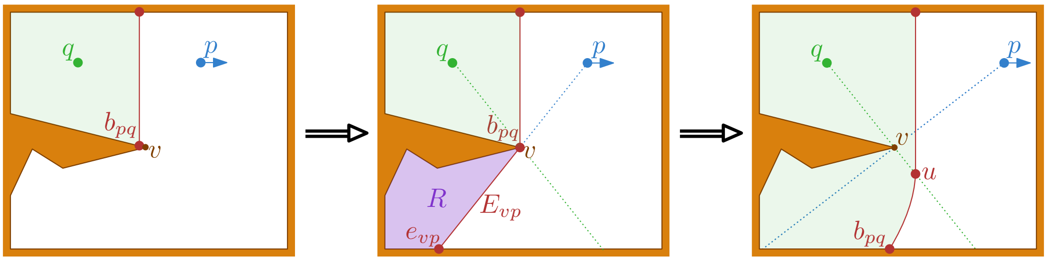


Figure 3 A vertex event at v may coincide with a $1,2$-expand event. At the time of the event all points in R are equidistant to p and q, and b_{pq} jumps from v to e_{vp}.

- $2,2$-*collapse* events at which a degree 2 vertex disappears by colliding with an other degree 2 vertex, and
- $2,2$-*expand* events, at which a new degree 2 vertex appears from a degree 2 vertex.

In Section 3.1 we prove that there are at most $O(m^2)$ vertex and $1,2$-collapse events, and at most $O(m^3)$ $2,2$-collapse events. The number of expand events can be similarly bounded. Some of these events may actually happen simultaneously. See for example Fig. 3, where B_{pq} changes when a vertex event and a $1,2$-expand event coincide. So we are double-counting these simultaneous events. Despite this, we show that our $O(m^3)$ bound on the number of changes of B_{pq} is tight in the worst case. In Section 3.2 we then argue that there is a KDS that can maintain B_{pq} efficiently.

3.1 Bounding the Number of Events

We start by showing that a bisector B_{pq} of p and q may change $\Omega(m^3)$ times. We then argue that there is also an $O(m^3)$ upper bound on the number of such changes.

▶ **Lemma 4.** *The bisector $B_{pq}(t)$ can change $\Omega(m^3)$ times.*

Proof. The main idea is to construct a bisector B_{pq}, a piecewise hyperbolic curve, of complexity $\Omega(m)$ in the middle of a region that consists of $\Omega(m^2)$ cells. These cells are defined by the extension segments in SPM_p and SPM_q that extend from the vertices on two convex chains of the polygon; one chain on either side of B_{pq}. See the top area in Fig. 4. In each cell the distance functions to p and q are different, thus if B_{pq} moves across a cell this causes a change in the bisector. The two upper convex chains in P have size $\Omega(m)$ and are placed such that as p moves towards the left (and q remains in place), the bisector sweeps from left to right over $\Omega(m^2)$ middle cells, thus causing $\Omega(m^2)$ changes to B_{pq}.

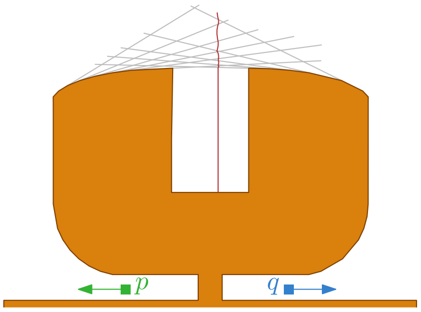


Figure 4 The bisector B_{pq} may be involved in $\Omega(m^3)$ $2,2$-collapse events.

Next, we argue that B_{pq} can be moved back and forth across these cells $\Omega(m)$ times by adding two convex chains of size $\Omega(m)$ to the bottom of the polygon, just above p and q. This way we can ensure that p and q alternate being the closest to the top of the polygon. Thus, when p is closest the bisector will move to the right and when q is the closest the bisector will move to the left. By making p and q move at the same speed and having the segments defining the convex chain on q's side start and end in the middle of where the segments of the convex chain on p's side, we can cause this alternation. It follows that the bisector sweeps over the $\Omega(m^2)$ middle cells $\Omega(m)$ times and thus it changes $\Omega(m^3)$ times. ◀

In the above proof one can observe that the counted events are only the 2,2-collapse events. For 2,2-collapse and 2,2-expand events there is also a $O(m^3)$ upper bound. For other events there is a $O(m^2)$ upper bound (proofs are in the full version [15]). Hence the bisector B_{pq} may change at most $O(m^3)$ times. Even though the entire bisector may change $O(m^3)$ times, the trajectories of its intersection points with the boundary of P have complexity at most $O(m^2)$. The following theorem summarizes these results.

▶ **Theorem 5.** *Let p and q be two points moving linearly inside P. The bisector B_{pq} of p and q can change $O(m^3)$ times. This bound is tight in the worst case. The trajectories of the endpoints of B_{pq} have $O(m^2)$ edges, each corresponding to a low-degree algebraic curve.*

3.2 A Kinetic Data Structure to Maintain a Bisector

We first describe a simple, yet naive, KDS to maintain B_{pq} that is not responsive and then show how to improve it to obtain a responsive KDS.

A Non-Responsive KDS to Maintain a Bisector. Our naive KDS for maintaining B_{pq} stores: (i) the extended shortest path maps of p and q using the data structure of Aronov et al. [5], (ii) the vertices of B_{pq}, ordered along B_{pq} from b_{pq} to b_{qp} in a balanced binary search tree, and (iii) for every vertex u of B_{pq}, the cell of SPM_p and of SPM_q that contains u. Since all cells in SPM_p and SPM_q are triangles, this requires only $O(1)$ certificates per vertex. We store these certificates in a priority queue $\mathcal{Q}$.

At any time where B_{pq} changes combinatorially (i.e. at an event) the shortest path to a vertex v of B_{pq} changes combinatorially, which indicates a change in the SPM cells that contain v. Hence, we detect all events. Conversely, when any vertex v of B_{pq} moves to a different SPM cell there is a combinatorial change in the bisector, so each event triggered by parts (ii) and (iii) of the KDS is an external event. The events at which SPM_p or SPM_q changes are internal (unless they also cause a change in a shortest path to a vertex of B_{pq}).

The changes to SPM_p and SPM_q are handled as in Aronov et al. and we update our other structures accordingly (see the full version for details [15]). This leads to a compact and efficient KDS to maintain the bisector. However, when the shortest path from p or q to some polygon vertex v changes this affects all bisector vertices whose shortest paths go through v. Hence, a single change in SPM_p or SPM_q may lead to a large number of updates to the certificates of bisector vertices. Therefore, the KDS is not responsive. To solve this issue we need a more refined data structure.

A Responsive KDS to Maintain a Bisector. First we dissect in some more detail the anatomy of a bisector. Each bisector consists of two endpoints which are degree 1 vertices and a chain of degree 2 vertices connecting them. We can further divide this chain based on which parts are directly visible from the sites defining the bisector. This division results in

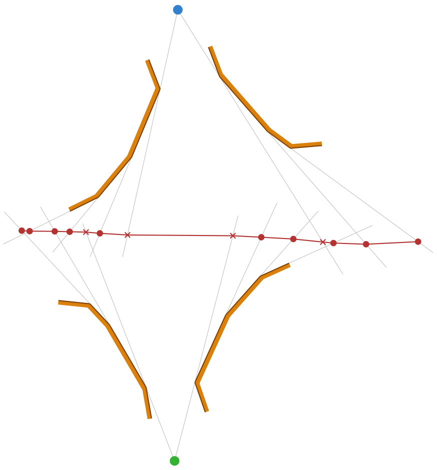


Figure 5 A bisector can be split into at most five pieces, here separated by degree 2 vertices marked as crosses.

at most 5 pieces, as illustrated in Fig. 5; some pieces may not be present in every bisector. First there is a *double-visible* piece that is visible from both sites p and q. Since P is a simple polygon, this piece consists of a single line segment. Adjacent to the double-visible piece on either side there may be a *single-visible* piece that is only visible to p or to q, but not both. Lastly, there are up to two *non-visible* pieces that are not directly visible from either p or q.

We will still store the bisector vertices in a balanced binary tree ordered along the bisector, but we will store the certificates for the degree 2 vertices a little differently. For each of the at most four degree 2 vertices that separate the pieces as well as the degree 1 endpoints, we store the cells of SPM_p and SPM_q that contain them. Then we observe that for internal vertices of the single-visible piece there can be no events. Each of these internal vertices lies on an extension segment of a single convex chain of vertices in the simple polygon and these extension segments do not intersect. Therefore no 2,2-collapses can occur.

The non-visible pieces are trickier, since 2,2-collapses may occur when a vertex moving on an extension segment of SPM_q moves to a different cell of SPM_p. Fortunately such potential events on a single non-visible bisector piece are related and form a strict ordering, regardless of the exact distance functions of the various vertices to p and q.

We define *event points* to be the locations at which 2,2-collapses that may occur. Consider two degree 2 vertices v and w that are internal to a non-visible piece of bisector between sites p and q, such that v and w are adjacent on the bisector and we have that v is on an extension segment of SPM_p and w is on an extension segment of SPM_q. Let the *event point* $\mathrm{ep}_{v,w}$ denote the intersection between these two extension segments. A 2,2-event between v and w corresponds to the event point being on the bisector between p and q. Without loss of generality assume that the event point currently lies in the Voronoi cell of p. We can then use the certificate $\pi(\mathrm{ep}_{v,w}, p) < \pi(\mathrm{ep}_{v,w}, q)$ to detect the 2,2-event between v and w. As we saw above maintaining these certificates explicitly is not efficient as any change in the shortest path towards p or q requires us to recompute the failure time. Instead we will store all event points in the Voronoi cell of p in one balanced binary tree ordered along the bisector and those in q in another. For each node in such a tree, we maintain the event point in its subtree that will be the first to be on the bisector, similar to a kinetic tournament (this, in turn requires maintaining distances between some of the relevant event points in the subtree). With this representation, we have to compute explicit failure times only for the two event points stored in the roots of the trees. Using these ideas we obtain the following result:

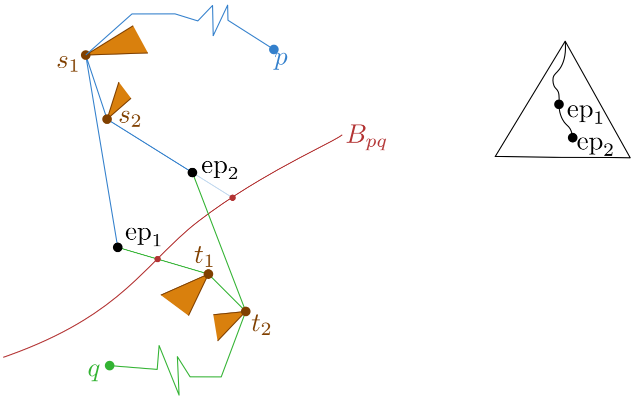


Figure 6 On the left is a schematic drawing of the event points ep_1 and ep_2 with their shortest paths towards p and q. On the right how (the certificates of) ep_1 and ep_2 are stored in the BST.

▶ **Theorem 6.** *Let p and q be two sites moving linearly inside a simple polygon P with m vertices. There is a KDS that maintains the bisector B_{pq} that uses $O(m)$ space and processes at most $O(m^3)$ events, each of which can be handled in $O(\log m)$ time. Additionally it can support movement changes of p and q in $O(\log m)$ time and splitting the bisector at any given vertex in $O(\log^2 m)$ time.*

Proof. For a single non-visible bisector piece between sites p and q, Consider event points ep_1 and ep_2 where ep_2 is a child of ep_1 in the tree. Let s_1 and t_1 denote the first polygon vertex on the shortest path from ep_1 towards p and q respectively and let s_2 and t_2 be defined symmetrically. See Fig. 6. Then we can rewrite the certificate for ep_1 as

$$\pi(\mathrm{ep}_1, s_1)+\pi(s_1, p) < \pi(\mathrm{ep}_1, t_1)+\pi(t_1, q) \quad \equiv \quad \pi(\mathrm{ep}_1, s_1)-\pi(\mathrm{ep}_1, t_1) < \pi(t_1, q)-\pi(s_1, p),$$

and the certificate for ep_2 similarly. Then observe that if $s_1 = s_2$ and $t_1 = t_2$, then ep_1 will be on the bisector before ep_2 if and only if $\pi(\mathrm{ep}_1, s_1) - \pi(\mathrm{ep}_1, t_1) > \pi(\mathrm{ep}_2, s_2) - \pi(\mathrm{ep}_2, t_2)$. This creates a strict ordering of the event points in the Voronoi cell of p. Unfortunately in many cases the first vertex on the path towards p or q will not be the same for every vertex on the bisector. Therefore we introduce an offset value to allow comparing event points that have different first vertices on their paths towards p and q.

If $s_1 \neq s_2$ and $t_1 \neq t_2$, we should compare based on a common node on the paths towards p and q, which may be any combination of s_1 or s_2 and t_1 or t_2. As these cases are analogous, we consider the case where s_1 and t_2 are on the shortest paths towards p and q respectively for both event points. (Intuitively s_1 and t_2 are further towards p and q). Now the values we would like to compare are $\pi(\mathrm{ep}_1, s_1) - \pi(\mathrm{ep}_1, t_2) > \pi(\mathrm{ep}_2, s_1) - \pi(\mathrm{ep}_2, t_2)$. However these are not what we stored. With some rewriting, we find that the above inequality holds if and only if

$$\pi(\mathrm{ep}_1, s_1) - \pi(\mathrm{ep}_1, t_1) > \pi(\mathrm{ep}_2, s_2) - \pi(\mathrm{ep}_2, t_2) - \pi(s_1, s_2) - \pi(t_1, t_2).$$

We call $-\pi(s_1, s_2) - \pi(t_1, t_2)$ the offset of ep_2 with respect to ep_1. Each node will store the maximum event value in its subtree as follows. For a leaf the maximum is its own event value. For an internal node, it is the maximum over its own event value and the maximum values of its children with the offset added. The maximum value of the root can then be used to determine the first time an $2, 2$-event happens among the bisector vertices stored in the tree.

Note that the above data structure stores only a constant number of certificates directly involving p or q, all of which are stored at the root of the tree. Therefore, it can be made to support changes in the movement of p and q in $O(\log m)$ time. Furthermore, we can support splitting the bisector at a vertex in $O(\log^2 m)$ time, since a split affects $O(\log m)$ nodes in the balanced binary search tree, and recomputing the offsets (and thus updating the certificates) takes $O(\log m)$ time per node.

By replacing part (iii) of the naive structure with this data-structure we are still guaranteed to detect all events, but now when SPM_p changes, we have to update only a constant number of certificates (rather than $\Theta(m)$). As the certificates are stored in a binary tree it is easy to add or remove vertices when the bisector is expanded or shrinks. This proves the theorem. ◀

4 A Voronoi Center

Let $c_{pqs}(t)$ be the point equidistant to $p(t)$, $q(t)$, and $s(t)$ if it exists. By Aronov et al. [4] (Lemma 2.3.5) there is indeed at most one such a point. We refer to c_{pqs} as the *Voronoi center* of p, q, and s. Note that there may be times at which c_{pqs} does not exist. We identify five types of events at which c_{pqs} may appear or disappear, or at which the movement of c_{pqs} can change (see Fig. 7). They are:

- $1,3$-*collapse* events in which c_{pqs} collides with the boundary of the polygon (in a bisector endpoint) and disappears from P,
- $1,3$-*expand* events in which c_{pqs} appears on the boundary of P as two bisector endpoints intersect, creating a point equidistant to all three sites,
- *vertex-events* where c_{pqs} appears or disappears strictly inside P, as two sites, say p and q, are equidistant to a vertex v that appears on the shortest paths to c_{pqs},
- $2,3$-*collapse* events where one of the geodesics from either p, q, or s to c_{pqs} loses a vertex,
- $2,3$-*expand* events where one of the shortest paths gains a new vertex.

Observe that, as the name suggests, at a $1,3$-collapse event the Voronoi center (a degree 3 vertex in $\mathrm{VD}_P(\{p,q,s\})$) disappears as it collides with the endpoint of a bisector (a degree 1 vertex). Similarly, at a $2,3$-collapse event a degree 2 vertex on one of the bisectors disappears as it collides with a degree 3 vertex (the Voronoi center c_{pqs}). As in case of the bisector, some of these events may coincide. In the next section, we bound the number of events, and thus the complexity of the trajectory of c_{pqs}. We then present a kinetic data structure to maintain c_{pqs} in Section 4.2.

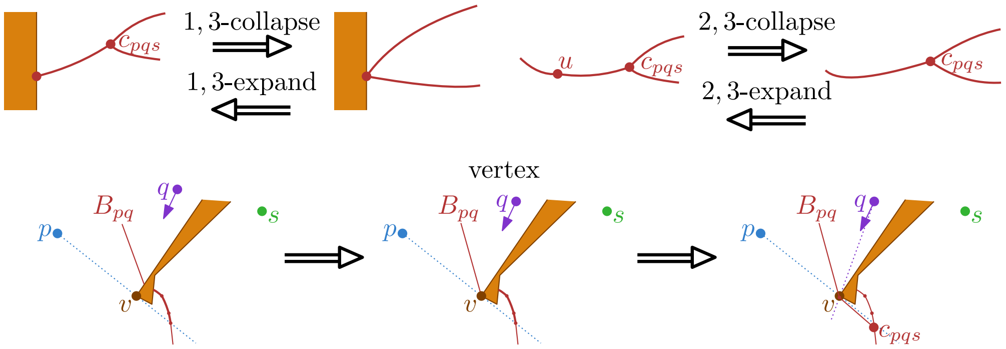


Figure 7 The events that can happen during the movement of a Voronoi center.

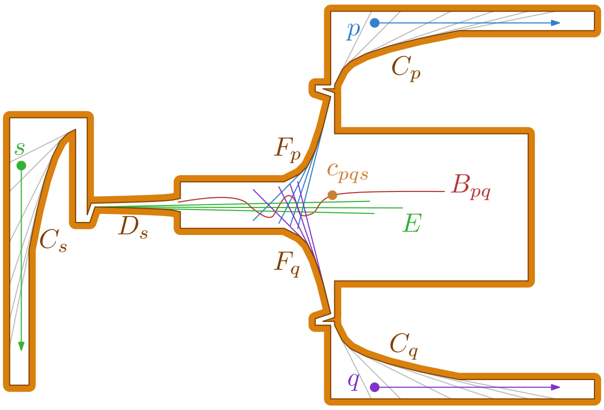


Figure 8 A polygon in which the trajectory of a voronoi center c_{pqs} has complexity $\Omega(m^3)$.

4.1 Bounding the Number of Events

We give a construction in which the trajectory of c_{pqs} has complexity $\Omega(m^3)$, and then prove a matching upper bound.

▶ **Lemma 7.** *The trajectory of the Voronoi center c_{pqs} of three points p, q, and s, each moving linearly, may have complexity $\Omega(m^3)$.*

Proof. The main idea is that we can construct a trajectory for c_{pqs} of complexity $\Omega(m^2)$, even when two of the three sites, say p and q, are static. We place p and q so that their bisector B_{pq}, a piecewise hyperbolic curve of complexity $\Omega(m)$, intersects an (almost) horizontal line E $\Omega(m)$ times. We can realize this using two convex chains F_p and F_q in ∂P. See Fig. 8 for an illustration. We now construct a third convex chain D_s in ∂P and place the third site s so that the extension segments in SPM$_s$ incident to the vertices of D_s all lie very close to E. Thus, each such segment intersects B_{pq} $\Omega(m)$ times. We choose the initial distances so that the voronoi center c_{pqs} lies on the rightmost segment of B_{pq}. Now observe that as s moves away from D_s, the center $c_{pqs}(t)$ will move to the left on B_{pq}, and thus it will pass over all $\Omega(m^2)$ intersection points of B_{pq} with the extension segments of the vertices in D_s. At each such time, the structure of one of the shortest paths $\pi(p(t), c_{pqs}(t))$, $\pi(q(t), c_{pqs}(t))$, or $\pi(s(t), c_{pqs}(t))$ changes (they gain or lose a vertex from F_p, F_q, or D_s, respectively). Hence, the trajectory of c_{pqs} changes $\Omega(m^2)$ times.

Next, we argue that we can make c_{pqs} "swing" back and forth $\Omega(m)$ times by having p and q move as well. The voronoi center c_{pqs} will then encounter every intersection point on B_{pq} $\Omega(m)$ times. It follows that the complexity of the trajectory of c_{pqs} is $\Omega(m^3)$ as claimed.

The idea is to add two additional convex chains, C_s and C_p, that make the bisector B_{ps} between p and s "zigzag" $\Omega(m)$ times throughout the movement of p and s. We can achieve this using a similar construction as in Lemma 4. To make sure that the bisector $B_{pq} = B_{pq}(t)$ between p and q remains static, we create a third chain C_q, which is a mirrored copy of C_p, and we make q move along a trajectory identical to that of p. See Fig. 8. Finally, observe that $c_{pqs}(t) = B_{pq} \cap B_{ps}(t)$, and thus $c_{pqs}(t)$ will indeed encounter all $\Omega(m^2)$ intersection points on B_{pq} $\Omega(m)$ times. The lemma follows. ◀

▶ **Lemma 8.** *The number of $1,3$-collapse events is at most $O(m^2)$.*

▶ **Theorem 9.** *The trajectory of the Voronoi center c_{pqs} has complexity $O(m^3)$. Each edge is a constant degree algebraic curve.*

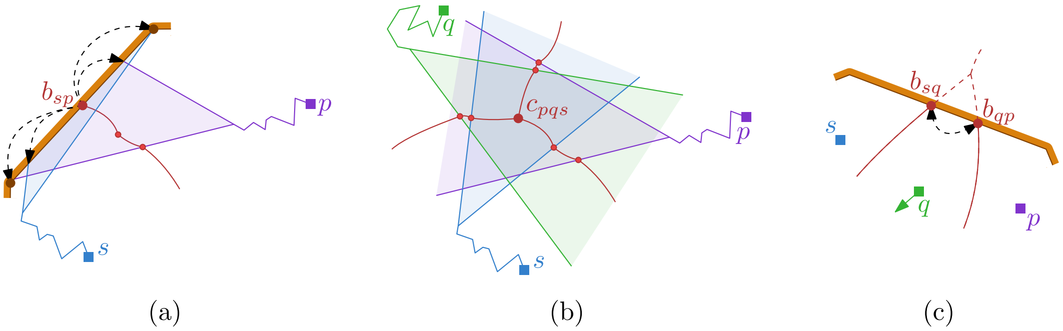


Figure 9 In black the certificates that we maintain in order to detect: (a) events where b_{sp} changes movement, (b) $1,3$-collapse, $2,3$-collapse and $2,3$-expand events, and (c) $1,3$-expand events.

4.2 A Kinetic Data Structure to Maintain a Voronoi Center

Our KDS for maintaining c_{pqs} stores: (i) the extended shortest path maps of p, q, and s, (ii) the cells of these shortest path maps containing c_{pqs} (when c_{pqs} lies inside P), and (iii) the endpoints of all bisectors (for all pairs), and their cyclic order on ∂P. In particular, for each such endpoint b_{sp} we keep track of the cells of SPM_p and SPM_s that contain it. See Fig. 9. At any time we maintain $O(m)$ certificates, which we store in a global priority queue.

Observe that at $1,3$-collapse, $2,3$-collapse, and $2,3$-expand events the shortest path from c_{pqs} to one of the sites changes combinatorially. Hence, we can detect all such events. At a vertex event a vertex is equidistant to two sites, say p and q. At such a time, one of the two endpoints of B_{pq} leaves an edge of P, and thus exits a shortest path map cell in SPM_p (and SPM_q). Since we explicitly track all bisector endpoints, we can thus detect this vertex event of c_{pqs}. Finally, at every $1,3$-expand event two such bisector endpoints collide, and thus change their cyclic order along ∂P. We detect such events due to certificates of type (iii).

Any time at which c_{pqs} changes cells in a shortest path map the movement of c_{pqs} changes combinatorially. Hence, any failure of a certificate of type (ii) is an external event (a $1,3$-collapse, $2,3$-collapse, or $2,3$-expand). The certificates of types (i) and (iii) may be internal or external.

▶ **Theorem 10.** *Let p, q and s be three sites moving linearly inside a simple polygon P with m vertices. There is a KDS that maintains the Voronoi center c_{pqs} that uses $O(m)$ space and processes at most $O(m^3)$ events, each of which can be handled in $O(\log^2 m)$ time. Updates to the movement of p, q, and s, can be handled in $O(\log^2 m)$ time.*

Proof. Certificate failures of type (i) are handled exactly as described by Aronov et al. [5]. This takes $O(\log^2 m)$ time. Note that changes to the shortest path maps may affect the certificates that guarantee that c_{pqs} or a bisector endpoint lies in a particular SPM cell. In these cases we trigger a type (ii) or type (iii) certificate failure. At a certificate failure of type (ii) at which c_{pqs} exits a shortest path map cell, we remove all certificates of type (ii) from the event queue. Next, for each site p, q, and s, we compute the new cell in the shortest path map containing c_{pqs} (if c_{pqs} still lies inside P). Finally, we create the appropriate new type (ii) certificates. Since all cells have constant complexity, the total number of certificates affected is also $O(1)$. Computing them can easily be done in $O(\log^2 m)$ time.

Certificate failures of type (iii) where the movement of a bisector endpoint changes are handled using the same approach as in Section 3.2. Furthermore, at such an event we check if c_{pqs} appears or disappears, that is, if the event is actually a vertex event of c_{pqs}. This can be done in $O(\log^2 m)$ time [21]. If c_{pqs} disappears then we delete all type (ii) certificates.

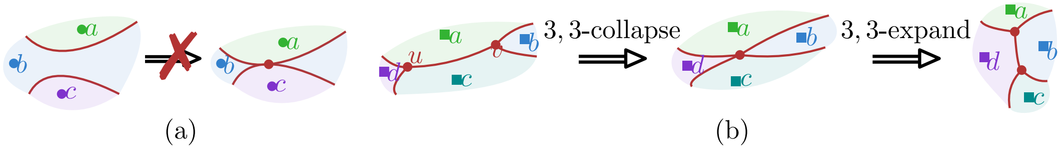


Figure 10 (a) Voronoi edges cannot intersect in their interior. (b) The 3, 3-collapse/expand events.

If c_{pqs} appears then we locate the cell of SPM_p, of SPM_q, and of SPM_s that contains c_{pqs}, and insert new type (ii) certificates that certify this. Finding the cells and updating the certificates can be done in $O(\log^2 m)$ time. At a certificate failure of type (iii) where two bisector endpoints collide, we check if the intersection point is equidistant to all three sites, and is thus a 1, 3-expand event. Similarly to the approach described above, we add new type (ii) certificates in this case. Again this takes $O(\log^2 m)$ time.

Maintaining the extended shortest path maps requires handling $O(m)$ events [5]. Events where c_{pqs} crosses a boundary of an extended SPM correspond to changes in the trajectory of c_{pqs}. By Theorem 9 there are at most $O(m^3)$ such events. This dominates the $O(m^2)$ events that we have to handle to maintain the bisector endpoints in cyclic order around ∂P (Theorem 5 and Lemma 8).

Since in addition to SPM_p, SPM_q, and SPM_s, we maintain only a constant amount of extra information. Since the KDS to maintain such a shortest path map SPM_s is local and can be updated to changes in the movement of s in $O(\log^2 m)$ time. The same applies for our data structure as well. Thus we obtain a compact, responsive, local, and efficient KDS. ◀

5 The Geodesic Voronoi Diagram

In this section we consider maintaining the geodesic Voronoi diagram $\mathrm{VD}_P(S)$ as the sites in S move. As a result of the sites in S moving, the Voronoi vertices and edges in $\mathrm{VD}_P(S)$ will also move. However, we observe that all events involving Voronoi edges involve their endpoints; two edges cannot start to intersect in their interior as this would split a Voronoi region, see Fig. 10(a). Similarly, the interior of a Voronoi edge cannot start to intersect the polygon boundary. This means we can distinguish the following types of events that change the combinatorial structure of the Voronoi diagram.

- Edge collapses, at which an edge between vertices u and v shrinks to length zero. Let d_u, d_v, with $d_u \leq d_v$, be the degrees of u and v, respectively. We then have a d_u, d_v*-collapse.*
- Edge expands. These are symmetric to edge collapses.
- Vertex events, where a degree 1 vertex of $\mathrm{VD}_P(S)$ crosses over a polygon vertex.

Indeed, we have seen most of these events when maintaining an individual bisector or Voronoi center (a degree 3 vertex in $\mathrm{VD}_P(S)$). The only new types of events are the 3, 3-collapse and 3, 3-expand events which involve two degree 3 vertices. They are depicted in Fig. 10.(b). We again note that some of these events may happen simultaneously.

▶ **Theorem 11.** *Let S be a set of n sites moving linearly inside a simple polygon P with m vertices. During the movement of the sites in S, the combinatorial structure of the geodesic Voronoi diagram $\mathrm{VD}_P(S)$ changes at most $O(m^3 n^3 \beta_z(n))$ times. In particular, the events at which $\mathrm{VD}_P(S)$ changes, and the number of such events, are listed in Table 1.*

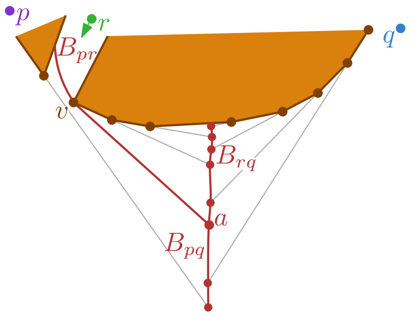


Figure 11 A vertex event may cause a bisector (here B_{pq}) to split, and a degree 3 vertex crossing an SPM extension segment may merge two bisectors.

We prove these bounds in the full version [15]. For most of the lower bounds we generalize the constructions from Sections 3 and 4. For the upper bounds we typically fix a site or vertex (or both), and map the remaining sites to a set of functions in which we are interested in the lower envelope. In Section 5.1 we develop a kinetic data structure to maintain $\mathrm{VD}_P(S)$.

5.1 A KDS for a Voronoi Diagram

In this section we develop a KDS to maintain the Voronoi diagram of S. Our KDS essentially stores for each site the extended shortest path map of its Voronoi cell, and a collection of certificates that together guarantee that the shortest paths from the sites to all Voronoi vertices remain the same (and thus the KDS correctly represents $\mathrm{VD}_P(S)$). The main difficulties that we need to deal with are shown in Fig. 11. Here, r becomes the site closest to vertex v, and as a result a part of the polygon moves from the Voronoi cell V_p of p to the Voronoi cell V_r of r. Our KDS should therefore support transplanting this region from the SPM representation of V_p into V_r or vice versa. Moreover, part of the bisector B_{pq} becomes a bisector B_{pr}, which means that any certificates internal to the bisector (such as those needed to detect 2,2-events) change from being dependent on the movement of p to being dependent on the movement of r. Next, we show how to solve the first problem, transplanting part of the shortest path map. Our KDS for the bisector from Theorem 6 essentially solves the second problem. All that then remains is to describe how to handle each event.

5.1.1 Maintaining Partial Shortest Path Maps

To support transplanting a part of SPM_s into SPM_q we extend the data structure of Aronov et al. [5]. Observe that SPM_s is a tree rooted at s, and we transplant only subtrees, rooted at some polygon vertex v. Our representation of SPM_s should support: (i) link operations in which we add the subtree rooted at v as a child of u, (ii) cut operations in which we cut an edge (u, v), (iii) shortest path queries in which we report the length of the shortest path from some vertex u to the root s, and (iv) principal-child queries in which we report the *principal child* c of some non-root node u. The principal child is the child of u for which the angle between $\overline{cu}$ and $\overline{up(u)}$, where $p(u)$ is the parent of u, is minimal. We need this operation to support updating the certificates of SPM_s[2]. To support these operations,

[2] Since the root is the only node storing a moving point, all certificates involve only nodes from the first three layers of the tree. Hence, it suffices to compute the principal child only for direct children of the root.

we store SPM_s twice: once in a link-cut tree [26] and once in an Euler tour tree [12]. Both these structures support link and cut operations in $O(\log m)$ time. The link-cut trees support query operations on node-to-root paths, and hence we use them to answer shortest path queries in $O(\log m)$ time (plus $O(k)$ time to report the actual path, if desired). The Euler tour trees support query operations on subtrees, and hence we use them to answer principal child queries. In particular, we maintain the children of u in cyclic order around u, starting with c. This way link and cut operations still take $O(\log m)$ time, and the principal child of u can be reported in constant time.

5.1.2 The data structure

The full KDS thus consists of an extended shortest path map for every Voronoi cell maintained as described above; and certificates for each degree three vertex, degree one vertex, and each bisector. For every degree three vertex c_{pqs} we maintain the cells of SPM_p, SPM_q and SPM_s that contain it and its distance to neighboring vertices. For every degree one vertex b_{pq}, we store the cells of SPM_p and SPM_q that contain it, which edge of P it is on, and if applicable its separation from neighboring degree one vertices on the same edge. For each bisector, we store the data structure of Theorem 6. Our data structure uses a total of $O(n + m)$ space.

It is not to difficult to see that this certificate structure captures all external events. For collapse and expand events involving degree three vertices we explicitly certify that the distance to its adjacent vertices is non-zero. For events involving degree one vertices we explicitly track which edge contains each such a vertex. This allows us to detect vertex events. Furthermore, we maintain distance of each degree one vertex to other degree one vertices on the same edge. Thus we can detect 1,3-expand events. Furthermore, we maintain which cells of the SPM the vertex is contained in, which allows us to detect 1,2-expand and $1, 2$-collapse events. What remains are the 2,2-events. These are detected by the data structure of Theorem 6.

5.1.3 Handling events

Handling the events is similar to what we described in Sections 3.2 and 4.2. Hence, we describe only what is new or different here.

At all external-events we have to update the shortest path map representations of the Voronoi cells. In most cases, this involves adding or removing a single vertex to the shortest path map. This can easily be handled using local computations in $O(\log^2 m)$ time. In case of vertex events, we may have to move an entire region in SPM_s to SPM_p. Since all shortest paths in such a region go via the vertex involved, we can perform these updates in $O(\log^2 m)$ time using the above data structure.

Since there are now n sites, we maintain $O(n + m)$ certificates, and thus updating the event queue takes $O(\log(n + m))$ time. Furthermore, we now have multiple degree three vertices, and thus we have to handle $3, 3$-collapse and expand events. These are handled in a similar fashion to the other events; we update the Voronoi regions, and compute new certificates certifying the movement of the vertices involved from scratch. All these updates can be done in $O(\log^2 m + \log n)$ time.

At a vertex event where p and r are equidistant to a vertex v, the region R that moves from SPM_p to SPM_r may now be bounded by a bisector B_{rq} rather than B_{pq} (see Fig. 11). Since, at the time of the event, the relevant parts of B_{pq} and B_{rq} coincide we can obtain the new part of B_{rq} by splitting B_{pq}, and updating the movement of the associated sites. In particular, replacing the function expressing the distance p to v by the distance from r to v. Our bisector KDS allows such updates in $O(\log^2 m)$ time.

Finally, we may have to update the certificates associated with the Voronoi vertices as a result of changes to the individual shortest path maps. For example, when a site s can no longer see polygon vertex v, this affects all Voronoi certificates of vertices for which the shortest path goes through v. While our KDS for the bisector (Theorem 6) can update the affected certificates of such a change efficiently, this unfortunately does not hold for the certificates associated with degree one or degree three vertices. Updating these requires $O(k(\log^2 m + \log n))$ time, where k is the number of neighbors of s in $\mathrm{VD}_S(P)$. It is an interesting open question to try and handle such events implicitly as well. We therefore obtain the following result:

▶ **Theorem 12.** *Let S be a set of n sites moving linearly inside a simple polygon P with m vertices. There is a KDS that maintains the geodesic Voronoi diagram $\mathrm{VD}_P(S)$ that uses $O(n+m)$ space and processes at most $O(m^3 n^3 \beta_z(n))$ events, each of which can be handled in $O(k(\log^2 m + \log n))$ time, where k is the number of neighbors of the affected Voronoi cell.*

References

1 Pankaj K. Agarwal, Lars Arge, and Frank Staals. Improved dynamic geodesic nearest neighbor searching in a simple polgyon. In *Proceedings of the 34th Annual Symposium on Computational Geometry*, volume 99 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 4:1–4:14. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.SoCG.2018.4.

2 Pankaj K. Agarwal, Haim Kaplan, Natan Rubin, and Micha Sharir. Kinetic voronoi diagrams and delaunay triangulations under polygonal distance functions. *Discrete & Computational Geometry*, 54(4):871–904, December 2015. doi:10.1007/s00454-015-9729-3.

3 Boris Aronov. On the Geodesic Voronoi Diagram of Point Sites in a Simple Polygon. *Algorithmica*, 4(1):109–140, 1989.

4 Boris Aronov, Steven Fortune, and Gordon Wilfong. The furthest-site geodesic voronoi diagram. *Discrete & Computational Geometry*, 9(3):217–255, March 1993.

5 Boris Aronov, Leonidas J. Guibas, Marek Teichmann, and Li Zhang. Visibility queries and maintenance in simple polygons. *Discrete & Computational Geometry*, 27(4):461–483, January 2002. doi:10.1007/s00454-001-0089-9.

6 Julien Basch, Leonidas J. Guibas, and John Hershberger. Data structures for mobile data. *J. Algorithms*, 31(1):1–28, 1999. doi:10.1006/jagm.1998.0988.

7 G. S. Brodal and R. Jacob. Dynamic planar convex hull. In *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.*, pages 617–626, November 2002. doi:10.1109/SFCS.2002.1181985.

8 Erik D. Demaine, Joseph S. B. Mitchell, and Joseph O'Rourke. The open problems project. http://maven.smith.edu/~orourke/TOPP/.

9 J. E. Goodman and J. O'Rourke. *Handbook of Discrete and Computational Geometry.* CRC Press series on discrete mathematics and its applications. Chapman & Hall/CRC, 2004.

10 Leonidas Guibas, John Hershberger, Daniel Leven, Micha Sharir, and Robert E. Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2(1):209–233, November 1987. doi:10.1007/BF01840360.

11 Leonidas J. Guibas, Joseph S. B. Mitchell, and Thomas Roos. Voronoi diagrams of moving points in the plane. In *Graph-Theoretic Concepts in Computer Science*, pages 113–125, Berlin, Heidelberg, 1992. Springer.

12 Monika R. Henzinger, Valerie King, and Valerie King. Randomized fully dynamic graph algorithms with polylogarithmic time per operation. *Journal of the ACM*, 46(4):502–516, July 1999. doi:10.1145/320211.320215.

13 John Hershberger and Subhash Suri. An Optimal Algorithm for Euclidean Shortest Paths in the Plane. *SIAM Journal on Computing*, 28(6):2215–2256, 1999.

14 Menelaos I. Karavelas and Leonidas J. Guibas. Static and kinetic geometric spanners with applications. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '01, pages 168–176, Philadelphia, PA, USA, 2001. Society for Industrial and Applied Mathematics.

15 Matias Korman, André van Renssen, Marcel Roeloffzen, and Frank Staals. Kinetic geodesic voronoi diagrams in a simple polygon. *CoRR*, abs/2002.05910, 2018. `arXiv:2002.05910`.

16 Chih-Hung Liu. A Nearly Optimal Algorithm for the Geodesic Voronoi Diagram of Points in a Simple Polygon. In *34th International Symposium on Computational Geometry (SoCG 2018)*, volume 99 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 58:1–58:14, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. `doi:10.4230/LIPIcs.SoCG.2018.58`.

17 Anna Lubiw, Jack Snoeyink, and Hamideh Vosoughpour. Visibility graphs, dismantlability, and the cops and robbers game. *Computational Geometry*, 66:14–27, 2017. `doi:10.1016/j.comgeo.2017.07.001`.

18 J. S. B. Mitchell. Shortest paths among obstacles in the plane. In *Proceedings of the 9th Annual ACM Symposium on Computational Geometry*, pages 308–317, 1993.

19 Joseph S. B. Mitchell. A new algorithm for shortest paths among obstacles in the plane. *Annals of Mathematics and Artificial Intelligence*, 3(1):83–105, March 1991. `doi:10.1007/BF01530888`.

20 Eunjin Oh. Optimal algorithm for geodesic nearest-point voronoi diagrams in simple polygons. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 391–409. SIAM, 2019. `doi:10.1137/1.9781611975482.25`.

21 Eunjin Oh and Hee-Kap Ahn. Voronoi diagrams for a moderate-sized point-set in a simple polygon. *Discrete & Computational Geometry*, March 2019. `doi:10.1007/s00454-019-00063-4`.

22 Eunjin Oh, Jean-Lou De Carufel, and Hee-Kap Ahn. The 2-center problem in a simple polygon. In *Algorithms and Computation*, pages 307–317, Berlin, Heidelberg, 2015. Springer.

23 Evanthia Papadopoulou and Der-Tsai Lee. A New Approach for the Geodesic Voronoi Diagram of Points in a Simple Polygon and Other Restricted Polygonal Domains. *Algorithmica*, 20(4):319–352, 1998.

24 Natan Rubin. On kinetic delaunay triangulations: A near-quadratic bound for unit speed motions. *Journal of the ACM*, 62(3):25:1–25:85, June 2015. `doi:10.1145/2746228`.

25 Micha Sharir and Pankaj K Agarwal. *Davenport-Schinzel sequences and their geometric applications.* Cambridge university press, 1995.

26 Daniel D. Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26(3):362–391, 1983. `doi:10.1016/0022-0000(83)90006-5`.

27 Subhash Suri. A linear time algorithm with minimum link paths inside a simple polygon. *Computer Vision, Graphics and Image Processing*, 35(1):99–110, 1986. `doi:10.1016/0734-189X(86)90127-1`.

Polytopes, Lattices, and Spherical Codes for the Nearest Neighbor Problem

Thijs Laarhoven
Eindhoven University of Technology, The Netherlands
http://www.thijs.com/
mail@thijs.com

Abstract

We study locality-sensitive hash methods for the nearest neighbor problem for the angular distance, focusing on the approach of first projecting down onto a random low-dimensional subspace, and then partitioning the projected vectors according to the Voronoi cells induced by a well-chosen spherical code. This approach generalizes and interpolates between the fast but asymptotically suboptimal hyperplane hashing of Charikar [STOC 2002], and asymptotically optimal but practically often slower hash families of e.g. Andoni–Indyk [FOCS 2006], Andoni–Indyk–Nguyen–Razenshteyn [SODA 2014] and Andoni–Indyk–Laarhoven–Razenshteyn–Schmidt [NIPS 2015]. We set up a framework for analyzing the performance of any spherical code in this context, and we provide results for various codes appearing in the literature, such as those related to regular polytopes and root lattices. Similar to hyperplane hashing, and unlike e.g. cross-polytope hashing, our analysis of collision probabilities and query exponents is *exact* and does not hide any order terms which vanish only for large d, thus facilitating an easier parameter selection in practical applications.

For the two-dimensional case, we analytically derive closed-form expressions for arbitrary spherical codes, and we show that the equilateral triangle is optimal, achieving a better performance than the two-dimensional analogues of hyperplane and cross-polytope hashing. In three and four dimensions, we numerically find that the tetrahedron and 5-cell (the 3-simplex and 4-simplex) and the 16-cell (the 4-orthoplex) achieve the best query exponents, while in five or more dimensions orthoplices appear to outperform regular simplices, as well as the root lattice families A_k and D_k in terms of minimizing the query exponent. We provide lower bounds based on spherical caps, and we predict that in higher dimensions, larger spherical codes exist which outperform orthoplices in terms of the query exponent, and we argue why using the D_k root lattices will likely lead to better results in practice as well (compared to using cross-polytopes), due to a better trade-off between the asymptotic query exponent and the concrete costs of hashing.

2012 ACM Subject Classification Theory of computation → Nearest neighbor algorithms; Theory of computation → Random projections and metric embeddings; Theory of computation → Computational geometry

Keywords and phrases (approximate) nearest neighbor problem, spherical codes, polytopes, lattices, locality-sensitive hashing (LSH)

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.76

Category Track A: Algorithms, Complexity and Games

Related Version A full version of the paper is available at https://arxiv.org/abs/1907.04628.

Funding The author is supported by an NWO Veni grant with project number 016.Veni.192.005. Part of this work was done while the author was visiting the Simons Institute for the Theory of Computing at the University of California, Berkeley.

1 Introduction

Given a large database of high-dimensional vectors, together with a target data point which does not lie in this database, a natural question to ask is: which item in the database is the

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 76; pp. 76:1–76:14
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

most similar to the query? And can we somehow preprocess and store the database in a data structure that allows such queries to be answered faster? These and related questions have long been studied in various contexts, such as machine learning, coding theory, pattern recognition, and cryptography [14,19,20,25,30,32], under the headers of *similarity search* and *nearest neighbor searching.* Observe that a naive solution might consist of simply storing the data set in a big list, and to search this list in linear time to find the nearest neighbor to a query point. This solution requires an amount of time and space scaling linearly in the size of the data set, and solutions we are interested in commonly require more preprocessed space (and time), but achieve a sublinear query time complexity to find the nearest neighbor.

Depending on the context of the problem, different solutions for these problems have been proposed and studied. For the case where the dimensionality of the original problem is constant, efficient solutions are known to exist [8]. Throughout the remainder of the paper, we will therefore assume that the dimensionality of the data set is superconstant. In the late 1990s, Indyk–Motwani [22] proposed the *locality-sensitive hashing* framework, and until today this method remains one of the most prominent and popular methods for nearest neighbor searching in high-dimensional vector spaces, both due to its asymptotic performance when using theoretically optimal hash families [4,5], and due to its practical performance when instantiated with truly efficient locality-sensitive hash functions [1,13,16]. And whereas many other methods scale poorly as the dimensionality of the problem increases, locality-sensitive hashing remains competitive even in high-dimensional settings.

Although solutions for both asymptotic and concrete settings have been studied over the years, there is often a separation between both worlds: some methods work well in practice but do not scale optimally when the parameters increase (e.g. Charikar's hash family [16]); while some other methods are known to achieve a superior performance for sufficiently large parameter sizes, but may not be quite as practical in some applications due to large hidden order terms in the asymptotic analysis (e.g. hash families studied in [3–5] and filter families from [6,11,17]). Moreover, the latter methods are often not as easy to deploy in practice due to these unspecified hidden order terms, making it hard to choose the scheme parameters that optimize the performance. A key problem in this area thus remains to close the gap between theory and practice, and to offer solutions that interpolate between *quick–and–dirty* simple approaches that might not scale well with the problem size, and more sophisticated methods that only start outperforming these simpler methods as the parameters are sufficiently large.

1.1 Related work

In this paper we will focus on methods for solving the nearest neighbor problem for the *angular distance*, which is closely related to the nearest neighbor problem in various ℓ_p-norms: as shown by Andoni and Razenshteyn [7], a solution for the nearest neighbor problem for the angular distance (or the Euclidean distance on the sphere) can be optimally extended to a solution for the ℓ_2-norm for all of $\mathbb{R}^d$. Solutions for the ℓ_2-norm can further be translated to e.g. solutions for the ℓ_1-norm via appropriate embeddings.

For the angular distance, perhaps the most well-known and widely deployed approach for finding similar items in a large database is to use the hyperplane hash family of Charikar [16]. For spherically-symmetric, random data sets on the sphere of size n, it can find a nearest neighbor at angle at most e.g. $\theta = \frac{\pi}{3}$ in sublinear time $\tilde{O}(n^\rho)$ and space $\tilde{O}(n^{1+\rho})$, with $\rho \approx 0.5850$. Various improvements later showed that smaller query exponents ρ can be achieved in sufficiently high dimensions [3,5], the most practical of which is based on cross-polytopes or orthoplices [4,24,35]: for large d and for the same target angle $\theta = \frac{\pi}{3}$, the query time complexity scales as $n^{\rho+o(1)}$ with $\rho = 1/3$. Note that the convergence to the limit is rather slow [4, Theorem 1] and depends on both d and n being sufficiently large – for fixed d and large n, the exponent is still larger than $1/3$. In certain practical applications with moderate-sized data sets, hyperplane hashing still appears to outperform cross-polytope hashing and other advanced hashing and filtering schemes [2, 11, 12, 28, 29] due to its low cost for hashing, and the absence of lower order terms in the exponent.

Related to the topic of this paper, various other works have further observed the relation between finding good locality-sensitive hash families for the angular distance and finding "nice" spherical codes that partition the sphere well, and allow for efficient decoding [3, 4, 11, 15, 35, 36]. The requirements on a spherical code to be a *good* spherical code are somewhat intricate to state, and to date it is an open problem to exactly quantify which spherical codes are the most suited for nearest neighbor searching. It may well be possible to improve upon the state-of-the-art orthoplex (cross-polytope) locality-sensitive hash family [4, 31] in practice with a method achieving the same asymptotic scaling, but with a faster convergence to the limit, and thus potentially a better performance in practice for large problem instances.

1.2 A framework for evaluating spherical codes

As a first contribution of this paper, we provide a framework for analyzing the performance of arbitrary spherical codes in the context of locality-sensitive hashing for the angular distance, where we focus on the approach of (1) projecting down onto a random low-dimensional subspace, and (2) partitioning the resulting subspace according to the Voronoi cells induced by a spherical code. More specifically, we relate the collision probabilities appearing in the analysis of these hash functions to so-called *orthant probabilities* of multivariate normal distributions with non-trivial correlation matrices. Below we informally state this relation for general spherical codes, which provides us a recipe for computing the collision probabilities (and the query exponent) as a function of the set of vertices $\mathcal{C}$ of the corresponding spherical code. Here a hash family being (θ, p_1, p_2)-sensitive means that uniformly random vectors on the sphere collide with probability at most p_2, and target nearest neighbors at angle at most θ from a random query vector collide in a random hash function with probability at least p_1. Further details and a more formal statement can be found in the full version.

▶ **Theorem 1** (Spherical code locality-sensitive hashing). *Let $\mathcal{C} \subset \mathcal{S}^{k-1}$ be a k-dimensional spherical code, and consider a hash family where:*

- *We first project onto a k-dimensional subspace using a random matrix $\mathbf{A} \sim \mathcal{N}(0,1)^{k \times d}$;*
- *We then assign hash values based on which $\mathbf{c} \in \mathcal{C}$ is nearest to the projected vector.*

Then for any $\theta \in (0, \frac{\pi}{2})$, this family is (θ, p_1, p_2)-sensitive, where p_2 can be expressed in terms of the relative volumes of the Voronoi cells induced by $\mathcal{C}$, and p_1 can be expressed as a sum of orthant probabilities $\Pr_{\boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_i)}(\boldsymbol{z} \geq \mathbf{0})$, where each $\boldsymbol{\Sigma}_i$ has size $(2k) \times (2k)$.

The locality-sensitive hashing exponent $\rho = \log p_1 / \log p_2$ describes the distinguishing power of a hash family, as for large n this tells us that we can solve the (average-case) nearest neighbor problem with target angle θ in query time $\tilde{O}(n^\rho)$ with space $\tilde{O}(n^{1+\rho})$. The theorem above essentially provides us a recipe which, given any spherical code $\mathcal{C}$ and target angle θ as input, tells us how to compute these probabilities p_1 and p_2 (and thus ρ) exactly.

While the above theorem is somewhat abstract, we show how to explicitly construct the correlation matrices $\boldsymbol{\Sigma}_i$ appearing in the theorem, allowing us to always at least obtain numerical estimates on the performance of different spherical codes in the context of nearest neighbor searching. We further study how to reduce the dimensionality of the problem (and in particular, the sizes of the matrices $\boldsymbol{\Sigma}_i$) when the spherical code exhibits many symmetries, such as being isogonal, and we show how using only the *relevant vectors* of each code word can further simplify the computations. In most cases the resulting orthant probabilities will still remain too complex to evaluate analytically, but in some cases we can obtain exact expressions this way.

1.3 A survey of known spherical codes

One and two dimensions. Using this new framework, in the full version we then apply it to various spherical codes, starting in very low dimensions. For the one-dimensional case we rederive the celebrated result of Charikar [16] for one-dimensional spherical codes, noting that the collision probability analysis in fact dates back to an old result from the late 1800s [33]. Applying the same framework to two-dimensional codes, among others we establish a general formula for collision probabilities and query exponents for arbitrary polygons, as well as a more compact description of the collision probabilities for regular polygons.

▶ **Theorem 2** (Collision probabilities for regular polygons). *Let $\mathcal{C} \subset \mathcal{S}^1$ consist of the vertices of the regular c-gon, for $c \geq 2$, and let $\theta \in (0, \frac{\pi}{2})$. Then the corresponding project–and–partition hash family $\mathcal{H}$ is (θ, p_1, p_2)-sensitive, with:*

$$p_1 = \frac{1}{c} + c\left(\frac{\pi - \theta}{2\pi}\right)^2 - c\left(\frac{\arccos(-\cos\theta\cos\frac{2\pi}{c})}{2\pi}\right)^2, \qquad p_2 = 1/c. \tag{1}$$

As a direct corollary of the above theorem, we establish that *triangular locality-sensitive hashing* achieves a superior asymptotic performance to hyperplane hashing, and achieves the lowest query exponents ρ among all regular polygons.

▶ **Corollary 3** (Triangular locality-sensitive hashing). *Consider the hash family where we first project onto a random plane using a random projection matrix $\mathbf{A} \sim \mathcal{N}(0,1)^{2\times d}$, and then decode to the nearest corner of a fixed equilateral triangle centered at $(0,0)$. Then, for target angles $\theta \in (0, \frac{\pi}{2})$, this hash family achieves query exponents ρ of the form:*

$$\rho = \ln\left[\frac{1}{3} + 3\left(\frac{\pi - \theta}{2\pi}\right)^2 - 3\left(\frac{\arccos(\frac{1}{2}\cos\theta)}{2\pi}\right)^2\right] \Big/ \ln\left[\frac{1}{3}\right]. \tag{2}$$

Among all such project–and–partition hash families based on regular k-gons, this family achieves the lowest values ρ for any target angle $\theta \in (0, \frac{\pi}{2})$.

Note again that the above statement of ρ is exact, and does not hide any order terms in d or n – in fact, the collision probabilities do not depend on d at all, due to our choice of $\mathbf{A}$ being Gaussian. Further note that the 2-gon in the plane corresponds to the one-dimensional antipodal code of Charikar, and triangular hashing therefore strictly improves upon hyperplane hashing for any θ in terms of the query exponent ρ. For instance, we can find neighbors at angle at most $\frac{\pi}{3}$ in time $\tilde{O}(n^\rho)$ with $\rho \approx 0.56996$, offering a concrete but minor improvement over the hyperplane hashing approach of Charikar [16] with query exponent $\rho \approx 0.5850$. This is the best we can do with any two-dimensional isogonal spherical code, and we numerically predict that this code achieves the lowest values ρ among all (not necessarily isogonal) two-dimensional spherical codes as well.

Three and four dimensions. For three-dimensional codes, through numerical integration of the resulting orthant probabilities we conclude that the *tetrahedron* appears to minimize the query exponent ρ out of all three-dimensional codes, beating the three-dimensional analogues of e.g. the cross-polytope and hypercube, as well as various sphere packings and other regular polytopes appearing in the literature, such as the Platonic and Archimedean solids. In four dimensions an interesting phenomenon occurs: the so-called 5-*cell* (4-simplex) and 16-*cell* (4-orthoplex) are optimal in different regimes, with the 5-cell inducing a more coarse-grained partition of the space and achieving a better performance when the nearest neighbor lies relatively far away from the data set, and the 16-cell inducing a more fine-grained partition, and working better when the nearest neighbor lies relatively close to the target vector. This crossover effect is also visualized in Figure 1, and it strengthens our intuition that as the dimensionality goes up, or as the nearest neighbor lies closer to the query, more fine-grained partitions are necessary to obtain the best performance. An overview of some of the exponents ρ for various low-dimensional spherical codes, for different target angles $\theta \in \frac{\pi}{12}\{1, 2, 3, 4, 5\}$, is given in Table 1. The best exponents ρ are highlighted in bold.

Five and more dimensions. For higher-dimensional spherical codes, we obtain further improvements in the query exponents ρ through the use of suitable spherical codes, as shown in Table 1 and Figures 1–4. For dimensions 5 and 6, the corresponding orthoplices achieve a better performance than the simplices, and the exotic polytopes 1_{21} and 2_{21}, related to the root lattice E_6, seem useful in the context of nearest neighbor searching as well.

In the full version we further study the performance of the following five non-trivial infinite families of spherical codes. Color codes below correspond to the same colors used in Table 1 and Figures 1–4 to differentiate these families of codes.

- The **simplices** S_k on $k + 1$ vertices;
- The **orthoplices** O_k on $2k$ vertices (also known as cross-polytopes [4, 36]);
- The **hypercubes** C_k on 2^k vertices (as studied in [27]);
- The **expanded simplices** A_k on $k(k + 1)$ vertices;
- The **rectified orthoplices** D_k on $2k(k - 1)$ vertices.

The latter two families are connected to the root lattices A_k and D_k, while the first three are related to the lattice $\mathbb{Z}^k$. We conjecture that, should other nice families of dense lattices be found (e.g. the recent [37]), these may give rise to suitable spherical codes in nearest neighbor applications as well. For each of the above five families we give closed-form expressions on the correlation matrices $\mathbf{\Sigma}$, but the resulting orthant probabilities that need to be evaluated for computing ρ do not appear to admit simple closed-form expressions. Apart from the family of hypercubes, these are all asymptotically optimal (with $\rho \to (1 - \cos\theta)/(1 + \cos\theta)$ as $k \to \infty$), although the convergence to the limit may differ for each family.

1.4 Lower bounds via spherical caps

Although this work tries to be exhaustive in covering as many (families of) spherical codes as possible, better spherical codes may exist, achieving even lower query exponents ρ. As these codes may be hard to find, and as it may be difficult to rule out the existence of other, better spherical codes, the next best thing one might hope for is a somewhat tight lower bound on the performance of any k-dimensional spherical code in our framework, which hopefully comes close to the performance of the spherical codes we have considered in this survey.

As has been established in several previous works on nearest neighbor searching on the sphere [3, 5, 6, 11, 17, 26], ideally we would like the hash regions induced by the partitions to take the shape of a spherical cap. Such a shape minimizes the angular radius, given that the region has a fixed volume, and in a sense it is the most natural and smoothest shape that a region on a sphere can take. So in a utopian world, one might hope that a spherical code partitions the sphere into c regions, and each region corresponds exactly to a spherical cap of volume $\mathrm{Vol}(\mathcal{S}^{k-1})/c$. Clearly such spherical codes do not exist for $k > 2$ and $c > 2$, but such an extremal example does give us an indication on the limits of what might be achievable in dimension k, and how the optimal ρ decreases with k. Note that random spherical codes might approach this utopian setting in high dimensions.

Following the above reasoning, and using a classic result of Baernstein–Taylor [10], we state a formal lower bound on the performance parameter ρ of any k-dimensional spherical code of size c, and by minimizing over c for given k, one obtains a lower bound for any spherical code living in k dimensions. Here $I_x(a, b)$ denotes the regularized incomplete beta function, which comes from computing the volume of a sphere in k dimensions, while $\| \cdot \|$ denotes the Euclidean norm.

▶ **Theorem 4** (Spherical cap lower bounds). *Let $\mathcal{C} \subset \mathcal{S}^{k-1}$ be a spherical code, and let $\mathcal{H}$ be the associated project–and–partition hash family. Then the parameter ρ for $\mathcal{C}$, for target angle $\theta \in (0, \frac{\pi}{2})$, must satisfy:*

$$\rho(\theta) \geq \rho_k(\theta) := \min_{c \geq 2} \rho_k(c, \theta), \tag{3}$$

where, with $\alpha_k(c)$ denoting the solution α to $\frac{1}{2} \cdot I_{1-\alpha^2}(\frac{k-1}{2}, \frac{1}{2}) = \frac{1}{c}$, $\rho_k(c, \theta)$ is given by:

$$\rho_k(c, \theta) := \log \left\{ \Pr_{\boldsymbol{x}, \boldsymbol{y} \sim \mathcal{N}(0,1)^k} \left(\frac{x_1}{\|\boldsymbol{x}\|} \geq \alpha_k(c), \frac{x_1 \cos\theta + y_1 \sin\theta}{\|\boldsymbol{x} \cos\theta + \boldsymbol{y} \sin\theta\|} \geq \alpha_k(c) \right) \right\} \Big/ \log \left(\frac{1}{c} \right).$$

The above minimization over $c \geq 2$ concerns the possible code sizes, and $\rho_k(c, \theta)$ describes the parameter ρ one would obtain when indeed, the code consisted of c equivalent regions of equal volume, and each region was shaped like a spherical cap. Equivalently, one could state that any spherical code of size exactly c must satisfy $\rho(\theta) \geq \rho_k(c, \theta)$.

Numerical evaluation of these expressions $\rho_k(c, \theta)$, and the resulting minimization over c, leads to the values in Table 1 in the rows indicated by *spherical caps.* The superscripts denote the values c that numerically solve the minimization problems. These results in low dimensions suggest that, especially for small angles, the optimal code size should increase superlinearly with the dimension. This inspires the following conjecture, stating that the use of orthoplices is likely not optimal in higher dimensions.

▶ **Conjecture 5** (Orthoplices are suboptimal for large k). *For arbitrary $\theta \in (0, \frac{\pi}{2})$, there exists a dimension $k_0 \in \mathbb{N}$ such that, for all dimensions $k \geq k_0$, there exist spherical codes $\mathcal{C} \subset \mathcal{S}^{k-1}$ whose query exponents ρ in the project–and–partition framework are smaller than the exponents ρ of the k-orthoplex.*

Actually finding such spherical codes, or finding families of spherical codes that outperform orthoplices may again be closely related to the problem of finding nice families of dense lattices with efficient decoding algorithms. We informally conjecture that in high dimensions, and for sufficiently large code sizes c, random spherical codes may be close to optimal. If we care only about minimizing ρ, then a further study might focus on (1) getting a better grip of the optimal scaling of $c = c(k)$ with k, and (2) estimating the asymptotic performance of using random spherical codes of size $c(k)$, for large k. We predict this will lead to better values ρ than those obtained with cross-polytope hashing.

1.5 Selecting spherical codes in practice

Although the exponent ρ, and therefore the probabilities p_1 and p_2, directly imply the main performance parameters to assess the asymptotic performance of a locality-sensitive hash family, in practice we are always dealing with concrete, non-asymptotic values n, d, and θ – if the convergence to the optimal asymptotic scaling is slow, or if the hash functions are too expensive to evaluate in practice, then hash families with lower query exponents ρ may actually be less practical for small, concrete values of n and d than fast hash families with larger ρ. For example, the hash family from [3] may be "optimal", but appears to be only of theoretical interest. In practice one needs to find a balance between decreasing ρ and using hash functions which are fast to evaluate.

As a more concrete example, consider how hyperplane hashing [16] allows us to partition a sphere in 2^k regions with a single random projection matrix $\mathbf{A} \in \mathbb{R}^{k \times d}$ (or equivalently k matrices $\mathbf{A}_1, \dots, \mathbf{A}_k \in \mathbb{R}^{1 \times d}$ merged into one large matrix). For cross-polytope hashing [4] a k-dimensional projection would only divide the k-dimensional sphere into $2k$ regions. As the total required number of hash buckets in the data structure is often roughly the same, regardless of the chosen hash family,[1] this means that if we wish to divide the sphere into 2^k hash regions with cross-polytope hashing, we would need $m = k/\log_2(2k)$ independent random projection matrices $\mathbf{A}_1, \dots, \mathbf{A}_m \in \mathbb{R}^{k \times d}$ to hash a vector to one of 2^k buckets. So even though the resulting partitions for cross-polytope hashing generate smaller exponents ρ, in practice this improvement may not offset the additional costs of computing the projections/rotations, which is almost a linear factor $O(d)$ more than for hyperplane hashing. So especially for data sets of small/moderate sizes, hyperplane hashing may be more practical than cross-polytope hashing, as also observed in e.g. [12, 25, 28, 29].

We explicitly quantify the trade-off between the complexity of the hash functions and the asymptotic query exponents ρ in Figure 3, where on the vertical axis we plotted the query exponents ρ as computed in this paper, and on the horizontal axis we plotted the number of bits extracted per row of a projection matrix; for hyperplane hashing we extract 1 bit per projection, while e.g. for cross-polytope hashing in dimension k we only extract $\log(2k)/k$ bits per projection. Depending on the application, it may be desirable to choose hash families with slightly larger values ρ, if this means the cost of the hashing becomes less. Figure 3 makes a case for using hashes induced by the root lattices D_k, as well as those induced by exotic polytopes such as the 2_{21}-polytope; compared to e.g. the 5-orthoplex, the D_{10} lattice achieves lower values ρ and extracts more bits per projection, while the 2_{21}-polytope further improves upon D_{10} by extracting more hash bits per projection.

To further illustrate how these trade-offs might look in a real-world setting, Figure 4 describes a case study for average-case nearest neighbor searching with different sizes n for the data sets. For small data sets, hyperplane hashing is quite competitive, and the best other spherical codes are those induced by the D_k lattices (for small k), and spherical codes generated by the polytopes $1_{21}, 1_{31}$ and 2_{21}. This case study matches the recommendations from Figure 3. For large n, the role of ρ becomes more prominent, and higher-dimensional orthoplices and D_k lattices attain the best performance. In the full version we further describe how one might choose the best codes in practice.

1.6 Summary and open problems

With the project–and–partition framework outlined in this paper, we can now easily formalize and analyze the performance of any spherical code in this framework, and analyze the collision probabilities and query exponents either analytically (by attempting to simplify the corresponding multivariate orthant probabilities) or numerically (by evaluating these multivariate integrals with mathematical software, such as the mvtnorm package [21]). We observed that already in two dimensions, it is possible to improve upon hyperplane hashing with a smaller exponent ρ by using triangular partitions, and we described closed-form

[1] While a hash family with lower query exponents ρ does require a smaller number of hash buckets per hash table, this effect is marginal compared to the increase in the number of projections/rotations required by e.g. cross-polytope hashing compared to hyperplane hashing.

formulas for the resulting parameters ρ. In three and four dimensions the improvements in the exponent ρ become more significant, with e.g. the tetrahedron, the 5-cell, and the 16-cell achieving the best exponents ρ in these dimensions. For higher dimensions the simplices and orthoplices seem to achieve the best performance in theory, thus making a strong case for the use of these partitions as advertised in [4, 24, 35, 36], and as observed in nearest neighbor benchmarks [9, 13, 31]. The family of D_k lattices and the generalized m-max hash functions however seem to offer better practical trade-offs between the asymptotic performance in terms of ρ and the costs of computing hashes, as discussed in the full version.

Finding better spherical codes. An important open problem, both for our project–and–partition framework and for arbitrary hash families for the angular distance, is to find (if they exist) other nice families of spherical codes which achieve an even better performance than the family of orthoplices. Numerics in the full version suggest that as k increases, the optimal code size c should increase faster than the linear scaling of $c = 2k$ offered by orthoplices. Finding the optimal scaling of c as a function of k, and finding nicer spherical codes closely matching this appropriate scaling of c is left for future work.

Faster pseudorandom projections. To make e.g. hyperplane and cross-polytope hashing more practical, various ideas were proposed in e.g. [1, 4] to work with sparse projections and pseudorandom rotation matrices. Similar ideas can be used for any hash family, to reduce the cost of multiplication by a random Gaussian matrix $\mathbf{A}$. Concretely, one can often replace it with a sparse, "sufficiently random" pseudorandom matrix $\mathbf{A}$ while still achieving good query exponents ρ in practice. Depending on how these pseudorandom projections are instantiated, this may lead to a different practical evaluation than what we described in our practical case analyses. In particular, as this reduces the relative cost of the hashing, this will further favor schemes which reduce ρ at the cost of increasing the (naive) complexity of hashing.

Using orthogonal projection matrices. In our framework, we focused on projecting down onto a low-dimensional subspace using Gaussian matrices $\mathbf{A} \sim \mathcal{N}(0,1)^{k \times d}$. For $k \ll d$, using Gaussian matrices or orthogonal matrices (i.e. $\mathbf{A}\mathbf{A}^T = I_k$) is essentially equivalent [18, 23], but for large k it may be beneficial to use proper rotations induced by orthogonal matrices. For instance, recent work [27] showed that for hypercube hashing in the ambient space, there is a clear gap between using random or orthogonal matrices; using orthogonal matrices generally works better than using random Gaussian projection matrices.

The main issue when analyzing the same framework with orthogonal matrices is that the dependence on d then does not disappear; the distribution of $\mathbf{A}\boldsymbol{x}$ then relies on both k and d, rather than only on $\boldsymbol{k}$. Our framework is in a sense dimensionless, as the performance can be computed without knowledge of d, and for $k = 2$ this even allowed us to obtain explicit analytic expressions for the collision probabilities for arbitrary k-gons. Using orthogonal matrices, the collision probabilities will likely become complicated functions of d, and comparing different spherical codes may then become a more difficult task. We leave it as an open problem to adjust the above framework to the setting where $\mathbf{A}$ is orthogonal, and to see how much can be gained by using orthogonal rather than Gaussian matrices[2].

The remainder of the paper can be found in the full version at `[arXiv:1907.04628]`.

[2] Note that one obtains different asymptotics when analyzing the hypercube for Gaussian [16] and orthogonal projection/rotation matrices [27]. For constant $k = O(1)$ and large d both approaches are asymptotically equivalent, but for large $k = O(d)$ we expect orthogonal matrices to give better results.

Table 1 Parameters ρ for various spherical codes. Sphere packings are from [34]. Rows labeled "spherical caps" are lower bounds (Theorem 4). Superscripts refer to the number of caps minimizing ρ_θ. Bold values indicate the best values ρ encountered up to this dimension. Results for $k = 1, 2, d$, were obtained analytically; for $k = 3, 4, 5, 6$ most results were obtained through numerical integration and Monte Carlo simulation.

k	spherical code	c	$\rho(\frac{\pi}{12})$	$\rho(\frac{\pi}{6})$	$\rho(\frac{\pi}{4})$	$\rho(\frac{\pi}{3})$	$\rho(\frac{5\pi}{12})$
1	*spherical caps*		*0.1255*$^{(2)}$	*0.2630*$^{(2)}$	*0.4150*$^{(2)}$	*0.5850*$^{(2)}$	*0.7776*$^{(2)}$
	hyperplane	2	**0.1255**	**0.2630**	**0.4150**	**0.5850**	**0.7776**
2	*spherical caps*		*0.1194*$^{(3)}$	*0.2518*$^{(3)}$	*0.4005*$^{(3)}$	*0.5700*$^{(3)}$	*0.7666*$^{(3)}$
	triangle (S_2)	3	**0.1194**	**0.2518**	**0.4005**	**0.5700**	**0.7666**
	square (O_2, C_2, D_2)	4	0.1255	0.2630	0.4150	0.5850	0.7776
	pentagon	5	0.1343	0.2788	0.4346	0.6040	0.7905
	hexagon (A_2)	6	0.1438	0.2954	0.4544	0.6222	0.8022
3	*spherical caps*		*0.1117*$^{(5)}$	*0.2389*$^{(4)}$	*0.3846*$^{(4)}$	*0.5548*$^{(4)}$	*0.7561*$^{(4)}$
	tetrahedron (S_3)	4	**0.1155**	**0.2445**	**0.3910**	**0.5600**	**0.7592**
	sphere packing	5	0.1170	0.2481	0.3965	0.5664	0.7644
	octahedron (O_3)	6	0.1159	0.2465	0.3952	0.5661	0.7649
	sphere packing	7	0.1207	0.2554	0.4065	0.5772	0.7725
	cube (C_3)	8	0.1255	0.2630	0.4150	0.5850	0.7776
	sphere packing	9	0.1217	0.2591	0.4129	0.5850	0.7786
	icosahedron	12	0.1255	0.2678	0.4255	0.5983	0.7883
	cuboctahedron (A_3, D_3)	12	0.1294	0.2728	0.4301	0.6017	0.7900
	dodecahedron	20	0.1509	0.3077	0.4692	0.6360	0.8112
4	*spherical caps*		*0.1050*$^{(7)}$	*0.2285*$^{(6)}$	*0.3730*$^{(6)}$	*0.5433*$^{(5)}$	*0.7466*$^{(5)}$
	5-cell (S_4)	5	0.1126	0.2392	0.3840	**0.5527**	**0.7537**
	sphere packing	6	0.1128	0.2401	0.3861	0.5555	0.7564
	sphere packing	7	0.1120	0.2392	0.3852	0.5553	0.7567
	16-cell (O_4)	8	**0.1107**	**0.2368**	**0.3822**	0.5528	0.7553
	sphere packing	10	0.1136	0.2426	0.3909	0.5623	0.7622
	sphere packing	13	0.1133	0.2439	0.3939	0.5666	0.7663
	tesseract (C_4)	16	0.1255	0.2630	0.4150	0.5850	0.7776
	runcinated 5-cell (A_4)	20	0.1216	0.2586	0.4128	0.5855	0.7795
	octacube (D_4)	24	0.1202	0.2577	0.4140	0.5877	0.7823
5	*spherical caps*		*0.0997*$^{(13)}$	*0.2203*$^{(10)}$	*0.3630*$^{(8)}$	*0.5342*$^{(7)}$	*0.7415*$^{(6)}$
	5-simplex (S_5)	6	0.1105	0.2354	0.3785	0.5469	0.7493
	5-orthoplex (O_5)	10	**0.1076**	**0.2299**	**0.3733**	**0.5433**	**0.7483**
	1_{21}-polytope	16	0.1080	0.2330	0.3794	0.5516	0.7554
	expanded 5-simplex (A_5)	30	0.1167	0.2494	0.4007	0.5735	0.7713
	5-hypercube (C_5)	32	0.1255	0.2630	0.4150	0.5850	0.7776
	rectified 5-orthoplex (D_5)	40	0.1139	0.2471	0.4009	0.5757	0.7739
6	*spherical caps*		*0.0955*$^{(18)}$	*0.2135*$^{(15)}$	*0.3552*$^{(11)}$	*0.5263*$^{(9)}$	*0.7357*$^{(8)}$
	6-simplex (S_6)	7	0.1089	0.2319	0.3742	0.5422	0.7458
	6-orthoplex (O_6)	12	0.1065	0.2260	**0.3670**	**0.5361**	**0.7431**
	2_{21}-polytope	27	**0.1038**	**0.2258**	0.3712	0.5442	0.7510
	1_{31}-polytope	32	0.1062	0.2314	0.3788	0.5520	0.7564
	expanded 6-simplex (A_6)	42	0.1133	0.2427	0.3917	0.5642	0.7647
	rectified 6-orthoplex (D_6)	60	0.1107	0.2404	0.3915	0.5661	0.7673
	hypercube (C_6)	64	0.1255	0.2630	0.4150	0.5850	0.7776
d	*lower bound*		*0.0173*	*0.0718*	*0.1716*	*0.3333*	*0.5888*
	simplex (S_d)	$d+1$	**0.0173**	**0.0718**	**0.1716**	**0.3333**	**0.5888**
	orthoplex (O_d)	$2d$	**0.0173**	**0.0718**	**0.1716**	**0.3333**	**0.5888**
	expanded simplex (A_d)	$d(d+1)$	**0.0173**	**0.0718**	**0.1716**	**0.3333**	**0.5888**
	rectified orthoplex (D_d)	$2d(d-1)$	**0.0173**	**0.0718**	**0.1716**	**0.3333**	**0.5888**
	hypercube (C_d; $\mathbf{A}$ orth.)	2^d	0.0799	0.1800	0.3151	0.5201	0.7686
	hypercube (C_d; $\mathbf{A}$ Gaussian)	2^d	0.1255	0.2630	0.4150	0.5850	0.7776

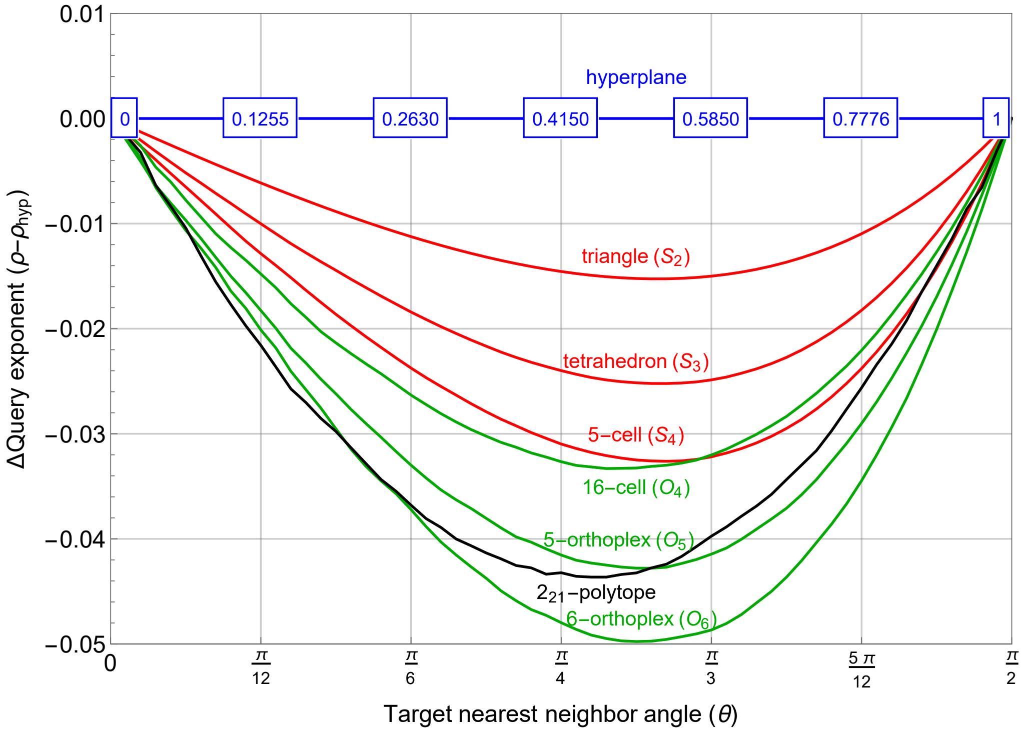


Figure 1 Comparison of the improvements in the query exponent ρ over hyperplane hashing [16] with query exponents ρ_{hyp}. The horizontal blue line denotes the baseline hyperplane hashing approach, with $\rho - \rho_{\text{hyp}} \equiv 0$. Lower curves denote improvements to ρ_{hyp} using various spherical codes. The measurements at the five vertical gridlines correspond to the values in Table 1. For instance, at $\theta = \pi/6$ the triangle has query exponent approximately 0.011 lower than hyperplane hashing (which has exponent 0.2630), leading to $\rho \approx 0.2520$; from Table 1 we get the more precise estimate $\rho \approx 0.2518$; while the theory in the full version allows us to compute ρ exactly through a closed-form expression. The polytopes in this figure are those achieving the lowest exponents ρ in their respective dimensions at one of these grid lines, as highlighted in boldface in Table 1.

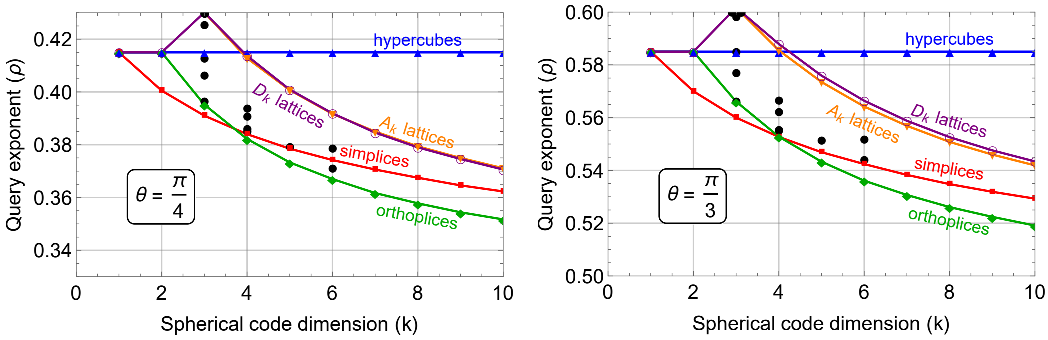


Figure 2 Query exponents ρ for project–and–partition hash families based on the spherical codes listed in Table 1. Lower query exponents ρ are generally better, and for these hash families the best exponents ρ are achieved by the simplex for $k \leq 3$ and by the orthoplex for $k \geq 5$. For $k = 4$ the simplex and orthoplex are close, and which of the two achieves a better performance depends on the target nearest neighbor angle θ.

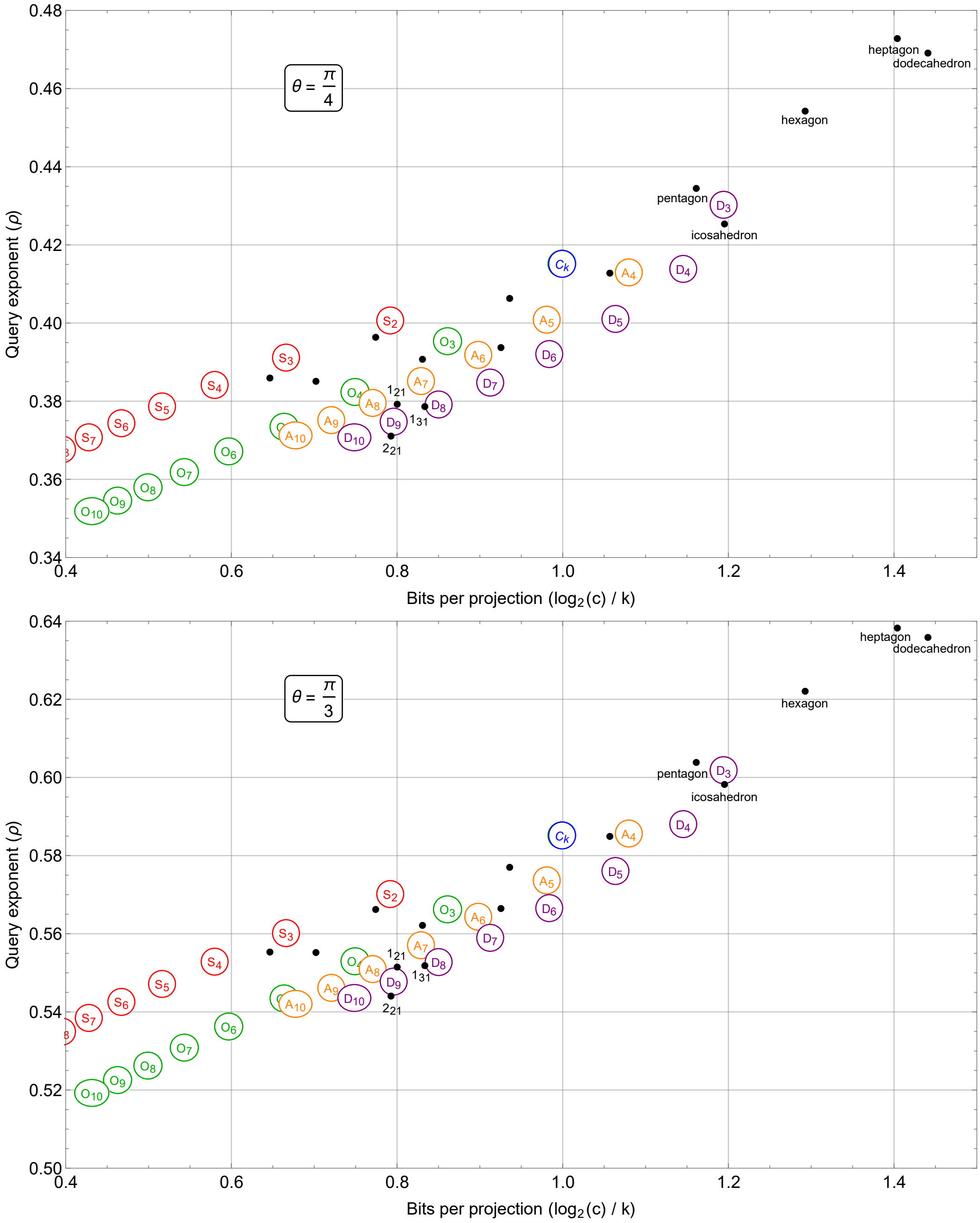


Figure 3 A comparison between different spherical codes in terms of the query exponent ρ (vertical axis) and the bits of information extracted from each row of the projection matrix $\mathbf{A}$ (horizontal axis). The top figure corresponds to target angle $\theta = \frac{\pi}{4}$ (or approximation factor $c = \sqrt{2 + \sqrt{2}}$), the bottom figure to target angle $\theta = \frac{\pi}{3}$ (or $c = \sqrt{2}$). Codes further down generate hash functions with a more discriminative power (a lower value ρ), while codes further to the right extract more bits per projection, therefore requiring fewer inner product computations to compute hash values. So for our purposes, the best codes would be as far to the right and as far down as possible. Hypercubes C_k all achieve the same value ρ and the same value $\log_2(c)/k = 1$.

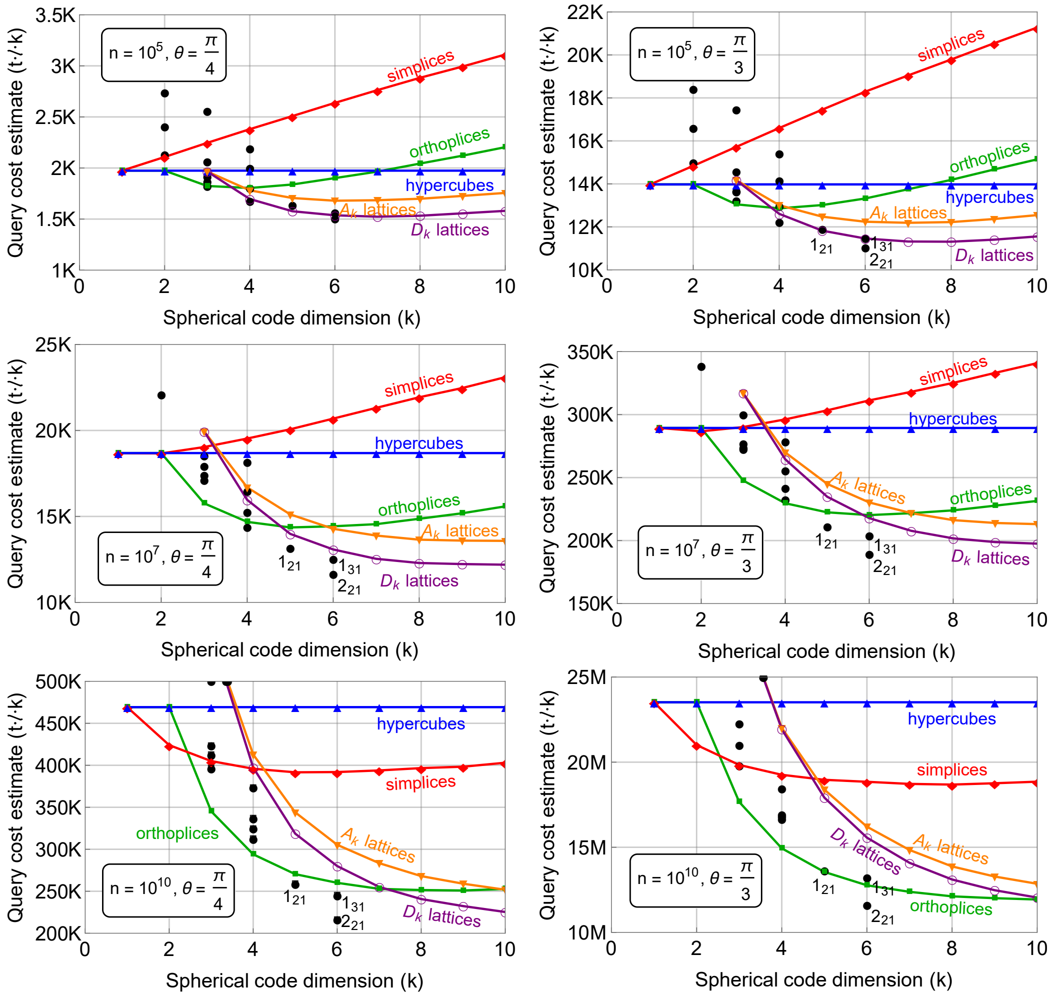


Figure 4 Comparison of the query cost estimates $t \cdot \ell \cdot k$ for different parameters $n \in \{10^5, 10^7, 10^{10}\}$ and $\theta \in \{\frac{\pi}{4}, \frac{\pi}{3}\}$, when using $t = n^\rho$ hash tables and a hash length $\ell = \log(n)/\log(1/p_2)$. The curves correspond to the regular convex polytope families and the root lattice families A_k and D_k, while single black points for $k \le 6$ correspond to spherical codes described in Table 1. As n increases, the role of a smaller ρ becomes bigger, and so larger spherical codes with smaller values ρ become more suitable choices than those with bigger values ρ but with lower hash costs. The cost of the projections increases linearly with k, while ρ decreases rather slowly with k, suggesting that for each n and θ there is an optimal spherical code dimension $k \ll d$ to project down to, and an optimal spherical code in this dimension to use. Note that other spherical codes than those from the five families of codes sometimes achieve better query cost estimates; in particular, the 2_{21}-polytope (which is connected to the E_6 lattice) might be useful in practice as well, and we cannot rule out the existence of other exotic spherical codes with good properties for nearest neighbor searching.

One of the conclusions one might draw from these figures is that indeed orthoplices (cross-polytopes) perform very well [4], but depending on the parameters it may be better to use the D_k lattices instead – the trends suggest that as k increases further, the query costs of using the D_k lattices will be smaller than those of the orthoplices.

References

1 Dimitris Achlioptas. Database-friendly random projections. In *PODS*, pages 274–281, 2001. doi:10.1145/375551.375608.

2 Martin R. Albrecht, Léo Ducas, Gottfried Herold, Elena Kirshanova, Eamonn Postlethwaite, and Marc Stevens. The general sieve kernel and new records in lattice reduction. In *EUROCRYPT*, pages 717–746, 2019.

3 Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *FOCS*, pages 459–468, 2006. doi:10.1109/FOCS.2006.49.

4 Alexandr Andoni, Piotr Indyk, Thijs Laarhoven, Ilya Razenshteyn, and Ludwig Schmidt. Practical and optimal LSH for angular distance. In *NIPS*, pages 1225–1233, 2015. URL: https://papers.nips.cc/paper/5893-practical-and-optimal-lsh-for-angular-distance.

5 Alexandr Andoni, Piotr Indyk, Huy Lê Nguyên, and Ilya Razenshteyn. Beyond locality-sensitive hashing. In *SODA*, pages 1018–1028, 2014. doi:10.1137/1.9781611973402.76.

6 Alexandr Andoni, Thijs Laarhoven, Ilya Razenshteyn, and Erik Waingarten. Optimal hashing-based time-space trade-offs for approximate near neighbors. In *SODA*, pages 47–66, 2017. doi:10.1137/1.9781611974782.4.

7 Alexandr Andoni and Ilya Razenshteyn. Optimal data-dependent hashing for approximate near neighbors. In *STOC*, pages 793–801, 2015. doi:10.1145/2746539.2746553.

8 Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. In *SODA*, pages 573–582, 1994. URL: http://dl.acm.org/citation.cfm?id=314464.314652.

9 Martin Aumueller, Erik Bernhardsson, and Alexander Faithfull. ANN benchmarks – available online at http://sss.projects.itu.dk/ann-benchmarks/, 2017. URL: http://sss.projects.itu.dk/ann-benchmarks/.

10 Albert Baernstein and B.A. Taylor. Spherical rearrangements, subharmonic functions, and $*$ -functions in n -space. *Duke Math. J.*, 43(2):245–268, June 1976. doi:10.1215/S0012-7094-76-04322-2.

11 Anja Becker, Léo Ducas, Nicolas Gama, and Thijs Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. In *SODA*, pages 10–24, 2016. doi:10.1137/1.9781611974331.ch2.

12 Anja Becker and Thijs Laarhoven. Efficient (ideal) lattice sieving using cross-polytope LSH. In *AFRICACRYPT*, pages 3–23, 2016. doi:10.1007/978-3-319-31517-1_1.

13 Erik Bernhardsson. ANN benchmarks – available online at https://github.com/erikbern/ann-benchmarks, 2016. URL: https://github.com/erikbern/ann-benchmarks.

14 Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag, 2006. URL: https://www.springer.com/us/book/9780387310732.

15 Karthekeyan Chandrasekaran, Daniel Dadush, Venkata Gandikota, and Elena Grigorescu. Lattice-based locality sensitive hashing is optimal. In *ITCS*, 2018.

16 Moses S. Charikar. Similarity estimation techniques from rounding algorithms. In *STOC*, pages 380–388, 2002. doi:10.1145/509907.509965.

17 Tobias Christiani. A framework for similarity search with space-time tradeoffs using locality-sensitive filtering. In *SODA*, pages 31–46, 2017. doi:10.1137/1.9781611974782.3.

18 Persi Diaconis and David Freedman. A dozen de Finetti-style results in search of a theory. *Annales de l'IHP Probabilités et statistiques*, 23(2):397–423, 1987.

19 Moshe Dubiner. Bucketing coding and information theory for the statistical high-dimensional nearest-neighbor problem. *IEEE Transactions on Information Theory*, 56(8):4166–4179, August 2010. doi:10.1109/TIT.2010.2050814.

20 Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification (2nd Edition)*. Wiley, 2000. URL: https://dl.acm.org/citation.cfm?id=954544.

21 Alan Genz, Frank Bretz, Tetsuhisa Miwa, Xuefei Mi, Friedrich Leisch, Fabian Scheipl, and Torsten Hothorn. *mvtnorm: Multivariate normal and t distributions*, 2019. R package version 1.0-10. URL: https://CRAN.R-project.org/package=mvtnorm.

22 Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *STOC*, pages 604–613, 1998. doi:10.1145/276698.276876.

23 Tiefeng Jiang. How many entries of a typical orthogonal matrix can be approximated by independent normals? *The Annals of Probability*, 34(4):1497–1529, 2006. doi:10.1214/009117906000000205.

24 Christopher Kennedy and Rachel Ward. Fast cross-polytope locality-sensitive hashing. In *ITCS*, pages 1–16, 2017. doi:10.4230/LIPIcs.ITCS.2017.53.

25 Thijs Laarhoven. Sieving for shortest vectors in lattices using angular locality-sensitive hashing. In *CRYPTO*, pages 3–22, 2015. doi:10.1007/978-3-662-47989-6_1.

26 Thijs Laarhoven. Tradeoffs for nearest neighbors on the sphere. *arXiv:1511.07527 [cs.DS]*, pages 1–16, 2015. arXiv:1511.07527.

27 Thijs Laarhoven. Hypercube LSH for approximate near neighbors. In *MFCS*, pages 1–20, 2017. doi:10.4230/LIPIcs.MFCS.2017.7.

28 Artur Mariano, Thijs Laarhoven, and Christian Bischof. Parallel (probable) lock-free HashSieve: a practical sieving algorithm for the SVP. In *ICPP*, pages 590–599, 2015. doi:10.1109/ICPP.2015.68.

29 Artur Mariano, Thijs Laarhoven, and Christian Bischof. A parallel variant of LDSieve for the SVP on lattices. In *PDP*, pages 23–30, 2017. doi:10.1109/PDP.2017.60.

30 Alexander May and Ilya Ozerov. On computing nearest neighbors with applications to decoding of binary linear codes. In *EUROCRYPT*, pages 203–228, 2015. doi:10.1007/978-3-662-46800-5_9.

31 Ilya Razenshteyn and Ludwig Schmidt. FALCONN – available online at https://falconn-lib.org/, 2016. URL: https://falconn-lib.org/.

32 Gregory Shakhnarovich, Trevor Darrell, and Piotr Indyk. *Nearest-Neighbor Methods in Learning and Vision: Theory and Practice*. MIT Press, 2005. URL: http://ttic.uchicago.edu/~gregory/annbook/book.html.

33 William F. Sheppard. On the application of the theory of error to cases of normal distribution and normal correlation. *Philosophical Transactions of the Royal Society A*, 192:101–167, 1899.

34 Neil J.A. Sloane. Spherical codes: nice arrangements of points on a sphere in various dimensions. URL: http://neilsloane.com/packings/.

35 Kengo Terasawa and Yuzuru Tanaka. Spherical LSH for approximate nearest neighbor search on unit hypersphere. In *WADS*, pages 27–38, 2007. doi:10.1007/978-3-540-73951-7_4.

36 Kengo Terasawa and Yuzuru Tanaka. Approximate nearest neighbor search for a dataset of normalized vectors. In *IEICE Transactions on Information and Systems*, volume 92(9), pages 1609–1619, 2009. URL: http://search.ieice.org/bin/summary.php?id=e92-d_9_1609.

37 Serge Vladut. Lattices with exponentially large kissing numbers. *Moscow J. Comb. Number Th.*, 8:163–177, 2019.

Deterministic Sparse Fourier Transform with an ℓ_∞ Guarantee

Yi Li
Nanyang Technological University, Singapore, Singapore
yili@ntu.edu.sg

Vasileios Nakos
Universität des Saarlandes, Saarbrücken, Germany
Max Planck Institut für Informatik, Saarland Informatics Campus, Saarbrücken, Germany
vnakos@mpi-inf.mpg.de

Abstract

In this paper we revisit the deterministic version of the Sparse Fourier Transform problem, which asks to read only a few entries of $x \in \mathbb{C}^n$ and design a recovery algorithm such that the output of the algorithm approximates $\widehat{x}$, the Discrete Fourier Transform (DFT) of x. The randomized case has been well-understood, while the main work in the deterministic case is that of Merhi et al. (J Fourier Anal Appl 2018), which obtains $O(k^2 \log^{-1} k \cdot \log^{5.5} n)$ samples and a similar runtime with the ℓ_2/ℓ_1 guarantee. We focus on the stronger ℓ_∞/ℓ_1 guarantee and the closely related problem of incoherent matrices. We list our contributions as follows.

1. We find a deterministic collection of $O(k^2 \log n)$ samples for the ℓ_∞/ℓ_1 recovery in time $O(nk \log^2 n)$, and a deterministic collection of $O(k^2 \log^2 n)$ samples for the ℓ_∞/ℓ_1 sparse recovery in time $O(k^2 \log^3 n)$.
2. We give new deterministic constructions of incoherent matrices that are row-sampled submatrices of the DFT matrix, via a derandomization of Bernstein's inequality and bounds on exponential sums considered in analytic number theory. Our first construction matches a previous randomized construction of Nelson, Nguyen and Woodruff (RANDOM'12), where there was no constraint on the form of the incoherent matrix.

Our algorithms are nearly sample-optimal, since a lower bound of $\Omega(k^2 + k \log n)$ is known, even for the case where the sensing matrix can be arbitrarily designed. A similar lower bound of $\Omega(k^2 \log n / \log k)$ is known for incoherent matrices.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis; Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases Fourier sparse recovery, derandomization, incoherent matrices

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.77

Category Track A: Algorithms, Complexity and Games

Related Version A full version of the paper is available at `https://arxiv.org/abs/1903.00995`.

Funding *Vasileios Nakos*: This work is part of the project TIPEA that has received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme (grant agreement No. 850979). Part of the work was completed when the author was a Ph.D. student in Harvard University and supported in part by NSF CAREER award CCF-1350670.

Acknowledgements We would like to thank anonymous reviewers for their valuable feedback.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 77; pp. 77:1–77:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Compressed sensing is a subfield of discrete signal processing, based on the principle that a high-dimensional signal can be approximately reconstructed, by exploiting its sparsity, in fewer samples than those demanded by the Shannon-Nyquist theorem. An important subtopic is the Sparse Fourier Transform, where we desire to detect and approximate the largest coordinates of a high-dimensional signal, given a few samples from its Fourier spectrum. Fewer samples play a crucial role, for example, in medical imaging, where reconstructing an image corresponds exactly to reconstructing a signal from its Fourier representation. Thus, the number of Fourier coefficients needed for (approximate) reconstruction is proportional to the radiation dose a patient receives as well as the time the patient needs to remain in the scanner. Furthermore, exploiting the sparsity of the signal has given researchers the hope of defeating the FFT algorithm of Cooley and Tukey, in the special (but of high practical value) case where the signal is approximately sparse. Thus, since FFT serves as an important computational primitive, and has been recognized as one of the 10 most important algorithms of the 20th century [16], every place where it has found application can possibly be benefited from a faster algorithm. The main intuition and hope is that signals arising in practice often exhibit certain structure, such as concentration of energy in a small number of Fourier coefficients.

Since vectors in practice are never exactly sparse, and it is impossible to reconstruct a generic vector $\widehat{x} \in \mathbb{C}^n$ from $o(n)$ samples, researchers resort to approximation. More formally, a sparse recovery scheme consists of a sample set $S \subseteq \{1, \ldots, n\}$ and a recovery algorithm $\mathcal{R}$ such that for any given $x \in \mathbb{C}^n$, the scheme approximates $\widehat{x}$ by $\widehat{x}' = \mathcal{R}(x_S)$, where x_S denotes the vector of x restricted to the coordinates in S. The fineness of approximation is measured with respect to the best k-sparse approximation to $\widehat{x}$. The breakthrough work of Candès, Tao and Donoho [12, 20] first showed that $k \log^{O(1)} n$ samples of $x \in \mathbb{C}^n$ suffices to reconstruct a $O(k)$-sparse vector $\widehat{x}'$ which is "close" to the best k-approximation of $\widehat{x}$. More formally, the reconstruction $\widehat{x}'$ satisfies the so-called ℓ_2/ℓ_1 guarantee, i.e.,

$$\|\widehat{x} - \widehat{x}'\|_2 \leq \frac{1}{\sqrt{k}} \|\widehat{x}_{-k}\|_1,$$

where $\widehat{x}_{-k}$ is the tail vector, obtained from restricting $\widehat{x}$ to its smallest $n-k$ coordinates in magnitude. The strength of their algorithms lies in the uniformity, in the sense that the samples at the same coordinates can be used to approximate every $x \in \mathbb{C}^n$. However, the running time is polynomial in the vector length n, giving thus only sample-efficient, but not necessarily time-efficient algorithms. Furthermore, the samples are not obtained via a deterministic procedure, but are chosen at random. Regarding non-uniform randomized algorithms that run in sublinear time, numerous researchers have worked on the problem and obtained a series of algorithms with different recovery guarantees [29, 46, 42, 24, 3, 27, 30, 31, 43, 38, 53, 34, 33, 39, 40, 41, 50]. See Table 1 for a list of common recovery guarantees. The state of the art is the seminal algorithm of Kapralov [40], which shows that $O(k \log n)$ samples and $O(k \log^{O(1)} n)$ time are simultaneously possible for the ℓ_2/ℓ_2 guarantee (which is strictly stronger[1] than the ℓ_2/ℓ_1). The fastest algorithm is due to [30], needing $O(k \log n \cdot \log(n/k))$ time and samples. We note also the algorithm of Indyk and Kapralov [33] that runs in $O(n \log^2 n)$ time, uses $O(k \log n)$ samples but gives a stronger

[1] Here we mean that given an algorithm giving the ℓ_2/ℓ_2 guarantee, one can create an algorithm, using the ℓ_2/ℓ_2 algorithm as a black box, with sparsity parameter $k' = O(k)$, achieving the ℓ_2/ℓ_1 guarantee with the same order of number of samples.

Table 1 Common guarantees of sparse recovery. Only the ℓ_2/ℓ_2 case requires a parameter $C > 1$. The guarantees are listed in the descending order of strength.

Guarantee	Formula	Deterministic Lower Bound
ℓ_∞/ℓ_2	$\|\widehat{x} - \widehat{x}'\|_\infty \leq \|\widehat{x}_{-k}\|_2/\sqrt{k}$	$\Omega(n)$ [17]
ℓ_2/ℓ_2	$\|\widehat{x} - \widehat{x}'\|_2 \leq C\|\widehat{x}_{-k}\|_2$	$\Omega(n)$ [17]
ℓ_∞/ℓ_1	$\|\widehat{x} - \widehat{x}'\|_\infty \leq \|\widehat{x}_{-k}\|_1/k$	$\Omega(k^2 + k\log n)$ [23, 21]
ℓ_2/ℓ_1	$\|\widehat{x} - \widehat{x}'\|_2 \leq \|\widehat{x}_{-k}\|_1/\sqrt{k}$	$\Omega(k\log(n/k))$ [23, 21]

ℓ_∞/ℓ_2 guarantee than the ℓ_2/ℓ_2 guarantee in the previous two papers. We refer the reader to the next section for comparison of the different guarantees appearing in the literature. Recently there has been also considerable work on recovering k-sparse signals from their continuous Fourier Transform, see [9, 55, 14, 6].

Although our understanding on randomized algorithms is almost complete, there are still important gaps in our knowledge regarding deterministic schemes. The following natural open-ended question has theoretical and practical interest and remains in principle highly unexplored, touching a variety of fields including (sublinear-time) algorithms, pseudorandomness and computational complexity, Additive Combinatorics [10] and analytic number theory.

▶ **Question 1.** *What are the best bounds we can obtain for the different versions of the deterministic Sparse Fourier Transform problem?*

With sublinear runtime, the earliest work of Iwen [36, 37] gives $O(k^2\log^4 n)$ samples and time, albeit in a significantly easier (although similar) model: where one wants to learn a band-limited function $f : [0, 2\pi) \to \mathbb{C}$ and can evaluate f at any point. In the discrete case which we are interested in, the state of the art is the work of Merhi et al. [47], which obtains $O(k^2\log^{11/2} n/\log k)$ samples and the same runtime. A recent work of Bittens et al. [8] showed that the quadratic dependence can be dropped if the signals are sufficiently structured, namely, if the Fourier coefficients are generated by an unknown but small degree polynomial. On the related problem of the Walsh-Hamadard Transform, Indyk and Cheraghchi [15] showed that roughly $O(k^{1+\alpha}\log^{O(1)+6/\alpha} n)$ samples and similar run-time are possible, if one resorts to a slightly weaker guarantee. Interestingly, their approach resides in a novel connection between the Walsh-Hadamard matrix and linear lossless condensers. However, this connection does not extend to the Fourier Transform over $\mathbb{Z}_n$, which is our focus and the most interesting case. Interesting ideas appear also in the work of Akavia [1, 2], where it is shown how to approximate the Fourier Transform of an arithmetic progression in poly-logarithmic time in the length of the progression; due to the worse dependence on the quality of approximation, however, that work obtained an algorithm with sample complexity $(k \cdot \text{(signal-to-noise ratio)})^4$.

The papers above showed how to achieve the ℓ_2/ℓ_1 guarantee in a number of samples that is quadratic in the signal sparsity. It is already known that a nearly linear dependence is possible [12]; however, we do not have efficient deterministic algorithms for finding these samples. The work of [12], as well as subsequent works, proceeds by sampling with repetition rows of the DFT matrix, and showing that the RIP condition (see Definition 5) holds, which in turn implies the desired result, but via a super-linear algorithm. The state-of-the-art analysis of such row subsampling is due to Haviv and Regev [32], who showed that $O(k\log^2 k\log n)$ samples suffice. A lower bound of $\Omega(k\log n)$ rows for this subsampling process has been shown in [7]. In this paper, we follow a different avenue and give a new set of schemes for the Sparse Fourier Transform which allow uniform reconstruction. Although our dependence is still quadratic in k, it is necessary, in contrast to the previous works: our results satisfy the strictly stronger ℓ_∞/ℓ_1 guarantee, for which a quadratic lower bound is known [23], and hence one cannot hope for a sub-quadratic dependence. We also note the deterministic algorithm of [41], which needs a cubic dependence on k but solves a somewhat different problem of finding the multidimensional sparse Fourier transform of a signal with at most k non-zeros in the frequency domain, and thus is not robust to noise.

The focus of our work is the ℓ_∞/ℓ_1 guarantee, defined formally as follows.

▶ **Definition 2** (ℓ_∞/ℓ_1 guarantee). *A sparse recovery scheme is said to satisfy the ℓ_∞/ℓ_1 guarantee with parameter k, if given access to vector x, it outputs a vector $\widehat{x}'$ such that*

$$\|\widehat{x} - \widehat{x}'\|_\infty \leq \frac{1}{k}\|\widehat{x}_{-k}\|_1. \tag{1}$$

ℓ_∞/ℓ_1 versus ℓ_2/ℓ_1: A matter of "find all" versus "miss all"

As we have discussed, previous works satisfied the ℓ_2/ℓ_1 guarantee, while our target is the ℓ_∞/ℓ_1 guarantee. Any algorithm for the latter guarantee also satisfies the former one. But, as we shall demonstrate in Section 2.3, the ℓ_∞/ℓ_1 guarantee is much stronger: there exists an infinite family of vectors for which an ℓ_2/ℓ_1 algorithm might detect none of the heavy frequencies, while an ℓ_∞/ℓ_1 algorithm must detect all of them. This happens because the ℓ_∞/ℓ_1 is a **worst-case** guarantee, in the sense that it requires detection of every frequency just above the noise level, in contrast to the ℓ_2/ℓ_1, which should be regarded as an **average-case** guarantee in the sense that it allows missing a subset of the heavy frequencies if they carry the energy proportional to the noise level.

Previous Work on ℓ_∞/ℓ_1 with arbitrary linear measurements

All approaches described above concerned Fourier measurements, but compressed sensing has a long history using arbitrary linear measurements, for example [19, 56, 35, 25, 28, 48, 26, 45, 44, 49]. Regarding ℓ_∞/ℓ_1, the work of [51] indicated a connection between the aforementioned guarantee and incoherent matrices. More specifically, it was shown that given a $(1/k)$-incoherent matrix one can design an algorithm satisfying the ℓ_∞/ℓ_1 guarantee. The existence of a matrix with $O(k^2 \min\{\log n, (\log n/\log k)^2\})$ rows was also proved. Reconstruction needed $\Omega(nk)$ time, something which was partially remedied by Li and Nakos [44] with a scheme of $O(k^2 \log n \cdot \log^* k)$ measurements and $\mathrm{poly}(k, \log n)$ decoding time. Incoherent matrices are interesting objects on their own, and have been studied before, as they can be used to obtain RIP matrices. Deterministic constructions of $O(k^2(\log n/\log k)^2)$ rows were obtained by DeVore [18] using deep results from the theory of Gelfand widths and by Amini and Marvasti [5] via binary BCH code vectors, where the zeros are replaced by -1s. We note that incoherent matrices matching this bound also follow immediately from the famous Nisan-Wigderson combinatorial designs [52], and serve as a cornerstone for constructions of pseudorandom generators and extractors [59]. Incoherent matrices are also connected with ϵ-biased codes, and thus an almost optimal strongly explicit construction can be obtained by the recent breakthrough work of [57]. On the lower bound side, Alon has shown that $\Omega(k^2 \log n/\log k)$ rows are necessary for a $(1/k)$-incoherent matrix [4].

Our Contribution

In this work we offer several new results for the Sparse Fourier Transform problem across different axis, some of which are nearly optimal. We show how to find in polynomial time a deterministic collection of samples from the time domain, such that we can solve the Sparse Fourier Transform problem in linear and sublinear time and achieve nearly optimal sample complexity. For the closely related problem of incoherent matrices from DFT rows, which is of independent interest, we obtain a nearly optimal derandomized construction via Bernstein's inequality. We also demonstrate strongly explicit constructions, by invoking heavy number-theoretical machinery.

We note that the bounds of our constructions have been known for more than a decade if the sensing/incoherent matrix is allowed to be arbitrary. However, the previous arguments did not facilitate the frequent and relevant scenario where we have access to rows only from the Fourier ensemble. Part of our work is to show that some of these results carry over to the significantly more constrained case. We also note that any progress to deterministic ℓ_2/ℓ_1 schemes with subquadratic sample complexity is connected to the very challenging problem of obtaining a deterministic DFT row-subsampled RIP matrices with subquadratic number of rows[2] which possibly out of reach at the moment.

2 Technical Results

2.1 Preliminaries

For a positive integer n, we define $[n] = \{0, 1 \ldots, n-1\}$ and we shall index the coordinates of a n-dimensional vector or the rows/columns of an $n \times n$ matrix from 0 to $n-1$. We define the Discrete Fourier Transform (DFT) matrix $F \in \mathbb{C}^{n\times n}$ to be the unitary matrix such that $F_{ij} = \frac{1}{\sqrt{n}} e^{2\pi\sqrt{-1}\cdot ij/n}$, and the Discrete Fourier Transform of a vector $x \in \mathbb{C}^n$ to be $\widehat{x} = Fx$.

For a set $S \subseteq [n]$ we define x_S to be the vector obtained from x after zeroing out the coordinates not in S. We also define $H(x, k)$ to be the set of the indices of the largest k coordinates (in magnitude) of x, and $x_{-k} = x_{[n]\setminus H(x,k)}$. We say x is k-sparse if $x_{-k} = 0$. We also define $\|x\|_p = \left(\sum_{i=0}^{n-1} |x_i|^p\right)^{1/p}$ for $p \geq 1$ and $\|x\|_0$ to be the number of nonzero coordinates of x.

For a matrix $F \in \mathbb{C}^{n\times n}$ and subsets $S, T \subseteq [n]$, we define $F_{S,T}$ to be the submatrix of F indexed by rows in S and columns in T.

The median of a collection of complex numbers $\{z_i\}$ is defined to be $\operatorname{median}_i z_i = \operatorname{median}_i \operatorname{Re}(z_i) + \sqrt{-1}\operatorname{median}_i \operatorname{Im}(z_i)$, i.e., taking the median of the real and the imaginary component separately.

For two points x and y on the unit circle, we use $|x-y|_\circ$ to denote the circular distance (in radians, i.e. modulo 2π) between x and y.

2.1.1 ℓ_∞/ℓ_1 Gurantee and incoherent matrices

The quality of the approximation is usually measured in different error metrics, and the main recovery guarantee we are interested in is called the ℓ_∞/ℓ_1 guarantee, as defined in Definition 2. Other types of recovery guarantee, such as the ℓ_∞/ℓ_2, the ℓ_2/ℓ_2 and the

[2] Note that [10] breaks the quadratic barrier for RIP matrices but does not use the Fourier ensemble; the rows are picked from the *discrete chirp-Fourier* ensemble, where the linear functions are substituted by quadratic polynomials.

ℓ_2/ℓ_1, are defined similarly, where (1) is replaced with the respective expression in Table 1. Note that these are definitions of the error guarantee per se and do not have algorithmic requirements on the scheme.

Highly relevant with the ℓ_∞/ℓ_1 guarantee is a matrix condition which we call incoherence.

▶ **Definition 3** (Incoherent Matrix). *A matrix $A \in \mathbb{C}^{m \times n}$ is called ϵ-incoherent if $\|A_i\|_2 = 1$ for all i (where A_i denotes the i-th column of A) and $|\langle A_i, A_j \rangle| \leq \epsilon$.*

▶ **Lemma 4** ([51]). *There exist an absolute constant $c > 0$ such that for any (c/k)-incoherent matrix A, there exists a ℓ_∞/ℓ_1-scheme which uses A as the measurement matrix and whose recovery algorithm runs in polynomial time.*

2.1.2 The Restrictred Isometry Property and its connection with incoherence

Another highly relevant condition is called the renowned restricted isometry property, introduced by Candès et al. in [11]. We show how incoherent matrices are connected to it.

▶ **Definition 5** (Restricted Isometry Property). *A matrix $A \in \mathbb{C}^{m \times n}$ is said to satisfy the (k, ϵ) Restricted Isometry Property (RIP), if for all $x \in \mathbb{C}^n$ with $\|x\|_0 \leq k$, it holds that $(1 - \epsilon)\|x\|_2 \leq \|Ax\|_2 \leq (1 + \epsilon)\|x\|_2$.*

Candès et al. proved in their breakthrough paper [11] that any RIP matrix can be used for sparse recovery with the ℓ_2/ℓ_1 error guarantee. The following formulation comes from [22, Theorem 6.12].

▶ **Lemma 6.** *Given a $(2k, \epsilon)$-RIP matrix A with $\epsilon < 4/\sqrt{41}$, we can design a ℓ_2/ℓ_1-scheme that uses A as the measurement matrix and has a recovery algorithm that runs in polynomial time.*

Although randomly subsampling the DFT matrix gives an RIP matrix with $O(k\text{log}^2k\text{log}n)$ rows [32], no algorithm for finding these rows in polynomial time is known; actually, even for $o(k^2) \cdot \text{poly}(\log n)$ rows the problem remains wide open[3]. It is a very important and challenging problem whether one can have an explicit construction of RIP matrices from Fourier measurements that break the quadratic barrier on k.

We state the following two folklore results, connecting the two different guarantees, and their associated combinatorial objects. This indicates the importance of incoherent matrices for the field of compressed sensing.

▶ **Proposition 7** (folklore). *An ℓ_∞/ℓ_1 scheme with a measurement matrix of m rows and recovery time T induces an ℓ_2/ℓ_1 scheme of a measurement matrix of $O(m)$ rows and recovery time $O(T + \|\widehat{x}'\|_0)$, where $\widehat{x}'$ is the output of the ℓ_∞/ℓ_1 scheme.*

▶ **Proposition 8** (folklore). *A (c/k)-incoherent matrix is also a (k, c)-RIP matrix.*

[3] In fact, one of the results of our paper gives the state-of-the-art result even for this problem, with $O(k^2 \log n)$ rows, see Theorem 12.

Table 2 Comparison of our results and the previous results. All O- and Ω-notations are suppressed. The result in the first row follows from Lemma 6 and the RIP matrix in [32].Our algorithms adopt the common assumption in the sparse FT literature that the signal-to-noise ratio is bounded by n^c for some absolute constant $c > 0$.

	Samples	Run-time	Guarantee	Explict Construction	Lower Bound
[32]	$k \log^2 k \log n$	$\mathrm{poly}(n)$	ℓ_2/ℓ_1	No	$k \log(n/k)$
[47]	$k^2 \log^{5.5} n / \log k$	$k^2 \log^{5.5} n / \log k$	ℓ_2/ℓ_1	Yes	$k \log(n/k)$
Theorem 9	$k^2 \log n$	$nk \log^2 n$	ℓ_∞/ℓ_1	Yes	$k^2 + k \log n$[51]
Theorem 10	$k^2 \log^2 n$	$k^2 \log^3 n$	ℓ_∞/ℓ_1	Yes	$k^2 + k \log n$[51]

2.2 Our results

2.2.1 Sparse Fourier Transform Algorithms

▶ **Theorem 9** (Deterministic SFT with super-linear time). *Let n be a power of 2. There exist a set $S \subseteq [n]$ with $|S| = O(k^2 \log n)$ and an absolute constant $c > 0$ such that the following holds. For any vector $x \in \mathbb{C}^n$ with $\|\widehat{x}\|_\infty \leq n^c \|\widehat{x}_{-k}\|_1/k$, one can find an $O(k)$-sparse vector $\widehat{x}' \in \mathbb{C}^n$ such that*

$$\|\widehat{x} - \widehat{x}'\|_\infty \leq \frac{1}{k}\|\widehat{x}_{-k}\|_1,$$

in time $O(nk \log^2 n)$ by accessing $\{x_i\}_{i \in S}$ only. Moreover, the set S can be found in $\mathrm{poly}(n)$ *time.*

▶ **Theorem 10** (Deterministic SFT with sublinear time,). *Let n be a power of 2. There exist a set $S \subseteq [n]$ with $|S| = O(k^2 \log^2 n)$ and an absolute constant $c > 0$ such that the following holds. For any vector $x \in \mathbb{C}^n$ with $\|\widehat{x}\|_\infty \leq n^c \|\widehat{x}_{-k}\|_1/k$, one can find an $O(k)$-sparse vector $\widehat{x}' \in \mathbb{C}^n$ such that*

$$\|\widehat{x} - \widehat{x}'\|_\infty \leq \frac{1}{k}\|\widehat{x}_{-k}\|_1,$$

in time $O(k^2 \log^3 n)$ by accessing $\{x_i\}_{i \in S}$ only. Moreover, the set S can be found in $\mathrm{poly}(n)$ *time.*

▶ Remark 11. The condition $\|\widehat{x}\|_\infty \leq n^c \|\widehat{x}_{-k}\|_1/k$ upper bounds the "signal-to-noise ratio", a common measure in engineering that compares the level of a desired signal to the level of the background noise. This is a common assumption in most algorithms in the Sparse Fourier Transform literature, see, e.g. [30, 33, 39, 13, 40], where the ℓ_2-norm variant $\|\widehat{x}\|_\infty \leq n^c \|\widehat{x}_{-k}\|_2/\sqrt{k}$ was assumed.

2.2.2 From DFT to incoherent matrices

This section contains deterministic constructions of incoherent matrices.

An Explicit Construction: Derandomization in poly(n) time

▶ **Theorem 12** (Incoherent matrices by derandomized subsampling of DFT). *There exists a set $S \subseteq [n]$ with of cardinality $O(k^2 \log n)$ such that the matrix $\sqrt{\frac{n}{m}} F_{S,[n]}$ is $(1/k)$-incoherent. Moreover, S can be found in* $\mathrm{poly}(n)$ *time.*

The above Theorem yields immediately a different algorithm for ℓ_∞/ℓ_1 Sparse Fourier Tranform with $O(k^2 \log n)$ samples, via the reduction in [51].

Strongly explicit constructions: Derandomization in sub-linear time

▶ **Theorem 13** (Incoherent matrices from DFT via low-degree polynomials). *Let $\epsilon > 0$ be a constant small enough, p be a prime and $d \geq 2$ be an integer. There exists a strongly explicit construction of an $O(m^\epsilon(\frac{1}{m} + \frac{p}{m^d})^{2^{1-d}})$-incoherent matrix $M \in \mathbb{C}^{m\times p}$ such that the rows of $\sqrt{m}M$ are rows of the DFT matrix (a row may appear more than once). The hidden constant in the O-notation depends on d and ϵ. Finding the indices of the rows takes $\widetilde{O}(m)$ time.*

To get an idea of the above result one could for example set $d = 3$ and observe that the results translates to the following: for every $k \geq p^{1/8}$ one can get a $(1/k)$-incoherent matrix with $O(k^{4+\epsilon})$ rows. One needs the condition on k (or equivalently the condition on m) to bound the term p/m^d. The larger the degree d, the looser this condition, but also the worse the dependence of m on k. For example, when $d = 4$, we can expand the regime of k to approximately $k \geq p^{1/24}$, but obtain approximately $m = O(k^{8+\epsilon})$.

The following is a different construction, incomparable with Theorem 13 in multiple ways. First, the construction runs in sublinear time in p but it is not strongly explicit. Second, it gives different trade-offs between the sparsity parameter and the number of rows. Last but not least, the construction depends on the factorization of $p-1$.

▶ **Theorem 14** (Incoherent matrices from DFT via multiplicative subgroupss). *Let p be a prime number. For every divisor d of $p-1$ such that $d > \sqrt{p}$ we can find in time $O(d\log p)$ a matrix $M \in \mathbb{C}^{d\times p}$ with rows being the rows of the DFT matrix such that $\frac{1}{d}M$ is $(\sqrt{p}/d)$-incoherent.*

This result could give (depending on the factorization of $p-1$) a better polynomial dependence of m on k in the high-sparsity regime. If $p-1$ has a large divisor about $p^{1-\gamma}$, this would yield a matrix with sparsity parameter $k \approx p^\gamma$ and $m \approx k^{1/\gamma-1}$ rows. For example, when $\gamma = 1/4$, we obtain $k \approx p^{1/4}$ and $m \approx k^3$, which cannot be obtained from Theorem 13. In general, Theorem 14 will yield useful matrices as long as $p-1$ has divisors in the range $[\sqrt{p}, p-1]$, ideally as many as possible. An extreme case is Fermat primes, which have $(\log p)/2$ divisors in the aforesaid interval.

The reader might ask the question if the polynomial dependence of k on p is necessary; ideally one would like a logarithmic dependence, since the polynomial dependence is interesting only in the high-sparsity regime. Regarding strongly explicit constructions, we provide some evidence why this might be a very hard problem in the remark below.

▶ Remark 15. The inferiority of our bounds in the low-sparsity regime is justifiable to some extent: it is because of a common obstacle that has persisted more than a century in the theory of exponential sums, due to the lack of techniques to account for sparse character sums (either additive or multiplicative). In general, the fewer summands the sum has, the harder it is to prove a tight cancellation bound. Thus, owing to the use of heavy machinery from analytic number theory and more specifically the theory of exponential sums over finite fields, our bounds for strongly explicit constructions are quite suboptimal.

2.3 Comparing ℓ_2/ℓ_1 with ℓ_∞/ℓ_1

In this subsection we elaborate why ℓ_∞/ℓ_1 is much stronger than ℓ_2/ℓ_1, and not just a guarantee that implies ℓ_2/ℓ_1. Let $\gamma < 1$ be a constant and consider the following scenario. There are three sets A, B, C of size γk, $(1-\gamma)k$, $n-k$ respectively, and for every $i \in A$ we have $|\widehat{x}_i| = \frac{2}{k}\|\widehat{x}_C\|_1 = \frac{2}{k}\|\widehat{x}_{-k}\|_1$, while every coordinate in B and C has the equal magnitude. It follows immediately that

$$\|\widehat{x}_C\|_1 = \frac{n-k}{n-\gamma k}\|\widehat{x}_{B\cup C}\|_1.$$

Now assume that $k \le \gamma n$, then $(n - \gamma k)/(n - k) \le 1 + \gamma$. We claim that the zero vector is a valid solution for the ℓ_2/ℓ_1 guarantee, since

$$\begin{aligned}\|\vec{0} - \widehat{x}\|_2^2 &= \|\widehat{x}_A\|_2^2 + \|\widehat{x}_{B\cup C}\|_2^2 \\ &\le \gamma k \cdot \frac{4}{k^2}\|\widehat{x}_{-k}\|_1^2 + \frac{1}{(n-\gamma k)}\|\widehat{x}_{B\cup C}\|_1^2 \\ &\le \frac{4\gamma}{k}\|\widehat{x}_{-k}\|_1^2 + \frac{n-\gamma k}{(n-k)^2}\|\widehat{x}_C\|_1^2 \\ &\le \left(\frac{4\gamma}{k} + \frac{1+\gamma}{n-k}\right)\|\widehat{x}_{-k}\|_1^2 \\ &\le \frac{5\gamma}{k}\|\widehat{x}_{-k}\|_1^2,\end{aligned}$$

where the last inequality follows provided it further holds that $k \le \gamma n/(2\gamma + 1)$. Hence when $\gamma \le 1/5$, we see that the zero vector satisfies the ℓ_2/ℓ_1 guarantee.

Since $\vec{0}$ is a possible output, we may not recover any of the coordinates in S, which is the set of "interesting" coordinates. On the other hand, the ℓ_∞/ℓ_1 guarantee does allow the recovery of **every** coordinate in S. This is a difference of recovering all γk versus 0 coordinates. We conclude from the discussion above that in the case of too much noise, the ℓ_2/ℓ_1 guarantee becomes much weaker than the ℓ_∞/ℓ_1, possibly giving meaningless results in some cases.

3 Overview

Sparse Fourier Transform Algorithms

We first show how to achieve the for-all schemes, i.e., schemes that allow universal reconstruction of all vectors, and then derandomize them. Similarly to the previous works [31, 33, 40], our algorithm hashes, with the filter in [40], the spectrum of x to $O(k)$ buckets using pseudorandom permutations, and repeat $O(k \log n)$ times with fresh randomness. The main part of the analysis is to show that for any vector $\widehat{x} \in \mathbb{C}^n$ and any set $S \subseteq [n]$ with $|S| \le k$, each $i \in S$, in a constant fraction of the repetitions, receives "low noise" from all other elements, under the pseudorandom permutations. This will boil down to a set of $\Theta(n^2)$ inequalities involving the filter and the pseudorandom permutations. We prove these inequalities with full randomness, and then derandomize the pseudorandom permutations using the method of conditional expectations. This will give us Theorem 9. To do so, we choose the pseudorandom permutations one at a time, repetition by repetition, and keep an (intricate) pessimistic estimator , which we update accordingly. Our argument extends the arguments in [51] and [54], and could be of independent interest. To compare with [51] we have the following observation. The construction in [51] consists of $O(k \log n)$ matrices, joined vertically, each having $O(k)$ rows and exactly one 1 per column. This ensures a small incoherence of the concatenated matrix and gives the ℓ_∞/ℓ_1 guarantee. In the Fourier case, the convolution with the filter functions behaves analogously: instead of having exactly one non-zero element, each column in the ℓ-th matrix has a contiguous segment of 1s of size $\approx n/k$ (where the center of that segment is depends on the choice of the ℓ-th pseudorandom permutation) and polynomially decaying entries away from this segment. Moreover, the positions of the segments across the columns are not fully independent and are defined via the pseudorandom permutations. We show that even in this more restricted setting, derandomization is possible in polynomial time. Several details are omitted in the preceding high-level discussion and we suggest the reader look at the corresponding sections for the complete argument.

The sublinear-time algorithm (Theorem 10) is obtained by bootstrapping the derandomized scheme above with an identification procedure in each bucket, as most previous algorithms have done (e.g. [30]). The major difference is that our identification procedure needs to be deterministic. We show an explicit set of samples that allow the implementation of the desired routine. To illustrate our idea, let us focus on the following 1-sparse case: $\widehat{x} \in \mathbb{C}^n$ and $|\widehat{x}_{i^*}| \geq 3\|\widehat{x}_{[n]\setminus i^*}\|_1$ for some i^*, which we want to locate. Let

$$\theta_j = \left(\frac{2\pi}{n}j\right) \bmod 2\pi,$$

and consider the $\log n$ samples $x_0, x_1, x_2, x_4, \ldots, x_{2^{r-1}}, \ldots$.

Observe that (ignoring $1/\sqrt{n}$ factors)

$$x_\beta = \widehat{x}_{i^*} e^{\sqrt{-1}\beta\theta_{i^*}} + \sum_{j \neq i^*} \widehat{x}_j e^{\sqrt{-1}\beta\theta_j},$$

we can find $\beta\theta_{i^*} + \arg \widehat{x}_{i^*}$ up to $\pi/8$, just by estimating the phase of x_β. Thus we can estimate $\beta\theta_{i^*}$ up to $\pi/4$ from the phase of x_β/x_0. If $i^* \neq j$, then there exists a $\beta \in \{1, 2, 2^2, \ldots, 2^{r-1}, \ldots\}$ such that $|\beta\theta_{i^*} - \beta\theta_j|_\circ > \pi/2$, and so $\beta\theta_j$ will be more than $\pi/4$ away from the phase of the measurement. Thus, by iterating over all $j \in [n]$, we keep the index j for which $\beta\theta_j$ is within $\pi/4$ from $\arg(x_\beta/x_0)$, for every β that is a power of 2 in $\mathbb{Z}_n$.

Unfortunately, although this is a deterministic collection of $O(\log n)$ samples, the above argument gives only $O(n \log n)$ time. For sublinear-time decoding we use x_1/x_0 to find a sector S_0 of the unit circle of length $\pi/4$ that contains θ_{i^*}. Then, from x_2/x_0 we find two sectors of length $\pi/8$ each, the union of which contains θ_{i^*}. Because these sectors are antipodal on the unit circle, the sector S_0 intersects exactly one of those, let the intersection be S_1. The intersection is a sector of length at most $\pi/8$. Proceeding iteratively, we halve the size of the sector at each step, till we find θ_{i^*}, and infer i^*. Plugging this idea in the whole k-sparse recovery scheme yields the desired result. Our argument crucially depends on the fact that in the ℓ_1 norm the phase of θ_{i^*} will always dominate the phase of all samples we take.

Incoherent Matrices from the Fourier ensemble

Our first result for incoherent matrices (Theorem 12) is more general and works for any matrix that has orthonormal columns with entries bounded by $O(1/\sqrt{n})$. We subsample the matrix, invoke a Chernoff bound and Bernstein's inequality to show the small incoherence of the subsampled matrix. We follow a derandomization procedure which essentially mimics the proof of Bernstein's inequality, keeping a pessimistic estimator which corresponds to the sum of the generating functions of the probabilities of all events we want to hold, evaluated at specific points. We obtain an explicit construction, i.e. a derandomization in poly(n) time. This argument could be of independent interest for its generality. As there are many technical obstacles to overcome, we suggest the reader take a careful look at the proof to gain a clearer picture of the argument.

Our next results (Theorem 13 and Theorem 14) construct *strongly explicit* incoherent matrices by making use of technology from the fruitful theory of exponential sums in analytic number theory and additive combinatorics. Roughly speaking, to bound a complex exponential sum over a set S, one would expect that specific choices of the set S lead to non-trivial bounds, i.e. $o(|S|)$, since cancellation takes place in the summation. Ideally, one would desire that the exponentials behave like a random walk and give the optimal cancellation of $O(\sqrt{|S|})$. This intuition is clearly not true, but the results by Weyl and others show that certain sets S can exhibit a nicer behaviour. We exploit their results to build incoherent matrices by taking the rows of the DFT matrix indexed by the "nice" sets. This connection also yields an immediate improvement on the lower bound of an exponential sum obtained by Winterhof [60].

4 Open Problems and Future Direction

A direction of research is to design deterministic schemes that break the quadratic barrier for signals with structured Fourier support. For example, subsampling the rows of the DFT matrix to obtain RIP matrices depends highly on the structure of the vectors we would like to preserve. The more additive structure the support of a k-sparse vector x has, the worse is the concentration of a random Fourier coefficient of x. Equivalently, the less additive structure the support of x has, the flatter its Fourier transform is, and hence, the better concentration bounds we obtain. The concentration in the extreme case, when the support of x is "dissociated", is captured by the renowned Rudin's inequality in additive combinatorics (see, e.g. [58, Lemma 4.33]). We thus believe that it is an interesting direction to use machinery from the field of additive combinatorics and the relevant fields in order to obtain new constructions and algorithms, at least for interesting subclasses of structured signals.

References

1 Adi Akavia. Deterministic sparse Fourier approximation via fooling arithmetic progressions. In *COLT*, pages 381–393, 2010.

2 Adi Akavia. Deterministic sparse Fourier approximation via approximating arithmetic progressions. *IEEE Transactions on Information Theory*, 60(3):1733–1741, 2014.

3 Adi Akavia, Shafi Goldwasser, and Shmuel Safra. Proving hard-core predicates using list decoding. In *FOCS*, volume 44, pages 146–159, 2003.

4 Noga Alon. Perturbed identity matrices have high rank: Proof and applications. *Combinatorics, Probability and Computing*, 18(1-2):3–15, 2009.

5 Arash Amini and Farokh Marvasti. Deterministic construction of binary, bipolar, and ternary compressed sensing matrices. *IEEE Transactions on Information Theory*, 57(4):2360–2370, 2011.

6 Haim Avron, Michael Kapralov, Cameron Musco, Christopher Musco, Ameya Velingker, and Amir Zandieh. A universal sampling method for reconstructing signals with simple Fourier transforms. *arXiv preprint*, 2018. arXiv:1812.08723.

7 Afonso S Bandeira, Megan E Lewis, and Dustin G Mixon. Discrete uncertainty principles and sparse signal processing. *Journal of Fourier Analysis and Applications*, pages 1–22, 2017.

8 Sina Bittens, Ruochuan Zhang, and Mark A Iwen. A deterministic sparse FFT for functions with structured Fourier sparsity. *Advances in Computational Mathematics, to appear.*, 2017.

9 Petros Boufounos, Volkan Cevher, Anna C Gilbert, Yi Li, and Martin J Strauss. What's the frequency, Kenneth?: Sublinear Fourier sampling off the grid. In *Algorithmica(A preliminary version of this paper appeared in the Proceedings of RANDOM/APPROX 2012, LNCS 7408, pp.61–72)*, pages 1–28. Springer, 2014.

10 Jean Bourgain, Stephen J Dilworth, Kevin Ford, Sergei V Konyagin, and Denka Kutzarova. Breaking the k^2 barrier for explicit RIP matrices. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 637–644. ACM, 2011.

11 Emmanuel J Candes, Justin K Romberg, and Terence Tao. Stable signal recovery from incomplete and inaccurate measurements. *Communications on pure and applied mathematics*, 59(8):1207–1223, 2006.

12 Emmanuel J Candes and Terence Tao. Near-optimal signal recovery from random projections: Universal encoding strategies? *IEEE transactions on information theory*, 52(12):5406–5425, 2006.

13 Volkan Cevher, Michael Kapralov, Jonathan Scarlett, and Amir Zandieh. An adaptive sublinear-time block sparse Fourier transform. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 702–715. ACM, 2017.

14 Xue Chen, Daniel M Kane, Eric Price, and Zhao Song. Fourier-sparse interpolation without a frequency gap. In *Foundations of Computer Science (FOCS), 2016 IEEE 57th Annual Symposium on*, pages 741–750. IEEE, 2016.

15 Mahdi Cheraghchi and Piotr Indyk. Nearly optimal deterministic algorithm for sparse walsh-hadamard transform. *ACM Transactions on Algorithms (TALG)*, 13(3):34, 2017.

16 Barry A Cipra. The best of the 20th century: Editors name top 10 algorithms. *SIAM news*, 33(4):1–2, 2000.

17 Albert Cohen, Wolfgang Dahmen, and Ronald DeVore. Compressed sensing and best k-term approximation. *Journal of the American mathematical society*, 22(1):211–231, 2009.

18 Ronald A DeVore. Deterministic constructions of compressed sensing matrices. *Journal of complexity*, 23(4):918–925, 2007.

19 Khanh Do Ba, Piotr Indyk, Eric Price, and David P Woodruff. Lower bounds for sparse recovery. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 1190–1197. SIAM, 2010.

20 David L. Donoho. Compressed sensing. *IEEE Trans. Information Theory*, 52(4):1289–1306, 2006.

21 Simon Foucart, Alain Pajor, Holger Rauhut, and Tino Ullrich. The gelfand widths of ℓ_p-balls for $0 < p \leq 1$. *Journal of Complexity*, 26(6):629–640, 2010.

22 Simon Foucart and Holger Rauhut. *A Mathematical Introduction to Compressive Sensing*. Applied and Numerical Harmonic Analysis. Birkhäuser Basel, 2013.

23 Sumit Ganguly. Lower bounds on frequency estimation of data streams. In *International Computer Science Symposium in Russia*, pages 204–215. Springer, 2008.

24 Anna C Gilbert, Sudipto Guha, Piotr Indyk, S Muthukrishnan, and Martin Strauss. Near-optimal sparse Fourier representations via sampling. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 152–161. ACM, 2002.

25 Anna C Gilbert, Yi Li, Ely Porat, and Martin J Strauss. Approximate sparse recovery: optimizing time and measurements. *SIAM Journal on Computing 2012 (A preliminary version of this paper appears in STOC 2010)*, 41(2):436–453, 2010.

26 Anna C Gilbert, Yi Li, Ely Porat, and Martin J Strauss. For-all sparse recovery in near-optimal time. *ACM Transactions on Algorithms (TALG)*, 13(3):32, 2017.

27 Anna C Gilbert, S Muthukrishnan, and Martin Strauss. Improved time bounds for near-optimal sparse Fourier representations. In *Optics & Photonics 2005*, pages 59141A–59141A. International Society for Optics and Photonics, 2005.

28 Anna C Gilbert, Hung Q Ngo, Ely Porat, Atri Rudra, and Martin J Strauss. ℓ_2/ℓ_2-foreach sparse recovery with low risk. In *International Colloquium on Automata, Languages, and Programming*, pages 461–472. Springer, 2013.

29 Oded Goldreich and Leonid A Levin. A hard-core predicate for all one-way functions. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 25–32. ACM, 1989.

30 Haitham Hassanieh, Piotr Indyk, Dina Katabi, and Eric Price. Nearly optimal sparse Fourier transform. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 563–578. ACM, 2012.

31 Haitham Hassanieh, Piotr Indyk, Dina Katabi, and Eric Price. Simple and practical algorithm for sparse Fourier transform. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 1183–1194. SIAM, 2012.

32 Ishay Haviv and Oded Regev. The restricted isometry property of subsampled Fourier matrices. In *SODA*, pages 288–297. arXiv preprint, 2016. `arXiv:1507.01768`.

33 Piotr Indyk and Michael Kapralov. Sample-optimal Fourier sampling in any constant dimension. In *Foundations of Computer Science (FOCS), 2014 IEEE 55th Annual Symposium on*, pages 514–523. IEEE, 2014.

34 Piotr Indyk, Michael Kapralov, and Eric Price. (Nearly) Sample-optimal sparse Fourier transform. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 480–499. SIAM, 2014.

35 Piotr Indyk, Eric Price, and David P Woodruff. On the power of adaptivity in sparse recovery. In *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*, pages 285–294. IEEE, 2011.

36 Mark A Iwen. A deterministic sub-linear time sparse Fourier algorithm via non-adaptive compressed sensing methods. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 20–29. Society for Industrial and Applied Mathematics, 2008.

37 Mark A Iwen. Combinatorial sublinear-time Fourier algorithms. *Foundations of Computational Mathematics*, 10(3):303–338, 2010.

38 Mark A Iwen. Improved approximation guarantees for sublinear-time Fourier algorithms. *Applied And Computational Harmonic Analysis*, 34(1):57–82, 2013.

39 Michael Kapralov. Sparse Fourier transform in any constant dimension with nearly-optimal sample complexity in sublinear time. In *Symposium on Theory of Computing Conference, STOC'16, Cambridge, MA, USA, June 19-21, 2016*, 2016.

40 Michael Kapralov. Sample efficient estimation and recovery in sparse FFT via isolation on average. In *Foundations of Computer Science (FOCS), 2017 IEEE 58th Annual Symposium on*, pages 651–662. Ieee, 2017.

41 Michael Kapralov, Ameya Velingker, and Amir Zandieh. Dimension-independent sparse Fourier transform. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2709–2728. SIAM, 2019.

42 Eyal Kushilevitz and Yishay Mansour. Learning decision trees using the Fourier spectrum. *SIAM Journal on Computing*, 22(6):1331–1348, 1993.

43 David Lawlor, Yang Wang, and Andrew Christlieb. Adaptive sub-linear time Fourier algorithms. *Advances in Adaptive Data Analysis*, 5(01):1350003, 2013.

44 Yi Li and Vasileios Nakos. Deterministic heavy hitters with sublinear query time. In *APPROX-RANDOM*, volume 116 of *LIPIcs*, pages 18:1–18:18. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.

45 Yi Li, Vasileios Nakos, and David P. Woodruff. On low-risk heavy hitters and sparse recovery schemes. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2018, August 20-22, 2018 - Princeton, NJ, USA*, pages 19:1–19:13, 2018. `doi:10.4230/LIPIcs.APPROX-RANDOM.2018.19`.

46 Yishay Mansour. Randomized interpolation and approximation of sparse polynomials. In *International Colloquium on Automata, Languages, and Programming*, pages 261–272. Springer, 1992.

47 Sami Merhi, Ruochuan Zhang, Mark A Iwen, and Andrew Christlieb. A new class of fully discrete sparse Fourier transforms: Faster stable implementations with guarantees. *Journal of Fourier Analysis and Applications*, 2018.

48 Vasileios Nakos, Xiaofei Shi, David P. Woodruff, and Hongyang Zhang. Improved algorithms for adaptive compressed sensing. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, pages 90:1–90:14, 2018. `doi:10.4230/LIPIcs.ICALP.2018.90`.

49 Vasileios Nakos and Zhao Song. Stronger l_2/l_2 compressed sensing; without iterating. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 289–297, 2019. `doi:10.1145/3313276.3316355`.

50 Vasileios Nakos, Zhao Song, and Zhengyu Wang. (nearly) sample-optimal sparse fourier transform in any dimension; ripless and filterless. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1568–1577. IEEE, 2019.

51 Jelani Nelson, Huy L Nguyên, and David P Woodruff. On deterministic sketching and streaming for sparse recovery and norm estimation. *Linear Algebra and its Applications*, 441:152–167, 2014.

52 Noam Nisan and Avi Wigderson. Hardness vs randomness. *Journal of computer and System Sciences*, 49(2):149–167, 1994.

53 Sameer Pawar and Kannan Ramchandran. A robust R-FFAST framework for computing a k-sparse n-length DFT in O(k log n) sample complexity using sparse-graph codes. In *Information Theory (ISIT), 2014 IEEE International Symposium on*, pages 1852–1856. IEEE, 2014.

54 Ely Porat and Amir Rothschild. Explicit non-adaptive combinatorial group testing schemes. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *Automata, Languages and Programming*, pages 748–759, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

55 Eric Price and Zhao Song. A robust sparse Fourier transform in the continuous setting. In *Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on*, pages 583–600. IEEE, 2015.

56 Eric Price and David P Woodruff. (1+ eps)-approximate sparse recovery. In *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*, pages 295–304. IEEE, 2011.

57 Amnon Ta-Shma. Explicit, almost optimal, epsilon-balanced codes. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2017, pages 238–251, New York, NY, USA, 2017. ACM.

58 Terence Tao and Van H Vu. *Additive combinatorics*, volume 105. Cambridge University Press, 2006.

59 Luca Trevisan. Extractors and pseudorandom generators. *Journal of the ACM*, 48(4):860–879, 2001.

60 Arne Winterhof. Incomplete additive character sums and applications. In *Finite fields and applications*, pages 462–474. Springer, 2001.

Faster Random k-CNF Satisfiability

Andrea Lincoln
MIT, Cambridge, MA, USA
andreali@mit.edu

Adam Yedidia
MIT, Cambridge, MA, USA
adamy@mit.edu

Abstract

We describe an algorithm to solve the problem of Boolean CNF-Satisfiability when the input formula is chosen randomly.

We build upon the algorithms of Schöning 1999 and Dantsin et al. in 2002. The Schöning algorithm works by trying many possible random assignments, and for each one searching systematically in the neighborhood of that assignment for a satisfying solution. Previous algorithms for this problem run in time $O(2^{n(1-\Omega(1)/k)})$.

Our improvement is simple: we count how many clauses are satisfied by each randomly sampled assignment, and only search in the neighborhoods of assignments with abnormally many satisfied clauses. We show that assignments like these are significantly more likely to be near a satisfying assignment. This improvement saves a factor of $2^{n\Omega(\lg^2 k)/k}$, resulting in an overall runtime of $O(2^{n(1-\Omega(\lg^2 k)/k)})$ for random k-SAT.

2012 ACM Subject Classification Theory of computation → Random search heuristics

Keywords and phrases Random k-SAT, Average-Case, Algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.78

Category Track A: Algorithms, Complexity and Games

Related Version https://arxiv.org/abs/1903.10618

Funding *Andrea Lincoln*: Some of this work was completed while this student was an intern at VMware. This work supported in part by NSF Grants CCF-1909429, CCF-1417238, CCF-1528078 and CCF-1514339, and BSF Grant BSF:2012338.

Acknowledgements We are very grateful to Greg Valiant, Virginia Williams, and Nikhil Vyas for their helpful conversations and kind support. Additionally, we are very appreciative of the email correspondence we had with Amin Coja-Oghlan, Allan Sly, and Nike Sun, who answered our many questions. We would also like to thank reviewers for their comments and suggestions. We would especially like to thank an exceptionally detailed and helpful review that we received on our paper last year.

1 Introduction

The Boolean Satisfiability problem, known as SAT for short, is one of the best-known and most well-studied problems in computer science (e.g. [28, 1, 19, 3, 14]). In its general form, it describes the following problem: given an input formula ϕ composed of conjunctions, disjunctions, and negations of a list of Boolean-valued variables $(x_1, x_2, \ldots, x_n)$, determine whether or not there exists an assignment of variables to Boolean values such that ϕ evaluates to TRUE. SAT was the first problem shown to be NP-complete [12, 24].

Every Boolean formula ϕ can be written in *conjunctive normal form*, meaning that it is written as the logical conjunction of a series of disjunctive clauses. Each disjunctive clause takes as its value the logical disjunction of a series of *literals*, which takes on either the same value of one of the variables x_i or the negation of that value.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 78; pp. 78:1–78:12
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

If we constrain the input formula ϕ to contain only disjunctive clauses that are of size k or smaller, then that more constrained problem is known as *k-Satisfiability*, or *k-SAT* for short. When $k > 2$ it is known to be NP-complete [22]. As k grows, the best known runtime of the worst-case k-SAT problem, $O(2^{1-1/\Theta(k)})$, grows [21, 28].

It is well-known that in real-world Boolean Satisfiability problems, SAT solvers often vastly outperform the best known theoretical runtimes [15, 31]. One possible explanation for this gap in performance is that most input formulas are easily solved without much computation being necessary, but that there exists a "hard core" of difficult-to-solve formulas that are responsible for the apparent difficulty of worst-case SAT.

Another possible explanation for this gap in performance is that, in practice, people usually care about highly structured formulas that are much easier to solve than typical formulas – according to this explanation, there would be an "easy core" of tractable formulas that are responsible for the apparent simplicity of most practical SAT problems.

To try to distinguish between these two explanations, one can study *random Satisfiability*: Boolean Satisfiability for which the input formula ϕ is chosen according to some known uniform probability distribution D_Φ, and where we expect to be able to return the correct answer (satisfiable or unsatisfiable) with probability that is exponentially close to 1 in the size of the input. Random k-SAT is a very well studied problem (e.g. [3, 16, 27, 10, 7, 26, 33]).

Typically, attention is restricted to k-CNF formulas whose ratio of clauses to variables is *at the threshold*, meaning that the number of clauses m is drawn from a Poisson distribution centered at $d_k n$, where n is the number of variables and d_k is a function of k close to $2^k \ln(2) - \frac{1}{2}(1 + \ln(2))$ [16]. Such formulas are conjectured to be the hardest instances for a given n [13, 31]. It was shown by Ding, Sly, and Sun [16] that away from this threshold, formulas are either overwhelmingly satisfied or overwhelmingly unsatisfied, making the problem less interesting. Notably, away from this threshold one can simply return True or False based on the number of clauses and give the correct answer with high probability. For our purposes, this is a very useful guarantee to have; this is why we use their definition throughout the paper. We go into much greater detail about D_Φ and the threshold in Section 2.

Away from the threshold, polynomial-time algorithms for SAT have been found and analyzed, first by Chao and Franco [26], and later by Coja-Oghlan et al. [10, 7]. Additionally, a recent result by Vyas and Williams [33] re-analyzes the algorithm of Paturi et al. [29] in the case when the input is drawn from a random distribution, and finds the algorithm to run faster on average in this case by a factor of $2^{n\Omega(\lg k)/k}$, giving a running time of $O(2^{n(1-\Omega(\lg k)/k)})$.

We build upon the work of Schöning [30] and Dantsin et al [14] to solve random k-SAT in time $O(2^{n(1-\Omega(\lg^2 k)/k)})$. This represents an algorithmic improvement of $2^{n\Omega(\lg^2 k)/k}$ over the runtime of the algorithm of Paturi et al. as analyzed by Vyas and Williams in [33].

1.1 A New Algorithm

In this paper, we restrict our attention to the problem of random k-CNF Satisfiability in the limit of large k, which approaches general Boolean CNF-Satisfiability. Our algorithm improves upon the previous best known algorithm for solving random k-SAT in the limit of large k, assuming that the input formulas are chosen according to a known uniform distribution.

Our algorithm improves the running time of k-CNF Satisfiability at the threshold by modifying the algorithms of Schöning and Dantsin et al. to only explore in the neighborhood of those sampled assignments that pass an additional test. By adding this test, we get a $2^{n\Omega(\lg^2 k)/k}$ improvement in the runtime of the algorithm. The test is simple: we count how many clauses are satisfied, and if that number is large, only then do we search in the neighborhood of the assignment. In Appendix G.2 of our full version [25], we provide additional motivation for why our improved running time is remarkable.

▶ **Theorem 1** (Main Theorem Informal). *Let ϕ be drawn uniformly at random from formulas at the threshold (defined formally in Section 2). There exists an algorithm, α-*SampleAndTest *(described in Section 3), such that:*

- *If ϕ is satisfiable, then with probability at least $1-3\cdot 2^{-n/(3\ln(2)2^k)}$, α-*SampleAndTest *returns an assignment $\vec{a}$ that satisfies ϕ.*
- *If ϕ is not satisfiable, then α-*SampleAndTest *will return False with certainty.*
- *α-*SampleAndTest*(ϕ) will run in time $O(2^{n(1-\Omega(\lg^2 k)/k)})$.*

A key technique in the proof of our result is connecting a different distribution over inputs (the *planted k-SAT distribution*) to the uniformly random k-SAT distribution. Reductions between planted k-SAT and random k-SAT have been shown in previous work as well [6, 4, 33]. In the planted k-SAT distribution, an assignment, $\vec{a}$, is picked first. The formula ϕ is selected uniformly over k-SAT clauses *conditioned on* $\vec{a}$ satisfying those clauses. As a result, the planted distribution has a bias towards picking formulas that have many satisfying assignments, relative to the uniform distribution over all satisfiable formulas. For this reason, the planted distribution tends to generate easier-to-solve formulas ϕ than the uniform distribution [18]. We also find that the planted distribution is more easily analyzed.

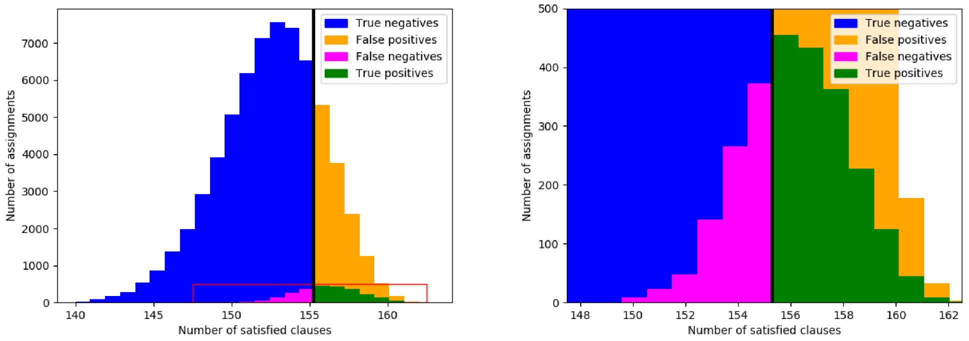
It would be possible, and simpler, to analyze our algorithm only in the planted distribution over formulas. This would *not*, however, correspond to a complete analysis of the algorithm in the random case. In this work, we begin by analyzing the performance of our algorithm when run on inputs drawn from the planted distribution. We show that algorithms with a sufficiently low probability of failure in the planted distribution over input formulas continue to have a low probability of failure in the uniform distribution over input formulas; see Lemma 37 of our full version [25]. Similar reductions have been proven in previous work [6, 4, 33].

The bulk of the analysis of our algorithm presented in this paper will focus on four quantities. Informally:

1. The *true positive rate p_{TP}* describes the fraction of all assignments that are both close to a satisfying assignment in Hamming distance and satisfy a large number of clauses.
2. The *false negative rate p_{FN}* describes the fraction of all assignments that are close to a satisfying assignment in Hamming distance, but do not satisfy a large number of clauses.
3. The *false positive rate p_{FP}* describes the fraction of all assignments that are not close to any satisfying assignment in Hamming distance, but satisfy a large number of clauses.
4. The *true negative rate p_{TN}* describes the fraction of all assignments that are neither close to any assignment in Hamming distance, nor satisfy a large number of clauses.

By showing that the true positive rate is large enough relative to the false positive rate, we show that we do not too often perform a "useless search," i.e. one that will not find a satisfying assignment. And by showing that the true positive rate is large enough relative to the total number of possible assignments, we show that we eventually do find a satisfying assignment without needing to take too many samples. See Fig. 1 for an illustration of these concepts.

To show that our algorithm achieves the desired runtime, we must demonstrate two things. First, we must show that false positives are sufficiently rare; in other words, conditioned on an assignment passing our test, it is sufficiently likely to be a small-Hamming-distance assignment. We prove this in Appendix A of our full version [25]. Second, we must show



(a) The full histogram.

(b) A magnification of the red region shown on the left.

Figure 1 A histogram of how many clauses are satisfied by every possible assignment. In this example, there are $n = 16$ variables, $m = 163$ clauses, and $k = 4$ literals per clause. For the example, we take $T = 155.5$ to be the clause-satisfaction threshold above which we explore further, and $\alpha n = 4$ to be the small-Hamming-distance threshold at which the exhaustive search algorithm finishes. (In actual runs of the algorithm, both of these parameters are selected more conservatively; we chose these parameters for clarity.)

that true positives are sufficiently common; in other words, conditioned on an assignment being close in Hamming distance to a satisfying assignment, it is sufficiently likely to pass our test. We prove this in Appendix B of our full version [25].

We also note that our algorithm can potentially be used as the seed for a worst-case algorithm. Informally, the correctness of the analysis in this paper depends only on the false positive and false negative rates being sufficiently low. As long as the inputs are guaranteed to come from a family of formulas for which this is the case, our algorithm will work even in the worst case. Or, to put it another way, to build a working worst-case algorithm using our algorithm as a template, one may now restrict one's attention to solving input formulas for which assignments in the neighborhood of the solution do *not* have an abnormally-high number of satisfied clauses; our algorithm can solve the others.

1.2 Previous work

Satisfiability and k-SAT have been thoroughly studied. We will cover some of the previous work in the area, focusing on the Random k-SAT problem.

Structural Results About Random k-SAT

To make the study of the Satisfiability of random formulas interesting, it is important to choose the probability distribution over formulas D_ϕ judiciously. In particular, D_ϕ must contain formulas where the ratio of Boolean variables to disjunctive clauses is such that the resulting formulas are neither overwhelmingly satisfiable, nor overwhelmingly unsatisfiable. Let n be the number of variables, and m be the number of clauses. If $n \gg m$, then nearly all formulas chosen uniformly from D_ϕ will be satisfiable; if $m \gg n$, then nearly all formulas will be unsatisfiable. In order for the problem of correctly identifying formulas as satisfiable or unsatisfiable to be nontrivial, we must choose m and n to be at the right ratio. Throughout this paper we will refer to the ratio of m to n as the *density* of a formula.

In work by Ding, Sly and Sun [16], it was shown that a sharp threshold exists between formulas which are satisfied with high probability and those that are unsatisfied with high probability. More precisely, they describe what happens when the number of clauses m is drawn from a Poisson distribution with mean $d_k n$. When the number of clauses drawn is below $(d_k - \epsilon)n$, only an exponentially small fraction of formulas will be unsatisfiable; when the number of clauses drawn is greater than $(d_k + \epsilon)n$, only an exponentially small fraction of formulas will be satisfiable. This holds true for any $\epsilon > 0$ constant in n.

Previous Average-Case k-SAT Algorithms

Feldman et al. studied planted random k-SAT and found that given $m = \Omega(n \lg n)$ clauses, the planted solution can be determined using statistical queries [18]. Feldman et al. also conjecture that planted k-SAT is easier than random k-SAT more generally. Previous work has shown a connection between algorithms that work in the planted distribution and algorithms that work in the random distribution [4, 6, 33]. An algorithm was found by Valiant which runs in time $O(2^{n(1-\Omega(\log(k))/k)})$ at the threshold [32], improving upon PPSZ [28]. Additionally, Vyas and Williams [33] obtained the same runtime by re-analyzing the algorithm of Paturi et al. [29] in the random case.

Some studies of random k-SAT have focused on refutation [9, 8, 20, 5]. Refutation aims to return a short certificate of unsatisfiability. For example Coja-Oghlan, Goerdt, and Lanka give an algorithm that provides refutations with high probability when $k = 3$ and $m > \ln(n)^6 n^{3/2}$. Refutation is quite difficult; note that the m is much larger than the threshold which sits at $\Theta(2^k n)$. We, however, focus on returning a satisfying assignment if the formula is satisfied with high probability.

Some studies of random k-SAT focus on the case where k is small (e.g. [9, 8, 2, 17]). We, however, focus on the asymptotic behavior when k is large.

Worst-Case k-SAT Algorithms

The previously best-known worst-case k-SAT algorithms for large k are due to Paturi et al. who get a running time of $O(2^{n(1-\Omega(1)/k)})$ [28]. Previous work by Schöning gave an algorithm to solve k-SAT in the worst case with an expected running time of $O(2^{n(1-\Omega(1)/k)})$ [30]. Dantsin et al. make the algorithm deterministic [14]. Our algorithm is a modification of the algorithms of both Schöning's and Dantsin et al. Their algorithm runs by choosing an assignment at random, and searching in the immediate neighborhood of that assignment by repeatedly choosing an unsatisfied clause and flipping a variable in that clause to satisfy it. They perform the search near the randomly chosen assignment via an exhaustive search. Their algorithm is an improvement over a naive brute-force algorithm because of the savings that result from only considering variable-flips that could possibly cause the formula to become satisfied (rather than also exploring variable-flips that can't possibly be helpful).

2 Preliminaries

In this section we will give the definition of random k-CNF Satisfiability (random k-CNF SAT) at the threshold. We additionally present definitions of several important distributions and functions that are used later in the paper.

Notation for This Paper

We use $x \sim D$ to indicate that x is a random value drawn from the distribution D.

We use the standard notation that $\lg(n) = \log_2(n)$.

We use $f(x) = O^*(g(x))$ to denote that there exists some constant c such that $f(x) = O(g(x)x^c)$. So, to say it another way, $f(x)$ grows at most as quickly as $g(x)$, ignoring polynomial and constant factors.

We often use "small-Hamming-distance assignment" to mean an assignment that is a small Hamming distance from a satisfying assignment.

Definitions for This Paper

▶ **Definition 2.** *Let $D_{replace}(n, k)$ be the uniform distribution over clauses on k variables where those k variables are chosen with replacement (e.g. $(\bar{x} \vee x \vee y)$ would be a valid clause). Under this definition, there are $n^k 2^k$ possible clauses.*

▶ **Definition 3.** *The density of a formula ϕ with m clauses and n variables is m/n.*

We will now define the *satisfiability threshold.* Informally, this is a density of clauses such that formulas drawn from below this threshold are with high probability (whp) satisfied, and those formulas drawn from above the threshold are whp unsatisfied.

▶ **Definition 4.** *The satisfiability threshold, d_k, is a ratio of clauses to variables such that for all $\epsilon > 0$:*

- *If m is drawn from $Pois[(d_k - \epsilon)n]$, the Poisson distribution with mean $(d_k - \epsilon)n$, and ϕ is formed by picking m clauses independently at random from $D_{replace}(n, k)$, then ϕ is whp **satisfied**.*
- *If m is drawn from $Pois[(d_k + \epsilon)n]$, the Poisson distribution with mean $(d_k + \epsilon)n$, and ϕ is formed by picking m clauses independently at random from $D_{replace}(n, k)$, then ϕ is whp **unsatisfied**.*

▶ **Definition 5.** *Let d_k be the density of clauses such that SAT is at its threshold.*

Note that it is not immediate that a satisfiability threshold exists for any given k. However, Jian Ding, Allan Sly, and Nike Sun showed that this threshold exists for sufficiently large k [16]. It has been proven that

$$2^k \ln(2) - \frac{1}{2}(1 + \ln(2)) - \epsilon_k \leq d_k \leq 2^k \ln(2) - \frac{1}{2}(1 + \ln(2)) + \epsilon_k,$$

where ϵ_k is a term that tends to 0 as k grows [23, 11]. It follows that there exists a large enough k such that $2^k \ln(2) - 1 \leq d_k$. Also, there exists a large enough k such that $2^k \ln(2) \geq d_k$.

This density determines the distribution over the number of clauses put in the formula. Specifically, m is drawn from $Pois[d_k n]$. However, we can say that with high probability the number of clauses is nearly $d_k n$ (see Lem. 9).

For many proofs it is convenient to assume k is large (e.g. when k is large, $2^k > 10k$ not just asymptotically but also numerically). We will now define $k^\star$. It will be a value such that $k = k^\star$ is large enough that both d_k is known to be close to $2^k \ln(2) - \frac{1}{2}(1 + \ln(2))$ and large enough for our proofs that depend on k being large.

▶ **Definition 6.** *Let $\epsilon_k = |d_k - 2^k \ln(2) + \frac{1}{2}(1 + \ln(2))|$.*

▶ **Definition 7.** *Let k_ϵ be the minimum value such that for all $k \geq k_\epsilon$ we have that $\epsilon_k < \frac{1+\ln(2)}{2}$.*

▶ **Definition 8.** *Let* $k^\star = \max(60, k_\epsilon)$.

Our choice of 60 in the above is somewhat arbitrary. When $k \geq 60$ the proofs in Appendix C of our full version [25] are simpler, so we analyze our core algorithm in that regime.

▶ **Lemma 9.** *If* $\epsilon_k < \frac{1+\ln(2)}{2}$ *then* $Pr[m > (d_k+1)n] + Pr[m < (d_k-1)n] \leq 2 \cdot 2^{-n/(3\ln(2)2^k)}$.

Proof. We apply the multiplicative form of the Chernoff bound. We have that $(d_k + 1)n/(d_k n) = 1 + 1/d_k$. We also have that $(d_k - 1)n/(d_k n) = 1 - 1/d_k$. This gives us

$$Pr[m > (d_k+1)n] + Pr[m < (d_k-1)n] \leq 2^{-(d_k)^{-2} d_k n/3} + 2^{-(d_k)^{-2} d_k n/2}.$$

Which means

$$Pr[m > (d_k+1)n] + Pr[m < (d_k-1)n] \leq 2^{-n/(3d_k)} + 2^{-n/(2d_k)}.$$

$$Pr[m > (d_k+1)n] + Pr[m < (d_k-1)n] \leq 2 \cdot 2^{-n/\left(3(\ln(2)2^k - \frac{1}{2}(1+\ln(2))\right)}.$$

$$Pr[m > (d_k+1)n] + Pr[m < (d_k-1)n] \leq 2 \cdot 2^{-n/(3\ln(2)2^k)}.$$ ◀

It follows that if our algorithm works efficiently for all values of $m \in [(d_k-1)n, (d_k+1)n]$, then it works with high probability at the threshold.

Below are some definitions used in later sections.

▶ **Definition 10.** *Let* $D_\Phi(n,k)$ *be the distribution over formulas* ϕ *where all clauses are chosen independently from* $D_{replace}$ *and the number of clauses is chosen from a Poisson distribution with mean* $d_k n$.

▶ **Definition 11.** *Let* $D_R(m,n,k)$ *be the distribution over formulas* ϕ *where all* m *clauses are chosen independently from* $D_{replace}$.

▶ **Definition 12.** *Let* $D_S(m,n,k)$ *be the uniform distribution over* satisfied *formulas* ϕ *where all* m *clauses are chosen from* $D_{replace}$.

▶ **Definition 13.** *Let* $D_{pc}(n,k,\vec{a})$ *(which we refer to as "the planted-clause distribution") be the uniform distribution over the* $(2^k-1)n^k$ *clauses* c *which are satisfied by* $\vec{a}$.

▶ **Definition 14.** *Let* $D_{pa}(m,n,k,\vec{a})$ *(which we refer to as "the planted distribution") be the distribution over formulas* ϕ *where every clause is picked IID from* $D_{pc}(n,k,\vec{a})$. *Note that this is equivalent to the uniform distribution over formulas* ϕ *which are satisfied by* $\vec{a}$ *and where all* m *clauses are in the support of* $D_{replace}$.

▶ **Definition 15.** *Let* $U_{\vec{a}}(n)$ *be the uniform distribution over assignments of length* n, $\{0,1\}^n$.

▶ **Definition 16.** *Let* $\textsc{NumClausesSAT}(\phi,\vec{v})$ *be the number of clauses in* ϕ *satisfied by the assignment* $\vec{v}$.

▶ **Definition 17.** *Let* $\textsc{NumClausesUnSAT}(\phi,\vec{v})$ *be the number of clauses in* ϕ *left unsatisfied by the assignment* $\vec{v}$.

3 Algorithm

We will describe our algorithm for random k-SAT in this section.

Informally, our algorithm works as follows. Given an input formula, we will sample many randomly-chosen assignments. On those that have a high number of satisfied clauses, we will run the deterministic algorithm for finding a satisfying assignment given an assignment that is within a Hamming distance of at most αn of that satisfying assignment (i.e. a small-Hamming-distance assignment).

Unsurprisingly, in the average case, small-Hamming-distance assignments satisfy more clauses than random assignments[1]. In fact, for many choices of criterion there will be a discrepancy between the values achieved by small-Hamming-distance assignments and random assignments. Lemma 30 of our full version [25], which characterizes this discrepancy, is general enough to be applied immediately to analyzing algorithms that make use of any clause-specific criterion.

We note the following from previous work:

▶ **Lemma 18** (Small Hamming Distance Search [14]). *There is a deterministic algorithm SAT-FROM-α-SMALL-HD$(\phi, \vec{v})$ which given*

- *a k-CNF formula ϕ on m clauses and n variables, and*
- *an assignment $\vec{v}$ which has Hamming distance αn from a true satisfying assignment $\vec{a}^*$,*

will return a satisfying assignment within Hamming distance αn of $\vec{v}$ if one exists in $k^{\alpha n}$ time.

This algorithm simply takes the assignment $\vec{a}$ and branches on the first unsatisfied clause, trying all possible variable flips. For each assignment resulting from these possible variable flips, the algorithm repeats the process in what is now the first unsatisfied clause, until it either finds a satisfying assignment or has searched αn flips from the original assignment. This will deterministically yield a satisfying assignment, should one exist, within a Hamming distance of αn of the original assignment.

So, if we find a small-Hamming-distance assignment and run SAT-FROM-α-SMALL-HD$(\phi, \vec{a})$ on this assignment, we are guaranteed to find the satisfying assignment. Therefore, we could randomly sample points until we expect to find an assignment at Hamming distance αn from the satisfying assignment (call this an α-small-Hamming-distance assignment). This is indeed what Schöning's algorithm does for $\alpha = \Theta(1/k)$ [30].

A general class of improvements to this algorithm work by running SAT-FROM-α-SMALL-HD$(\phi, \vec{a})$ on only a cleverly-chosen subset of these sampled assignments. In our case, we choose this set to be assignments that satisfy an unusually large number of clauses, but in principle one could use any membership criterion for this set.

Let M be the runtime of the membership test for the set of assignments, and let p_{TP}, p_{FP}, p_{FN}, and p_{TN} represent the fraction of assignments that are true positives, false positives, false negatives, and true negatives respectively. Here, just as in Section 1.1, we use "positive" or "negative" to mean an assignment that passes or doesn't pass the test for membership, respectively. The truth or falsehood of that positive or negative represents whether or not that assignment actually has a satisfying assignment within small Hamming distance.

[1] Consider changing one variable's assignment at random; in this case, almost all clauses will remain satisfied. This phenomenon persists even when we flip several variables at once.

We will have to draw samples until we would have found a satisfying assignment with high probability were one to exist. Next, we will have to run SAT-FROM-α-SMALL-HD$(\phi, \vec{a})$ at least once to find the satisfying assignment itself. Finally, we will have to run it once more for every false positive we find. Hence, the the generalized running time of this class of algorithms is

$$O^*\left(\frac{M}{p_{TP}} + k^{\alpha n} + \left(\frac{p_{FP}}{p_{TP}}\right) k^{\alpha n}\right). \tag{1}$$

This general formula is a powerful tool for analyzing the runtimes of algorithms from this class. For example, if we apply it to analyzing the algorithm of [14], i.e. the special case where the test we use always returns a positive, we see that the third term in Equation (1) dominates, and that $p_{FP} \approx 1$ and $p_{TP} \approx \frac{\binom{n}{\alpha n}}{2^n}$, giving an overall expected runtime of $O^*(2^{n(1-\Theta(\frac{1}{k}))})$ when we choose $\alpha = \Theta(\frac{1}{k+1})$ (using tail bounds to convert $\binom{n}{\alpha n}$ to an exponential). In Appendix G.3 of our full version [25] we discuss a different deterministic search algorithm with a slightly improved runtime (yielding no relevant improvement on the runtime of the overall algorithm for our analysis).

Our algorithm presents improvements for large k, but for small k we will simply use the previous algorithm of Dantsin et al [14].

▶ **Lemma 19** (Algorithm for Small k [14]). *For $k \leq k^\star$ there exists a deterministic algorithm, DANTSINLS, that solves k-SAT in the worst case in time $2^{n(1-\gamma)}$ for some constant $\gamma > 0$.*

We will now give pseudocode for the α-SAMPLEANDTEST algorithm in Algorithm 1. Let NUMCLAUSESSAT$(\phi,\vec{a})$ return the number of clauses in ϕ satisfied by the assignment $\vec{a}$. In Appendix G.1 of our full version [25] we describe a different set of concepts with which the algorithm can be understood.

Note that our algorithm as stated is non-constructive due to our use of the constant $k^\star$. Other than this constant, our algorithm is explicit. While $k^\star$ is known to be constant [11], its exact value is currently unknown. We note in Appendix E of our full version [25] that finding the value of $k^\star$ is an open problem which, if solved, would make our algorithm constructive.

3.1 Correctness and Running Time

We will include the theorem statement of correctness and running time here. Its proof depends on bounds on the false positive rate and the true positive rate, which we prove in later sections. In particular, we show in Appendix A of our full version [25] that conditioned on an assignment passing the test, it is sufficiently likely to be an α-small-Hamming-distance assignment. We additionally show in Appendix B of our full version [25] that conditioned on an assignment being an α-small-Hamming-distance assignment, it is sufficiently likely to pass the test.

Note that much of our probability of returning the wrong value comes from our bounds on the probability that we are drawing a formula with length $m < (d_k - 1)n$. If we knew m to be fixed and greater than $(d_k - 1)n$, we would have a lower error probability.

We will show that α-SAMPLEANDTEST(ϕ) has one-sided error and returns the correct answer with high probability. Note that it returns the correct answer with high probability even conditioned on the input being unsatisfied or satisfied. We use Theorem 26 of our full version [25] to bound the false positive rate and use Lemma 43 of our full version [25] to bound the true positive rate, which gives us the desired result.

Algorithm 1 α-SampleAndTest(ϕ).

```
α-SampleAndTest(φ):
if $k < k^\star$ then
    return DantsinLS($\phi$)
end
Initialize $S$ to the empty set.
▷ For $k \geq k^\star$ we run our variant of local search:
for $i \in [0, n^2 \cdot 2^n / \binom{n}{\alpha n}]$ do
    Sample an assignment $\vec{a}$ uniformly at random from $\{0,1\}^n$.
    ▷ Only keep assignments which satisfy abnormally many clauses.
    if NumClausesSAT($\phi$,$\vec{a}$) $\geq (1 - \frac{1-(1-\alpha)^{2k}}{2^k-1})m$ then
        Add $\vec{a}$ to $S$.
        if $|S| > 4n^3 2^n / \left(\binom{n}{\alpha n} k^{\alpha n}\right) + 1$ then
            return False
        end
        Run SAT-from-$\alpha$-Small-HD($\phi, \vec{a}$).
        If an assignment was found, return it.
    end
end
return False
```

In the theorem that follows, we choose α such that αn is an integer. Specifically, we choose:

$$\alpha = \frac{\lfloor \frac{\lg(k)}{16k} n \rfloor}{n}.$$

Note that when we choose α to take on this value, it will always lie in the range $\frac{\lg(k)}{20k} \leq \alpha \leq \frac{\lg(k)}{16k}$ for large n.

▶ **Theorem 20.** *Assume ϕ is drawn from $D_\Phi(n,k)$. Let $\alpha = \frac{\lfloor n \lg(k)/(16k) \rfloor}{n}$.*

Conditioned on there being at least one satisfying assignment to ϕ, α-SampleAndTest(ϕ) will return some satisfying assignment with probability at least $1 - 3 \cdot 2^{-n/(3\ln(2)2^k)}$.

Conditioned on there being no satisfying assignment to ϕ, α-SampleAndTest(ϕ) will return False with probability 1.

α-SampleAndTest(ϕ) will run in time

$$O\left(2^{n(1-\Omega(\lg^2(k)/k))}\right).$$

Proof. Proof given in Appendix D of our full version [25]. ◀

References

1 Scott Aaronson. P=?NP. *Electronic Colloquium on Computational Complexity (ECCC)*, 24:4, 2017. URL: https://eccc.weizmann.ac.il/report/2017/004.

2 D. Achioptas and G. B. Sorkin. Optimal myopic algorithms for random 3-sat. In *Proceedings 41st Annual Symposium on Foundations of Computer Science*, pages 590–600, November 2000. doi:10.1109/SFCS.2000.892327.

3 Dimitris Achlioptas. Random satisfiability. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 245–270. IOS Press, 2009. doi:10.3233/978-1-58603-929-5-245.

4 Dimitris Achlioptas and Amin Coja-Oghlan. Algorithmic barriers from phase transitions. In *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*, pages 793–802, 2008. doi:10.1109/FOCS.2008.11.

5 Sarah R. Allen, Ryan O'Donnell, and David Witmer. How to refute a random CSP. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 689–708, 2015. doi:10.1109/FOCS.2015.48.

6 Eli Ben-Sasson, Yonatan Bilu, and Danny Gutfreund. Finding a randomly planted assignment in a random 3CNF. Technical report, In preparation, 2002.

7 Amin Coja-Oghlan. A better algorithm for random k-SAT. *SIAM Journal on Computing*, 39(7):2823–2864, 2010.

8 Amin Coja-Oghlan, Colin Cooper, and Alan M. Frieze. An efficient sparse regularity concept. *SIAM J. Discrete Math.*, 23(4):2000–2034, 2010. doi:10.1137/080730160.

9 Amin Coja-Oghlan, Andreas Goerdt, and André Lanka. Strong refutation heuristics for random k-sat. *Combinatorics, Probability & Computing*, 16(1):5–28, 2007. doi:10.1017/S096354830600784X.

10 Amin Coja-Oghlan, Michael Krivelevich, and Dan Vilenchik. Why almost all k-CNF formulas are easy. In *Proceedings of the 13th International Conference on Analysis of Algorithms, to appear*, 2007.

11 Amin Coja-Oghlan and Konstantinos Panagiotou. The asymptotic k-SAT threshold. *Advances in Mathematics*, 288:985–1068, 2016. doi:10.1016/j.aim.2015.11.007.

12 Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA*, pages 151–158, 1971. doi:10.1145/800157.805047.

13 Stephen A Cook and David G Mitchell. Finding hard instances of the satisfiability problem. In *Satisfiability Problem: Theory and Applications: DIMACS Workshop*, volume 35, pages 1–17, 1997.

14 Evgeny Dantsin, Andreas Goerdt, Edward A. Hirsch, Ravi Kannan, Jon M. Kleinberg, Christos H. Papadimitriou, Prabhakar Raghavan, and Uwe Schöning. A deterministic $(2-2/(\mathrm{k}+1))^{\mathrm{n}}$ algorithm for k-SAT based on local search. *Theor. Comput. Sci.*, 289(1):69–83, 2002. doi:10.1016/S0304-3975(01)00174-8.

15 Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

16 Jian Ding, Allan Sly, and Nike Sun. Proof of the Satisfiability Conjecture for large k. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 59–68, 2015. doi:10.1145/2746539.2746619.

17 Olivier Dubois, Yacine Boufkhad, and Jacques Mandler. Typical random 3-sat formulae and the satisfiability threshold. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '00, pages 126–127, Philadelphia, PA, USA, 2000. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=338219.338243.

18 Vitaly Feldman, Will Perkins, and Santosh Vempala. On the complexity of random satisfiability problems with planted solutions. *Electronic Colloquium on Computational Complexity (ECCC)*, 21:148, 2014. URL: http://eccc.hpi-web.de/report/2014/148.

19 Jun Gu, Paul W. Purdom, John Franco, and Benjamin W. Wah. Algorithms for the satisfiability (SAT) problem: A survey. In *Satisfiability Problem: Theory and Applications, Proceedings of a DIMACS Workshop, Piscataway, New Jersey, USA, March 11–13, 1996*, pages 19–152, 1996. doi:10.1090/dimacs/035/02.

20 Hiêp Hàn, Yury Person, and Mathias Schacht. Note on strong refutation algorithms for random k-sat formulas. *Electronic Notes in Discrete Mathematics*, 35:157–162, 2009. doi:10.1016/j.endm.2009.11.027.

21 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.

22 Richard M. Karp. Reducibility among combinatorial problems. In *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, pages 85–103, 1972. URL: http://www.cs.berkeley.edu/%7Eluca/cs172/karp.pdf, doi:10.1007/978-1-4684-2001-2_9.

23 Lefteris M. Kirousis, Evangelos Kranakis, Danny Krizanc, and Yannis C. Stamatiou. Approximating the unsatisfiability threshold of random formulas. *Random Struct. Algorithms*, 12(3):253–269, 1998. doi:10.1002/(SICI)1098-2418(199805)12:3<253::AID-RSA3>3.0.CO;2-U.

24 Leonid A. Levin. Universal search problems. *Problems of Information Transmission*, 9(3), 1973.

25 Andrea Lincoln and Adam Yedidia. Faster random k-cnf satisfiability. *arXiv preprint*, 2019. arXiv:1903.10618.

26 Chao Ming-Te and John Franco. Probabilistic analysis of a generalization of the unit-clause literal selection heuristics for the k-satisfiability problem. *Information Sciences*, 51(3):289–314, 1990.

27 Eugene Nudelman, Kevin Leyton-Brown, Holger H. Hoos, Alex Devkar, and Yoav Shoham. Understanding random SAT: beyond the clauses-to-variables ratio. In *Principles and Practice of Constraint Programming - CP 2004, 10th International Conference, CP 2004, Toronto, Canada, September 27 - October 1, 2004, Proceedings*, pages 438–452, 2004. doi:10.1007/978-3-540-30201-8_33.

28 Ramamohan Paturi, Pavel Pudlák, Michael E. Saks, and Francis Zane. An improved exponential-time algorithm for k-SAT. *J. ACM*, 52(3):337–364, 2005. doi:10.1145/1066100.1066101.

29 Ramamohan Paturi, Pavel Pudlák, and Francis Zane. Satisfiability coding lemma. *Chicago J. Theor. Comput. Sci.*, 1999, 1999. URL: http://cjtcs.cs.uchicago.edu/articles/1999/11/contents.html.

30 Uwe Schöning. A probabilistic algorithm for k-sat and constraint satisfaction problems. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*, pages 410–414, 1999. doi:10.1109/SFFCS.1999.814612.

31 Bart Selman, David G. Mitchell, and Hector J. Levesque. Generating hard satisfiability problems. *Artificial Intelligence*, 81(1):17–29, 1996. Frontiers in Problem Solving: Phase Transitions and Complexity. doi:10.1016/0004-3702(95)00045-3.

32 Greg Valiant. Faster random SAT. Personal communication.

33 Nikhil Vyas and Ryan Williams. On super strong ETH. In Mikolás Janota and Inês Lynce, editors, *Theory and Applications of Satisfiability Testing - SAT 2019 - 22nd International Conference, SAT 2019, Lisbon, Portugal, July 9-12, 2019, Proceedings*, volume 11628 of *Lecture Notes in Computer Science*, pages 406–423. Springer, 2019. doi:10.1007/978-3-030-24258-9_28.

Succinct Filters for Sets of Unknown Sizes

Mingmou Liu
State Key Laboratory for Novel Software Technology, Nanjing University, China
http://tcs.nju.edu.cn/files/people/mingmou
liu.mingmou@smail.nju.edu.cn

Yitong Yin
State Key Laboratory for Novel Software Technology, Nanjing University, China
http://tcs.nju.edu.cn/yinyt/
yinyt@nju.edu.cn

Huacheng Yu
Princeton University, NJ, USA
https://www.cs.princeton.edu/~hy2/
yuhch123@gmail.com

Abstract

The membership problem asks to maintain a set $S \subseteq [u]$, supporting insertions and *membership queries*, i.e., testing if a given element is in the set. A data structure that computes exact answers is called a *dictionary*. When a (small) false positive rate ϵ is allowed, the data structure is called a *filter*.

The space usages of the standard dictionaries or filters usually depend on the upper bound on the size of S, while the actual set can be much smaller.

Pagh, Segev and Wieder [28] were the first to study filters with varying space usage based on the *current* $|S|$. They showed in order to match the space with the current set size $n = |S|$, any filter data structure must use $(1 - o(1))n(\log(1/\epsilon) + (1 - O(\epsilon)) \log \log n)$ bits, in contrast to the well-known lower bound of $N \log(1/\epsilon)$ bits, where N is an upper bound on $|S|$. They also presented a data structure with almost optimal space of $(1 + o(1))n(\log(1/\epsilon) + O(\log \log n))$ bits provided that $n > u^{0.001}$, with expected amortized constant insertion time and worst-case constant lookup time.

In this work, we present a filter data structure with improvements in two aspects:

- it has constant worst-case time for all insertions and lookups with high probability;
- it uses space $(1 + o(1))n(\log(1/\epsilon) + \log \log n)$ bits when $n > u^{0.001}$, achieving optimal leading constant for all $\epsilon = o(1)$.

We also present a dictionary that uses $(1 + o(1))n \log(u/n)$ bits of space, matching the optimal space in terms of the current size, and performs all operations in constant time with high probability.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis

Keywords and phrases Bloom filters, Data structures, Approximate set membership, Dictionaries

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.79

Category Track A: Algorithms, Complexity and Games

Related Version A full version of the paper is available at http://arxiv.org/abs/2004.12465.

Funding *Mingmou Liu*: Part of the research was done when Mingmou Liu was visiting the Princeton University. Mingmou Liu is supported by National Key R&D Program of China 2018YFB1003202 and NSFC under Grant Nos. 61722207 and 61672275.
Yitong Yin: Yitong Yin is supported by National Key R&D Program of China 2018YFB1003202 and NSFC under Grant Nos. 61722207 and 61672275.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 79; pp. 79:1–79:19
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Membership data structures are fundamental subroutines in many applications, including databases [9], content delivery network for web caching [24], image processing [17], scanning for viruses [14], etc. The data structure maintains a set of keys from a key space $[u]$,[1] supporting the following two basic operations:

- insert(x): insert x into the set;
- lookup(x): return YES if x is in the set, and NO otherwise.

When false positive errors are allowed, such a data structure usually is referred as a *filter*. That is, a filter with false positive rate ϵ may answer YES with probability ϵ when x is not in the set (but it still needs to always answer YES when x is in the set).

In the standard implementations, a initialization procedure receives the key space size u and a *capacity* N, i.e., an upper bound on the number of keys that can simultaneously exist in the database. Then it allocates sufficient space for the data structure, e.g., a hash table consisting of $\Theta(N)$ buckets. Thereafter, the memory usage is always staying at the maximum, as much space as N keys would take. It introduces inefficiency in the space, when only few keys have been inserted so far. On the other hand, it could also happen that only a rough estimation of the maximum size is known (e.g. [16, 1, 22]). Therefore, to avoid overflowing, one has to set the capacity conservatively. The capacity parameter given to the initialization procedure may be much more than the actual need. To avoid such space losses, a viable approach is to dynamically allocate space such that at any time, the data structure occupies space depending only on the *current* database size (rather than the maximum possible).

For exact membership data structures, it turns out that such promise is not too hard to obtain if one is willing to sacrifice an extra constant factor in space and accept amortization: When the current database has n keys, we set the capacity to $2n$; after n more keys are inserted, we construct a new data structure with capacity equal to $4n$ and transfer the whole database over. The amortized cost to transfer the database is $O(1)$ per insertion. Raman and Rao [29] showed that the extra constant factor in space is avoidable, they designed a *succinct*[2] membership data structure using space $(1 + o(1)) \log \binom{u}{n}$,[3] where n is the *current* database size, supporting insertions in expected amortized constant time, and lookup queries in worst-case constant time.

For filters, the situation is more complicated. The optimal space to store at most N keys while supporting approximate membership queries with false positive rate ϵ is $N \log 1/\epsilon$ [8, 23] (Pagh, Pagh and Rao [27] achieved $(1 + o(1))N \log 1/\epsilon$ bits). However, the above trick to reduce the space may not work in general. This is because the filter data structures do not store perfect information about the database, and therefore, it is non-trivial to transfer to the new data structure with capacity $4n$, as one might not be able to recover the whole database from the previous data structure. In fact, Pagh, Segev and Wieder [28] showed an information theoretical space lower bound of $(1 - o(1))n(\log 1/\epsilon + (1 - O(\epsilon)) \log \log n)$ bits, regardless of the insertion and query times. That is, one has to pay extra $\approx \log \log n$ bits per key in order to match the space with the current database size. They also proposed a data structure with a nearly matching space of $(1 + o(1))n \log 1/\epsilon + O(n \log \log n)$ bits when $n > u^{0.001}$, while supporting insertions in expected amortized constant time and lookup

[1] Throughout the paper, $[u]$ stands for the set $\{0, \ldots, u - 1\}$.

[2] A succinct data structure uses space equal to the information theoretical minimum plus an asymptotically smaller term called *redundancy*.

[3] All logarithms are base 2.

queries in worst-case constant time. When ϵ is at least $1/\text{poly}\log n$, the extra $\log\log n$ bits per key is dominating. It was proposed as an open problem in [28] whether one can make the $\log\log n$ term succinct as well, i.e., to pin down its leading constant.

On the other hand, an amortized performance guarantee is highly undesirable in many applications. For instances, IP address lookups in the context of router hardware [7, 19], and timing attacks in cryptography [21, 20, 26, 25]. When the database size is always close to the capacity (or when the space is not a concern), it was known how to support all operations in worst-case constant time [13, 3] with high probability. That is, except for a probability of $1/\text{poly}\, n$, the data structure handles *every* operation in a sequence of length $\text{poly}\, n$ in constant time.[4] However, it was not known how to obtain such a guarantee when the space is succinct with respect to the current database size, i.e., $(1+o(1))\log\binom{u}{n}$. For filters, Pagh et al. [28] showed it is possible to get worst-case constant time with high probability, at the price of a constant factor more space $O(n\log 1/\epsilon + n\log\log n)$. They asked if there is a data structure which enjoys the succinct space usage and the worst-case constant time with high probability simultaneously.

1.1 Main Results

In this paper, we design a new dynamic filter data structure that answers both questions. Our data structure has both worst-case constant time with high probability and is succinct in space in terms of the current database size.

▶ **Theorem 1** (Dynamic filter – informal). *There is a data structure for approximate membership with false positive rate ϵ that uses space $(1+o(1))n(\log(1/\epsilon)+\log\log n)$ bits, where $n > u^{0.001}$ is the current number of keys in the database, such that every insertion and lookup takes constant time in the worst case with high probability.*

We also present a dictionary data structure with the space depending on the current n. A dictionary is a generalization of membership data structures, it maintains a set of key-value pairs, supporting

- insert(x, y): insert a key-value pair (x, y) for $x \in [u]$ and v-bit y ;
- lookup(x): if $\exists(x, y)$ in the database, output y; otherwise output `NO`.

By setting $v = 0$, the lookup query simply tests if x is in the database.

▶ **Theorem 2** (Dynamic dictionary – informal). *There is a dictionary data structure that uses space $(1+o(1))n(\log(u/n)+v+O(\log\log\log u))$ bits, where $n > u^{0.001}$ is the current number of key-value pairs in the database, such that every insertion and lookup takes constant time in the worst case with high probability.*

1.2 Related Work

Membership with Constant Time Worst-Case Guarantee. The FKS perfect hashing [15] stores a set of n *fixed* (i.e., static) keys using $O(n)$ space, supporting membership queries in worst-case constant time. Dietzfelbinger, Karlin, Mehlhorn, Meyer auf der Heide, Rohnert and Tarjan [12] introduced an extension of the FKS hashing, which is the first dynamic membership data structure with worst-case constant query time and the expected amortized constant insertion time. Later, Dietzfelbinger and Meyer auf der Heide [13] improved

[4] This is stronger guarantee than expected constant time, since when the unlikely event happened, one could simply rebuild the data structure in linear time. The expected time is still a constant.

the insertion time to worst-case constant, with an overall failure probability of $1/\text{poly}\, n$. Demaine, Meyer auf der Heide, Pagh and Pǎtraşcu [11] improved the space to $O(n \log(u/n))$ bits of space. Arbitman, Naor and Segev [2] proved that a de-amortized version of cuckoo hashing [19] has constant operation time in the worst case with high probability.

On the other hand, filters can be reduced to dictionaries with a hash function $h : [u] \to [n/\epsilon]$, and thus, all the dictionaries imply similar upper bounds for filters [8].

Succinct Membership. Raman and Rao [29] presented the first succinct dictionary with constant time operations, while the insertion time is amortized. Arbitman, Naor and Segev [3] refined the schema of [2], suggested a succinct dictionary with worst case operation time with high probability.

By using the reduction from [8] and the succinct dictionary from [29], Pagh, Pagh and Rao [27] provided a succinct filter with constant time, while the insertion time is amortized due to [29]. Bender, Farach-Colton, Goswami, Johnson, McCauley and Singh [5] suggested a succinct *adaptive filter*[5] with constant time operation in the worst case with high probability.

Membership for Sets of Unknown Sizes. The data structure of Raman and Rao [29] can be implemented such that the size of the data structure always depends on the "current n". Pagh, Segev and Wieder [28] were the first to study dynamic filters in this setting from a foundational perspective. As we mentioned above, they proved an information-theoretical space lower bound of $(1 - o(1))n(\log(1/\epsilon) + (1 - O(\epsilon)) \log \log n)$ bits for filter, and presented a filter data structure using $n(\log(1/\epsilon) + O(\log \log n))$ bits of space with constant operation time when $n > u^{0.001}$. Indeed, the insertion time is expected amortized, since the succinct dictionary of Raman and Rao is applied as a black box (it was not clear if any succinct dictionary with worst-case operational time can be generalized to this setting).

Very recently, Bercea and Even [6] proposed a succinct membership data structure for maintaining *dictionaries* and *random multisets* with constant operation time. While their data structure is originally designed for the case where an upper bound N on the keys is given (and the space usage is allowed to depend on N), we note that it is possible to extend their solution and reduce the space to depend only on the current n. However, their data structure assumes free randomness, and straightforward extension results in an additive $\Omega(n \log \log u)$ term in space. The redundancy makes their data structure space-inefficient for filters, since the space lower bound is $(1 - o(1))n(\log(1/\epsilon) + (1 - O(\epsilon)) \log \log n)$.

1.3 Previous Construction

As we mentioned earlier, for dynamic membership data structures, if we are willing to pay an extra constant factor in space, one way to match the space with the "current" n is to set the capacity to be $2n$. When the data structure is full after another n insertions, we double the capacity, and transfer the database to the new data structure. However, the standard way to construct an efficient filter is to hash $[u]$ to $[n/\epsilon]$ (where ϵ is the false positive rate) and store all n hash values in a membership data structure, which takes $O(n \log 1/\epsilon)$ bits of space. As we insert more keys and increase the capacity to $4n$, the range of the hash value needs to increase as well. Unfortunately, it cannot be done, because the original keys are not stored, and we have lost the information in order to save space (this is exactly the point of a filter). On the other hand, we could choose to keep the previous

[5] In an adaptive filter, for a negative query x, the false positive event is independent of previous queries.

data structure(s), and only insert the future keys to the new data structure. For each query, if it appears in any of the (at most $\log n$) data structures, we output YES. By setting the false positive rate for the data structure with capacity 2^i to $O(\epsilon/i^2)$, the overall false positive rate is at most $\epsilon \cdot \sum_i O(1/i^2) \leq \epsilon$ by union bound. The total space usage becomes roughly $n\log(\log^2 n/\epsilon) = n(\log 1/\epsilon + O(\log\log n))$.

To avoid querying all $\log n$ filters for each query, the previous solution by Pagh et al. [28] uses a *single* global hash function h that maps $[u]$ to $\log(u/\epsilon)$-bit strings for all $\log n$ filters. For a key x in the i-th data structure (with capacity 2^i), one simply takes the first $i + \log 1/\epsilon + 2\log i$ bits of $h(x)$ as its hash value. Then querying the i-th data structure on y is to check whether the $(i + \log 1/\epsilon + 2\log i)$-bit prefix of $h(y)$ exists. Since all filters use the same hash function, the overall task is to check whether *some* prefix of $h(y)$ appears in the database, which now consists of strings of various lengths. Note that there are very few short strings in the database, the previous solution extends all short strings to length $\log(n/\epsilon)$ by duplicating the string and appending all possible suffixes, e.g., a string of length $\log(n/\epsilon) - c$ is duplicated into 2^c strings by appending all possible c-bit suffixes. Then all strings are stored in one single dictionary (longer strings are stored according to their first $\log(n/\epsilon)$ bits), and the query becomes to check if the $\log(n/\epsilon)$-bit prefix of $h(y)$ is in the dictionary, which is solved by invoking Raman and Rao [29]. One may verify that duplicating the short strings does not significantly increase the total space, and comparing only the $\log(n/\epsilon)$-bit prefix of a longer string does not increase the false positive rate by much.

1.4 Our Techniques

Our new construction follows a similar strategy, but the "prefix matching" problem is solved differently. Given a collection of $2^{i-1} < n \leq 2^i$ strings of various lengths, we would like to construct a data structure such that given any query $h(y)$, we will be able to quickly decide if any prefix of $h(y)$ appears in the database. The first observation is that the short strings in the database can be easily handled. In fact, all strings shorter than i bits can be stored in a "truth table" of size $2^i = O(n)$. That is, we simply store for all i-bit strings, whether any of its prefix appears in the database. For a query $h(y)$, by checking the corresponding entry of its i-bit prefix, one immediately resolves all short strings. On the other hand, for strings longer than $\log m$ bits, we propose a new (exact) membership data structure, and show that it in fact, automatically solves prefix matching when all strings are long. Before describing its high-level construction in Section 1.4.1, let us first see what it can do and how it is applied to our filter construction.

When the capacity is set to m, the membership data structure stores $n \leq m$ keys from $[u]$ using space $n(\log(u/m) + O(\log\log\log u)) + O(m)$ bits, supporting insertion and membership query in worst-case constant time with high probability. When applying to prefix matching, it stores n strings of length at most ℓ (and more than $\log m$) using $n(\log(2^\ell/m) + O(\log\log \ell)) + O(m)$ bits. Using this data structure with the capacity set to $m = 2^i$, we are able to store the database succinctly when $m/2 < n \leq m$. As we insert more keys to the database, the capacity needs to increase. Another advantage of our membership data structure is that the data can be transferred from the old data structure with capacity m to a new one with capacity $2m$ in $O(m)$ time. More importantly, the transfer algorithm runs almost "in-place", and the data structure remains "queryable" in the middle of the execution. That is, one does not need to keep both data structures in full, at any time the total memory usage is still $n(\log(2^\ell/n) + O(\log\log \ell)) + O(m)$, and the data structure can be queried. Therefore, as n is increasing from $m/2$ to m, we gradually build a new data structure with capacity $2m$. Every time a key is inserted, the background data-transfer algorithm is run for constant steps. By the time n reaches m, we will have already transferred everything to the new data structure, and will be ready to build the next one with capacity $4m$. Overall, the data structure is going to have $\log n$ stages, the i-th stage handles the $(2^{i-1}+1)$-th to the 2^i-th insertion. In each stage, the database size is doubled, and the data structure also gradually doubles its capacity. This guarantees that the total space is succinct with respect to the current database size, and every operation is handled in constant time with high probability.

Finally, to pin down the leading constant in the extra $O(\log\log n)$ bits, we show that for the n-th inserted key x for $2^{i-1} < n \leq 2^i$, storing the $(i + \log(i/\epsilon) + \log\log\log u)$-bit prefix of $h(x)$ balances the false positive rate and the space. Since our new membership data structure only introduces an extra $\approx \log\log i \approx \log\log\log n$ bits of space per key, it is not hard to verify that the total space of our construction is $(1+o(1))n(\log(1/\epsilon) + \log\log n)$.

1.4.1 Membership Data Structure

In the following, let us briefly describe how our new membership data structure works. The data structure works in the *extendable array* model, as the previous solution by Raman and Rao. See Section 2.2.2 or [29] for more details.

Our main technique contribution is the idea of *data block*. Without the data blocks, our data structure degenerates into a variant of the one proposed in [6]. Instead of a redundancy of $O(n\log\log\log u)$ bits, the degeneration contributes a redundancy of $O(n\log\log u)$ bits, which makes the data structure space-inefficienct for filters as we discussed early.

For simplicity, let us for now assume that we have free randomness, and the first step is to randomly permute the universe. Thus, we may assume that at any time, the database is a uniformly random set (of certain size). We divide the universe into $m/\log u$ *buckets*, e.g., according to the top $\log(m/\log u)$ bits of the key. Then with high probability, every bucket will have $O(\log u)$ keys. We will then dynamic allocate space for each bucket. Note that given that a key is a bucket b, we automatically know that its top $\log(m/\log u)$ bits is "b". Therefore, within each bucket, we may view the keys have lengths only $\log u - \log(m/\log u) = \log((u\log u)/m)$, or equivalently, the universe size being $(u\log u)/m$ (recall that the goal is to store each key using $\approx \log(u/m)$ bits on average).

To store the keys in a bucket, we further divide it into *data blocks* consisting of $O(\log u/\log\log u)$ keys each, based on the time of insertion. That is, the first $O(\log u/\log\log u)$ keys inserted to this bucket will form the first data block, the next $O(\log u/\log\log u)$ keys will be the second data block, etc. Since each data block has few enough keys, they can be stored using a *static* constructions (supporting only queries) using nearly optimal space of $\approx \log\binom{(u\log u)/m}{O(\log u/\log\log u)}$, which is $\log((u\log\log u)/m) = \log(u/m) + \log\log\log u$ bits per key, or a dynamic constructions use $\log(u/m) + O(\log\log u)$ bits per key. The latest data block, which we always insert the new key into, is maintained using the dynamic construction. When it becomes full, we allocate a new data block, and at the same time, we run a in-place *reorganization* algorithm in the background. The reorganization algorithm runs in $O(\log u/\log\log u)$ time, and convert the dynamic construction into the static construction, which uses less space. For each insertion in the future, the reorganization algorithm is run for constant steps, thus, it finishes before the next data block becomes full. Finally, for each bucket, we maintain an *adaptive prefixes* structure [4, 5] to navigate the query to the relevant data block. Roughly speaking, when all $O(\log u)$ keys in the bucket are random, most keys will have a unique prefix of length $\log\log u$. In fact, Bender et al. [4, 5] showed that for every keys, the shortest prefix that is unique in the bucket can be implicitly maintained in constant time, and the total space for all $O(\log u)$ keys is $O(\log u)$ bits with high probability.[6] We further store for each such unique prefix, which data block contains the corresponding key. It costs $O(\log\log\log u)$ bits per key. Given a query, the adaptive prefix structure is able to locate the prefix that matches the query in constant time, which navigates the query algorithm to the (only) relevant data block. We present the details in Section 4.

2 Preliminaries

2.1 String Notations

Let $\{0,1\}^{\le \ell} \triangleq \bigcup_{0\le i\le \ell}\{0,1\}^i$ and $\{0,1\}^* \triangleq \bigcup_{i\ge 0}\{0,1\}^i$. Given a string $x\in\{0,1\}^\ell$, we use $|x|=\ell$ to denote its length. We denote by $a\circ b$ the concatenation of two strings $a,b\in\{0,1\}^*$. We denote the concatenation of k ones or zeros by 1^k or 0^k, respectively.

For $x,y\in\{0,1\}^*$, we use $x\sqsubseteq y$ (or $y \sqsupseteq x$) to denote that y is a *prefix* of x, formally:

$$x\sqsubseteq y \iff x=y\circ a \text{ for some } a\in\{0,1\}^*. \tag{1}$$

Note that our notation is unconventional: we use $x\sqsubseteq y$ for y prefixing x, to reflect that the Hamming cube identified by x is contained by the Hamming cube for its prefix y.

For two strings x,y such that $|x|\le|y|$, to compare x and y in lexicographical order, we compare $x\circ \perp^{|y|-|x|}$ and y in lexicographical order, where $\perp$ is a special symbol which is smaller than any other symbol.

Recall that an injection (code) on strings is a *prefix-free code* if no codeword is a prefix of another codeword.

▷ Claim 3. There is a prefix-free code $\texttt{PFC}:\{0,1\}^{\le\ell}\to\{0,1\}^{\ell+1}$ for strings of length $\le \ell$.

Proof. Given any $x\in\{0,1\}^{\le\ell}$, the codeword $\texttt{PFC}(x)$ is $1^{\ell-|x|}\circ 0\circ x$. ◁

2.2 Computational Models

2.2.1 Random Access Machine

Throughout the paper, we use w to denote the word size: each memory word is a Boolean string of w bits. We assume that the total number of memory words is at most 2^w, and each memory word has an unique address from $[2^w]$, so that any pointer fits in one memory word. We also assume CPU has constant number of registers of size w, and any datapoint fits in constant number of words (i.e. $w=\Omega(v+\log u)$). During each CPU clock tick, CPU may load one memory word to one of its register, write the content of some register to some memory word, or execute the basic operations on the registers. Specifically, the basic operations include four arithmetic operations (addition, subtraction, multiplication, and division), bitwise operations (AND, OR, NOT, XOR, shifting), and comparison.

2.2.2 Memory Models

We use a memory access model known as the *extendable arrays* [29] to model the dynamic space usage.

The extendable array is one of the most fundamental data structures in practice. It is implemented by the standard libraries of most popular programming languages, such as `std::vector` in `C++`, `ArrayList` in `java` and `list` in `python`.

[6] The $O(\log u)$-bit representation is implicit.

▶ **Definition 4** (Extendable arrays). *An* extendable array *of length n maintains a sequence of n fixed-sized elements, each assigned a unique address from $[n]$, such that the following operations are supported:*

- `access`(i)*: access the element with address i;*
- `grow`*: increment the length n, creating an arbitrary element with address $n+1$;*
- `shrink`*: decrement the length n, remove the element with address n.*

A collection of extendable arrays *supports*

- `create`(r)*: create an empty extendable array with element of size r and return its name;*
- `destroy`(A)*: destroy the empty extendable array A;*
- `access`(A, i), `grow`(A), `shrink`(A)*: apply the corresponding operations on array A.*

Each of above operations takes constant time. The space overhead of an extendable array is $O(w) + nr$, where w, n, r are the word size, the length of the array, and the element size respectively. Indeed, the space overhead of a collection of extendable arrays is $O(|\mathcal{A}|w) + \sum_{A\in\mathcal{A}} n_A r_A$, where $\mathcal{A}$, n_A and r_A are the set of extendable arrays, the length of array A, and the element size of array A respectively.

We also consider the following *allocate-free* model.

▶ **Definition 5** (Allocate and free). *In the* allocate-free *model, there are two built-in procedures:*

- `allocate`(b)*: return a pointer to a block of b consecutive memory words which is uninitialized;*
- `free`(p)*: free the block of consecutive memory words which is pointed by p and have been initialized to 0s.*

Each of above operations takes constant time. The total space overhead is $O(|\mathcal{A}|w) + \sum_{A\in\mathcal{A}} n_A w$, where $\mathcal{A}$ is set of all memory blocks and n_A is the length of memory block A.

We discuss the space usages of our data structures in allocate-free model in Section 7.

To avoid the pointer being too expensive in the dynamic memory models, we assume $w = \Theta(\log u)$.

2.3 Random Functions

▶ **Definition 6** (k-wise independent random function). *A random function $h : [u] \to [r]$ is called k-*wise independent *if for any distinct $x_1, \cdots, x_k \in [u]$, and any $y_1, \cdots, y_k \in [r]$,*

$$\Pr_h\left[\bigwedge_{i\le k} h(x_i) = y_i\right] = 1/r^k.$$

▶ **Theorem 7** ([31, 10]). *Let $[u]$ be a universe, $w = \Omega(\log u)$, $c_1 > 0$, $r = \mathrm{poly}(u)$, and $k = u^{o(1)}$. There exists a data structure for a random function $h : [u] \to [r]$ such that*

- *with probability $\ge 1 - 1/u$, the data structure is constructed successfully;*
- *upon successful construction of the data structure, h is k-wise independent;*
- *the data structure uses space u^{c_1} bits;*
- *for each $x \in [u]$, $h(x)$ is evaluated in $\tilde{O}(1/c_1)$ time in the worst case in the RAM model.*

▶ **Theorem 8** (Chernoff bound with limited independence [30]). *Let $X_1, \cdots, X_n$ be arbitrary k-wise independent boolean random variables with $\Pr[X_i = 1] = p$ for any $i \in [n]$. Let $X \triangleq \sum_i X_i, \mu \triangleq \mathbb{E}[X] = np$, then for any $\delta > 0$, it holds that*

$$\Pr\left[\, X \ge (1+\delta)\mu \,\right] \le \exp(-\mu\delta^2/2),$$

as long as $k \ge \lceil \frac{\mu\delta}{1-p} \rceil$.

2.4 Adaptive Prefixes

Given a sequence $S = (x_1, x_2, \ldots)$ of strings, let $\alpha_m(S) = \{\alpha_m(x_1), \alpha_m(x_2), \ldots\}$ be a collection of prefixes, such that for every $x_i \in S$, the $\alpha_m(x_i)$ is the shortest prefix, of length at least m, of the binary representation of x_i, such that $\alpha_m(x_i)$ prefixes no other $x_j \in S$. Note that for any string y, there is at most one $x \in S$ such that $\alpha_m(x) \sqsupseteq y$ as long as $\alpha_m(S)$ exists. In particular, $\alpha_m(S)$ does not exist if there are $i \neq j$ such that $x_i = x_j$.

The prefixes are stored in lexicographical order, thus we refer k-th prefix as the prefix with rank k in lexicographical order.

▶ **Theorem 9** (Refined from [4, 5]). *Let $c_0, c_1 > 1$ be two constants where $c_0 > c_1$. For a random sequence $S = (x_1, \cdots)$ of strings drawn from $(\{0,1\}^{c_0 \log u})^{\leq c_3 \log u}$ uniformly at random with replacement, with probability at least $1 - u^{-c_1}$, the prefix collection $\alpha_{\log\log u}(S)$ exists and can be represented with at most $c_2 \log u$ bits, where $c_2 > 0$ is determined by c_1, c_3. Furthermore, the following operations are supported in constant time:*

- insert(y)*: update the representation by inserting a new string $y \in \{0,1\}^{c_0 \log u}$ to S, when there is at most one $x \in S$ such that $\alpha_{\log\log u}(x) \sqsupseteq y$;*
- lookup(y)*: given any query $y \in \{0,1\}^{c_0 \log u}$, return the rank of the only $z \in \alpha_{\log\log u}(S)$ that prefixes y, and return `NO` if there does not exist such a z;*
- lowerbound(y)*: given any query $y \in \{0,1\}^{\log\log u}$, return the lowest rank of all $z \in \alpha_{\log\log u}(S)$ that $z \sqsubseteq y$, and return 0 if there is no $z \sqsubseteq y$ in the collection;*

For completeness, a proof is provided in the full version of this paper.

3 Data Structures for Sets of Unknown Sizes

In this section, we present our filter and dictionary data structures for sets of unknown sizes.

3.1 The Succinct Dynamic Filters

The following theorem is a formal restatement of Theorem 1.

▶ **Theorem 10** (Dynamic filter – formal). *Let $0 < \epsilon < 1$, $[u]$ the data universe, and $\delta = u^{-C}$, where $C > 1$ is an arbitrary constant. Assume the word size $w = \Theta(\log u)$. There exists a data structure for approximate membership for subsets of unknown sizes of $[u]$, such that*

1. *for any $n = \omega(\log u)$ and $n < u$, the data structure uses $n(\log(1/\epsilon) + \log\log n + O(\log\log\log u))$ bits of space after insertions of any n key, and extra u^c precomputed bits that are independent of the input, where $0 < c < 1$ is an arbitrary small constant;*
2. *each insertion and membership query takes $O(1)$ time in the worst case;*
3. *after each insertion, a failure may be reported by the data structure with some probability, and for any sequence of insertions, the probability that a failure is ever reported is at most δ, where the probability is taken over the precomputed random bits;*
4. *conditioned on no failure, each membership query is answered with false positive rate at most ϵ.*

As we mentioned in the introduction, our data structure has $\log n$ stages when handling n insertions. The i-th stage is from the insertion of the $(2^{i-1}+1)$-th key to the 2^i-th key – the database size doubles after each stage.

The main strategy is to reduce the problem of (approximate) membership to (exact) *prefix matching*. More formally, in the *prefix matching* problem, we would like to maintain a set of binary strings $\{s_1, s_2, \ldots\}$ of possibly different lengths, supporting

- insert(s): add string s to the set;
- query(y): decide of any string s in the set is a prefix of y.

To this end, our filter first applies a *global* hash function h such that $h : [u] \to [u^{c_2}]$ is $(c_1 \log u)$-wise independent according to Theorem 7, where $c_1 > 0$ is a constant to be fixed later, and c_2 is a sufficiently large constant (which in fact, is the c_0 in Theorem 9). To insert a key x in stage i, we calculate its hash value $h(x)$, and then insert the ℓ_i-bit prefix of $h(x)$, for some parameter ℓ_i. To answer a membership query y, we simply calculate $h(y)$ and search if any prefix of $h(y)$ is in the database. If no prefix of $h(y)$ is in the database, we output `NO`; otherwise, we output `YES`. It is easy to see that this strategy will never output any false negatives. On the other hand, by union bound, if the query y is not in the set, the probability that the query algorithm outputs `YES` is at most

$$\sum_{i=1}^{\log u} 2^i \cdot 2^{-\ell_i},$$

since h is $(c_1 \log u)$-wise independent (in particular, it is pairwise independent), then the probability that the ℓ_i-bit prefix of $h(y)$ matches with the prefix of the hash value $h(x)$ of key x is $2^{-\ell_i}$. Hence, by setting

$$\ell_i \triangleq i + \log(1/\epsilon) + \log i + \log\log\log u + 2, \tag{2}$$

the false positive rate is at most

$$\sum_{i=1}^{\log u} 2^i \cdot 2^{-i-\log(1/\epsilon)-\log i-\log\log\log u-2} = \epsilon \cdot \sum_{i=1}^{\log u} \frac{1}{4i \log\log u} < \epsilon.$$

We use $\mathcal{D}_{c_1 \log u}$ to denote the distribution of the random insertion sequence $y_1, y_2, \dots, y_n$ for prefix matching constructed above. Formally, $\mathcal{D}_{c_1 \log u}$ is the distribution of a sequence of random strings $y_1, y_2, \dots, y_n$ obtained from $(c_1 \log u)$-wise independent sequence $z_1, z_2, \dots, z_n \in [u^{c_2}]$ by truncating: $\forall 1 \le j \le n$, $y_j = (z_j)_{\le \ell_i}$, where $i = \lceil \log j \rceil$.

▶ **Lemma 11** (Prefix matching). *Let $\delta = u^{-C}$, where $C > 1$ is an arbitrary constant. There exist a constant c_1 and a deterministic data structure for prefix matching such that*

1. *for any $n = \omega(\log u)$ and $n < u$, the data structure uses $n(\ell_{\lceil \log n \rceil} - \log n + O(\log\log\log u))$ bits of space after n insertions, and extra u^c precomputed bits, where $0 < c < 1$ is an arbitrary small constant;*
2. *each insertion and query takes $O(1)$ time in the worst case;*
3. *after each insertion, a failure may be reported by the data structure, and for a random sequence of insertions drawn from $\mathcal{D}_{c_1 \log u}$, the probability that a failure is ever reported is at most δ, where the probability is taken over $\mathcal{D}_{c_1 \log u}$;*
4. *every query is answered correctly if no "fail" is reported.*

We present the construction in Section 4. Using this prefix matching data structure, the space usage of the filter is

- $n(\ell_{\lceil \log n \rceil} - \log n + O(\log\log\log u)) = n(\log(1/\epsilon) + \log\log n + O(\log\log\log u))$ bits,
- and u^c bits for storing h by Theorem 7 and for the precomputed lookup tables described in the appendix of the full version of this paper, both independent of the operation sequence.

Each insertion and query can be handled in constant time given the data structure does not fail. This proves Theorem 10.

3.2 The Succinct Dynamic Dictionaries

The data structure for prefix matching also works well as a dictionary data structure for the insertions with keys are sampled uniformly at random. A worst-case instance can be converted into a random instance by a random permutation $\pi : [u] \to [u]$. Assuming an idealized $(c_1 \log u)$-wise independent random permutation whose representation and evaluation are efficient, the data structure for prefix matching in Lemma 11 can be immediately turned to a dictionary. However, the construction of k-wise independent random permutation with low space and time costs is a longstanding open problem [18].

We show that our data structure can solve the dictionary problem in the worst case unconditionally, at the expense of extra u^c bits of space for storing random bits which are independent of the input.

▶ **Theorem 12** (Dynamic dictionary – formal). *Let $[u] \times \{0,1\}^v$ be the data universe, and $\delta = u^{-C}$, where $C > 1$ is an arbitrary constant. Assume the word size $w = \Theta(v + \log u)$. There exists a data structure for dictionary for sets of unknown sizes of key-value pairs from $[u] \times \{0,1\}^v$, such that*

1. *for any $n = \omega(\log u)$ and $n < u$, the data structure uses $n(\log(u/n) + v + O(\log\log\log u))$ bits of space after insertions of any n key-value pairs, and extra u^c precomputed bits that are independent of the input, where $0 < c < 1$ is an arbitrary small constant;*
2. *each insertion and query takes $O(1)$ time in the worst case;*
3. *after each insertion, a failure may be reported by the data structure with some probability, and for any sequence of insertions, the probability that a failure is ever reported is at most δ, where the probability is taken over the precomputed random bits;*
4. *conditioned on no failure, each query is answered correctly.*

The details of the data structure are postponed to Secion 6.

4 Prefix Matching Upper Bound

In this section, we prove Lemma 11.

Recall the distribution $\mathcal{D}_{c_1 \log u}$ of random insertion sequence $y_1, y_2, \ldots, y_n$ assumed in Lemma 11. Given an insertion sequence $\bar{y} = (y_1, y_2, \ldots, y_n) \sim \mathcal{D}_{c_1 \log u}$, we define the *core set* $B(\bar{y}) \triangleq \{x \in \bar{y} : \forall x' \in \bar{y}, x = x' \vee x' \not\sqsupseteq x\}$, and its subset $B^{(a,b]} \triangleq \{x \in B : |x| \in (a,b]\}$ for any $a < b$. Let $\mathcal{D}^{(a,b]}_{c_1 \log u}$ denote the distribution of $B^{(a,b]}$. We say that a random sequence Y of strings is drawn from $\mathcal{D}^{(a,b]}_{c_1 \log u}$ if it can be obtained by permuting the random core set $B^{(a,b]}$.

We show that Lemma 11 is true as long as there exist a family of deterministic data structures for prefix matching with known capacity m. An instance of the data structure $D = D(m, \ell)$ is parameterized by capacity $m < u$, and string length upper bound $\ell \geq \log m$. The data structure uses u^c bits extra space whose contents are precomputed lookup tables, and supports following functionalities with good guarantees:

- initialize(D) and destroy(D): subroutines for initializing and destroying D respectively. The data structure is successfully initialized (or destroyed) after invoking initialize(D) (destroy(D)) consecutively for $O(m)$ times. When successfully initialized, D uses space $O(m)$ bits. The initialize(D)'s are invoked before all other subroutines and destroy(D)'s are invoked after all other subroutines.
- insert(D, x): insert string x to D, where $\log m < |x| \leq \ell$. After n insertions, D uses at most $n(\ell - \log m + 2\log\log\log u) + O(m)$ bits. Each insertion may cause D to fail. A failure ever occurs for a random insertion sequence Y with probability at most u^{-2C}, as long as Y is drawn from $\mathcal{D}^{(\log m, \ell]}_{c_1 \log u}$, where c_1 is suitably determined by constant C.

- $\mathsf{query}(D,x)$: return one bit to indicate whether there exists a prefix of x in D. The correct answer is always returned as long as D has not failed.
- $\mathsf{decrement}(D)$: try to delete an arbitrary string y in D and return the y if y is deleted. An invoking may delete nothing and hence nothing is returned, but it guarantees that the total number of such empty invoking is at most m. Each invoking that successfully deletes a string frees space $\ell - \log m$ bits. The $\mathsf{decrement}(D)$'s are invoked after all insertions.

▷ Claim 13. Given the deterministic data structures supporting above functionalities in constant time in the worst case, Lemma 11 is true.

Proof. We use an auxiliary structure called *truth table* to deal with short strings. A truth table T_i is a bitmap (i.e. array of bits) of length 2^i and supports the required functionalities in the worst cases:

- T_i is initialized to the all-0 string 0^{2^i}, where each invoking of $\mathsf{initialize}(T_i)$ extends T_i by one 0 until T_i is of length 2^i, and each invoking of $\mathsf{destroy}(T_i)$ shrinks T_i by one bit until T_i is fully destroyed;
- to insert x where $|x| = i$, we set $T_i[x+1] \leftarrow 1$;[7]
- to query x where $|x| = i$, we return `YES` if $T_i[x+1] = 1$ and return `NO` if otherwise;
- to decrement T_i, we maintain a j that traverses from 1 to 2^i, and at each time set $T_i[j] \leftarrow 0$, return $j-1$ if $T_i[j] = 1$, and increment j by 1.

Initially, the prefix matching data structure required by Lemma 11 consists of T_0, T_1 and $D_0 = D(1, \ell_0), D_1 = D(2, \ell_1)$ respectively with capacities $1, 2$, and string lengths ℓ_0, ℓ_1, where ℓ_i is defined in Eq(2).

To insert x, which is the n-th insertion, we set $i \leftarrow \lceil \log n \rceil$, invoke $\mathsf{insert}(D_i, x)$ if there is no prefix of x has been inserted. Then we execute the following procedure for 10 times to maintain our data structure:

1. If T_{i-1} is non-empty, we decrement it by invoking $\mathsf{decrement}(T_{i-1})$. If a y is returned, we insert it into T_i by invoking $\mathsf{insert}(T_i, y \circ 0)$ and $\mathsf{insert}(T_i, y \circ 1)$.
2. If D_{i-1} is non-empty, we invoke $\mathsf{decrement}(D_{i-1})$. If a y is returned, we insert it into D_i by invoking $\mathsf{insert}(D_i, y)$ when $|y| > i$ and insert y into T_i by invoking $\mathsf{insert}(T_i, y)$ otherwise.
3. If T_{i-1} (or D_{i-1}) is empty but not destroyed yet, we invoke $\mathsf{destroy}(T_{i-1})$ (or $\mathsf{destroy}(D_{i-1})$).
4. If T_{i-1} (or D_{i-1}) has been destroyed, we invoke $\mathsf{initialize}(T_{i+1})$ (or $\mathsf{initialize}(D_{i+1})$ for $D_{i+1} = D(2^{i+1}, \ell_{i+1})$ with capacity 2^{i+1} and string length upper bound ℓ_{i+1}), where ℓ_i is defined in Eq(2).

A failure is reported whenever a failure is reported during insertion to D_i.

Clearly, for any integer $n \in [2^i, 2^{i+1})$, after n insertions, all inserted strings are stored in either D_{i-1}, T_{i-1} or D_i, T_i. By the time n reaches 2^{i+1}, D_{i+1}, T_{i+1} have been initialized, all inserted strings are stored in D_i, T_i, and D_{i-1}, T_{i-1} have been destroyed.

Consider the insertion sequence for a fixed D_i. Observe that the strings inserted into D_i must be in the core set $B^{(i,\ell_i]}(\bar{y})$. Therefore the insertion sequence is drawn from $\mathcal{D}^{(i,\ell_i]}_{c_1 \log u}$, which means that insertions to each D_i ever failed with probability at most δ. By union bound, a failure is ever reported with probability at most $\sum_{i=1}^{\log u} u^{-2C} \le u^{-C} = \delta$.

Overall, the data structure uses at most $n(\ell_{\lceil \log n \rceil} - \log n + O(\log\log\log u)) \le n(\log(1/\epsilon) + \log\log n + O(\log\log\log u))$ bits after n insertions, besides the u^c precomputed bits.

[7] For A, a list or array of items, we let $A[i]$ denote the i-th item of A.

Suppose n strings has been inserted, let $i \leftarrow \lceil \log n \rceil$. To query x, we invoke $\mathsf{query}(D_{i-1}, x)$, $\mathsf{query}(D_i, x)$, $\mathsf{query}(T_{i-1}, x_{\leq i-1})$, $\mathsf{query}(T_i, x_{\leq i})$ simultaneously, and return YES if any one of the invokings returns YES.

Obviously each insertion and query takes constant time in the worst case, and it is easy to check that every query is correctly answered as long as no failure is reported. ◁

5 Succinct Prefix Matching with Known Capacity

We now describe the data structures required by Claim 13. The pseudocodes are given in the appendix of the full version of this paper.

The data structure consists of a main table and $m/\log u$ subtables.

We partition each binary string x into four consecutive parts: $st(x), hd(x), hs(x), rt(x)$ of lengths $\log(m/\log u), \log(\log u/\log\log u), \log\log\log u, |x| - \log m$ respectively. Roughly speaking, a datapoint x will be distributed into a subtable according to $st(x)$, then be put into a data block of size $\log u/\log\log u$ according to the order it is inserted, therefore we can save $|st(x)| + |hd(x)| - O(1)$ bits for each datapoint by properly encoding.

Main Table. The main table consists of $m/\log u$ entries, each of which contains a pointer to a subtable. Each insertion/query x is distributed into an entry of the main table addressed by $st(x)$. Recall the word size $w = \Theta(\log u)$. The main table uses $mw/\log u = O(m)$ bits.

Recall that $\bar{y} = (y_1, y_2, \ldots, y_n) \sim \mathcal{D}_{c_1 \log u}$ is transformed from a $(c_1 \log u)$-wise independent sequence $Z = (z_1, z_2, \cdots, z_n)$ by truncating. The insertion sequence Y is drawn from $\mathcal{D}_{c_1 \log u}^{(\log m, \ell]}$ by permuting $B^{(\log m, \ell]}$, the restriction of the core set $B(\bar{y})$ to the strings whose lengths ranges within $(\log m, \ell]$.

Let Y_i, Z_i denote the subsequences of Y, Z which contain all the strings that have prefix i, respectively. By definitions, $|Y_i| \leq |Z_i|$. Recall that Z are $(c_1 \log u)$-wise independent. Due to Theorem 8, the load of entry i exceeds $c_3 \log u$ with probability

$$\Pr[\, |Y_i| \geq c_3 \log u \,] \leq \Pr[\, |Z_i| \geq c_3 \log u \,] \leq \exp(-(c_3 - 1)^2 \log u/2), \tag{3}$$

as long as $c_1 \geq \lceil 2(c_3 - 1)^2 \rceil$. Therefore the max-load of entries of the main table is upper bounded by $c_3 \log u$ with probability at least $1 - (m/\log u)\exp(-(c_3 - 1)^2 \log u/2)$. The data structure reports failure if any entry of the main table overflows. In the rest of the proof, we fairly assume $|Y_i| \leq |Z_i| \leq c_3 \log u$ for all i.

Observe that a datapoint x can be identified with $hd(x) \circ hs(x) \circ rt(x)$ if the entry $i = st(x)$ it is distributed into is fixed. Therefore we let Y_i', Z_i' denote the subsequences generated from Y_i, Z_i by discarding the left-most $\log(m/\log u)$ bits.

Subtable. Each subtable i consists of the following parts to be specified later:

- a collection of fingerprints $\alpha_{\log\log u}(Y_i')$ and its indicator list I_i;
- an (extendable) array of navigators N_i;
- an (extendable) array of data blocks A_i;
- two buffers, $B_{i,u}, B_{i,r}$;
- constant many other local variables.

All the datapoints are stored in array A_i. Given a datapoint x, it is easy to see that the addresses of the entries which contains the information of x is high correlated with the order it is inserted, since any insertion takes constant time in the worst case. Hence we take the fingerprints $\alpha_{\log\log u}(Y_i')$, indicators I_i, navigators N_i, and a tricky way to encode a data block as clues to locate the entries which maintain x. Recall that new insertions is put into the latest data block using a dynamic construction, and we reorganize the full dynamic data block into a static construction. We use buffer $B_{i,u}$ to maintain the dynamic block, and use buffer $B_{i,r}$ to "de-amortize" the reorganization.

At first consider a static version of our data structure. In the static version, the buffers and the indicator list are unnecessary. Let $n_i \triangleq |Y_i|$.

Fingerprints. The collection of fingerprints $\alpha_{\log\log u}(Y_i')$ is obtained by applying Theorem 9 on Y_i' with guarantee $c_1 \geq c_3$. Note that Z_i' are mutually independent as long as $c_1 \geq c_3$. Due to Theorem 9, there exists a constant $c'' > 0$ such that a fingerprint collection $\alpha_{\log\log u}(Z_i')$ for Z_i' can be represented in $c'' \log u$ bits with probability $1 - u^{-c_5}$.

We show that there exists a injective function $P : [|Y_i'|] \to [|Z_i'|]$ such that $\forall j \in [|Y_i'|], Y_i'[j] \sqsupseteq Z_i'[P(j)]$. Due to the injective function P and the guarantee that Y is prefix-free, the fingerprint collection of Y_i' can be represented with the same space and probability guarantees as above.

We define $P : [|Y_i|] \to [|Z_i|]$ by $P(j) \triangleq \min\{k \in [|Z_i|] : Y_i[j] \sqsupseteq Z_i[k]\}$. By the definition, for any $y \in Y_i$, there is $z \in Z$ such that $y \sqsupseteq z$. Recall that all the strings in Y_i has prefix i. Hence for any $y \in Y_i, z \in Z$ such that $y \sqsupseteq z$, it holds that $i \sqsupseteq z$, i.e. $z \in Z_i$. Thus for any $j \in [|Y_i|], \{k \in [n] : Y_i[j] \sqsupseteq Z_i[k]\} \neq \emptyset$. Therefore P is well-defined. On the other hand, for distinct $j, l \in [|Y_i|]$, $\{k \in [n] : Y_i[j] \sqsupseteq Z_i[k]\}$ and $\{k \in [n] : Y_i[l] \sqsupseteq Z_i[k]\}$ are disjoint, since Y_i is prefix-free and there is no such z that x, y prefix z simultaneously for distinct $x, y \in Y_i$. Therefore P is injective. Recall that Y_i', Z_i' are generated by removing the prefix i from the strings in Y_i, Z_i: $\forall j, Y_i[j] = i \circ Y_i'[j], Z_i[j] = i \circ Z_i'[j]$. Therefore P works for Y_i', Z_i' too, i.e. $\forall j \in [|Y_i'|], Y_i'[j] \sqsupseteq Z_i'[P(j)]$.

A failure is reported if any fingerprint collection can not be represented within $c'' \log u$ bits, which occurs with probability at most u^{-c_5}.

The fingerprints are sorted lexicographically, so that by the j-th fingerprint we mean the j-th in lexicographical order. For simplicity, we write $\alpha_i \triangleq \alpha_{\log\log u}(Y_i')$.

A failure is reported if there are more than $c_4 \log u / \log\log u$ datapoints share identical $hd(x)$ and $hs(x)$, which occurs with probability at most

$$\binom{c_3 \log u}{c_4 \log u/\log\log u}\left(\frac{1}{\log u}\right)^{c_4 \log u/\log\log u} < u^{-(c_4 - 0.01)}. \tag{4}$$

The fingerprints cost $O(\log u)$ bits per subtable if no failures.

Navigators. N_i is an array of pointers. For any datapoint, the rank of its fingerprint is synchronized with the index of its navigator. In particular, for the k-th fingerprint in α_i, $N_i[k]$ is the address of the data block which maintains the datapoint with the fingerprint. A data block maintains up to $\log u/\log\log u$ datapoints, thus there are at most $c_3 \log\log u$ data blocks. The navigators cost at most $n_i \log\log\log u + O(n_i)$ bits of space.

Data Blocks. A_i is interpreted as an array of data blocks, with each data block holding up to $\log u/\log\log u$ datapoints.

Consider the following succinct binary representation (called *pocket dictionary* in [6]) of a collection of datapoints $F \in \binom{\{0,1\}^\ell}{m}$: The representation consists of two parts $header(F)$ and $body(F)$. Let $header(x), body(x)$ denote the left-most $\log m$ bits and the right-most $\ell - \log m$ bits of x. Let $n_i' \triangleq |\{x \in F | header(x) = i\}|$, and $F = (x_1, \cdots, x_m)$ sorted lexicographically. Then $header(F) \triangleq 0 \circ 1^{n_0'} \circ 0 \circ 1^{n_1'} \circ 0 \cdots 1^{n_{m-1}'}$ and $body(F) \triangleq body(x_1) \circ body(x_2) \cdots \circ body(x_m)$. It is easy to see that this representation uses $2m + m(\ell - \log m)$ bits of space.

Our static data block is a variant of this representation. Let $(x_1, \cdots, x_{\log u/\log\log u})$ be the sorted list of datapoints maintained by data block j. Data block j consists of a list of headers $(hd(x_1), \cdots, hd(x_{\log u/\log\log u}))$, a list of identities $(hs(x_1), \cdots, hs(x_{\log u/\log\log u}))$ and an array of the rest part of datapoints $(rt(x_1), \cdots, rt(x_{\log u/\log\log u}))$. The header list are represented in the same way as in the pocket dictionary, the identity list is the concatenation $hs(x_1) \circ hs(x_2) \cdots \circ hs(x_{x/\log\log u})$, and the rest part array is the concatenation $\mathtt{PFC}(rt(x_1)) \circ \mathtt{PFC}(rt(x_2)) \cdots \circ \mathtt{PFC}(rt(x_{x/\log\log u}))$, where $\mathtt{PFC}(\cdot)$ is the prefix-free code in Claim 3.

Recall that $|hd(x)| = \log(\log u/\log\log u), |hs(x)| = \log\log\log u$. Therefore the data blocks for the subtable cost at most $O(n_i) + n_i(\ell - \log m + \log\log\log u)$ bits of space.

Query in a Static Data Block. Recall that the fingerprints are sorted in lexicographical order, and the indices of the navigators are synchronized with the ranks of corresponding fingerprints. Also note that a navigator costs $\log\log\log u + O(1)$ bits, there are at most $c_4 \log u/\log\log u$ datapoints share $hd(x) \circ hs(x)$ with any query x. By putting everything together, we can retrieve all the navigators of the datapoints which have the same $hd(x) \circ hs(x)$ with query x, and learn a k' such that the unique suspected datapoint is the k'-th datapoint among the datapoints which share the $hd(x) \circ hs(x)$ in its data block. On the other hand, we can retrieve the header list and identity list with constant number of memory accesses. Consequently, we can retrieve the rest part of the suspected datapoint efficiently. See the pseudocode in the appendix of the full version of this paper for more details.

The space usage is upper bounded by $m + (m/\log u) \cdot O(\log u) + n \log\log\log u + O(n) + n(\ell - \log m + \log\log\log u) \leq n(\ell - \log m + 2\log\log\log u) + O(m)$ bits. And the query time is clearly constant.

Insertion. The new insertions will be put into a data block under construction temporarily, and the data block will be reorganized to the static version as long as the data block is full (which means, there are $\log u/\log\log u$ datapoints stored in it). A data block under construction consists of two incomplete lists for headers and identities, and an (extendable) array of the rest parts of datapoints. Note that the space usages of incomplete lists are identical with the ones of the complete lists, it wastes at most $O(\log u)$ bits per dynamic data block.

Reorganizing a data block (i.e. sorting a data block) can be expensive, therefore we finish this work during the procedure that a new data block under construction is being filled. Hence there are two dynamic data blocks, one under construction and one under reorganization, besides the static ones.

Note that the collection of fingerprints α_i can be updated dynamically with small costs. To retrieve the datapoint y with fingerprint $\alpha(y)$, the only things we need are the address of the data block which maintains y and the in-block index of y. (recall that $hd(y)$ and $hs(y)$ are known due to the fingerprint collection.) It is easy to learn the address as long as we know that y is in a dynamic data block, since there are at most two dynamic blocks. We use the buffers to record the in-block index, and use the indicators to inform whether y is in a dynamic data block.

The list of indicators is a string from $\{1, 2, 3\}^{n_i}$. The value of i-th indicator implies which kind of data block the datapoint corresponding to i-th fingerprint is stored in. For a static data block, the query algorithm works in previous way. For a dynamic data block, the address of the data block can be easily learnt with the counter n_i.

The two buffers are arrays of pointers from $[\log u/\log\log u]^{\log u/\log\log u}$, so they fit in constant number of memory cells. In particular, for a indicator $I_i[j]$ which is the k-th indicator has value 2 (or 3), $B_{i,u}[k]$ (or $B_{i,r}[k]$) is the in-block index of the rest part which corresponds to j-th fingerprint.

The new insertion is not a prefix of some preceded insertion due to our guarantees. Therefore, to insert x, we simply append $hs(x), rt(x)$ to the identity list and rest part array, update the header list, fingerprint collection, and indicator list, and insert a proper pointer into $B_{i,u}$, which overall takes constant time. And to query x in a dynamic data block, where x corresponds to a datapoint y stored in the data block, we need to retrieve the address of the data block with counter n_i and the in-block index of y with the buffers, then retrieve $rt(y)$. See the pseudocodes in the appendix of the full version of this paper for more details.

Reorganizing a Data Block. The reorganization procedure starts as long as the data block under construction is full. Informally, the reorganization procedure works as follows:

1. Update the list of indicators and copy $B_{i,r} \leftarrow B_{i,u}$. (We guarantee that the preceded reorganization process has been finished before the buffer $B_{i,u}$ is full)
2. Insert the address of the data block in proper positions of the navigator list.
3. Sort the array of rest parts and the list of identities according to the pointers in the buffer $B_{i,r}$ while keep the buffer updated. Note that the sorting can be done within time cost $O(\log u/\log\log u)$: we enumerate $j \in [\log u/\log\log u]$, find j' such that $B_{i,r}[j'] = j$, swap $hs_j, rt_j, B_{i,r}[j]$ with $hs_{B_{i,r}[j]}, rt_{B_{i,r}[j]}, B_{i,r}[j']$ one by one, where hs_j, rt_j is the j-th item of the corresponding list and array.
4. Update indicator list.

The total time cost is $O(\log u/\log\log u)$, which can be simulated by $\log u/\log\log u$ operations, each costing $O(1)$ time, within a data block. See the appendix of the full version of this paper for more details.

The two dynamic data blocks waste at most $O(\log u)$ bits on the two incomplete lists for headers and identities, therefore we uses at most extra $O(m)$ bits of space.

Setting the Constant Parameters. Our data structure may fail at the load balancing on subtables, constructing the fingerprint collections for subtables, and the load balancing on headers of fingerprint collections. By the union bound, the failure probability is at most

$$\frac{m}{\log u}\left(\exp(-(c_3-1)^2\log u/2) + u^{-c_5}\right) + m2^{-(c_4-0.01)\log u}, \tag{5}$$

when $c_1 \geq \max\{\lceil 2(c_3-1)^2\rceil, c_3\}$. Since $m < u$,the failure probability can be as small as $\delta = u^{-2C}$ if we set the constants c_1, c_3, c_4, c_5 to be sufficiently large.

The initialize, decrement, and destroy subroutines are easy to implement, which are postponed to the appendix of the full version of this paper

6 Unconditional Succinct Dictionary

We show that our data structure can solve the dictionary in the worst case unconditionally. In our data structure for prefix matching, the randomness is used only for:

1. load balancing on the subtables;
2. the representation of the adaptive prefixes;
3. load balancing on the $hd(x) \circ hs(x)$'s.

Note that we should decode $st(x)$ from subtable index i, decode $hd(x) \circ hs(x)$ from bucket index in adaptive prefixes, and decode $hd(x)$ from bucket index in data block. To achieve the identical guarantees with prefix matching, we apply a weaker but strong enough random permutation.

▶ **Definition 14** (Feistel permutation). *Given any $x \in [u]$, let x_L, x_R respectively denote the $\log m$ left-most bits and the $\log(u/m)$ right-most bits of the binary representation of x, so that $x = x_L \circ x_R$. Given any $f : \{0,1\}^{\log(u/m)} \to \{0,1\}^{\log m}$, the* Feistel permutation *$\pi_f : [u] \to [u]$ is defined as*

$$\pi_f(x) = (x_L \oplus f(x_R)) \circ x_R.$$

It is easy to verify that π_f is indeed a permutation. In fact, $\pi_f(\pi_f(x)) = x$ for any x and f.

Our dictionary data structure works with three $(c_1 \log u)$-wise independent hash functions $f : [\frac{u \log u}{m}] \to [m/\log u]$, $g : [u/m] \to [\log u]$, and $h : [u/m] \to [u^{c_2}]$. Given a key $x \in [u]$, we let $st'(x) = \pi_f(x), hd'(x) \circ hs'(x) = \pi_g(hd(x) \circ hs(x) \circ rt(x))$ and $hss(x) = h(rt(x))$. Then we distribute x into subtable $st'(x)$, insert/query $hd'(x) \circ hs'(x) \circ hss(x)$ to the fingerprint collection, and encode $hd'(x) \circ hs'(x) \circ rt(x)$, instead of $hd(x) \circ hs(x) \circ rt(x)$, in its data block.

Consider two datapoints x, x'. If $hd(x) \circ hs(x) \circ rt(x) \neq hd(x') \circ hs(x') \circ rt(x')$, then $st'(x)$ and $st'(x')$ are independent; otherwise $st'(x) \neq st'(x')$. Therefore for any i, c,

$$\Pr\left[\,|\{x \in Y : st'(x) = i\}| \geq c\,\right] \leq \Pr\left[\,|\{x \in Y' : st(x) = i\}| \geq c\,\right], \tag{6}$$

where Y are the insertion sequence, Y' are $(c_1 \log u)$-wise independent random insertion sequence. Hence the load balancing is not worse than the one in the prefix matching case.

Due to Theorem 9 and Eq(6), the fingerprint collection works with the same guarantees. Clearly $hd(x) \circ hs(x) = (hd'(x) \circ hs'(x)) \oplus g(rt(x)), st(x) = st'(x) \oplus f(hd(x) \circ hs(x) \circ rt(x))$, thus the keys can be retrieved precisely. For the values, we store $rt(x)$ and its value together as a tuple in the data block.

7 Upper Bounds in Allocate-Free Model

We mimic the extendable arrays in the allocate-free model. For simplicity, we modify the navigator list from extendable array to an array of length $c_3 \log u$.

Suppose we are dealing with D_i. The main table can be implemented easily since it has fixed length. Let $s = c_3(\log u)(\log(1/\epsilon) + \log i + O(\log\log\log u))$ be the space usage upper bound of any single subtable. For a subtable i, we maintain a pointers array of length $\lceil\sqrt{s/w}\rceil$ to mimic the extendable array. Every pointer in the array points to a memory block of $\lceil\sqrt{sw}\rceil$ bits. Therefore we waste at most

$$(2^i/\log u) \cdot O(w \cdot \sqrt{s/w} + \sqrt{sw}) = O(2^i\sqrt{\log(1/\epsilon) + \log i + \log\log\log u})$$

bits of space. In conclusion, after n insertions our data structure for filters uses at most

$$n(\log(1/\epsilon) + \log\log n + O(\log\log\log u)) + O(n\sqrt{\log(1/\epsilon) + \log\log n + \log\log\log u})$$

bits of space in the allocate-free model.

Similarly, our data structure for dictionaries uses at most

$$n(\log(u/n) + v + O(\log\log\log u)) + O(n\sqrt{\log(u/n) + v + \log\log\log u})$$

bits of space after n insertions in the allocate-free model.

References

1 Paulo Sérgio Almeida, Carlos Baquero, Nuno M Preguiça, and David Hutchison. Scalable bloom filters. *Information Processing Letters*, 101(6):255–261, 2007.

2 Yuriy Arbitman, Moni Naor, and Gil Segev. De-amortized cuckoo hashing: Provable worst-case performance and experimental results. In *International Colloquium on Automata, Languages, and Programming*, pages 107–118. Springer, 2009.

3 Yuriy Arbitman, Moni Naor, and Gil Segev. Backyard cuckoo hashing: Constant worst-case operations with a succinct representation. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 787–796. IEEE, 2010.

4 Michael A Bender, Martin Farach-Colton, Mayank Goswami, Rob Johnson, Samuel McCauley, and Shikha Singh. Bloom filters, adaptivity, and the dictionary problem. *arXiv preprint*, 2017. `arXiv:1711.01616`.

5 Michael A Bender, Martin Farach-Colton, Mayank Goswami, Rob Johnson, Samuel McCauley, and Shikha Singh. Bloom filters, adaptivity, and the dictionary problem. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 182–193. IEEE, 2018.

6 Ioana O Bercea and Guy Even. Fully-dynamic space-efficient dictionaries and filters with constant number of memory accesses. *arXiv preprint*, 2019. `arXiv:1911.05060`.

7 Andrei Broder and Michael Mitzenmacher. Using multiple hash functions to improve ip lookups. In *Proceedings IEEE INFOCOM*, volume 3, pages 1454–1463. IEEE, 2001.

8 Larry Carter, Robert Floyd, John Gill, George Markowsky, and Mark Wegman. Exact and approximate membership testers. In *Proceedings of the tenth annual ACM symposium on Theory of computing*, pages 59–65, 1978.

9 Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Trans. Comput. Syst.*, 26(2), June 2008.

10 Tobias Christiani, Rasmus Pagh, and Mikkel Thorup. From independence to expansion and back again. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 813–820. ACM, 2015.

11 Erik D Demaine, Friedhelm Meyer auf der Heide, Rasmus Pagh, and Mihai Pătraşcu. De dictionariis dynamicis pauco spatio utentibus. In *Latin American Symposium on Theoretical Informatics*, pages 349–361. Springer, 2006.

12 Martin Dietzfelbinger, Anna R. Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Heide, Hans Rohnert, and Robert Endre Tarjan. Dynamic perfect hashing: Upper and lower bounds. In *29th Annual Symposium on Foundations of Computer Science*, pages 524–531. IEEE, 1988.

13 Martin Dietzfelbinger and Friedhelm Meyer auf der Heide. A new universal class of hash functions and dynamic hashing in real time. In *International Colloquium on Automata, Languages, and Programming*, pages 6–19. Springer, 1990.

14 O. Erdogan and Pei Cao. Hash-av: fast virus signature scanning by cache-resident filters. In *GLOBECOM '05. IEEE Global Telecommunications Conference, 2005.*, volume 3, pages 6 pp.–, November 2005. `doi:10.1109/GLOCOM.2005.1577953`.

15 Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with 0(1) worst case access time. *J. ACM*, 31(3):538–544, 1984.

16 Deke Guo, Jie Wu, Honghui Chen, and Xueshan Luo. Theory and network applications of dynamic bloom filters. In *Proceedings IEEE INFOCOM*, pages 1–12. IEEE, 2006.

17 Mai Jiang, Chunsheng Zhao, Zaifeng Mo, and Jing Wen. An improved algorithm based on bloom filter and its application in bar code recognition and processing. *EURASIP Journal on Image and Video Processing*, 2018(1):139, December 2018. `doi:10.1186/s13640-018-0375-6`.

18 Eyal Kaplan, Moni Naor, and Omer Reingold. Derandomized constructions of k-wise (almost) independent permutations. *Algorithmica*, 55(1):113–133, 2009.

19 Adam Kirsch and Michael Mitzenmacher. Using a queue to de-amortize cuckoo hashing in hardware. In *Proceedings of the Forty-Fifth Annual Allerton Conference on Communication, Control, and Computing*, volume 75, 2007.

20 Paul C Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *Annual International Cryptology Conference*, pages 104–113. Springer, 1996.

21 Richard J Lipton and Jeffrey F Naughton. Clocked adversaries for hashing. *Algorithmica*, 9(3):239–252, 1993.

22 Yi Liu, Xiongzi Ge, David Hung-Chang Du, and Xiaoxia Huang. Par-bf: A parallel partitioned bloom filter for dynamic data sets. *The International Journal of High Performance Computing Applications*, 30(3):259–275, 2016. doi:10.1177/1094342015618452.

23 Shachar Lovett and Ely Porat. A lower bound for dynamic approximate membership data structures. In *IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 797–804, 2010.

24 Bruce M. Maggs and Ramesh K. Sitaraman. Algorithmic nuggets in content delivery. *SIGCOMM Comput. Commun. Rev.*, 45(3):52–66, July 2015. doi:10.1145/2805789.2805800.

25 Moni Naor and Eylon Yogev. Bloom filters in adversarial environments. *ACM Transactions on Algorithms (TALG)*, 15(3):1–30, 2019.

26 Dag Arne Osvik, Adi Shamir, and Eran Tromer. Cache attacks and countermeasures: the case of aes. In *Cryptographers' track at the RSA conference*, pages 1–20. Springer, 2006.

27 Anna Pagh, Rasmus Pagh, and S Srinivasa Rao. An optimal bloom filter replacement. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 823–829, 2005.

28 Rasmus Pagh, Gil Segev, and Udi Wieder. How to approximate a set without knowing its size in advance. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 80–89. IEEE, 2013.

29 Rajeev Raman and Satti Srinivasa Rao. Succinct dynamic dictionaries and trees. In *International Colloquium on Automata, Languages, and Programming*, pages 357–368. Springer, 2003.

30 Jeanette P Schmidt, Alan Siegel, and Aravind Srinivasan. Chernoff–hoeffding bounds for applications with limited independence. *SIAM Journal on Discrete Mathematics*, 8(2):223–250, 1995.

31 Mikkel Thorup. Simple tabulation, fast expanders, double tabulation, and high independence. In *54th Annual Symposium on Foundations of Computer Science*, pages 90–99. IEEE, 2013.

A $(2+\varepsilon)$-Factor Approximation Algorithm for Split Vertex Deletion

Daniel Lokshtanov
University of California, Santa Barbara, CA, USA
daniello@ucsb.edu

Pranabendu Misra
Max Planck Institut für Informatik, Saarland Informatics Campus, Saarbrücken, Germany
pmisra@mpi-inf.mpg.de

Fahad Panolan
IIT Hyderabad, India
fahad@iith.ac.in

Geevarghese Philip
Chennai Mathematical Institute, UMI ReLaX, Chennai, India
gphilip@cmi.ac.in

Saket Saurabh
Institute of Mathematical Sciences, Chennai, India
University of Bergen, Norway
saket@imsc.res.in

Abstract

In the Split Vertex Deletion (SVD) problem, the input is an n-vertex undirected graph G and a weight function $w\colon V(G) \to \mathbb{N}$, and the objective is to find a minimum weight subset S of vertices such that $G-S$ is a split graph (i.e., there is bipartition of $V(G-S) = C \uplus I$ such that C is a clique and I is an independent set in $G-S$). This problem is a special case of 5-Hitting Set and consequently, there is a simple factor 5-approximation algorithm for this. On the negative side, it is easy to show that the problem does not admit a polynomial time $(2-\delta)$-approximation algorithm, for any fixed $\delta > 0$, unless the Unique Games Conjecture fails.

We start by giving a simple quasipolynomial time ($n^{\mathcal{O}(\log n)}$) factor 2-approximation algorithm for SVD using the notion of *clique-independent set separating collection*. Thus, on the one hand SVD admits a factor 2-approximation in quasipolynomial time, and on the other hand this approximation factor cannot be improved assuming UGC. It naturally leads to the following question: Can SVD be 2-approximated in polynomial time? In this work we almost close this gap and prove that for any $\varepsilon > 0$, there is a $n^{\mathcal{O}(\log \frac{1}{\varepsilon})}$-time $2(1+\varepsilon)$-approximation algorithm.

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

Keywords and phrases Approximation Algorithms, Graph Algorithms, Split Vertex Deletion

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.80

Category Track A: Algorithms, Complexity and Games

Funding This work has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (via grant no. 819416 and no. 715744), the Norwegian Research Council(via grants MULTIVAL and CLASSIS), and Swarnajayanti Fellowship grant DST/SJF/MSA-01/2017-18.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 80; pp. 80:1–80:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

The Hitting Set problem encompasses a large number of well studied problems in Computer Science. Here, the input is a family $\mathcal{F}$ of sets over an n-element universe U and a weight function $w\colon U \to \mathbb{N}$, and the objective is to compute a hitting set of minimum weight. A *hitting set* is a subset $S \subseteq U$ such that for any $F \in \mathcal{F}$, $F \cap S \neq \emptyset$ and the weight of S is $w(S) = \sum_{u \in S} w(u)$. This problem generalizes a number of other well studied problems in computer science, and consequently it is very hard to approximate: it can not be approximated within a factor $2^{\log^{1-\delta_c(n)} n}$ in polynomial time, for any constant $c < 1/2$, unless SAT can be decided in slightly subexponential time, where $\delta_c(n) = 1/(\log\log n)^c$ [11]. A restricted version of this problem, is the d-Hitting Set problem, where $d \in \mathbb{N}$ and the cardinality of every set in $\mathcal{F}$ is at most d. This problem also generalizes a number of well studied problems, and it admits a simple factor d-approximation algorithm: Solve the natural LP relaxation and select all elements whose corresponding variable in the LP is set to at least $1/d$. Unfortunately, this simple algorithm is likely to be the best possible. That is, assuming Unique Game Conjecture (UGC), there is no c-factor approximation algorithm for d-Hitting Set, for any $c < d$ in the general case [7].

A number of vertex deletion problems on graphs can be considered as special cases of d-Hitting Set, and it is of great interest to devise factor-α approximation algorithm for them where $\alpha < d$, or rule out any such algorithm. For example, in the Vertex Cover problem, the input is a graph G and a weight function $w\colon V(G) \to \mathbb{N}$, and the objective is to find a subset of vertices of minimum weight that hits all edges in G. This is same as 2-Hitting Set, and assuming the Unique Games Conjecture we cannot do better than a factor-2 approximation in polynomial time. However, there are other examples of vertex deletion problems on graphs, that are special cases of d-Hitting Set, for which we can indeed do better than a factor-d approximation. Consider the Cluster Vertex Deletion problem, where the input is a graph G and a weight function $w\colon V(G) \to \mathbb{N}$, and the objective is to find a minimum weight subset S of vertices such that S is a cluster graph. Equivalently, S hits all induced paths of length 3 in G. Hence, it is a special case of 3-Hitting Set and admits a simple 3-approximation algorithm. You et al. [13] showed that the unweighted version of Cluster Vertex Deletion admits a 5/2 approximation algorithm. Recently, this was improved to factor 9/4 by Fiorini et al. [5]. The problem also admits an approximation-preserving reduction from Vertex Cover and hence there is a lower bound of 2 on the approximation-factor assuming UGC [5]. Fiorini et al. [5] have conjectured that Cluster Vertex Deletion admits a 2-approximation algorithm. Another example is the Tournament Feedback Vertex Set (TFVS) problem, which is equivalent to hitting all directed triangles in a digraph. It is very well studied in the realm of approximation algorithms [3, 1, 10, 9], and very recently a 2-approximation algorithm was designed by Lokshtanov et al. [9], matching the lower-bound under UGC [12]. Similarly, a number of such "implicit" d-Hitting Set problems are studied in Computer Science, and it is of great interest to settle their approximation complexity.

In this work we study another implicit d-Hitting Set problem called Split Vertex Deletion(SVD) (defined below). A subset S of vertices in a graph G is a split vertex deletion set if $G - S$ is a split graph (i.e., there is bipartition of $V(G-S) = C \uplus I$ such that C is a clique and I is an independent set in $G - S$).

Split Vertex Deletion (SVD)
Input: An undirected graph G and a weight function $w : V(G) \to \mathbb{N}$.
Output: A split vertex deletion set $S \subseteq V(G)$ of G of the smallest weight (an *optimum* split vertex deletion set of G).

A graph G is a split graph if and only if it does not contain C_4, C_5 and $2K_2$ as induced subgraphs in G [6]. This implies that SVD is special case of 5-Hitting Set and hence it admits a simple 5-approximation algorithm. Furthermore, it is interesting to note that we can obtain a 2-approximation algorithm for SVD in time $n^{O(\log n)}$ using the notion of *clique-independent set separating collection* [4]. For a graph G, a clique-independent set separating collection is a family $\mathcal{C}$ of vertex subsets of $V(G)$ such that for a clique C and an independent set I in G such that $C \cap I = \emptyset$, there is subset X in the collection $\mathcal{C}$ such that $C \subseteq X$ and $I \subseteq V(G) \setminus X$. Thus, if there is a "small" clique-independent set separating collection, then we can enumerate such a collection $\mathcal{C}$ and solve Vertex Cover of $\overline{G}[X]$ and $G - X$ for each $X \in \mathcal{C}$. Notice that for any $X \in \mathcal{C}$, the union of the two solutions of the two Vertex Cover instances on $\overline{G}[X]$ and $G - X$, respectively, is a solution to SVD. Moreover, the best c-approximation solutions over all choices of X, is a c-approximate solution of SVD. It is known that for any n-vertex graph, there is clique-independent set separating collection of size $n^{\mathcal{O}(\log n)}$ and this can be enumerated in time linear in the size of the collection [4]. This along with a 2-approximation algorithm of Vertex Cover leads to an $n^{\mathcal{O}(\log n)}$-time 2-approximation algorithm for SVD. There is also a simple approximation preserving reduction from Vertex Cover to SVD, which shows that we cannot improve upon factor 2-approximation algorithm, unless UGC fails. The reduction is as follows: Given an instance (G, w) of Vertex Cover, we add a large complete graph H of size $2|V(G)|$ into G with weight of each vertex in H to be $\max\{w(u) \,:\, u \in V(G)\}$. One can easily verify that this is an approximation preserving reduction.

Thus, on the one hand SVD admits a 2-approximation in quasipolynomial ($n^{\mathcal{O}(\log n)}$) time, and on the other hand this approximation factor cannot be improved assuming UGC. It naturally leads to the following question: Can SVD be 2-approximated in polynomial time? This is precisely the question we address in this paper, and obtain the following result.

▶ **Theorem 1.** *Let G be a graph on n vertices, w a weight function on $V(G)$ and let $\varepsilon > 0$ be a constant. Then there exists a randomized algorithm that runs in time $\mathcal{O}(n^{g(\varepsilon)})$ and outputs $S \subseteq V(G)$ such that $G - S$ is a split graph and $w(S) \leq 2(1+\varepsilon)w(OPT)$ with probability at least $1/2$. Here OPT is a minimum weight split vertex deletion set of G, and $g(\varepsilon) = 6 + 8\log(80(1+\frac{12}{\varepsilon})) \cdot \log(\frac{30}{\varepsilon})/\log(4/3)$.*

Overview of Theorem 1. At a very high level the algorithm described in Theorem 1 is inspired from the algorithm developed for factor 2-approximation algorithm for TFVS [9]. In TFVS knowing just one vertex is sufficient to completely split the instance into two independent sub-instances and thus leading to a natural divide and conquer scheme. However, in our case (SVD) the instances don't become truly independent before every vertex is classified as either *potential clique* or *potential independent set vertex.* Classifying all the vertices requires several new ideas and insights in the problem. This classification could be vaguely viewed as a polynomial time algorithm that quickly navigates through sets in clique-independent set separating collection $\mathcal{C}$, and almost reaches a correct partition.

Our algorithm in fact finds a $(2+\varepsilon)$-factor approximate solution for a more general *annotated* variant of the problem, where the solution must obey certain additional constraints.

Annotated Split Vertex Deletion (A-SVD)
Input: An undirected graph G, a weight function $w : V(G) \to \mathbb{N}$, and a partition of $V(G)$ into three parts $V(G) = C \uplus I \uplus U$, where at most two of these parts may be empty.
Output: A set $S^\star \subseteq V(G)$ of G of the smallest weight such that $G - S^\star$ is a split graph with a split partition $(C^\star, I^\star)$ where $C^\star \subseteq (C \cup U)$ and $I^\star \subseteq (I \cup U)$ hold.

A *feasible solution* to an instance $(G, w, (C, I, U))$ of Annotated Split Vertex Deletion is a split vertex deletion set S of G such that the split graph $G - S$ has a split partition (C', I') where no vertex in the specified set I goes to the split part C' and no vertex in the specified set C goes to the independent part I'. Thus, each vertex in the set I is either deleted as part of S or ends up in the independent set I' in graph $G - S$, and each vertex in C is either deleted or ends up in the clique C' in $G - S$. There are no restrictions on where the vertices in the "unconstrained" set U may go. We call a feasible solution of A-SVD an *annotated split vertex deletion set* of the instance $(G, w, (C, I, U))$; the A-SVD problem asks for an *optimum* annotated split vertex deletion set of the input instance.

First we show that we can, in polynomial time, find 2-factor approximate solutions to A-SVD instances which are of the form $(G, w, (C, I, U = \emptyset))$ (Lemma 12). Let $(G, w, (C, I, U))$ be an instance of A-SVD, let OPT be an (unknown) optimum solution to $(G, w, (C, I, U))$, let $(C^\star \subseteq (C \cup U), I^\star \subseteq (I \cup U))$ be a split partition of $G - OPT$, and let $C^\star_U = (C^\star \cap U), I^\star_U = (I^\star \cap U)$. We show that if $w(C^\star_U \setminus \{c^\star\}) \leq \frac{\varepsilon \cdot w(OPT)}{2}$ holds for some $c^\star \in C^\star_U$ or $w(I^\star_U \setminus \{i^\star\}) \leq \frac{\varepsilon \cdot w(OPT)}{2}$ holds for some $i^\star \in I^\star_U$ then we can, in polynomial time, find a $(2+\varepsilon)$-factor approximate solution to $(G, w, (C, I, U))$ (Lemma 16, Lemma 18). These constitute the base cases of our algorithm. It is not difficult to see that moving a vertex $x \in C^\star_U$ to the set C and moving a vertex $y \in I^\star_U$ to the set I are approximation-preserving transformations. At a high level, our algorithm starts with an arbitrary instance $(G, w, (C, I, U))$ of A-SVD, correctly identifies – with a constant probability of success – a good fraction of vertices which belong to the sets $C^\star_U$ or $I^\star_U$, and moves these vertices to the sets C or I, respectively. It then recurses on the resulting instance, till it reaches one of the base cases described above.

We now briefly and informally outline how our algorithm identifies vertices as belonging to $C^\star_U$ or $I^\star_U$. Consider the bipartite subgraph H of G induced by the pair $(C^\star_U, I^\star_U)$. Define the weight of an edge to be the product of the weights of its two end-points, and suppose the total weight of edges in H is at least half the maximum possible weight. Then each of a constant fraction (by weight) of the vertices in $I^\star_U$ has a constant fraction (by weight) of $C^\star_U$ in its neighborhood (Lemma 4). If we can identify one of these special vertices of $I^\star_U$ then we can safely move all its neighbors in U to the set C while reducing the weight of $C^\star_U$ by a constant fraction. The catch, of course, is that we have no idea what the set $I^\star_U$ is.

To get around this, we find an approximate solution X of the Split Vertex Deletion instance defined by the induced subgraph $G[U]$. Let (C_X, I_X) be a split partition of $G - X$. We show that we can, in polynomial time and with constant probability, sample a vertex from the set $X \cup (I_X \setminus C^\star_U)$ (Lemma 26). We further show that the weight of $X \cup (I_X \setminus C^\star_U)$ is at most a *constant multiple* of the weight of $I^\star_U$ (Lemma 22). So if $I^\star_U \subseteq (X \cup (I_X \setminus C^\star_U))$ holds then we can, with good probability, sample a vertex from the set $I^\star_U$. The hard part is when this condition does not hold. We show using a series of lemmas (summarized in Lemma 25) that we can, even in this case, sample a vertex from one of the two sets $C^\star_U, I^\star_U$ with constant probability. A symmetric analysis applies when the total weight of *non-edges* across $(C^\star_U, I^\star_U)$ is at least half the maximum possible weight.

2 Preliminaries

We use $\uplus$ to denote the disjoint union of sets. Moreover, when we write $X \uplus Y$ we implicitly assert that the sets X and Y are disjoint. We use $V(G)$ (respectively, $E(G)$) to denote the vertex set (respectively, the edge set) of graph G. For a subset $S \subseteq V(G)$ of vertices of G we use $G[S]$ to denote the subgraph of G *induced* by S and $G - S$ to denote the subgraph of G obtained by deleting all vertices in S (and their incident edges) from G. A *non-edge* in a graph G is any 2-subset $\{x, y\} \subseteq V(G)$ of vertices such that xy is not an edge in G. For the sake of brevity we use the notation xy to denote a non-edge $\{x, y\}$. For a finite set U, weight function $w : U \to \mathbb{N}$, and subset $X \subseteq U$ we use w_X to denote the weight function w *restricted to the subset* X, and $w(X)$ to denote the sum $\sum_{x \in X} w(x)$ of weights of all the elements in X. For the sake of brevity we drop the subscript X from the expression w_X when there is no risk of ambiguity.

The operation of *sampling (or picking) proportionately at random* from U according to the weight function w chooses one element from U, where each element $x \in U$ is chosen with probability $w(x)/w(U)$. We use $\overline{G}$ to denote the *complement* of a graph G, defined as follows: The vertex set of $\overline{G}$ is $V(G)$. For every two vertices $\{u, v\} \subseteq V(G)$ there is an edge uv in $\overline{G}$ if and only if uv is *not* an edge in graph G. A *vertex cover* of graph G is any subset $S \subseteq V(G)$ of its vertex set such that the graph $G - S$ has no edges. A *clique* in graph G is any non-empty subset $S \subseteq V(G)$ of its vertex set such that (i) $|S| = 1$, or (ii) if $|S| \geq 2$ then for every two vertices u, v in S, the edge uv is present in graph G.

▶ **Observation 2.** *For an undirected graph G and any $S \subseteq V(G)$, the vertex set $V(G) \setminus S$ is a clique in G if and only if S is a vertex cover of the complement graph $\overline{G}$.*

For a graph G and two disjoint vertex subsets $X, Y \subseteq V(G)$; $X \cap Y = \emptyset$ *the bipartite subgraph of G induced by the pair* (X, Y) has vertex set $X \cup Y$ and edge set $\{xy \mid x \in X, y \in Y, xy \in E(G)\}$. Note that the bipartite subgraph of G induced by the pair (X, Y) is not necessarily identical to the subgraph $G[X \cup Y]$ induced by the subset $X \cup Y$, and is defined even if the induced subgraph $G[X \cup Y]$ is not bipartite. For a bipartite graph H with vertex bipartition $V(H) = V_1 \uplus V_2$ we define $\widehat{E(H)} = \{v_1v_2 \mid v_1 \in V_1, v_2 \in V_2, v_1v_2 \notin E\}$ to be the set of all **non-edges** of H *with one end in* V_1 *and the other end in* V_2. Further, for a weight function $w : V(H) \to \mathbb{N}$ defined on the vertex set of a bipartite graph H we define the weight of its edge set to be $w(E(H)) = \sum_{v_1v_2 \in E(H)}(w(v_1) \cdot w(v_2))$ and the weight of its set of non-edges to be $w(\widehat{E(H)}) = \sum_{v_1v_2 \in \widehat{E(H)}}(w(v_1) \cdot w(v_2))$.

▶ **Definition 3.** *Let G be an undirected graph and $w : V(G) \to \mathbb{N}$ a weight function. Let X, Y be two disjoint vertex subsets of G and let H be the bipartite subgraph of G induced by the pair (X, Y). Let $w(E(H))$ and $w(\widehat{E(H)})$ be defined as above. We say that (X, Y) is a* heavy pair *if $w(E(H)) \geq \frac{w(X) \cdot w(Y)}{2}$ holds, and is a* light pair *if $w(\widehat{E(H)}) \geq \frac{w(X) \cdot w(Y)}{2}$ holds.*

▶ **Lemma 4 (♣).** [1] *Let $H = (V, E)$ be a bipartite graph, let $V = V_1 \uplus V_2$ be a bipartition of H, and let $w : V(H) \to \mathbb{N}$ be a weight function. Then (V_1, V_2) is either a heavy pair or a light pair. Moreover,*

1. *Suppose (V_1, V_2) is a heavy pair, and let $X = \{x \in V_1 \mid w(N(x)) \geq \frac{w(V_2)}{4}\}$ be the set of all vertices x in the set V_1 such that the total weight of the neighborhood of x in the set V_2 is at least one-fourth the total weight of the set V_2. Then $w(X) > \frac{w(V_1)}{4}$.*
2. *Suppose (V_1, V_2) is a light pair, and let $Y = \{y \in V_1 \mid w(V_2 \setminus N(y)) \geq \frac{w(V_2)}{4}\}$ be the set of all vertices y in the set V_1 such that the total weight of the* non-neighbors *of y in the set V_2 is at least one-fourth the total weight of the set V_2. Then $w(Y) > \frac{w(V_1)}{4}$.*

For a graph G given together with a weight function $w : V(G) \to \mathbb{N}$, an *optimum vertex cover* of G is any vertex cover of G with the least total weight.

[1] Proofs of statements labeled with a ♣ will appear in the full version of the paper.

Weighted Vertex Cover (wVC)
Input: An undirected graph G and a weight function $w : V(G) \to \mathbb{N}$.
Output: An optimum vertex cover $S \subseteq V(G)$ of G

▶ **Theorem 5** ([2]). *There is an algorithm which, given an instance (G, w) of Weighted Vertex Cover as input, runs in $\mathcal{O}(|E(G)|)$ time and outputs a vertex cover S of G whose weight is at most twice the weight of an optimum vertex cover of G.*

3 The Algorithm

Let (G, w) be an instance of Split Vertex Deletion. Since deleting vertices conserves the property of being a split graph one can safely add zero-weight vertices to any split vertex deletion set. So we assume without loss of generality that $w(v) \geq 1$ holds for every $v \in V(G)$. Split Vertex Deletion is NP-complete by the meta-result of Lewis and Yannakakis [8], and has a simple 5-factor approximation algorithm based on the Local Ratio Technique.

▶ **Theorem 6** (♣). *There is a deterministic algorithm which, given an instance (G, w) of SVD, runs in $\mathcal{O}(|V(G)|^6)$ time and outputs a split vertex deletion set $S \subseteq V(G)$ of G such that $w(S) \leq 5 \cdot w(OPT)$ where OPT is an optimum split vertex deletion set of G.*

We describe a randomized polynomial-time algorithm which outputs a $(2+\varepsilon)$-factor approximate solution for this problem for any fixed $\varepsilon > 0$.

Note that in an instance $(G, w, (C, I, U))$ of Annotated Split Vertex Deletion the set C is not necessarily a clique, nor is I necessarily an independent set in G. But we have the following.

▶ **Observation 7.** *Let S be a feasible solution of an A-SVD instance $(G, w, (C, I, U))$ and let (C', I') be a split partition of $G - S$ where $C' \subseteq (C \cup U)$ and $I' \subseteq (I \cup U)$ hold. Then $C \setminus S \subseteq C'$ and $I \setminus S \subseteq I'$ hold. Hence $C \setminus S$ is a clique in G and $I \setminus S$ is an independent set in G.*

From Observations 2 and 7 we get

▶ **Corollary 8.** *Let S be a feasible solution of an A-SVD instance $(G, w, (C, I, U))$. Let VC_C be an optimum solution of the wVC instance $(\overline{G[C]}, w)$ and let VC_I be an optimum solution of the wVC instance $(G[I], w)$. Then $w(VC_C) \leq w(S \cap C)$ and $w(VC_I) \leq w(S \cap I)$ hold.*

A-SVD is clearly a generalization of SVD: Given an instance (G, w) of SVD, construct the instance $(G, w, (C = \emptyset, I = \emptyset, U = V(G)))$ of A-SVD. Every split vertex deletion set of graph G is a feasible solution of the A-SVD instance, and every annotated split vertex deletion set of $(G, w, (\emptyset, \emptyset, V(G)))$ is a split vertex deletion set of graph G. It follows that for any constant c, a c-factor approximate solution to the A-SVD instance is a c-factor approximate solution to the SVD instance as well.

We can find feasible solutions to an A-SVD instance $(G, w, (C, I, U))$ by computing vertex covers for certain pairs of subgraphs derived from G.

▶ **Observation 9** (♣). *Let $(G, w, (C, I, U))$ be an instance of A-SVD.*

1. *Let V_1 be a vertex cover of the graph $G[I \uplus U]$ and let V_2 be a vertex cover of the graph $\overline{G[C]}$. Then $V_1 \uplus V_2$ is a feasible solution to $(G, w, (C, I, U))$.*
2. *Let V_3 be a vertex cover of the graph $G[I]$ and let V_4 be a vertex cover of the graph $\overline{G[C \uplus U]}$. Then $V_3 \uplus V_4$ is a feasible solution to $(G, w, (C, I, U))$.*

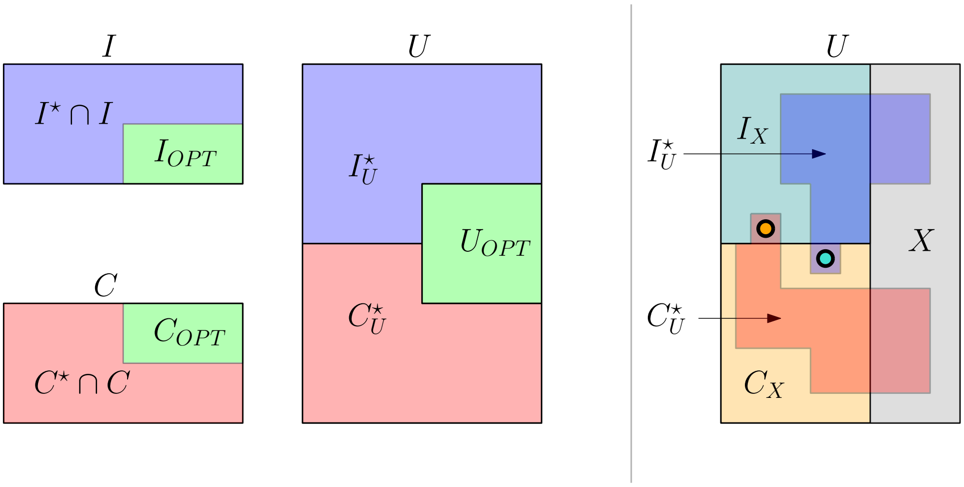


Figure 1 Illustration of Definition 13.

▶ **Observation 10** (♣). *Let $(G, w, (C, I, U))$ be an instance of A-SVD and let $u \in U$.*

1. *Let V_1 be a vertex cover of the graph $G[I \uplus (U \setminus \{u\})]$ and let V_2 be a vertex cover of the graph $\overline{G[C \cup \{u\}]}$. Then $V_1 \uplus V_2$ is a feasible solution to $(G, w, (C, I, U))$.*
2. *Let V_3 be a vertex cover of the graph $G[I \cup \{u\}]$ and let V_4 be a vertex cover of the graph $\overline{G[C \uplus (U \setminus \{u\})]}$. Then $V_3 \uplus V_4$ is a feasible solution to $(G, w, (C, I, U))$.*

Observation 9 has some interesting consequences. For instance, it implies that when the “unconstrained” set U in an A-SVD instance is *empty*, an optimum solution to the A-SVD instance corresponds to optimum solutions of two Weighted Vertex Cover instances derived from the A-SVD instance in a natural fashion.

▶ **Lemma 11** (♣). *Let $S^\star$ be an optimum solution to an A-SVD instance $(G, w, (C, I, U = \emptyset))$. Then the set $(S^\star \cap I)$ is an optimum solution to the wVC instance $(G[I], w)$, and the set $(S^\star \cap C)$ is an optimum solution to the wVC instance $(\overline{G[C]}, w)$.*

This in turn implies that given an A-SVD instance in which the unconstrained set U is empty, we can find a 2-factor approximate solution to the instance in polynomial time.

▶ **Lemma 12** (♣). *There is a deterministic algorithm that finds a 2-factor approximate solution to an A-SVD instance that is of the form $(G, w, (C, I, U = \emptyset))$, in $\mathcal{O}(|E(G)|)$ time.*

This idea generalizes as follows. Let OPT be an optimum solution to an A-SVD instance $(G, w, (C, I, U))$. Suppose the split graph $G - OPT$ has a split partition $(C^\star, I^\star)$ such that vertices from the unconstrained set U *contribute a small weight* to either the clique $C^\star$ or the independent set $I^\star$. Then a variant of the algorithm in the proof of Lemma 12 yields a small-factor approximate solution to the instance, in polynomial time. We state this formally in Lemma 16 below, for which we need some notation (see Figure 1).

▶ **Definition 13.** *Let $(G, w, (C, I, U))$ be an instance of A-SVD, and let $\varepsilon \geq 0$ be a constant. Let $OPT \subseteq V(G)$ be an optimum solution of $(G, w, (C, I, U))$ and let $(C^\star, I^\star)$ be a split partition of the split graph $G^\star = (G - OPT)$ such that $C^\star \subseteq (C \cup U)$ and $I^\star \subseteq (I \cup U)$ hold. Let $C^\star_U = (C^\star \cap U)$ be the set of vertices from the unconstrained set U which become part of the clique $C^\star$ and let $I^\star_U = (I^\star \cap U)$ be the set of vertices from U which become part of the independent set $I^\star$ in $G^\star$. Let $U_{OPT} = (U \cap OPT)$, $C_{OPT} = (C \cap OPT)$ and $I_{OPT} = (I \cap OPT)$.*

Further, let X be a 5-factor approximate solution of the Split Vertex Deletion *instance* $(G[U], w_U)$ defined by the induced subgraph $G[U]$*, and let* (C_X, I_X) *be a split partition of the split graph* $G[U] - X$*.*

▶ Remark 14. Given an instance $(G, w, (C, I, U))$ of A-SVD we can, using Theorem 6, compute such a set X and partition (C_X, I_X) in polynomial time.

▶ **Observation 15 (♣).** *Let* $(G, w, (C, I, U)), X, I_X, C_X, I_U^\star, C_U^\star$ *be as in Definition 13. Then both* $|I_U^\star \setminus (X \cup (I_X \setminus C_U^\star))| \leq 1$ *and* $|C_U^\star \setminus (X \cup (C_X \setminus I_U^\star))| \leq 1$ *hold.*

▶ **Lemma 16 (♣).** *Let* $(G, w, (C, I, U)), \varepsilon, OPT, C_U^\star, I_U^\star$ *be as in Definition 13. Let* S_1 *be a 2-factor approximate solution for the* WVC *instance* $(G[I \cup U], w)$ *and* S_2 *a 2-factor approximate solution for the* WVC *instance* $(\overline{G[C]}, w)$*. Let* $S_{12} = (S_1 \cup S_2)$*. Let* S_3 *be a 2-factor approximate solution for the* WVC *instance* $(\overline{G[C \cup U]}, w)$ *and* S_4 *a 2-factor approximate solution for the* WVC *instance* $(G[I], w)$*. Let* $S_{34} = (S_3 \cup S_4)$*. Then the sets* S_{12} *and* S_{34} *can be computed in* $\mathcal{O}(|E(G)|)$ *time. Further,*

1. *If* $w(C_U^\star) \leq \frac{\varepsilon \cdot w(OPT)}{2}$ *holds then the set* S_{12} *is a* $(2+\varepsilon)$*-factor approximate solution for the* Annotated Split Vertex Deletion *instance* $(G, w, (C, I, U))$*.*
2. *If* $w(I_U^\star) \leq \frac{\varepsilon \cdot w(OPT)}{2}$ *holds then the set* S_{34} *is a* $(2+\varepsilon)$*-factor approximate solution for the* Annotated Split Vertex Deletion *instance* $(G, w, (C, I, U))$*.*

▶ Remark 17. Note that these two cases are neither exclusive nor exhaustive.

By repeatedly applying the procedure in the proof of Lemma 16 and taking the minimum, we can find a $(2+\varepsilon)$-factor approximate solution in polynomial time even in the more general case where there is at most one "heavy" vertex in $C_U^\star$ or $I_U^\star$ that

▶ **Lemma 18 (♣).** *Let* $(G, w, (C, I, U)), \varepsilon, OPT, C_U^\star, I_U^\star$ *be as in Definition 13. For each vertex* $u \in U$ *let* S_1^u *be a 2-factor approximate solution for the* WVC *instance* $(G[I \cup (U \setminus \{u\})], w)$*,* S_2^u *a 2-factor approximate solution for the* WVC *instance* $(\overline{G[C \cup \{u\}]}, w)$*, and let* $S_{12}^u = S_1^u \cup S_2^u$*. Let* S_3^u *be a 2-factor approximate solution for the* WVC *instance* $(\overline{G[C \cup (U \setminus \{u\})]}, w)$*,* S_4^u *a 2-factor approximate solution for the* WVC *instance* $(G[I \cup \{u\}], w)$*, and let* $S_{34}^u = S_3^u \cup S_4^u$*. Finally, let* $S^\dagger$ *be a set of the form* S_{12}^u *of the minimum weight and let* $S^\ddagger$ *be a set of the form* S_{34}^u *of the minimum weight, both minima taken over all vertices* $u \in U$*.*

The sets $S^\dagger$ *and* $S^\ddagger$ *can be computed in* $\mathcal{O}(|V(G)| \cdot |E(G)|)$ *time. Further,*

1. *If* $w(C_U^\star \setminus \{c^\star\}) \leq \frac{\varepsilon \cdot w(OPT)}{2}$ *holds for some vertex* $c^\star \in C_U^\star$ *then the set* $S^\dagger$ *is a* $(2+\varepsilon)$*-factor approximate solution for the A-SVD instance* $(G, w, (C, I, U))$*.*
2. *If* $w(I_U^\star \setminus \{i^\star\}) \leq \frac{\varepsilon \cdot w(OPT)}{2}$ *holds for some vertex* $i^\star \in I_U^\star$ *then the set* $S^\ddagger$ *is a* $(2+\varepsilon)$*-factor approximate solution for the A-SVD instance* $(G, w, (C, I, U))$*.*

▶ Remark 19. Note that these two cases are neither exclusive nor exhaustive.

▶ **Definition 20.** *Let* $(G, w, (C, I, U)), \varepsilon, OPT, C^\star, I^\star, C_U^\star, I_U^\star$ *be as in Definition 13. We say that* $(G, w, (C, I, U))$ *is an* easy *instance if* $U = \emptyset$ *holds, or if* at least one *of the following holds: (i)* $w(C_U^\star) \leq \frac{\varepsilon \cdot w(OPT)}{2}$*, (ii)* $w(I_U^\star) \leq \frac{\varepsilon \cdot w(OPT)}{2}$*, (iii)* $w(C_U^\star \setminus \{c^\star\}) \leq \frac{\varepsilon \cdot w(OPT)}{2}$ *holds for some vertex* $c^\star \in C_U^\star$*, (iv)* $w(I_U^\star \setminus \{i^\star\}) \leq \frac{\varepsilon \cdot w(OPT)}{2}$ *holds for some vertex* $i^\star \in I_U^\star$*. We say that* $(G, w, (C, I, U))$ *is a* hard *instance otherwise.*

From Lemma 12, Lemma 16 and Lemma 18 we get

▶ **Corollary 21.** *There is an algorithm which, given an* easy *instance* $(G, w, (C, I, U))$ *of A-SVD and a constant* $\varepsilon > 0$ *as input, computes a* $(2+\varepsilon)$*-factor approximate solution for* $(G, w, (C, I, U))$ *in deterministic polynomial time.*

▶ **Lemma 22 (♣).** *Let $(G, w, (C, I, U))$ be a* hard *instance of A-SVD and let $\varepsilon, C^{\star}_{U}, I^{\star}_{U}, X, I_X, C_X$ be as in Definition 13. Then the following hold:*

1. $w(X \cup (I_X \setminus C^{\star}_{U})) < (1 + \frac{12}{\varepsilon}) \cdot w(I^{\star}_{U})$
2. $w(X \cup (C_X \setminus I^{\star}_{U})) < (1 + \frac{12}{\varepsilon}) \cdot w(C^{\star}_{U})$

Recall the notion of heavy and light pairs from Definition 3.

▶ **Lemma 23 (♣).** *Let $(G, w, (C, I, U))$ be a* hard *instance of A-SVD and let $\varepsilon, OPT, C^{\star}, I^{\star}, C^{\star}_{U}, I^{\star}_{U}$ be as in Definition 13. Suppose $(I^{\star}_{U}, C^{\star}_{U})$ is a* heavy *pair. Let $I^{\bigcirc} = \{v \in I^{\star}_{U} \; ; \; w(N(v) \cap C^{\star}_{U}) \geq \frac{w(C^{\star}_{U})}{4}\}$ be the set of vertices in $I^{\star}_{U}$ which have a "heavy" neighborhood in $C^{\star}_{U}$, and let $i^{\bigcirc}$ be a heaviest vertex in $I^{\bigcirc}$, i.e. a vertex of maximum weight. Let $C^{\bigcirc} = \{v \in C^{\star}_{U} \; ; \; w((I^{\star}_{U} \setminus \{i^{\bigcirc}\}) \setminus (N(v) \cap I^{\star}_{U})) \geq \frac{w(I^{\star}_{U} \setminus \{i^{\bigcirc}\})}{4}\}$ be the set of vertices in $C^{\star}_{U}$ which have a "heavy"* non-neighborhood *in the subset $I^{\star}_{U} \setminus \{i^{\bigcirc}\}$, and let $c^{\bigcirc}$ be a heaviest vertex in $C^{\bigcirc}$. Let $I^{\square} = \{v \in (I^{\star}_{U} \setminus \{i^{\bigcirc}\}) \; ; \; w(N(v) \cap (C^{\star}_{U} \setminus \{c^{\bigcirc}\})) \geq \frac{w(C^{\star}_{U} \setminus \{c^{\bigcirc}\})}{4}\}$ be the set of vertices in $I^{\star}_{U} \setminus \{i^{\bigcirc}\}$ which have a "heavy" neighborhood in $C^{\star}_{U} \setminus \{c^{\bigcirc}\}$, and let $C^{\square} = \{v \in (C^{\star}_{U} \setminus \{c^{\bigcirc}\}) \; ; \; w((I^{\star}_{U} \setminus \{i^{\bigcirc}\}) \setminus (N(v) \cap I^{\star}_{U})) \geq \frac{w(I^{\star}_{U} \setminus \{i^{\bigcirc}\})}{4}\}$ be the set of vertices in $(C^{\star}_{U} \setminus \{c^{\bigcirc}\})$ which have a "heavy"* non-neighborhood *in $I^{\star}_{U} \setminus \{i^{\bigcirc}\}$.*

Then at least one of the following statements holds:

(1a) *Picking a vertex proportionately at random from the set $X \cup (I_X \setminus C^{\star}_{U})$ yields a vertex $v \in I^{\bigcirc}$ with probability at least $1/(20(1 + \frac{12}{\varepsilon}))$.*

(1b) *Picking a vertex proportionately at random from the set $X \cup (I_X \setminus C^{\star}_{U})$ yields a vertex $v \in I^{\square}$ with probability at least $1/(4(1 + \frac{12}{\varepsilon}))$.*

(2a) *Picking a vertex proportionately at random from the set $X \cup (C_X \setminus I^{\star}_{U})$ yields a vertex $v \in C^{\bigcirc}$ with probability at least $1/(20(1 + \frac{12}{\varepsilon}))$.*

(2b) *Picking a vertex proportionately at random from the set $X \cup (C_X \setminus I^{\star}_{U})$ yields a vertex $v \in C^{\square}$ with probability at least $1/(4(1 + \frac{12}{\varepsilon}))$.*

▶ **Lemma 24 (♣).** *Let $(G, w, (C, I, U))$ be a* hard *instance of A-SVD and let $\varepsilon, OPT, C^{\star}, I^{\star}, C^{\star}_{U}, I^{\star}_{U}$ be as in Definition 13. Suppose $(I^{\star}_{U}, C^{\star}_{U})$ is a* light *pair. Let $C^{\|} = \{v \in C^{\star}_{U} \; ; \; w(I^{\star}_{U} \setminus (N(v) \cap I^{\star}_{U})) \geq \frac{w(I^{\star}_{U})}{4}\}$ be the set of vertices in $C^{\star}_{U}$ which have a "heavy"* non-neighborhood *in $I^{\star}_{U}$, and let $c^{\|}$ be a heaviest vertex in $C^{\|}$. Let $I^{\|} = \{v \in I^{\star}_{U} \; ; \; w(N(v) \cap (C^{\star}_{U} \setminus \{c^{\|}\})) \geq \frac{w(C^{\star}_{U} \setminus \{c^{\|}\})}{4}\}$ be the set of vertices in $I^{\star}_{U}$ which have a "heavy" neighborhood in the subset $C^{\star}_{U} \setminus \{c^{\|}\}$, and let $i^{\|}$ be a heaviest vertex in $I^{\|}$. Let $C^{\ddagger} = \{v \in (C^{\star}_{U} \setminus \{c^{\|}\}) \; ; \; w((I^{\star}_{U} \setminus \{i^{\|}\}) \setminus (N(v) \cap I^{\star}_{U})) \geq \frac{w(I^{\star}_{U} \setminus \{i^{\|}\})}{4}\}$ be the set of vertices in $C^{\star}_{U} \setminus \{c^{\|}\}$ which have a "heavy"* non-neighborhood *in $I^{\star}_{U} \setminus \{i^{\|}\}$, and let $I^{\ddagger} = \{v \in (I^{\star}_{U} \setminus \{i^{\|}\}) \; ; \; w(N(v) \cap (C^{\star}_{U} \setminus \{c^{\|}\})) \geq \frac{w(C^{\star}_{U} \setminus \{c^{\|}\})}{4}\}$ be the set of vertices in $(I^{\star}_{U} \setminus \{i^{\|}\})$ which have a "heavy" neighborhood in $C^{\star}_{U} \setminus \{c^{\|}\}$.*

Then at least one of the following statements is true.

(1a) *Picking a vertex proportionately at random from the set $X \cup (C_X \setminus I^{\star}_{U})$ yields a vertex $v \in C^{\|}$ with probability at least $1/(20(1 + \frac{12}{\varepsilon}))$, or*

(1b) *Picking a vertex proportionately at random from the set $X \cup (C_X \setminus I^{\star}_{U})$ yields a vertex $v \in C^{\ddagger}$ with probability at least $1/(4(1 + \frac{12}{\varepsilon}))$.*

(2a) *Picking a vertex proportionately at random from the set $X \cup (I_X \setminus C^{\star}_{U})$ yields a vertex $v \in I^{\|}$ with probability at least $1/(20(1 + \frac{12}{\varepsilon}))$, or*

(2b) *Picking a vertex proportionately at random from the set $X \cup (I_X \setminus C^{\star}_{U})$ yields a vertex $v \in I^{\ddagger}$ with probability at least $1/(4(1 + \frac{12}{\varepsilon}))$.*

From Lemma 23 and Lemma 24 we get

▶ **Lemma 25.** *Let $(G, w, (C, I, U))$ be a* hard *instance of A-SVD and let $\varepsilon, OPT, C^\star, I^\star, C^\star_U$, $I^\star_U$ be as in Definition 13. Then one of the following statements is true.*

(1a) *Picking a vertex proportionately at random from $X \cup (I_X \setminus C^\star_U)$ yields a vertex from $\{v \in I^\star_U \mid w(N(v) \cap C^\star_U) \geq \frac{w(C^\star_U)}{4}\}$ with probability at least $1/20(1+\frac{12}{\varepsilon})$.*

(1b) *Picking a vertex proportionately at random from $X \cup (I_X \setminus C^\star_U)$ yields a vertex from $\{v \in I^\star_U \mid w(N(v) \cap (C^\star_U \setminus \{c^\star\})) \geq \frac{w(C^\star_U \setminus \{c^\star\})}{4}\}$ with probability at least $1/20(1+\frac{12}{\varepsilon})$, for some vertex $c^\star \in C^\star_U$.*

(2a) *Picking a vertex proportionately at random from $X \cup (C_X \setminus I^\star_U)$ yields a vertex from $\{v \in C^\star_U \mid w(I^\star_U \setminus N(v)) \geq \frac{w(I^\star_U)}{4}\}$ with probability at least $1/20(1+\frac{12}{\varepsilon})$.*

(2b) *Picking a vertex proportionately at random from $X \cup (C_X \setminus I^\star_U)$ yields a vertex from $\{v \in C^\star_U \mid w((I^\star_U \setminus \{i^\star\}) \setminus N(v)) \geq \frac{w(I^\star_U \setminus \{i^\star\})}{4}\}$ with probability at least $1/20(1+\frac{12}{\varepsilon})$, for some vertex $i^\star \in I^\star_U$.*

Proof. From Lemma 4 we get that $(I^\star_U, C^\star_U)$ is either a heavy pair or a light pair. If $(I^\star_U, C^\star_U)$ is a heavy pair then Lemma 23 applies, and at least one of the four options of that lemma holds. Option **(1a)** of Lemma 23 implies option **(1a)** of the current lemma. Option **(1b)** of Lemma 23 implies option **(1b)** of the current lemma. Options **(2a)** and **(2b)** of Lemma 23 both imply option **(2b)** of the current lemma.

If $(I^\star_U, C^\star_U)$ is a light pair then Lemma 24 applies, and at least one of the four options of that lemma holds. Option **(1a)** of Lemma 24 implies option **(2a)** of the current lemma. Option **(1b)** of Lemma 24 implies option **(2b)** of the current lemma. Options **(2a)** and **(2b)** of Lemma 24 both imply option **(1b)** of the current lemma.

Thus in every case, one of the four options of the current lemma holds. ◀

▶ **Lemma 26 (♣).** *Let $(G, w, (C, I, U))$ be a* hard *instance of A-SVD and let $\varepsilon, OPT, C^\star, I^\star$, $C^\star_U, I^\star_U$ be as in Definition 13.*

1. *There is a randomized polynomial-time algorithm which, given $(G, w, (C, I, U))$ as input, picks a vertex proportionately at random from the set $X \cup (I_X \setminus C^\star_U)$ with probability at least $\frac{1}{2}$. That is, the algorithm runs in polynomial time and outputs a vertex v, and the following hold with probability at least $\frac{1}{2}$: (i) $v \in X \cup (I_X \setminus C^\star_U)$, and (ii) for any $x \in (X \cup (I_X \setminus C^\star_U))$, $Pr[v = x] = w(x)/w(X \cup (I_X \setminus C^\star_U))$.*
2. *There is a randomized polynomial-time algorithm which, given $(G, w, (C, I, U))$ as input, picks a vertex proportionately at random from the set $X \cup (C_X \setminus I^\star_U)$ with probability at least $\frac{1}{2}$. That is, the algorithm runs in polynomial time and outputs a vertex v, and the following hold with probability at least $\frac{1}{2}$: (i) $v \in X \cup (C_X \setminus I^\star_U)$, and (ii) for any $x \in (X \cup (C_X \setminus I^\star_U))$, $Pr[v = x] = w(x)/w(X \cup (C_X \setminus I^\star_U))$.*

3.1 Polynomially Bounded Weights

Let us first consider instances (G, w) of SVD which have polynomially bounded weights. Let $n = |V(G)|$. Recall that $w(v) \geq 1$ holds for each vertex v of G. We say that the weight function w is *polynomially bounded* if, in addition, $\sum_{v \in V(G)} w(v) \leq c_1 n^{c_0}$ holds for every $v \in V(G)$ and some constants c_0, c_1. For such instances we have the following theorem.

▶ **Theorem 27.** *There exists a randomized algorithm that given a graph G, a polynomially bounded weight function w on $V(G)$ and $\varepsilon > 0$, runs in time $\mathcal{O}(n^{f(\varepsilon)})$ and outputs $S \subseteq V(G)$ such that $G - S$ is a split graph and $w(S) \leq (2+\varepsilon)w(OPT)$ with probability at least $1/2$, where OPT is a minimum weight split vertex deletion set of G. Here, $f(\varepsilon) = 6 + \log(80(1 + \frac{12}{\varepsilon})) \cdot 4c_0 \log(c_1)/\log(4/3)$, where c_0, c_1 are constants such that $w(V(G)) \leq c_1 \cdot n^{c_0}$.*

Algorithm 1 Approximation algorithm for the case of polynomially bounded weights.

```
  Input: An instance $(G, w, (C, I, U))$ of A-SVD, a tuples $(\beta_1^C, \beta_2^C, \beta_1^I, \beta_2^I)$ and $\varepsilon > 0$.
  Output: A $(2+\varepsilon)$-factor approximate solution to $(G, w, (C, I, U))$.
procedure ASVD-Approx($(G, w, (C, I, U)), \varepsilon, \beta_1^C, \beta_2^C, \beta_1^I, \beta_2^I$)
    if $U = \emptyset$ then
        Compute a 2-approximation $S$ using Lemma 12
        return $S$
    end if
    $X \leftarrow$ 5-approximate solution to $(G[U], w)$ from Theorem 6
    $I_X, C_X \leftarrow$ the independent set and the clique in the split partition of $G[U] - X$.
    Compute the sets $S_{12}$ and $S_{34}$ as described in Lemma 16.
    Compute the sets $S^\dagger$ and $S^\ddagger$ as described in Lemma 18.
    if $\beta_1^C \geq 0$ and $\beta_2^C \geq 0$ and $\beta_1^I \geq 0$ and $\beta_2^I \geq 0$ then
        for all $j \in \{1, 2, \ldots, b(\varepsilon)\}$ do          ▷ $b(\varepsilon) = \lceil 80(1 + \frac{12}{\varepsilon})\rceil$.
            Sample a vertex $v_I$ proportionally at random from the set $X \cup (I_X \setminus C_U^\star)$ using Lemma 26.
            Set $Z_C \leftarrow N(v_I) \cap U$.
            Set $C' \leftarrow C \cup Z_C$
            Set $U' \leftarrow U \setminus Z_C$
            Set $S_{j,1}^C \leftarrow$ ASVD-Approx($(G, w, (C', I, U')), \varepsilon, \beta_1^C - 1, \beta_2^C, \beta_1^I, \beta_2^I$)
            Set $S_{j,2}^C \leftarrow$ ASVD-Approx($(G, w, (C', I, U')), \varepsilon, \beta_1^C, \beta_2^C - 1, \beta_1^I, \beta_2^I$)
            Sample a vertex $v_C$ proportionally at random from the set $X \cup (C_X \setminus I_U^\star)$ using Lemma 26.
            Set $Z_I \leftarrow U \setminus N(v_C)$.
            Set $I' \leftarrow I \cup Z_I$
            Set $U' \leftarrow U \setminus Z_I$
            Set $S_{j,1}^I \leftarrow$ ASVD-Approx($(G, w, (C, I', U')), \varepsilon, \beta_1^C, \beta_2^C, \beta_1^I - 1, \beta_2^I$)
            Set $S_{j,1}^I \leftarrow$ ASVD-Approx($(G, w, (C, I', U')), \varepsilon, \beta_1^C, \beta_2^C, \beta_1^I, \beta_2^I - 1$)
        end for
    else
        for all $j \in \{1, 2, \ldots, b(\varepsilon)\}$ do
            $S_{j,1}^C, S_{j,2}^C, S_{j,1}^I, S_{j,2}^I \leftarrow V(G), V(G), V(G), V(G)$
        end for
    end if
    $S \leftarrow$ a min weight set in $\bigcup_{j=1,2,\ldots b(\varepsilon)}\{S_{j,1}^C, S_{j,2}^C, S_{j,1}^I, S_{j,2}^I\} \bigcup \{S_{12}, S_{34}, S^\dagger, S^\ddagger\}$.
    return $S$
end procedure
```

Proof. Let us fix an optimum solution OPT to (G, w). We treat the instance (G, w) of SVD as an instance $(G, w, (C = \emptyset, I = \emptyset, U = V(G)))$ of A-SVD, and apply Algorithm 1 to it, along with the given value of ε and four integers $\beta_1^C, \beta_2^C, \beta_1^I, \beta_2^I$ each set to $\lceil \log_{4/3}(w(V(G)))\rceil$. Note that, as w is polynomially bounded, we have $w(V(G)) \leq c_1 n^{c_0}$ for some constants c_0, c_1, and hence $\beta' \leq c_2 \log(n)$ for every $\beta' \in \{\beta_1^C, \beta_2^C, \beta_1^I, \beta_2^I\}$ where c_2 is a constant. We will show that the value $\beta = 1 + \beta_1^C + \beta_2^C + \beta_1^I + \beta_2^I \leq 1 + 4c_2 \log(n)$ is an upper-bound on the depth of the recursion tree of Algorithm 1, and that in each recursive call this value drops by 1. Hence the depth of recursion is bounded by β. Each recursive call is made on more constrained sub-instances of A-SVD where the underlying graph G, weight function w, and the value of ε remain fixed. When one of $\{\beta_1^C, \beta_2^C, \beta_1^I, \beta_2^I\}$ falls to -1, we argue that the current instance must be an easy instance (see Definition 20), assuming all the recursive calls leading the current call were "good" (as defined below). During its run the algorithm also computes a 5-approximate solution X to $(G[U], w)$ using Theorem 6; let (I_X, C_X) be a fixed split partition of $G[U] - X$. We have a split partition $(C^\star, I^\star)$ of $G - OPT$ and we define $I_U^\star = I^\star \cap U, C_U^\star = C^\star \cap U$. These sets, introduced in Definition 13, play an important role in Algorithm 1 and its analysis.

To argue the correctness of Algorithm 1, we require the following definition. An invocation ASVD-Approx$(G, w, (C, I, U), \varepsilon, \beta_1^C, \beta_2^C, \beta_1^I, \beta_2^I)$ is *good* if the following conditions are true:

- $\beta_1^C \geq \log_{4/3}(w(C_U^\star))$,
- $\beta_2^C \geq \log_{4/3}(w(C_U^\star \setminus \{c\}))$ for some $c \in C_U^\star$,
- $\beta_1^I \geq \log_{4/3}(w(I_U^\star))$, and
- $\beta_2^I \geq \log_{4/3}(w(I_U^\star \setminus \{i\}))$ for some $i \in I_U^\star$.

Note that the definitions of $C_U^\star$ and $I_U^\star$ depend only on $(G, w, (C, I, U))$ and on the optimum solution OPT that was fixed at the beginning. These sets are hypothetical and unknown, and we can't directly test if an invocation of Algorithm 1 is a good invocation. However, observe that in the initial call, $U = V(G)$ and we set each of $\beta_1^C, \beta_2^C, \beta_1^I, \beta_2^I$ to $\lceil \log_{4/3}(w(V(G))) \rceil$, and hence the initial invocation is good. We will argue that if the current invocation is good and the instance of A-SVD is a hard instance (see Definition 20), then each recursive call made by the algorithm is good with a constant probability (which depends on ε). Then (via an induction) we argue that a good recursive call will return a $(2+\varepsilon)$-approximate solution with probability at least $\frac{1}{2}$, and hence with constant probability we obtain a $(2+\varepsilon)$-approximate solution from a recursive call. To boost the probability of success to $\frac{1}{2}$, we need to repeat this process constantly many times, so we make constantly many recursive calls. Finally, to bound the running time, we argue that the depth of the recursion tree is bounded by $\beta = \mathcal{O}(\log n)$, and we make constantly many recursive calls in each invocation of the algorithm. So the total number of calls made to this algorithm, which is upper-bounded by the size of the recursion tree, is $n^{\mathcal{O}(1)}$. This means that in polynomial time, with probability at least 1/2, we obtain a $(2+\varepsilon)$-approximate solution to (G, w). Let us now present these arguments formally.

Let us recall the optimum solution OPT to (G, w) that was fixed at the beginning. We say that an instance $(G, w, (C, I, U))$ is a *nice instance* if the solution OPT is also an optimum solution to this A-SVD instance. This means that a split partition $(C^\star, I^\star)$ of $G - OPT$ satisfies, $C^\star \cap I = \emptyset$ and $I^\star \cap C = \emptyset$. Note that this condition is trivially satisfied at the beginning for the starting instance $(G, w, (C = \emptyset, I = \emptyset, U = V(G))$. Let us consider an invocation of Algorithm 1 on a nice instance of $(G, w, (C, I, U))$ with polynomially bounded weight function w and $\beta_1^C, \beta_2^C, \beta_1^I, \beta_2^I$ such that it is a good invocation. Let S denote the solution returned by it. We will show that S is a $(2+\varepsilon)$-approximate solution with probability at least $\frac{1}{2}$, by an induction on $|U|$. Suppose that $|U| = 0$, i.e. $U = \emptyset$. Then Lemma 12 ensures that S is a 2-approximate solution. This forms the base case of our induction on $|U|$.

Now suppose that $|U| > 0$, and we have two cases depending on whether $(G, w, (C, I, U))$ is an easy instance or not. If it is an easy instance, then either the premise of Lemma 16 or the premise of Lemma 18 holds. Hence, one of $S_{12}, S_{34}, S^\dagger, S^\ddagger$ is a $(2+\varepsilon)$-approximation to $(G, w, (C, I, U))$. Moreover, we claim that if any one of $\beta_1^C, \beta_2^C, \beta_1^I, \beta_2^I$ drops to -1, then the instance $(G, w, (C, I, U))$ is an easy instance. Consider the case when $\beta_2^C = -1$. Then $\log_{4/3}(w(C_U^\star \setminus \{c\})) = -1$ for some $c \in C_U^\star$. This means $w(C_U^\star \setminus \{c\}) < 3/4$, and since $w(v) \geq 1$ for every $v \in V(G)$, it must be the case that $C_U^\star = \{c\}$. Hence, the premise of Lemma 18 holds and we obtain a $(2+\varepsilon)$-approximate solution for $(G, w, (C, I, U))$. Similar arguments apply to the other cases, i.e. when $\beta_1^C = -1$, or $\beta_1^I = -1$ or $\beta_2^I = -1$, and we can obtain a $(2+\varepsilon)$-approximation in all these cases. Therefore, in all these cases S is a $(2+\varepsilon)$-approximation to $(G, w, (C, I, U))$.

Now, consider the case when the given instance is a hard instance, i.e. $U \neq \emptyset$ and the premises of Lemma 16 and Lemma 18 don't hold. In this case $\beta_1^C, \beta_2^C, \beta_1^I, \beta_2^I \geq 0$. Recall that X is a 5-approximate solution to SVD in the subgraph $G[U]$, and hence $w(X) \leq 5 \cdot w(OPT)$. We will make recursive calls on instances of A-SVD of the form $(G, w, (C', I', U'))$ such that $C \subseteq C', I \subseteq I'$ and $U' \subsetneq U$. Suppose that $(G, w, (C', I', U'))$ is a nice instance. Then by the induction hypothesis, as $|U'| < |U|$, we can assume that Algorithm 1 returns a $(2+\varepsilon)$-approximate solution $\widehat{S}$ to this instance with probability at least 1/2. This is an approximate solution to the current instance as well:

▷ Claim 28. $\widehat{S}$ is a $(2+\varepsilon)$-approximate solution to $(G, w, (C, I, U))$

Proof. Observe that, since $\widehat{S}$ is feasible solution to the nice instance $(G, w, (C', I', U'))$, there is a split partition $(C_{\widehat{S}}, I_{\widehat{S}})$ of $G - \widehat{S}$ such that $C' \cap I_{\widehat{S}} = \emptyset$ and $I' \cap C_{\widehat{S}} = \emptyset$. Therefore, we have $C \cap I_{\widehat{S}} = \emptyset$ and $I \cap C_{\widehat{S}} = \emptyset$, i.e. $\widehat{S}$ is a feasible solution to $(G, w, (C, I, U))$. Since $w(\widehat{S}) \leq (2+\varepsilon)w(OPT)$, the claim is true. ◁

Let us now consider the recursive calls made by the algorithm for each $j \in \{1, 2, \ldots, b(\varepsilon) = \lceil 80(1 + \frac{12}{\varepsilon})\rceil\}$, and argue that with a constant probability (depending on ε) we can obtain a $(2+\varepsilon)$-approximation to the given instance. In each recursive call, one of $\beta_1^C, \beta_2^C, \beta_1^I, \beta_2^I$ drops by exactly 1. Let us fix $j \in \{1, 2, \ldots, b(\varepsilon)\}$ and consider the two vertices v_I, v_C sampled using Lemma 26. Since $(G, w, (C, I, U))$ is a hard instance, the following hold.

- With probability at least 1/2, $v_I \in X \cup (I_X \setminus C_U^\star)$, and for any $x \in (X \cup (I_X \setminus C_U^\star))$, $Pr[v_I = x] = w(x)/w(X \cup (I_X \setminus C_U^\star))$.
- With probability at least 1/2, $v_C \in X \cup (C_X \setminus I_U^\star)$, and for any $x \in (X \cup (C_X \setminus I_U^\star))$, $Pr[v_C = x] = w(x)/w(X \cup (C_X \setminus I_U^\star))$.

By the induction hypothesis, any good invocation ASVD-Approx$(G, w, (C', I', U'), \varepsilon, \widehat{\beta_1^C}, \widehat{\beta_2^C}, \widehat{\beta_1^I}, \widehat{\beta_2^I})$ where $(G, w, (C', I', U'))$ is a nice instance and $|U'| < |U|$ holds, returns a $(2+\varepsilon)$-approximate solution to $(G, w, (C', I', U'))$ with probability at least $\frac{1}{2}$. We now have four cases, depending on which of the four statements in Lemma 25 is true for $(G, w, (C, I, U))$. In each case we will argue that with constant probability, we make a good recursive call on a nice instance and obtain a $(2+\varepsilon)$-approximate solution from it.

(i) Suppose that statement (1a) of Lemma 25 is true. That is, picking a vertex proportionally at random from $X \cup (I_X \setminus C_U^\star)$ yields a vertex from $\{v \in I_U^\star \mid w(N(v) \cap C_U^\star) \geq \frac{w(C_U^\star)}{4}\}$ with probability at least $1/20(1+\frac{12}{\varepsilon})$. Then $v_I \in \{v \in I_U^\star \mid w(N(v) \cap C_U^\star) \geq \frac{w(C_U^\star)}{4}\}$ with probability at least $1/40(1+\frac{12}{\varepsilon})$. As $v_I \in I_U^\star$, every vertex in $Z_C = N(v_I) \cap U$ must either be in OPT_U or in $C_U^\star$. Furthermore, $w(Z_C \cap C_U^\star) \geq \frac{w(C_U^\star)}{4}$. Let $U' = U \setminus Z_C$, $C' = C \cup Z_C$ and consider the invocation ASVD-Approx$(G, w, (C', I, U'), \varepsilon, \beta_1^C - 1, \beta_2^C, \beta_1^I, \beta_2^I)$. Let us argue that it is a good invocation. By definition $C_{U'}^\star = C^\star \cap U'$ satisfies $w(C_{U'}^\star) \leq \frac{3}{4} w(C_U^\star)$. Therefore, as $\beta_1^C \geq \log_{4/3}(w(C_U^\star))$, we have $\beta_1^C - 1 \geq \log_{4/3}(w(C_{U'}^\star))$. Furthermore, observe that $\beta_C^2 \geq \log_{4/3}(w(C_{U'}^\star \setminus \{c^\star\}))$, and I, β_1^I, β_2^I remain unchanged. Hence, assuming that the current invocation is good, this invocation is also good. Let us argue that $(G, w, (C', I, U'))$ is a nice instance, i.e. OPT is an optimum solution to it. Towards this, recall that $C' = C \cup Z_C$ where $Z_C = N(v_I) \cap U$ and $v_I \in I_U^\star \subseteq I^\star$. Hence, every vertex in Z_C is either in OPT or in $C^\star$, i.e. $Z_C \cap I^\star = \emptyset$. Since OPT is feasible for $(G, w, (C, I, U))$ we have that $C \cap I^\star = \emptyset$. Therefore, $C' \cap I^\star = (C \cup Z_C) \cap I^\star = \emptyset$, and hence OPT is a feasible solution for $(G, w, (C', I, U'))$. Finally, as any feasible solution for $(G, w, (C', I, U'))$ is also feasible for (G, w), OPT is an optimum solution for $(G, w, (C', I, U'))$. Now $|U'| < |U|$, and by the induction hypothesis, this invocation returns a solution $S^C_{j,1}$ to $(G, w, (C', I, U'))$ with probability at least 1/2. By Claim 28, $S^C_{1,j}$ is a $(2+\varepsilon)$-approximate solution to $(G, w, (C, I, U))$. Hence, we obtain a solution $S^C_{1,j}$ that is a $(2+\varepsilon)$-approximation to $(G, w, (C, I, U))$, and this event happens with probability at least $1/80(1+\frac{12}{\varepsilon})$. Note that β^C_1 drops by 1 in the recursive call .

(ii) Suppose that statement (1b) of Lemma 25 is true. That is, picking a vertex proportionately at random from $X \cup (I_X \setminus C^\star_U)$ yields a vertex from $\{v \in I^\star_U \mid w(N(v) \cap (C^\star_U \setminus \{c^\star\})) \geq \frac{w(C^\star_U)\setminus\{c^\star\}}{4}\}$ with probability at least $1/20(1+\frac{12}{\varepsilon})$, for some vertex $c^\star \in C^\star_U$ (as determined by Lemma 25). Then, with probability at least $1/40(1+\frac{12}{\varepsilon})$, $v_I \in \{v \in I^\star_U \mid w(N(v) \cap (C^\star_U \setminus \{c^\star\})) \geq \frac{w(C^\star_U)\setminus\{c^\star\}}{4}\}$. As $v_I \in I^\star_U$, every vertex in $Z_C = N(v_I) \cap U$ must either be in OPT or in $C^\star_U$. Let $C' = C \cup Z_C, U' = U \setminus Z_C$ and consider the invocation ASVD-Approx$(G, w, (C', I, U'), \varepsilon, \beta^C_1, \beta^C_2 - 1, \beta^I_1, \beta^I_2)$. Let us argue that it is a good invocation. Let $\widehat{C} = (C^\star_U \setminus \{c^\star\}) \setminus N(v_I)$ and $C^\star_{U'} = C^\star \cap U'$, and note that either $C^\star_{U'} = \widehat{C}$ or $C^\star_{U'} = \widehat{C} \cup \{c^\star\}$. Since $w(\widehat{C}) \leq \frac{3}{4}w(C^\star_U \setminus \{c^\star\})$ by the choice of v_I, we have $\log_{4/3}(w(\widehat{C})) \leq \log_{4/3}(w(C^\star_U \setminus \{c^\star\}) - 1 \leq \beta^C_2 - 1$. Therefore, if $C^\star_{U'} = \widehat{C}$, then for any arbitrary $c' \in C^\star_{U'}$ we have $\beta^C_2 - 1 \geq \log_{4/3}(w(C^\star_{U'} \setminus \{c'\}))$; otherwise $C^\star_{U'} = \widehat{C} \cup \{c^\star\}$, and $\beta^C_2 - 1 \geq \log_{4/3}(w(C^\star_{U'} \setminus \{c^\star\}))$. Furthermore, observe that β^C_1 is unchanged and $C^\star_{U'} \subseteq C^\star_U$, we have $\log_{4/3}(w(C^\star_{U'})) \leq \beta^C_1$. Similarly, I, β^I_1, β^I_2 are also unchanged. Hence, this invocation is good. Next, as in the previous case, we can argue that $(G, w, (C', I, U')$ is a nice instance. Then, as $|U'| < |U|$, by the induction hypothesis the invocation returns a $(2+\varepsilon)$-approximate solution $S^C_{j,2}$ to $(G, w, (C', I, U'))$ with probability at least 1/2. By Claim 28, $S^C_{j,2}$ is a $(2+\varepsilon)$-approximate solution to $(G, w, (C, I, U))$. Hence, we obtain a solution $S^C_{j,2}$ that is a $(2+\varepsilon)$-approximation to $(G, w, (C, I, U))$, and this event happens with probability at least $1/80(1+\frac{12}{\varepsilon})$. Note that β^C_2 drops by 1 in recursive call made here.

(iii) Suppose that statement (2a) of Lemma 25 is true. This case is symmetric to Case-(i), above, where the arguments are made with respect to $v_C \in X \cup (C_X \setminus I^\star_U)$. Here $v_C \in \{v \in C^\star_U \mid w(I^\star_U \setminus N(v)) \geq \frac{w(I^\star_U)}{4}\}$ with probability at least $1/40(1+\frac{12}{\varepsilon})$. We consider the instance $(G, w, (C, I', U'))$ where $Z_I = U \setminus N(v_C)$, $I' = I \cup Z_I$ and $U' = U \setminus Z_I$. We can argue that this invocation is good and the instance $(G, w, (C, I', U'))$ is nice. Then, as $|U'| \leq |U|$, by the induction hypothesis, this invocation returns a $(2+\varepsilon)$-approximate solution to $(G, w, (C, I', U'))$ with probability at least 1/2. Let $S^I_{j,1}$ denote this solution, and we argue that it is also a $(2+\varepsilon)$-approximate solution to $(G, w, (C, I, U))$. In conclusion, we obtain a solution $S^I_{j,1}$ that is a $(2+\varepsilon)$-approximation to $(G, w, (C, I, U))$, and this event happens with probability at least $1/80(1+\frac{12}{\varepsilon})$. Note that β^I_1 drops by 1 in recursive call made here.

(iv) Suppose that statement (2b) of Lemma 25 is true. This case is symmetric to Case-(ii) above. Here we have a vertex $v_C \in \{v \in C^\star_U \mid w(I^\star_U \setminus N(v)) \geq \frac{w(I^\star_U)}{4}\}$ with probability at least $1/40(1+\frac{12}{\varepsilon})$. We make a recursive call ASVD-Approx$(G, w, (C, I', U'), \varepsilon, \beta^C_1, \beta^C_2, \beta^I_1, \beta^I_2 - 1)$, where $I' = I \cup Z_I$, $U' = U \setminus Z_I$ and $Z_I = U \setminus N(v_C)$. Here, we obtain a solution $S^I_{j,2}$ that is a $(2+\varepsilon)$-approximation to $(G, w, (C, I, U))$, and this event happens with probability at least $1/80(1+\frac{12}{\varepsilon})$. Note that β^I_2 drops by 1 in recursive call made here.

Therefore, if $(G, w, (C, I, U))$ is a hard instance, then for each $j \in \{1, 2, \ldots, b(\varepsilon)\}$, one of $S^C_{j,1}, S^C_{j,2}, S^I_{j,1}, S^I_{j,2}$ is a $(2+\varepsilon)$-approximate solution to it with probability at least $1/80(1+\frac{12}{\varepsilon})$. Note that the recursive calls made for any two distinct $j, j' \in \{1, 2, \ldots, b(\varepsilon)\}$ are independent events. Therefore, by setting $b(\varepsilon) = \lceil 80(1+\frac{12}{\varepsilon})\rceil$, we obtain that with probability at least $1/2$ there exists $j \in \{1,2,\ldots,b(\varepsilon)\}$ such that one of $S^C_{j,1}, S^C_{j,2}, S^I_{j,1}, S^I_{j,2}$ is a $(2+\varepsilon)$-approximate solution to $(G,w,(C,I,U))$.

Finally, let us bound the running time of this algorithm. Towards this, we must bound the total number of calls made to Algorithm 1, when run on an instance (G,w) with polynomially bounded weights. Observe that, we start with an instance $(G,w,(C=\emptyset, I=\emptyset, U=V(G)))$ of A-SVD along with $\beta^C_1, \beta^C_2, \beta^I_1, \beta^I_2$ set to $\lceil \log_{4/3}(w(V(G))\rceil = c_2\log(n)$ for some constant c_2. Then, for each instance $(G,w,(C,I,U))$, we make $b(\varepsilon)$ recursive calls and at least one of $\beta^C_1, \beta^C_2, \beta^I_1, \beta^I_2$ drops by 1 in each of these calls. Additionally U drops to a strict subset in each of these calls. Hence in a finite number of steps, either U becomes empty, or one of $\beta^C_1, \beta^C_2, \beta^I_1, \beta^I_2$ becomes equal to -1, and we reach an easy instance. Observe that this must happen at some point before the depth of recursion exceeds $\beta = 1+4c_2\log(n)$. Hence, the number of recursive calls made for the instance (G,w) is upper bounded by $b(\varepsilon)^\beta = \mathcal{O}(n^{h(\varepsilon)})$ where $h(\varepsilon) = \log(80(1+\frac{12}{\varepsilon}))\cdot 4c_0\log(c_1)/\log(4/3)$. Recall that c_0, c_1 are constants such that $w(V(G)) \leq c_1 \cdot n^{c_0}$. Observe that in each recursive call, we spend $O(n^6)$ time (excluding the recursive calls). Hence the total running time is upper-bounded by $n^{f(\varepsilon)}$ where $f(\varepsilon) = 6+\log(80(1+\frac{12}{\varepsilon}))\cdot 4c_0\log(c_1)/\log(4/3)$. Alternatively, this bound on the running time can be obtained from the Master Theorem. ◀

3.2 General Weight Functions

In this section, we extend Theorem 27 to instances of SVD with general weight function. In particular we show that given an instance with general weights, we can construct an instance with polynomially-bounded weights such that an approximate solution to the new instance can be lifted back to the original instance.

▶ **Lemma 29** (♣). *Let (G,w) be an instance of SVD, and $\varepsilon > 0$ be a constant. Then we can construct another instance (G',w') of SVD such that G' is a subgraph of G and given any α-approximate solution to (G',w') where $\alpha \leq 5$, we can obtain an $(\alpha+\varepsilon)$-approximate solution to (G,w). Moreover, the weight function w' is polynomially bounded, and $w'(V(G')) \leq \frac{30n^2}{\varepsilon}$.*

We have the following corollary of Theorem 27 and Lemma 29.

▶ **Theorem 30.** *There exists a randomized algorithm that given a graph G, a weight function w on $V(G)$ and $\varepsilon > 0$, runs in time $\mathcal{O}(n^{g(\varepsilon)})$ and outputs $S \subseteq V(G)$ such that $G-S$ is a split graph and $w(S) \leq 2(1+\varepsilon)w(OPT)$ with probability at least $1/2$, where OPT is a minimum weight split vertex deletion set of G. Here, $g(\varepsilon) = 6+8\log(80(1+\frac{12}{\varepsilon}))\cdot\log(\frac{30}{\varepsilon})/\log(4/3)$.*

Proof. Given the instance (G,w) and ε, we apply Lemma 29 and obtain an instance (G',w'), where $w'(V(G')) \leq \frac{30n^2}{\varepsilon}$. We then apply Theorem 27 to (G',w') and ε and obtain a solution S' to it. This algorithm runs in time $|V(G')|^{g(\varepsilon)} \leq n^{g(\varepsilon}$, where $g(\varepsilon) = 6+8\log(80(1+\frac{12}{\varepsilon}))\cdot\log(\frac{30}{\varepsilon})/\log(4/3)$, and with probability at least $1/2$ S' is a $(2+\varepsilon)$-approximate solution to (G',w'). Then by Lemma 29, S' can be lifted to a $2(1+\varepsilon)$-approximate solution S to (G,w). ◀

4 Conclusion

One of the natural open question is to obtain a polynomial time 2-approximation algorithm for SVD and match the lower bound obtained under UGC. It will be interesting to find other implicit d-Hitting Set problems and find its correct "approximation complexity". Towards this we restate the conjecture of Fiorini et al. [5] about a concrete implicit 3-Hitting Set problem: there is a 2-approximation algorithm for Cluster Vertex Deletion matching the lower bound under UCG.

References

1 R Bar-Yehuda and S Even. A linear-time approximation algorithm for the weighted vertex cover problem. *Journal of Algorithms*, 2(2):198–203, 1981. doi:10.1016/0196-6774(81)90020-1.

2 Reuven Bar-Yehuda and Shimon Even. A linear-time approximation algorithm for the weighted vertex cover problem. *Journal of Algorithms*, 2(2):198–203, 1981.

3 Mao-cheng Cai, Xiaotie Deng, and Wenan Zang. An approximation algorithm for feedback vertex sets in tournaments. *SIAM J. Comput.*, 30(6):1993–2007, 2000.

4 Marek Cygan and Marcin Pilipczuk. Split vertex deletion meets vertex cover: New fixed-parameter and exact exponential-time algorithms. *Inf. Process. Lett.*, 113(5-6):179–182, 2013.

5 Samuel Fiorini, Gwenaël Joret, and Oliver Schaudt. Improved approximation algorithms for hitting 3-vertex paths. *CoRR*, abs/1808.10370, 2018. arXiv:1808.10370.

6 Stephane Foldes and Peter L. Hammer. Split graphs. *Proceedings of the 8th Southeastern Conference on Combinatorics, Graph Theory, and Computing*, pages 311–315, 1977.

7 Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within $2-\epsilon$. *Journal of Computer and System Sciences*, 74(3):335–349, 2008. Computational Complexity 2003. doi:10.1016/j.jcss.2007.06.019.

8 John M Lewis and Mihalis Yannakakis. The node-deletion problem for hereditary properties is np-complete. *Journal of Computer and System Sciences*, 20(2):219–230, 1980.

9 Daniel Lokshtanov, Pranabendu Misra, Joydeep Mukherjee, Fahad Panolan, Geevarghese Philip, and Saket Saurabh. 2-approximating feedback vertex set in tournaments. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1010–1018. SIAM, 2020. doi:10.1137/1.9781611975994.61.

10 Matthias Mnich, Virginia Vassilevska Williams, and Lászlo A Végh. A 7/3-approximation for feedback vertex sets in tournaments. In *24th Annual European Symposium on Algorithms (ESA 2016)*. Schloss Dagstuhl, 2016.

11 Jelani Nelson. A note on set cover inapproximability independent of universe size. *Electronic Colloquium on Computational Complexity (ECCC)*, 14(105), 2007. URL: http://eccc.hpi-web.de/eccc-reports/2007/TR07-105/index.html.

12 Ewald Speckenmeyer. On feedback problems in diagraphs. In *Graph-Theoretic Concepts in Computer Science, 15th International Workshop, WG '89, Castle Rolduc, The Netherlands, June 14-16, 1989, Proceedings*, volume 411 of *Lecture Notes in Computer Science*, pages 218–231. Springer, 1989. doi:10.1007/3-540-52292-1_16.

13 Jie You, Jianxin Wang, and Yixin Cao. Approximate association via dissociation. *Discrete Applied Mathematics*, 219:202–209, 2017. doi:10.1016/j.dam.2016.11.007.

Near Optimal Algorithm for the Directed Single Source Replacement Paths Problem

Shiri Chechik

Blavatnik School of Computer Science, Tel Aviv University, Israel
shiri.chechik@gmail.com

Ofer Magen

Blavatnik School of Computer Science, Tel Aviv University, Israel
ofermagen98@gmail.com

Abstract

In the Single Source Replacement Paths (SSRP) problem we are given a graph $G = (V, E)$, and a shortest paths tree $\widehat{K}$ rooted at a node s, and the goal is to output for every node $t \in V$ and for every edge e in $\widehat{K}$ the length of the shortest path from s to t avoiding e.

We present an $\tilde{O}(m\sqrt{n} + n^2)$ time randomized combinatorial algorithm for unweighted directed graphs[1]. Previously such a bound was known in the directed case only for the seemingly easier problem of replacement path where both the source and the target nodes are fixed.

Our new upper bound for this problem matches the existing conditional combinatorial lower bounds. Hence, (assuming these conditional lower bounds) our result is essentially optimal and completes the picture of the SSRP problem in the combinatorial setting.

Our algorithm naturally extends to the case of small, rational edge weights. In the full version of the paper, we strengthen the existing conditional lower bounds in this case by showing that any $O(mn^{1/2-\epsilon})$ time (combinatorial or algebraic) algorithm for some fixed $\epsilon > 0$ yields a truly sub-cubic algorithm for the weighted All Pairs Shortest Paths problem (previously such a bound was known only for the combinatorial setting).

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms

Keywords and phrases Fault tolerance, Replacement Paths, Combinatorial algorithms, Conditional lower bounds

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.81

Category Track A: Algorithms, Complexity and Games

Related Version A full version of the paper is available at `https://arxiv.org/abs/2004.13673`.

Funding This publication is part of a project that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 803118 UncertainENV).

1 Introduction

In the replacement paths (RP) problem, we are given a graph G and a shortest path P between two vertices s and t and the goal is to return for every edge e in P the length $d(s, t, G - e)$, where $G - e$ is the graph obtained by removing the edge e from G, and $d(s, t, G - e)$ is the distance between s and t in the resulted graph. In some cases the goal is to provide the shortest path itself and not only its length. The interest in replacement path problems stems from the fact that failures and changes in real world networks are inevitable, and in many cases we would like to have a solution or a data structure that can

[1] As usual, n is the number of vertices, m is the number of edges and the $\tilde{O}$ notation suppresses poly-logarithmic factors.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 81; pp. 81:1–81:17
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

adapt to these failures. The replacement paths problem is a notable example where we would like to have backup paths between two distinguished vertices in the event of edge failures. The replacement paths problem is also very well motivated as it is used as an important ingredient in other applications such as the Vickrey pricing of edges owned by selfish agents from auction theory [14, 6]. Another application of the replacement path problem is finding the k shortest simple paths between a pair of vertices. The k shortest simple paths problem can be solved by invoking the replacement paths algorithm k times and adding a very small weight to the path found in each invocation. The k shortest simple paths problem has many applications by itself [4]. The replacement paths problem has been extensively studied and the literature covers many aspects of this problem with many near optimal solutions in many of the cases (see e.g. [11, 13, 12, 15, 3, 9, 19, 16, 2]).

In this paper we consider a natural and important generalization of the replacement paths problem, referred to as the single source replacement paths (SSRP) problem, which is defined as follows. Given a graph G and a shortest paths tree $\widehat{K}$ rooted at a node s, the SSRP problem is to compute the values of $d(s, t, G - e)$ for every vertex $t \in V(G)$ and for every edge $e \in E(\widehat{K})$. Note that as the number of edges in $\widehat{K}$ is $n - 1$, there are $O(n^2)$ different distances we need to evaluate. It follows that the size of the SSRP output is $O(n^2)$.

Despite of its natural flavor, the picture of the SSRP problem is not yet complete in many of the cases. To the best of our knowledge the first paper that considered the SSRP problem is by Hershberger et al. [7] who referred to the problem as edge-replacement shortest paths trees and showed that in the path-comparison model of computation of Karger et al. [8], there is a lower bound of $\Omega(mn)$ comparisons in order to solve the SSRP problem for arbitrarily weighted directed graphs.

For the directed weighted case it was shown by Vassilevska Williams and Williams [18] that any truly sub-cubic algorithm for the simpler problem of RP in **directed**, arbitrarily weighted graph admits a truly sub-cubic algorithm for the arbitrarily weighted All Pairs Shortest Paths (APSP) problem. The conditional lower bound from [18] holds only for the directed case, and quite interestingly for the undirected arbitrarily weighted case, the classical RP problem admits a near linear time algorithm [11, 13, 12]. However, the SSRP problem in undirected graphs appears to be much harder than the RP problem. In [2] it was shown by Chechik and Cohen that any truly sub-cubic solution for the SSRP problem in **undirected** arbitrarily weighted graphs, admits a truly sub-cubic algorithm for the arbitrarily weighted APSP problem. Therefore, it seems there is no hope to solve the SSRP problem in weighted graphs, both in the directed and undirected case. Meaning that if we seek for truly sub-cubic algorithms for the SSRP problem we must either consider unweighted graphs or restrict the edge weights in some other way.

One way to restrict the weights is to consider only bounded **integer** edge weights. This restriction was considered by Grandoni and Vassilevska Williams [5], who were also the ones to name this problem the single source replacement paths problem. Grandoni and Vassilevska Williams [5] gave the first non trivial upper bound for the SSRP problem. They showed that one can bypass the cubic lower bounds by using fast matrix multiplications and by restricting the weights to be integers in a bounded range. More precisely, they showed that for graphs with positive integer edge weights in the range $[1, M]$, SSRP can be computed in $\tilde{O}(Mn^{\omega})$ time (here $\omega < 2.373$ is the matrix multiplication exponent [17, 10]). This matches the current best known bound for the simpler problem of RP for directed graph with weights $[-M, M]$, by Vassilevska Williams [16]. Quite interestingly, for integer edge weights in the range $[-M, M]$, the authors of [5] gave a higher upper bound of $\tilde{O}(M^{\frac{1}{4-\omega}} n^{2+\frac{1}{4-\omega}})$ time, which creates an interesting gap between the SSRP problem and the RP problem for negative integer weights. Grandoni and Vassilevska Williams [5] conjectured that the gap between these two problems is essential and in fact they conjectured that the SSRP problem with negative weights is as hard as the directed APSP problem.

The algorithm described in [5] uses fast matrix multiplication tricks in order to break the trivial cubic upper bound, such algorithms are known as "algebraic algorithms". Algorithms that do not use any matrix multiplication tricks are known as "combinatorial algorithms". The interest in combinatorial algorithms mainly stems from the assumption that in practice combinatorial algorithms are much more efficient since the constants and sub-polynomial factors hidden in the matrix multiplication bounds are considered to be very high.

The SSRP problem was also recently considered in the combinatorial setting by Chechik and Cohen in [2] for undirected unweighted graphs. Specifically, Chechik and Cohen in [2] gave an $\tilde{O}(m\sqrt{n}+n^2)$ time randomized algorithm for SSRP in undirected unweighted graphs. Moreover, using conditional lower bounds Chechik and Cohen also showed that under some reasonable assumptions any combinatorial algorithm for the SSRP problem in unweighted undirected graphs requires $\tilde{\Omega}(m\sqrt{n})$ time.

Since there is little hope to solve the weighted case, the only missing piece in the picture of combinatorial SSRP is the case of directed unweighted graphs.

For the directed unweighted case it was shown earlier by Vassilevska Williams and Williams [18], using a conditional combinatorial lower bound that under some reasonable assumptions any combinatorial algorithm for the directed unweighted RP (and hence SSRP) problem requires $\tilde{\Omega}(m\sqrt{n})$ time. For the seemingly easier problem of replacement paths Roditty and Zwick [15] showed a near optimal solution of $\tilde{O}(m\sqrt{n})$ time for directed unweighted graphs.

Note that in the undirected unweighted case there is an essential gap between the RP and the SSRP problems. A natural question is whether such a gap also exists in the directed unweighted case. In this paper we show that this is not the case by providing a combinatorial near optimal $\tilde{O}(m\sqrt{n}+n^2)$ time algorithm for the case of directed unweighted graphs, which up to the n^2 factor (that is unavoidable as the output itself is of size $O(n^2)$) matches the running time of the algorithm in [15] (and also matches the running time of the undirected case in [2]). We therefore (up to poly-logarithmic factors) complete the picture of combinatorial SSRP.

Our main result is as follows.

▶ **Theorem 1.1.** *There exists an $\tilde{O}(m\sqrt{n}+n^2)$ time combinatorial algorithm for the SSRP problem on unweighted directed graphs. Our randomized algorithm is Monte Carlo with a one-sided error, as we always output distances which are at least the exact distances, and with high probability (of at least $1-n^{-C}$ for any constant $C>0$) we output the exact distance.*

Note that for unweighted directed graphs where $m=\tilde{O}(n^{1.5})$ our algorithm runs in $\tilde{O}(n^2)$ time, which is the time it takes just to output the result. Namely, in this range of density our algorithm surpasses the current best algebraic SSRP algorithm [5] (which has a running time complexity of $\tilde{O}(n^\omega)$) as long as $\omega>2$.

We will note that while we focus on the case of edge failures, in the directed case there is a well known reduction showing that edge failures can be used to simulate vertex failures. The reduction is as follows, replace every vertex v with two vertices v_{in} and v_{out}, and connect them by a direct edge $(v_{\text{in}}, v_{\text{out}})$. Then, for every incoming edge (u,v) add the edge (u, v_{in}), and for every outgoing edge (v,u) add the edge (v_{out}, u). The failure of the vertex v is now simulated by the failure of the edge $(v_{\text{in}}, v_{\text{out}})$.

Our main novelty is in the introduction of a tool which we refer to as *weight functions*. This tool proved to be very useful in order to apply a divide and conquer approach and could perhaps be utilized in other related problems.

1.1 Rational Weights

While we describe an algorithm for the problem of SSRP in unweighted graphs, our algorithm (much like the directed RP algorithm [15]) can be easily generalized to solve the case of weighted graphs for **rational** edge weights in the range $[1, C]$, for every constant $C \geq 1$, in the same time complexity. This is because the only place our algorithm (and the algorithm from [15]) uses the fact that the graph is unweighted is in the claim that a path of length l contains $\Theta(l)$ vertices, which is used in order to utilize sampling techniques. As this is also true for rational weights in the range $[1, C]$, our algorithm generalizes for this case trivially.

Algebraic algorithms inherently can not perform on graphs with rational weights. This is since algebraic algorithms use a reduction from a problem known as min-plus product [2] to the problem of matrix product, and this reduction works only for **integer** weights. Since in some use-cases (like the k-simple paths problem) it is very useful to have rational weights, this shows another potential interest in combinatorial algorithms.

We note that in order to store **rational** numbers, we must make some common assumptions regarding the model of computation. More specifically, we assume that computing the summation of n edge weights can be performed in $\tilde{O}(n)$ time and that all numbers we are dealing with can be stored in one (or $O(1)$) space unit. A realistic option is working in a word-RAM model, and considering only rational edge weights which are of the form $\frac{m}{2^k}$, where the two integers m and 2^k fit in the size of $O(1)$ computer words. This way, the summation of $O(n)$ numbers also fits in $O(1)$ computer words. This way of representing rational numbers is reminiscent of the floating-point representation, that is commonly used in practical applications.

In Section 7 in the full version of this paper, we show that any algorithm (combinatorial or not) for the SSRP problem for graphs with rational edge weights from the range $[1, 2)$, that runs in $O(mn^{1/2-\epsilon})$ time for any fixed $\epsilon > 0$ implies a truly sub-cubic algorithm for APSP over graphs with arbitrary integer weights. The claim is formally stated in Theorem 7.1. Previously such a conditional lower bound was only known for combinatorial algorithms using a reduction from Boolean Matrix Multiplication (see [2]).

2 Preliminaries

We will use the following notation: $\mathbb{N} = \{0, 1, 2, ...\}, \mathbb{N}^+ = \{1, 2, 3, ...\}, \forall a \in \mathbb{N}^+ : [a] = \{1, 2, ..., a\}$. Let G be a weighted directed graph then $E(G)$ denotes the set of edges in G and $V(G)$ the set of nodes. For a vertex v we say that $v \in G$ if $v \in V(G)$ and for an edge e we say that $e \in G$ if $e \in E(G)$. Let $u, v \in V(G)$ be two vertices, we denote by $d(u, v, G)$ the distance from u to v in the graph G, and denote by $R(u, v, G)$ **some** shortest path from u to v in G. Let $P \subseteq G$ be a path from u to v, we define $|P| = |E(P)| = |V(P)| - 1$. We also denote the length of P by $d(P)$. Note that $d(R(u, v, G)) = d(u, v, G)$. For a set of edges $A \subseteq E(G)$ we denote the graph $(V(G), E(G) \setminus A)$ by $G - A$. For an edge e we shortly denote $G - \{e\}$ by $G - e$, and for a path P we shortly denote $G - E(P)$ by $G - P$.

We denote by G^R the graph obtained by reversing the directions of all edges - that is the graph obtained by replacing each edge $(v, u) \in E(G)$ with the edge (u, v) with the same weight. Given a sub-graph $H \subseteq G$ we denote by $G[H]$ the sub-graph of G induced by the nodes in $V(H)$.

[2] Also known as funny matrix multiplication or distance product, see [1, 20]

Let $P \subseteq G$ be a shortest path from a node s to a node t. Let $u, v \in V(P)$ be two nodes in P, we say that u is **before** v in P if $d(u,t,G) \geq d(v,t,G)$ and that u is **after** v in P if $d(u,t,G) \leq d(v,t,G)$, For an edge $e = (u,v) \in E(P)$ and a node $a \in V(P)$ we say that a is **before** e in the path P if a is **before** u in P and say that a is **after** e in P if a is **after** v in the path P.

The following sampling Lemma is a folklore.

▶ **Lemma 2.1** (Sampling Lemma). *Consider n balls of which R are red and $n - R$ are blue. Let $C > 0, N > 0$ be two numbers such that $R > C \cdot \ln(N)$. Let B be a random set of balls such that each ball is chosen to be in B independently at random with probability $\frac{C \cdot \ln(N)}{R}$. Then w.h.p (with probability at least $1 - \frac{2}{N^c}$) there is a red ball in B and the size of B is $\tilde{O}(n/R)$.*

The following separation Lemma was used extensively in many divide and conquer algorithms on graphs including the algebraic SSRP algorithm from [5].

▶ **Lemma 2.2** (Separator Lemma). *Given a tree K with n nodes rooted at a node s, one can find in $O(n)$ time a node t that separates the tree K into 2 edge disjoint sub-trees S, T such that $E(S) \cup E(T) = E(K)$, $V(T) \cap V(S) = \{t\}$ and $\frac{n}{3} \leq |V(T)|, |V(S)| \leq \frac{2n}{3}$*

WLOG we always assume that $s \in V(S)$, which implies that t must be the root of T. Note that it might be the case that $t = s$.

2.1 The Generalized SSRP Problem

We next describe a generalization of the directed-SSRP problem that our algorithm works with. We start by describing the notation of weight functions, a new concept we developed that allows us to compress a lot of information into one recursive call of the algorithm. In the next section we will give more intuition about the weight functions and this specific generalization.

▶ **Definition 2.3** (Weight Function). *Let G be an* ***unweighted*** *directed graph. Let $s \in V(G)$ be some special source node. A function $w : V(G) \to \mathbb{N} \cup \{\infty\}$ is a weight function (with respect to the source node s) if $w(v) \geq d(s,v,G)$ for every vertex $v \in V(G)$. We refer to this requirement as the weight requirement.*

For a source node $s \in V(G)$ and a weight function w (with respect to the source node s) we define the **weighted** directed graph G_w by taking the unweighted graph G, and assigning each edge the weight 1. We then add for every node $v \in V(G)$ the edge (s,v) and assign to it the weight $w(v)$. Note that G is a sub-graph of G_w. Also, note that by the weight requirement, for every two nodes $u, v \in V(G) : d(u,v,G) = d(u,v,G_w)$.

The generalized SSRP problem is now defined as follows. The input consists of the following:

- An unweighted directed graph H and a source vertex $s \in V(H)$
- A BFS tree K in H rooted at the source s ($E(K) \subseteq E(H)$)
- A set of weight functions W (with respect to source node s)
- A set of queries $Q \subseteq E(K) \times V(H) \times W$

The goal is to output for every $(e,x,w) \in Q$ the distance $d(s,x,H_w - e)$. Note that this problem is indeed a generalization of the classic SSRP problem. In order to solve the SSRP problem on the initial graph G and the BFS tree $\widehat{K}$, we simply define a single weight function $w : V(G) \to \mathbb{N} \cup \{\infty\}$ that is defined to be $w \equiv \infty$. We then invoke our algorithm with the graph G, the BFS tree $\widehat{K}$, the set of weight functions $W = \{w\}$, and the query set $Q = E(\widehat{K}) \times V(G) \times W$. Note that G and G_w are the same graph in the sense that for every edge $e \in E(G)$ and destination $x \in V(G)$ we have that $d(s, x, G - e) = d(s, x, G_w - e)$. Hence, invoking our algorithm for the generalized SSRP would suffice. As we will only work with the generalized SSRP problem, we here after refer to it as the SSRP problem for simplicity.

3 Overview

Our algorithm uses a divide and conquer approach. Each recursive call works on a different sub-tree (K) of the original BFS tree ($\widehat{K}$), where both the destination and the edge failure are within this sub-tree (for the case when the edge failure and the destination are not in the same sub-tree our algorithm solves this in a non recursive manner to be described later in case 1 of the algorithm overview). The vertices of the sub-tree K induce a sub-graph H of the original graph G. Denote by $n = |V(G)|, m = |E(G)|, n_H = |V(H)|, m_H = |E(H)|$.

The first step of our algorithm is to separate the input BFS tree K into two edge disjoint sub-trees S and T using a balanced tree separator (see Lemma 2.2). We denote the root of the BFS tree K by s. We assume WLOG that the root of S is s and the root of T is some node t. It might be the case that $s = t$. We define P as the path from s to t in the BFS tree K. Note that $P \subseteq S$. An illustration of this separation can be found in Figure 1.

Let K' be one of the two sub-trees of K (that is S or T). If a replacement path is fully contained in the graph induced by K' then simply using the recursive call is enough in order to compute its length. The more challenging case is when the replacement path contains vertices that are not in K'.

In [5] the authors used a somewhat similar divide and conquer approach. Consider a recursive call on a sub-tree K' and consider the case when the edge failure and destination node are both in K'. In their algorithm, the authors of [5] used sampling techniques and a truncated version of the algebraic APSP algorithm (as presented in [21]) in order to create a compressed version of the subgraph induced over K' (by adding shortcuts between vertices in K'), which (w.h.p) preserves all information needed in order to compute the true distance.

However, in the combinatorial setting, one cannot use this sort of compression process for several reasons. Firstly, after the first call to the compression step (as described in the algorithm in [5]), the resulted graph could be very dense, maybe even complete. Since the conditional lower bound of $\tilde{\Omega}(m\sqrt{n} + n^2)$ for a combinatorial SSRP ([2, 18]) depends on the number of edges, we do not want to receive such dense graphs. Secondly, as we are in the combinatorial setting, we cannot use fast matrix multiplication in the compression step, which is a critical part of the algorithm described in [5]. Lastly, after the compression step the resulted graph is weighted which leaves us with a substantially more difficult problem. In fact in the combinatorial setting there is no sub-cubic time algorithm that solves the even seemingly easier problem of weighted RP (see [18] for conditional lower bounds).

So in the combinatorial setting we must devise a new, more restricted, compression technique. We will essentially show that if we add weighted edges only from the source s to all other vertices, and restrict the weights to be such that the weight of the edge (s, v) is at least $d(s, v, H)$, then solving replacement path on such a graph still requires only $\tilde{O}(m_H\sqrt{n_H} + n_H^2)$ time. We therefore would like to add only edges between s and all other nodes. However, this quickly proves to be difficult, and it seems that if we add only weighted edges from s to the compressed graph we either "under-shoot" and do not represent all replacement paths, or we "over-shoot" and represent replacement paths that does not really exist in the graph $H - e$ (for some edge failure e) - such paths will be called untruthful paths.

We have devised a technique to fix the over-shooting. That is, we give the recursive call weights that may represent untruthful replacement paths in $H - e$, but we force the recursive call to restrict the replacement paths it searches for, so we will be able to fix them before the algorithm outputs them, while maintaining optimality. The way we do so is by a novel concept we call weight functions. The idea is that the unweighted graph H will come equipped with a set of functions W, such that every $w \in W$ is a function from $V(H)$ to $\mathbb{N} \cup \{\infty\}$ and for every vertex v it holds that $w(v) \geq d(s, v, H)$. For every weight function $w \in W$ the weighted graph H_w is defined by adding for every node $v \in V$ the edge (s, v) with weight $w(v)$. The goal of the algorithm is then outputting $d(s, v, H_w - e)$ for every triplet $x \in V, w \in W, e \in E(K)$. By restricting the algorithm to only use a single, specific weight function we achieve enough "control" to fix the untruthful paths. In order to maintain the desired running time it will be critical to keep the number of weight functions ($|W|$) at most $\tilde{O}(\sqrt{n})$ (where n is the number of nodes of the original graph G).

3.1 Algorithm Overview

In the remainder of this section we sketch the ideas of our algorithm in high level. For the sake of simplicity, the algorithm in this section runs in $\tilde{O}(n^{2.5})$ time rather than $\tilde{O}(m\sqrt{n} + n^2)$ time. At the end of this section we will briefly describe how one can use some simple techniques to reduce the running time to the near optimal of $\tilde{O}(m\sqrt{n}+n^2)$. While sketching the algorithm, we also ignore the query set Q, as it is only necessary when reducing the running time of the algorithm to $\tilde{O}(m\sqrt{n} + n^2)$. So the goal of the algorithm in this section is to estimate $d(s, x, H_w - e)$ **for every** $e \in E(K), x \in V(H)$ and $w \in W$. The complete algorithm and proof of correctness can be found in Sections 4 and 5 in the full version of this paper.

In our algorithm we distinguish between a few cases according to where the edge failure and the destination are with respect to S and T. Note that for each edge failure e and destination node x we clearly know in which case we are. In each such case we distinguish between different sub-cases according to different properties of the replacement path. Clearly we do not know the replacement path a-priori, meaning that we do not know in which sub-case we are. So when proving the correctness of our algorithm in Section 5 we show that the estimation created for every sub-case is always at least the real value of $d(s, x, H_w - e)$, that is, we do not underestimate. Then we show that for the true sub-case (the sub-case describing the true replacement path) our estimation matches the true value of $d(s, x, H_w - e)$ w.h.p. By returning the minimum estimation from all of the sub-cases we are guaranteed to return the true distance w.h.p. To distinguish between the different sub-cases we first define two useful characterization of replacement paths in H_w.

▶ **Definition 3.1** (Weighted paths). *Let $e \in E(K), x \in V(H), w \in W$, and let R be a path from s to x in the graph $H_w - e$. The path R will be called weighted if it uses some edge from $E(H_w) - E(H)$. R will be called unweighted if it is fully contained in $H - e$.*

The following crucial observation allows us to handle many cases involving weighted replacement paths

▶ **Observation 3.2.** Let $e \in P$ be an edge failure, $x \in H$ be a destination node and $w \in W$ be a weight function. If the replacement path $R(s, x, H_w - e)$ is weighted then it leaves P at s and does not intersect with P until **after** the edge failure.

To see why this observation is true, first note that all the edges in $E(H_w) - E(H)$ begin at s by definition, so $R(s, x, H_w - e)$ indeed leaves P at s. Also, $R(s, x, H_w - e)$ does not intersect with P until after the edge failure as otherwise $R(s, x, H_w - e)$ could have used the path P to get from s to the intersection point, which is a shortest path by the weight requirements. In other words, the use of the weighted edge is unnecessary. An illustration of such path can be seen in Figure 6.

For edge failures from P we also define the following useful characterization

▶ **Definition 3.3** (Jumping and Departing Paths). *Let $e \in E(P), x \in V(H), w \in W$, and let R be a path from s to x in the graph $H_w - e$. The path R will be called jumping if it uses some node u such that $u \in P$ and u is **after** the edge failure e in the path P. A path that is not jumping will be called departing.*

First case – the failure is in P and the destination is in T

This case can be solved in a non-recursive manner, using observation 3.2 and somewhat similar observations to those that were used in [5]. We distinguish between 3 different forms the path $R(s, x, H_w - e)$ can take:

Case 1.1: $R(s, x, H_w - e)$ is departing and weighted.

Using observation 3.2, we can conclude that $R(s, x, H_w - e)$ is edge-disjoint from P as it does not intersect with P before the edge failure nor after (since it is departing). This implies that the length of $R(s, x, H_w - e)$ is $d(s, x, H_w - P)$. This value can easily be computed by running Dijkstra's algorithm from s in the graph $H_w - P$ for every weight function w.

Case 1.2: $R(s, x, H_w - e)$ is departing and unweighted.

An illustration of this case can be seen in Figure 3. In this case one can use a technique similar to the one used in [5] in order to compute length of the replacement path w.h.p. That is, if e is among the last $\sqrt{n_H}$ edges of P, then we can use a brute force solution to compute $d(s, x, H - e)$. If e is of distance at least $\sqrt{n_H}$ from t, then the length of the detour of $R(s, x, H_w - e)$ is at least $\sqrt{n_H}$ as this path departs before e and gets to T. So by sampling a set of nodes B of size $\tilde{O}(\sqrt{n_H})$, we hit every such detour w.h.p. Assuming we hit the detour using the pivot node $b \in B$, we can compute $d(s, b, H - e)$ rather easily, and have that $d(s, x, H - e) = d(s, b, H - e) + d(b, x, H - P)$.

In the full algorithm we denote the estimation obtained by the pivots sampling by $\text{Depart}(s, x, e)$. We show how to compute this estimation in step 5 of the full algorithm and prove its correctness in Claims 5.1 and 5.2.

Case 1.3: $R(s, x, H_w - e)$ is jumping.

As observed by the authors of [5], taking care of jumping replacement paths in the case when $x \in T$ essentially reduces to solving the RP problem, where the source node is s and the destination node is t. This is since a jumping replacement path passes WLOG through the separator node t.

So we focus on computing the length of $R(s, t, H_w - e)$ for every $e \in P$ and $w \in W$. Using observation 3.2, if $R(s, t, H_w - e)$ is weighted then its length is $\min\limits_{u \text{ after } e \text{ in } P} \{d(s, u, H_w - P) + d(u, t, H)\}$ as it does not intersect with P until after the edge failure and from the intersection node the replacement path can go to t using the shortest path P (as this subpath does not contain the edge failure e). Computing this value naively for every $w \in W$ and $e \in P$ takes $\tilde{O}(|W| {n_H}^2)$ time.

If $R(s, t, H_w - e)$ is unweighted, the algorithm of [15] can be used to compute its length.

Second case – the failure is in T and the destination is in T

We solve this case recursively. The recursive call will be invoked over the subgraph $H[T]$. Because the root of the tree T is t and not s, we must change the source of our SSRP. This implies that replacement paths that use the path P to get from s to t will be $d(s,t,H)$ units shorter in the recursive call than they truly are. So when we compress different forms of replacement paths using weight functions, for normalization reasons we must also subtract $d(s,t,H)$ from the weight function. For simplicity, we ignore this issue in the overview, but keep in mind that we always need to subtract $d(s,t,H)$ from every weight function before the algorithm passes them to the recursion call, and add this value back when it receives the recursion's estimation.

We distinguish between two possible forms of the replacement path: weighted and unweighted. Rather interestingly we will see that this separation provides enough information about the structure of the replacement path in order to compress it, and find its length recursively.

Case 2.1: The path $R(s,x,H_w - e)$ is weighted.

We claim that in this case, the only node from S that $R(s,x,H_w - e)$ uses is s. To see this note that for every node u from S that is not s, the path from s to u in the BFS tree K does not contain the edge failure e, since $e \in T$. By the weight requirement this path is a shortest path in H_w. Hence, if a weighted replacement path uses a node u from S, it can use the path from s to u in K. In other words, the use of a weighted edge was unnecessary. So in this case the replacement path is almost completely contained within $H[T]$. Therefore, in order to take care of this case, we simply need to pass the weight function w to the recursive call. We formally prove the correctness of this case in Claim 5.20.

Case 2.2: The path $R(s,x,H_w - e)$ is unweighted.

Let u be the last node of $R(s,x,H_w - e)$ that is from S. If u is t, then we can separate $R(s,x,H_w - e)$ into two subpaths: a path from s to t - that is the shortest path P WLOG, and the shortest path from t to x in $H[T] - e$. We can use the recursive call over $H[T]$ to compute the length of the second sub-path, and when we add $d(s,t,H)$ we will get the length of $R(s,x,H_w - e)$.

The more interesting case is when $u \neq t$. Let v denote the node right after u on $R(s,x,H_w - e)$. In this case we say that v gets "helped from above" by u, as illustrated in Figure 8. Since u is the last node in $R(s,x,H_w - e)$ that belongs to S the sub-path of $R(s,x,H_w - e)$ from v to x is fully contained in $H[T] - e$. So we only need to compress the sub-path of $R(s,x,H_w - e)$ from s to v. In order to do so we define a new weight function $c_T : V(T) \to \mathbb{N} \cup \{\infty\}$ where for every vertex v , $c_T(v)$ is defined to be $\min_{u \in V(S) - \{t\} : (u,v) \in E(H)} \{d(s,u,H) + 1\}$. The sub-path of $R(s,x,H_w - e)$ from s to v is represented in the graph $H[T]_{c_T} - e$ as the weighted edge (t,v). So by passing c_T to the recursion and computing $d(t,x,H[T]_{c_T} - e)$ we will be able to obtain the length of the replacement path. We formally prove the correctness of this case in Claim 5.19.

We note that c_T is truthful, in the sense that for every edge failure $e \in T$, $c_T(v)$ is the length of some path from s to v in $H - e$. This is since the path from s to u in K is of length $d(s,u,H)$ and does not contain e (as previously claimed), and the edge (u,v) is of length 1 and is not in T because u is not in T. We formally prove that c_T is truthful as part of Claim 5.18.

Third case – the failure is in $E(S) - E(P)$ and the destination is in S

We handle this case similarly to the way we handled the second case. However we still sketch the algorithm for this case as it will introduce the notation of a "help from bellow" replacement path, which will be useful in the fourth case. We solve this case recursively. The recursive call will be invoked over the subgraph $H[S]$. In order to take care of this case we distinguish between two forms of the replacement path $R(s, x, H_w - e)$.

Case 3.1: The path $R(s, x, H_w - e)$ uses only nodes from S.
In this case simply passing the weight function w to the recursive call would suffice in order to compute the length of $R(s, x, H_w - e)$.

Case 3.2: $R(s, x, H_w - e)$ uses a node from $V(T) - \{t\}$.
Let u be the last node in $R(s, x, H_w - e)$ that belongs to $V(T) - \{t\}$. Note that the path from s to u in the BFS tree K uses only edges from P and T, meaning that it does not use the edge failure $e \in E(S) - E(P)$. Hence, WLOG we may assume that the sub-path from s to u in $R(s, x, H_w - e)$ is the path from s to u in the BFS tree K as this is a shortest path by the weight requirements. Note that this implies in particular that $R(s, x, H_w - e)$ is unweighted. An illustration of this case can be found in Figure 4. We name this kind of paths "help from bellow" replacement paths. Let v be the node right after u in $R(s, x, H_w - e)$. Since u was the last node in $R(s, x, H_w - e)$ that belongs to $V(T) - \{t\}$ the sub-path of $R(s, x, H_w - e)$ from v to x is fully contained in $H[S] - e$.

So we only need to compress the sub-path of $R(s, x, H_w - e)$ from s to v. In order to do so we define a new weight function $c_S : V(S) \to \mathbb{N} \cup \{\infty\}$ where for every vertex v , $c_S(v)$ is defined to be $\min_{u \in V(T) - \{t\} : (u,v) \in E(H)} \{d(s, u, H) + 1\}$. The sub-path of $R(s, x, H_w - e)$ from s to v is represented in the graph $H[S]_{c_S}$ by the weighted edge (s, v). So by passing the weight function c_S to the recursive call over $H[S]$, and computing $d(s, x, H[S]_{c_S} - e)$, we will be able to compute the length of $R(s, x, H_w - e)$. We formally prove the correctness of this case in Claim 5.23.

We also claim that this function is truthful for edge failures from $E(S) - E(P)$ in the sense that for every $e \in E(S) - E(P)$, $c_S(v)$ is the weight of some path from s to v in $H - e$. This is since the path from s to u in K is of length $d(s, u, H)$ and does not contain e (as previously claimed), and the edge (u, v) is of length 1 and is not in S because u is not in S. We formally prove this fact as part of Claim 5.22.

Note that the weight function c_S is untruthful for edge failures from P, as the path from s to u in K contains the entire path P. But if we consider the recursion's estimation for $d(s, x, H[S]_{c_S} - e)$ **only** for an edge failure $e \in E(S) - E(P)$, we are promised that this estimation represents the length of a true path in $H - e$. If we were to add weighted edges instead of weight functions, we would lose the ability to consider $d(s, x, H[S]_{c_S} - e)$ as an estimation for $d(s, x, H_w - e)$ only for specific edge failures.

Fourth case – the failure is in P and the destination is in S

This case is the most complicated case in our algorithm. Since we cannot allow three recursive calls (in order to obtain the desired running time) and because we see no efficient way to solve this case in a non-recursive manner, we will need to use the same recursive call over $H[S]$ as in the previous case (the third case). We will do so by adding more weight functions.

We begin by making two simple observations that take care of some easy cases, so we could focus on the more involved ones.

- If the replacement path $R(s, x, H_w - e)$ uses no nodes from T then one can simply use a recursive call over the graph $H[S]$ to compute its length.
- If $R(s, x, H_w - e)$ is departing, then since we may assume it contains nodes from T, we can use observations similar to those made in cases (1.1) and (1.2) in order to compute its length.

So we now focus on the more interesting case when $R(s, x, H_w - e)$ uses nodes from T and is jumping. Note that since $R(s, x, H_w - e)$ is jumping it must leave the path P at some node v_i before the edge failure and return to P at some node v_j after the edge failure. We will in fact still need to separate this case into 3 further sub-cases, depending on the order $R(s, x, H_w - e)$ uses nodes from T. These 3 cases present the true power of weight functions, and their ability to compress graphs in a way that is sometimes untruthful but fixable.

Case 4.1: $R(s, x, H_w - e)$ uses a node from T after it uses v_j. An illustration for this case can be found in Figure 5.

We claim that in this case the length of $R(s, x, H_w - e)$ is $d(s, x, H[S]_{c_S} - e) - d(s, t, H) + d(s, t, H_w - e)$. While formally proving the correctness of this claim is rather technical we attempt to give some intuition for this claim. Note that since $R(s, x, H_w - e)$ uses a node from T after it uses v_j, it passes WLOG through t (as v_j is after the edge failure). So we can split $R(s, x, H_w - e)$ into two sub-paths: the replacement path from s to t - which is of length $d(s, t, H_w - e)$, and the path from t to x - which we denote by $R[t, x]$.

Lets us consider the path $P \circ R[t, x]$. We claim that even though the path $R[t, x]$ contains nodes from T, the recursive call over $H[S]$ can evaluate the length of the path $P \circ R[t, x]$. This is because, roughly speaking, the path $P \circ R[t, x]$ is a sort of "help from bellow" replacement path - as described in the the third case in which $e \in E(S) - E(P)$. So like in the "help from bellow" case, the path $P \circ R[t, x]$ would be represented in $H[S]_{c_S} - e$ as a weighted replacement path. When we receive the length of this weighted replacement path we remove P and replace it with the replacement path from s to t. That is, we subtract $d(s, t, H)$ and add $d(s, t, H_w - e)$. We formally prove the correctness of this estimation in Claim 5.12.As stated in the beginning of the overview, we do not know a-priori if the replacement path indeed falls in this sub-case, so we have to make sure that we never underestimate $d(s, x, H_w - e)$. We formally prove this in Claim 5.6.In this case we see that weight functions allow us to assign weights that are untruthful for some edge failures, but give us enough control in order to fix the untruthful replacement paths.

Note that $d(s, x, H[S]_{c_S} - e)$ is used regardless of which weight function w the true replacement path uses. The fact that we use one recursive call over all weight functions, allows us to compute this term only once, which we could not do if the algorithm would have used a different recursive call for each weight function.

Case 4.2: $R(s, x, H_w - e)$ is weighted and it uses no nodes from T after v_j. An illustration for this case can be found in Figure 6.

Let (s, v) be the weighted edge in the replacement path $R(s, x, H_w - e)$. Note that (s, v) is not in P (as P contains only unweighted edges from H). Hence, by definition the replacement path leaves the path P at s, that is, $v_i = s$.

This implies that the sub-path from s to v_j is edge disjoint to P and so its length is $d(s, v_j, H_w - P)$. So for every weight function $w \in W$, we would have wished to define a new weight function $w|_P$ such that $w|_P(v_j) = d(s, v_j, H_w - P)$ for every $v_j \in P$. We will then ask the recursive call to estimate $d(s, x, H[S]_{w|_P} - e)$. This will indeed suffice in order to compute the length of the replacement path recursively, as $R(s, x, H_w - e)$ uses no nodes from T after v_j.

However, by doing so we increase the number of weight function passed to the recursive call by a factor of 2. This sort of exponential growth will prevent us from achieving the desired running time. So instead we define a new weight function $w|_S$ such that $w|_S(x) = d(s, x, H_w - P)$ if $x \in P$ and $w|_S(x) = w(x)$ if $x \notin P$. Note that for every x it holds that $w|_S(x) \leq w(x)$, since the distance $d(s, x, H_w - P)$ is at most the weight of the edge $(s, x) \in H_w$ which is $w(x)$. This implies that the $w|_S$ function preserves information from both w and $w|_P$. So instead of passing w to the recursive call, we pass $w|_S$. Later in Claim 5.5 we prove that the new $w|_S$ function is truthful in the sense that for every $e \in S, x \in S$ it holds that $d(s, x, H[S]_{w|_S} - e)$ is at least $d(s, x, H_w - e)$, meaning we do not create underestimations by using $w|_S$ instead of w. In the full version of the algorithm, we prove the correctness of this case in Claim 5.13.

So as one can see, weight functions allow us to specifically choose special nodes and decrease their weights in order to compress more information, without sacrificing the truthfulness of the weight function.

Case 4.3: $R(s, x, H_w - e)$ is unweighted, it uses no nodes from T after v_j.

This is the most involved and interesting case our algorithm handles. Note that since we assume $R(s, x, H_w - e)$ uses a node from T, and since $R(s, x, H_w - e)$ uses no nodes from T after v_j, then the sub-path of $R(s, x, H_w - e)$ from v_i to v_j must contain a node from T. An illustration for this case can be found in Figure 7. Similarly to Case 1.2, we may assume that the edge failure is not among the last $\sqrt{n_H}$ edges of P as otherwise we can use a brute force solution to compute the length of the replacement path. Since the sub-path from v_i to v_j uses a node from T, its length is at least $d(v_i, t, H)$ that is at least $\sqrt{n_H}$. So w.h.p we have sampled some pivot node $b \in B$ on this sub-path. Note that the sub-path from s to b is departing as the replacement path returns to P only at v_j. So we can easily compute $d(s, b, H - e)$ as stated before in Case 1.2.

To compress the sub-path from b to v_j we define a weight function w_b for every pivot node. We would have wished to define $w_b(v_j) = d(b, v_j, H - P)$, recursively compute $d(s, x, H[S]_{w_b} - e)$ and add $d(s, b, H - e)$ when receiving the answer from the recursion. This will indeed suffice in order to compress the length of the sub-path from v_i to v_j as it is edge disjoint to P. However this is not a valid weight function as it does not necessarily fulfill the weight requirements. So instead we define $w_b(v_j) = d(s, b, H) + d(b, v_j, H - P)$ which is a valid weight function, and we fix the output of the recursion by replacing $d(s, b, H)$ with $d(s, b, H - e)$, i.e. subtracting the former and adding the latter. As in case 4.1, we need to prove that we never underestimate $d(s, x, H - e)$. This is formally done in Claim 5.7.

As we can see, while the weight function w_b is untruthful, in the sense that $w_b(v_j)$ is not necessarily the distance of a path from s to v_j in $H - e$, we are able to fix this untruthfulness as we know what pivot $b \in B$ is used in each weight function w_b. In a sense if we could have used a different recursive call for every $b \in B$ we could have used edges from s rather than weight functions but this would be very inefficient. The weight functions allow us to compress all these recursive calls into one.

In the full version of the algorithm we denote the estimation made using the w_b functions by Pivot(s, x, e). We show how to compute this estimation in step 9 of the algorithm and prove the correctness of this case in Claim 5.14.

3.2 Running Time Analysis

First we note that the number of weight functions in each recursive call increases by $\tilde{O}(\sqrt{n_H})$ in each level of the recursion, as we add the c_S, c_T and $\{w_b\}_{b\in B}$ weight functions, and $|B| = \tilde{O}(\sqrt{n_H})$. Since the number of the weight functions in the first call to the algorithm is 1 (the function $w \equiv \infty$), and since the depth of the recursion is logarithmic we have that $|W| = \tilde{O}(\sqrt{n})$ at all times. So if we simply analyze the non-recursive parts of the algorithm, we can conclude the algorithm spends $\tilde{O}(n_H^2\sqrt{n})$ times on the recursive call over the sub-graph H. One can rather easily see that since the BFS trees in each level of the recursion are edge disjoint sub-trees of the original BFS tree $\widehat{K}$, the total number of vertices in each level of the recursion is at most $2n$. We prove this formally in Section 6. So the total time the algorithm spends on each level of the recursion is $\tilde{O}(n^{2.5})$. Since the depth of the recursion is logarithmic the running time of the algorithm is $\tilde{O}(n^{2.5})$.

3.3 Going From $\tilde{O}(n^{2.5})$ to $\tilde{O}(m\sqrt{n} + n^2)$

In this section we sketch the ideas of improving the running time from $\tilde{O}(n^{2.5})$ to $\tilde{O}(m\sqrt{n}+n^2)$.

We first note that as the number of weight functions in our algorithm is $\tilde{O}(\sqrt{n})$ in each recursive call then even outputting $d(s, x, H_w - e)$ for every triplet $w \in W, x \in V(H), e \in E(K)$ is impossible (in the desired running time) as there are $\tilde{O}(n_H^{2.5})$ such triplets. In order to overcome this issue, the algorithm does not output distances to all such triplets but rather each recursive call is given as input a set of queries Q that is a small subset of all possible triplets (that is $Q \subseteq E(K) \times V(H) \times W$) and the goal is to output the distances only for the given set of queries. Initially, Q is set to be $E(\widehat{K}) \times V(G) \times \{w\}$, where $w \equiv \infty$, and so its size is $|Q| = \tilde{O}(n^2)$. Each recursive call over a graph H will make sure to ask only $\tilde{O}({n_H}^2)$ new queries (queries which it didn't received). Since the total number of vertices in each level is at most $2n$, the number of new queries added at each level of the recursion is $\tilde{O}(n^2)$. Since the depth of the recursion is logarithmic, the total number of queries asked is $\tilde{O}(n^2)$.

Secondly, recall that in Case 1.3, where the edge failure is in $E(P)$ and the destination is t, the algorithm computes the value of $\min_{u \text{ is after } e \text{ in } P}\{d(s, u, H_w - P) + d(u, t, H)\}$ naïvely for every $e \in E(P), w \in W$. This computation costs $\tilde{O}(|W|n_H^2)$ time. However for a specific function $w \in W$, this value can be computed for all $e \in E(P)$ in $\tilde{O}(n_H)$ time using a simple dynamic programming argument which will be shown in Section 5.2 in the complete algorithm. Hence, we can reduce the running time of this part to $\tilde{O}(|W|n_H)$.

Finally, and most importantly, when handling departing unweighted paths (Case 1.2) and when using the w_b weight function (Case 4.3) the algorithm samples a set B of pivots of size $\tilde{O}(\sqrt{n_H})$. Then for every edge failure $e \in P$ and destination node $x \in V(H)$ we iterate over B and find the pivot that provides the smallest distance estimation. This implies that the algorithm spends $\tilde{O}(|B||P||V(H)|)$ time to find these pivots, which is again $\tilde{O}(n_H^{2.5})$ time. The problem is that our estimation for the distance between an edge failure and the separator node t is too loose. On the one hand when sampling B we say that this distance is at least $\sqrt{n_H}$, but on the other hand when bounding $|P|$ we say that it is at most $O(n_H)$.

In order to solve this problem we use a standard scaling trick. More specifically, we consider a logarithmic number of sub-paths $\{P_k\}$, where P_k is the sub-path of P induced by the vertices $\{v \in V(P) : 2^{k+1}\sqrt{n_H} \geq d(v, t, H) \geq 2^k\sqrt{n_H}\}$. P_0 is defined to be the sub-path of P induced by the last $2\sqrt{n_H}$ vertices of P. Note that the set of paths $\{P_k\}$ is an edge disjoint partition of P, and that $|P_k| = O(2^k\sqrt{n_H})$. An illustration for this partition can be seen in Figure 2. For every index $k \neq 0$ we then sample a random set B_k of size $\tilde{O}(\frac{\sqrt{n_H}}{2^k})$

using the sampling lemma 2.1. Now, if we consider an edge failure $e \in P_k$ for $k \neq 0$, we know that the distance from e to t is at least $2^k\sqrt{n_H}$. So when we wish to estimate the length of the departing replacement path in Case 1.2 or send the query (e, x, w_b) to the recursive call over $H[S]$ in Case 4.3, we only need consider pivot nodes b that are from B_k.

Figures

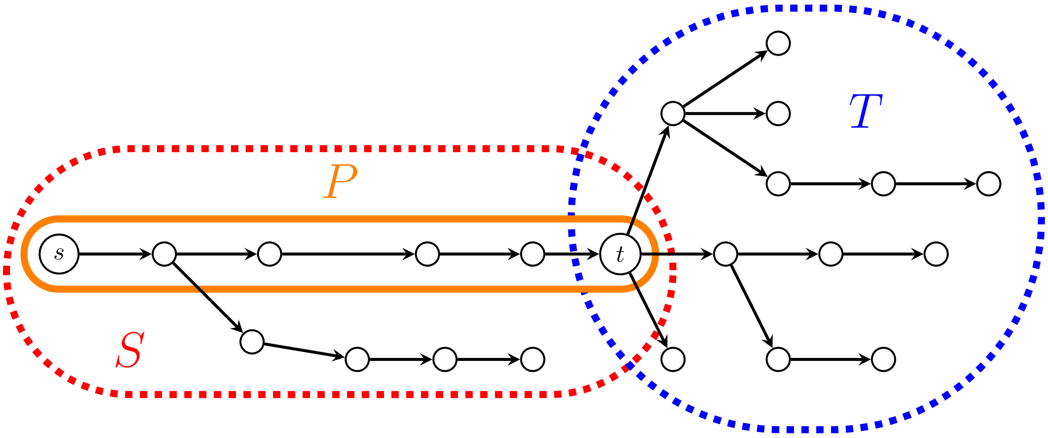


Figure 1 Tree separation.

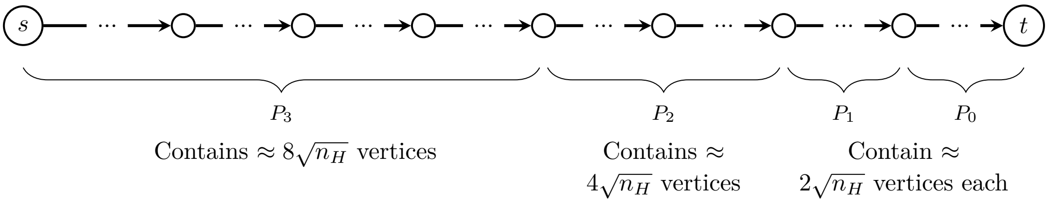

s
t
P_3
Contains ≈ $8\sqrt{n_H}$ vertices
P_2
Contains ≈ $4\sqrt{n_H}$ vertices
P_1
P_0
Contain ≈ $2\sqrt{n_H}$ vertices each

Figure 2 The P_k partition.

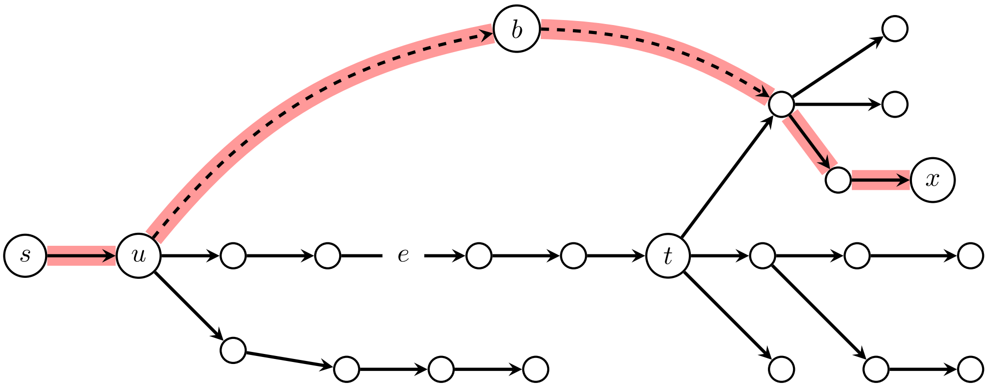


Figure 3 Departing replacement path in H.

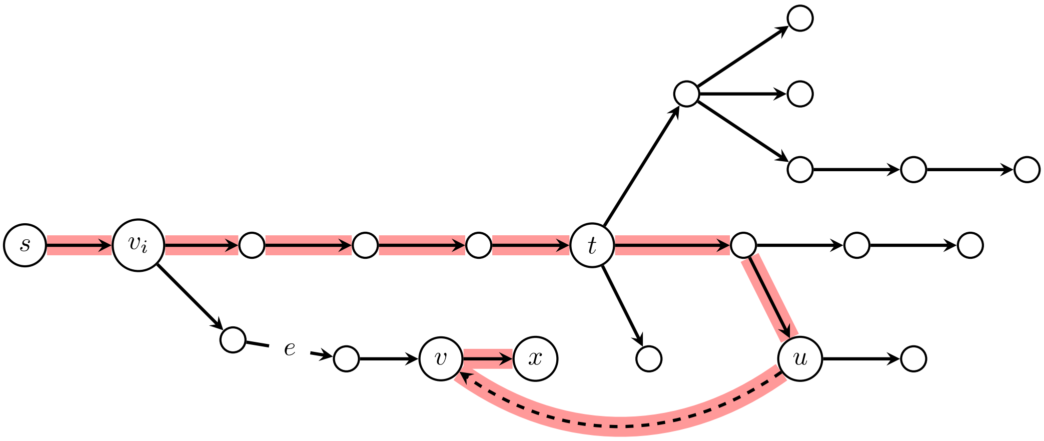


Figure 4 $R(s, x, H_w - e)$ is a “help from below” path.

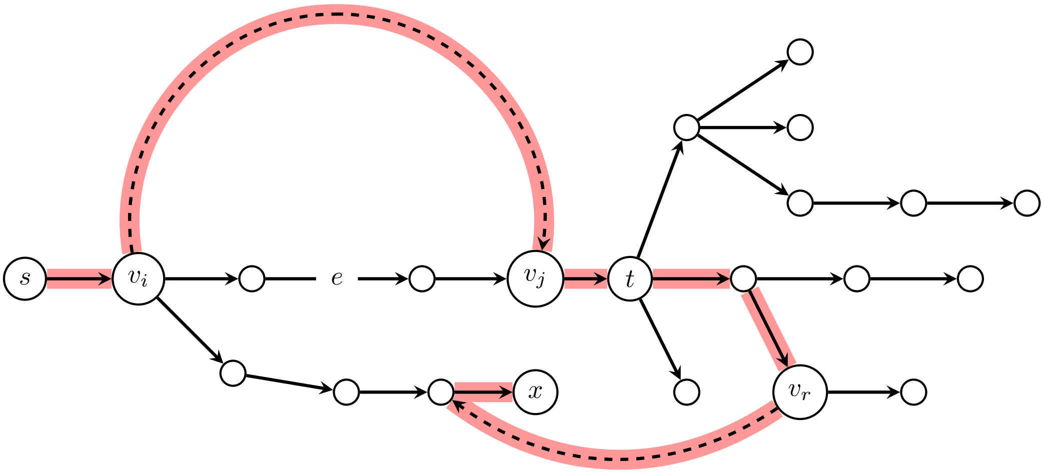


Figure 5 Case 4.1: $R(s, x, H_w - e)$ uses a node from T after it uses v_j.

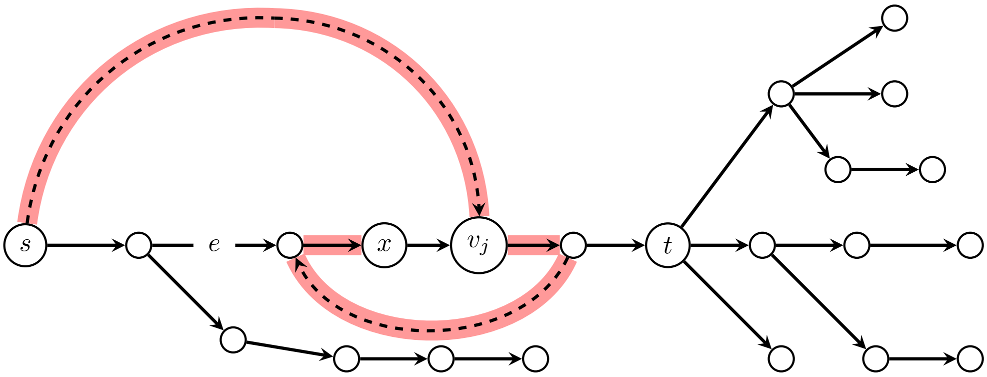


Figure 6 Case 4.2: $R(s, x, H_w - e)$ is weighted, it uses no nodes from T after v_j.

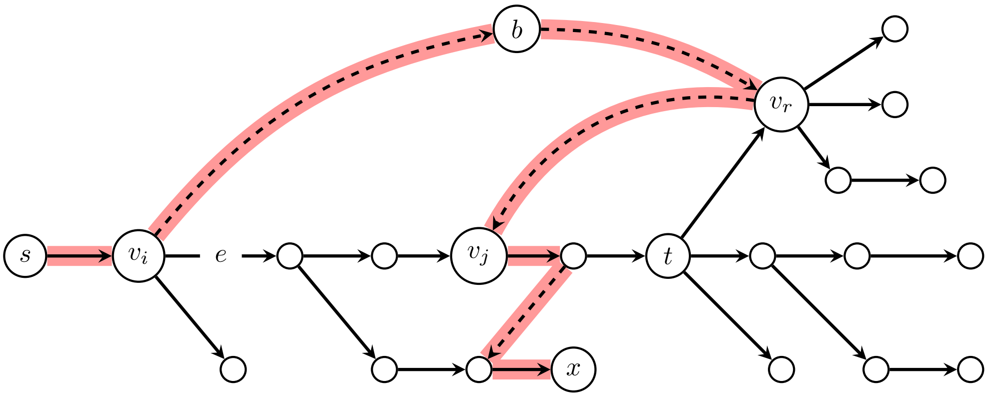


Figure 7 Case 4.3: $R(s, x, H_w - e)$ is unweighted, it uses a node from T in the sub-path from v_i to v_j, and uses no nodes from T after v_j.

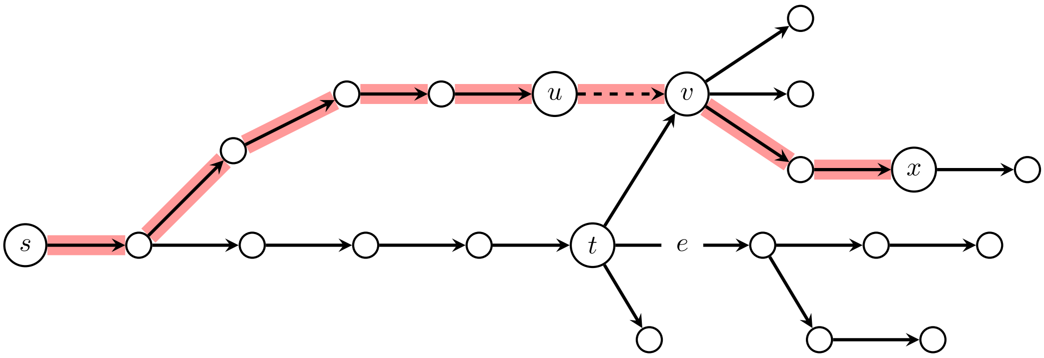


Figure 8 $R(s, x, H_w - e)$ is a "help from above" path.

References

1 Noga Alon, Zvi Galil, and Oded Margalit. On the exponent of the all pairs shortest path problem. *J. Comput. Syst. Sci.*, 54(2):255–262, April 1997. doi:10.1006/jcss.1997.1388.

2 Shiri Chechik and Sarel Cohen. Near optimal algorithms for the single source replacement paths problem. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '19, pages 2090–2109, Philadelphia, PA, USA, 2019. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=3310435.3310561.

3 Yuval Emek, David Peleg, and Liam Roditty. A near-linear time algorithm for computing replacement paths in planar directed graphs. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '08, pages 428–435, 2008. URL: http://dl.acm.org/citation.cfm?id=1347082.1347129.

4 David Eppstein. Finding the k shortest paths. *SIAM Journal on Computing*, 28(2):652–673, 1998. doi:10.1137/S0097539795290477.

5 F. Grandoni and V. V. Williams. Improved distance sensitivity oracles via fast single-source replacement paths. In *2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*, pages 748–757, October 2012. doi:10.1109/FOCS.2012.17.

6 J. Hershberger and S. Suri. Vickrey prices and shortest paths: what is an edge worth? In *Proceedings 2001 IEEE International Conference on Cluster Computing*, pages 252–259, October 2001. doi:10.1109/SFCS.2001.959899.

7 John Hershberger, Subhash Suri, and Amit Bhosle. On the difficulty of some shortest path problems. *ACM Trans. Algorithms*, 3(1):5:1–5:15, February 2007. doi:10.1145/1186810.1186815.

8 David R. Karger, Daphne Koller, and Steven J. Phillips. Finding the hidden path: Time bounds for all-pairs shortest paths. *SIAM Journal on Computing*, 22(6):1199–1217, 1993. doi:10.1137/0222071.

9 Philip N. Klein, Shay Mozes, and Oren Weimann. Shortest paths in directed planar graphs with negative lengths: A linear-space $o(n \log^2 n)$-time algorithm. *ACM Trans. Algorithms*, 6(2):30:1–30:18, April 2010. doi:10.1145/1721837.1721846.

10 François Le Gall. Powers of tensors and fast matrix multiplication. In *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*, ISSAC '14, pages 296–303, New York, NY, USA, 2014. ACM. doi:10.1145/2608628.2608664.

11 K. Malik, A. K. Mittal, and S. K. Gupta. The k most vital arcs in the shortest path problem. *Oper. Res. Lett.*, 8(4):223–227, August 1989. doi:10.1016/0167-6377(89)90065-5.

12 Enrico Nardelli, Guido Proietti, and Peter Widmayer. A faster computation of the most vital edge of a shortest path. *Inf. Process. Lett.*, 79(2):81–85, June 2001. doi:10.1016/S0020-0190(00)00175-7.

13 Enrico Nardelli, Guido Proietti, and Peter Widmayer. Finding the most vital node of a shortest path. *Theor. Comput. Sci.*, 296(1):167–177, March 2003. doi:10.1016/S0304-3975(02)00438-3.

14 Noam Nisan and Amir Ronen. Algorithmic mechanism design (extended abstract). In *Proceedings of the Thirty-first Annual ACM Symposium on Theory of Computing*, STOC '99, pages 129–140, New York, NY, USA, 1999. ACM. doi:10.1145/301250.301287.

15 Liam Roditty and Uri Zwick. Replacement paths and k simple shortest paths in unweighted directed graphs. *ACM Trans. Algorithms*, 8(4):33:1–33:11, October 2012. doi:10.1145/2344422.2344423.

16 Virginia Vassilevska Williams. Faster replacement paths. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '11, page 1337–1346, USA, 2011. Society for Industrial and Applied Mathematics.

17 Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing*, STOC '12, pages 887–898, New York, NY, USA, 2012. ACM. doi:10.1145/2213977.2214056.

18 Virginia Vassilevska Williams and R. Ryan Williams. Subcubic equivalences between path, matrix, and triangle problems. *J. ACM*, 65(5), August 2018. doi:10.1145/3186893.

19 Christian Wulff-Nilsen. Solving the replacement paths problem for planar directed graphs in $o(n \log n)$ time. In *Proceedings of the Twenty-first Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '10, pages 756–765, 2010. URL: http://dl.acm.org/citation.cfm?id=1873601.1873663.

20 Gideon Yuval. An algorithm for finding all shortest paths using $n^{2.81}$ infinite-precision multiplications. *Inf. Process. Lett.*, 4:155–156, 1976.

21 Uri Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *Journal of the ACM*, 49(3):289–317, May 2002. doi:10.1145/567112.567114.

Quantum Distributed Complexity of Set Disjointness on a Line

Frédéric Magniez
Université de Paris, IRIF, CNRS, France
magniez@irif.fr

Ashwin Nayak
Dept. of Combinatorics and Optimization, University of Waterloo, Canada
IQC, University of Waterloo, Canada
ashwin.nayak@uwaterloo.ca

Abstract

Given $x, y \in \{0,1\}^n$, Set Disjointness consists in deciding whether $x_i = y_i = 1$ for some index $i \in [n]$. We study the problem of computing this function in a distributed computing scenario in which the inputs x and y are given to the processors at the two extremities of a path of length d. Each vertex of the path has a quantum processor that can communicate with each of its neighbours by exchanging $\mathrm{O}(\log n)$ qubits per round. We are interested in the number of rounds required for computing Set Disjointness with constant probability bounded away from $1/2$. We call this problem "Set Disjointness on a Line".

Set Disjointness on a Line was introduced by Le Gall and Magniez [12] for proving lower bounds on the quantum distributed complexity of computing the diameter of an arbitrary network in the CONGEST model. However, they were only able to provide a lower bound when the local memory used by the processors on the intermediate vertices of the path is severely limited. More precisely, their bound applies only when the local memory of each intermediate processor consists of $\mathrm{O}(\log n)$ qubits.

In this work, we prove an unconditional lower bound of $\widetilde{\Omega}\big(\sqrt[3]{nd^2}+\sqrt{n}\,\big)$ rounds for Set Disjointness on a Line with $d+1$ processors. This is the first non-trivial lower bound when there is no restriction on the memory used by the processors. The result gives us a new lower bound of $\widetilde{\Omega}\big(\sqrt[3]{n\delta^2}+\sqrt{n}\,\big)$ on the number of rounds required for computing the diameter δ of any n-node network with quantum messages of size $\mathrm{O}(\log n)$ in the CONGEST model.

We draw a connection between the distributed computing scenario above and a new model of query complexity. In this model, an algorithm computing a bi-variate function f (such as Set Disjointness) has access to the inputs x and y through two separate oracles $\mathcal{O}_x$ and $\mathcal{O}_y$, respectively. The restriction is that the algorithm is required to alternately make d queries to $\mathcal{O}_x$ and d queries to $\mathcal{O}_y$, with input-independent computation in between queries. The model reflects a "switching delay" of d queries between a "round" of queries to x and the following "round" of queries to y. The technique we use for deriving the round lower bound for Set Disjointness on a Line also applies to this query model. We provide an algorithm for Set Disjointness in this query model with query complexity that matches the round lower bound stated above, up to a polylogarithmic factor. In this sense, the round lower bound we show for Set Disjointness on a Line is optimal.

2012 ACM Subject Classification Theory of computation → Distributed algorithms; Theory of computation → Quantum computation theory; Theory of computation → Quantum communication complexity; Theory of computation → Quantum query complexity

Keywords and phrases Quantum distributed computing, Set Disjointness, communication complexity, query complexity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.82

Category Track A: Algorithms, Complexity and Games

Related Version An extended version of the article is available at `http://arxiv.org/abs/2002.11795`.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 82; pp. 82:1–82:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Funding *Frédéric Magniez*: Research supported in part by the ERA-NET Cofund in Quantum Technologies project QuantAlgo and the French ANR Blanc project RDAM.
Ashwin Nayak: Research supported in part by NSERC Canada.

1 Introduction

1.1 Context

The field of Distributed Computing aims to model a collection of processors or computers communicating with each other over some network with the goal of collectively solving a global computational task. This task may depend on the structure of the network and on some additional data distributed among the computers. For instance, one may want to compute the distance between two nodes of the network, or its diameter, a proper colouring, a spanning tree, or even all-pairs shortest paths. In the context of cloud computing, data centres serve as special nodes of the network where data are stored. These centres are usually spread all over the world in order to minimise access time by clients. Since some operations need to be performed in order to synchronise the centres, the distance between these centres influence the quality of the network. For instance, one may want to decide if there is any inconsistency between two or more remote databases, or check for the availability of a common slot for booking some service.

In this work, we focus on the case of two remote data centres deployed on two nodes of a distributed network, and consider the problem of computing Set Disjointness. This fundamental problem, which we denote by D_n, consists in deciding whether two n-bit input strings x and y modelling two remote databases have the bit 1 at the same position. (This may indicate a schedule conflict, for instance.) The problem has been studied extensively in Communication Complexity [20], due to its many applications in other contexts (see, for example, the survey by Chattopadhyay and Pitassi [6]). In the most basic setting, two remote parties, Alice and Bob, hold the inputs x and y, respectively. They communicate with each other directly in order to solve the problem, while minimising the total length of the messages exchanged. Depending upon the model of computation and the type of communication channel connecting the players, the messages may be deterministic, randomised, or quantum.

The two-party communication model may be too simplistic in some scenarios, since it assumes instantaneous communication and full access to the input (by the party that is "given" the input). To address the first issue, we may include the communication delay as a multiplicative factor in the communication complexity. However, this would not account for a potentially more sophisticated use of the communication channel between the two parties. Consider the case when the channel consists of a chain of d devices, say, repeaters. One could use the channel as a network of processors in order to minimise the communication delay, for instance using cached memories. With regard to the second issue – pertaining to access to the input – the standard two-party model may not be suitable when the inputs are massive, and may only be accessed in small parts. Such access is better modelled as in Query Complexity, in which inputs are accessed by querying *oracles* (see, e.g., Refs. [5, 7, 2]).

Motivated by a concrete problem in distributed computing, we define a new model of query complexity, two-oracle query complexity with a "switching delay". In this model we consider a single computer with access to two oracles, one for each input x or y, such that switching between queries to the two inputs involves a time delay d. The delay accounts for the lag in communication between the parties holding the inputs, for instance when the inputs are not physically at the same place. It might be advantageous to balance this delay by making several accesses to the same input, say x, before switching to the other input y; we also incorporate this feature in the model. The new model attempts to address both the issues discussed above, and is described more precisely in Section 4.1.

There are several bridges between query complexity and communication complexity, but we are not aware of any previous work in a query model such as the one above. The two models – communication through a chain of d devices, and two-oracle query algorithms with a switching delay of d – share some similarities but also have fundamental differences. In the first model, one node has full access to half of the input. In the second model, all the information obtained so far from the inputs x and y is kept in the same memory registers, even when the algorithm switches between inputs.

In this work, we show that the above refinements of the two-party communication model and the query model differ significantly from their standard versions for solving Set Disjointness in the quantum setting. Such a difference does not occur in the setting of deterministic or randomised computing, and we do not know whether such a difference arises for another "natural" problem.

1.2 Application to quantum distributed computing

This study was initially motivated by a problem left open by Le Gall and Magniez [12] in the context of distributed computing with congestion (CONGEST model). They demonstrated the superiority of quantum computing for computing the diameter δ of some n-node network (Diameter problem). They designed a quantum distributed algorithm using $\widetilde{\mathrm{O}}(\sqrt{n\delta}\,)$ synchronised rounds, where simultaneous messages of $\mathrm{O}(\log n)$ qubits are exchanged at each round between neighbouring nodes in the network. They also established a lower bound of $\widetilde{\Omega}(\sqrt{n}+\delta)$ rounds.

Classically the congested distributed complexity of Diameter is well understood, and requires $\widetilde{\Theta}(n)$ rounds [9, 16, 8]. The lower bound is based on the construction of a two-party communication protocol for Set Disjointness from any distributed algorithm for Diameter. From n-bit inputs x, y, two pieces of a $\widetilde{\Theta}(n)$-node network are constructed by the two players. Then the pieces are connected by paths of length d. The diameter of the resulting network is either $d+4$ or $d+5$ depending on the solution to Set Disjointness with inputs (x, y). Finally, the classical lower bound of $\Omega(n)$ for the communication complexity of Set Disjointness implies the same lower bound on the number of rounds for any distributed algorithm for computing Diameter.

In the quantum setting, the situation is much more complex since Set Disjointness has communication complexity $\Theta(\sqrt{n}\,)$ [17, 1]. This leads to the lower bound of $\widetilde{\Omega}(\sqrt{n}+\delta)$ rounds for computing the diameter of a quantum congested network, which is significantly smaller than the upper bound stated above. Nonetheless, Le Gall and Magniez improved the lower bound for a restricted set of protocols in which each node has memory of size at most $\mathrm{poly}(\log n)$ qubits. For this, they used a more refined lower bound for Set Disjointness for bounded-round protocols.

Recall that the number of rounds in a two-party protocol is the number of messages exchanged, where the length of the messages may vary. Braverman, Garg, Ko, Mao, and Touchette [3] showed that the communication complexity of r-round two-party quantum protocols for Set Disjointness is $\widetilde{\Omega}(n/r+r)$. Using this, Le Gall and Magniez showed that any quantum distributed protocol for Diameter with congestion $\mathrm{O}(\log n)$ and memory-size $\mathrm{poly}(\log n)$ per node requires $\widetilde{\Omega}(\sqrt{n\delta}\,)$ rounds. However, without any restriction on the memory size of the nodes, no better bound than $\widetilde{\Omega}(\sqrt{n}+\delta)$ was known.

1.3 Contributions

We prove that solving Set Disjointness with the two n-bit inputs given to the processors at the extremities of a line of $d+1$ quantum processors requires $\widetilde{\Omega}(\sqrt[3]{nd^2}\,)$ rounds of communication of messages of size $\mathrm{O}(\log n)$ (**Theorem 3**). As a corollary, we get a new lower bound of

$\widetilde{\Omega}(\sqrt[3]{n\delta^2}\,)$ rounds for quantum distributed protocols computing the diameter δ of an n-node network with congestion $\mathrm{O}(\log n)$ (**Corollary 4**). This bound improves on the previous bound of $\widetilde{\Omega}(\sqrt{n}\,)$ rounds when $\delta \in \widetilde{\Omega}(\sqrt[4]{n}\,)$. The improvement is obtained by a more refined analysis of a reduction similar to one due to Le Gall and Magniez [12].

We observe that the technique used to derive the above round lower bound for Set Disjointness also applies to two-oracle query algorithms with switching delay d (**Theorem 11**). We show that this bound, and the bound of $\Omega(\sqrt{n}\,)$ coming from the standard query complexity model, are tight to within polylogarithmic factors in different ranges of the parameters n and d (**Theorem 12**). This hints at the possibility that the round lower bound for Set Disjointness may be tight. We hope that these results and, more generally, the models we study also provide a better understanding of quantum distributed computing.

2 Preliminaries

We assume that the reader is familiar with the basic notions of quantum information and computation. We recommend the texts by Nielsen and Chuang [14] and Watrous [19], and the lecture notes by de Wolf [7] for a good introduction to these topics. We briefly describe some notation, conventions, and the main concepts that we encounter in this work.

We write pure quantum states using the ket notation, for example as $|\psi\rangle$. By a quantum register, we mean a sequence of quantum bits (qubits). We assume for simplicity (and without loss of generality) that the computations do not involve any intermediate measurements, i.e., they are unitary until the measurement that is made to obtain the output in the final step of the computation.

We use the notation $\widetilde{\mathrm{O}}(\cdot)$ to indicate that we are suppressing factors that are polylogarithmic in the stated expression. For a positive integer k, we denote the set $\{1, 2, \ldots, k\}$ by $[k]$. In the sequel, we consider the computation of Boolean bi-variate functions $f\colon \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ in several models of computation.

2.1 Quantum distributed computing in the CONGEST model

We consider the quantum analogue of the standard CONGEST communication model [15]. The topology of the network is given by some graph $G = (V, E)$. Each node in the network has a distinct identifier and represents a processor. Initially, the nodes know nothing about the topology of the network except the set of edges incident on them, and the total number of nodes $|V|$. The nodes of the network do not share any entanglement at the beginning of the protocol. Communication protocols are executed with round-based synchrony. In each round, each node may perform some quantum computation on its local memory and the message registers it uses to communicate with its neighbours. Then each node transfers one message with b qubits to each adjacent node to complete that round. The parameter b is called the *congestion* or *bandwidth* of the communication channels. Unless explicitly mentioned, we assume that the congestion b is of order $\log n$, and $|V| = \mathrm{poly}(n)$. All links and nodes in the network (corresponding to the edges and vertices of G, respectively) are reliable and do not suffer any faults.

In this paper we consider the special case of a d-line network, where G consists in a single path of length d. The nodes/processors at the extremities receive inputs $x, y \in \{0,1\}^n$, respectively, and the intermediate nodes get no input. In this setting, the quantum distributed complexity of f on a d-line is the minimum number of rounds of any quantum protocol that computes f with probability at least $2/3$ and congestion $\mathrm{O}(\log n)$. We assume that $d \le n$; otherwise the complexity of any non-trivial function f of both its arguments is $\Theta(d)$.

2.2 Quantum information theory

We refer the reader to the texts by Nielsen and Chuang [14] and Watrous [19] for the basic elements of quantum information theory.

Unless specified, we take the base of the logarithm function to be 2. Whenever we consider information-theoretic quantities involving quantum registers, we assume they are in a quantum state that is implied by the context. For ease of notation, we identify the register with the quantum state.

For a register X the *von Neumann entropy* of X is defined as $\mathrm{S}(X) \coloneqq -\operatorname{Tr}(X \log X)$. If the state space of X has dimension k, then $\mathrm{S}(X) \leq \log k$. Suppose that the registers $WXYZ$ are in some joint quantum state. The *mutual information* $\mathrm{I}(X : Y)$ of X and Y is defined as $\mathrm{I}(X : Y) \coloneqq \mathrm{S}(X) + \mathrm{S}(Y) - \mathrm{S}(XY)$. The *conditional mutual information* $\mathrm{I}(X : Y \,|\, Z)$ of X and Y given Z is defined as $\mathrm{I}(X : Y \,|\, Z) \coloneqq \mathrm{I}(X : YZ) - \mathrm{I}(X : Z)$. Conditional mutual information is invariant under the application of an isometry to any of its three arguments. The quantity also satisfies the following important property.

▶ **Lemma 1** (Data Processing Inequality). *$\mathrm{I}(X : WY \,|\, Z) \;\geq\; \mathrm{I}(X : Y \,|\, Z)$.*

We may also bound conditional mutual information as follows.

▶ **Lemma 2.** *$\mathrm{I}(X : WY \,|\, Z) \;\leq\; 2\,\mathrm{S}(W) + \mathrm{I}(X : Y \,|\, Z)$.*

2.3 Quantum communication complexity

We informally describe a two-party quantum communication protocol *without shared entanglement* for computing a bi-variate Boolean function $f(x, y)$ of n-bit inputs x, y. For a formal definition, we refer the reader to an article by Touchette [18]. In such a protocol, we have two parties, Alice and Bob, each of whom gets an input in registers X and Y, respectively. In the protocols we consider, the inputs are *classical*, i.e., the joint quantum state in the input registers XY is diagonal in the basis $(|x, y\rangle \;:\; x, y \in \{0, 1\}^n)$. Alice and Bob's goal is to compute the value of the function on the pair of strings in the input registers by interacting with each other.

The protocol proceeds in some number $m \geq 1$ of *rounds*. At the cost of increasing the number of rounds by 1, we assume that Alice sends the message in the first round, after which the parties alternate in sending messages. Each party holds a *work* register in addition to the input register. Initially, Alice has work register A_0, Bob has B_0. We denote the work register with Alice at the end of round $k \in [m]$ by A_k and that with Bob by B_k.

The qubits in the work registers $A_0 B_0$ are initialised to a fixed pure state $|\bar{0}\rangle$. Suppose that Alice is supposed to send the message in the k-th round, for some $k \in [m]$. Alice applies an isometry controlled by her input register X to the work register A_{k-1} to obtain registers $A_k M_k$. She then sends the message register M_k to Bob. Bob's work register at the end of the k-th round is then $B_k \coloneqq M_k B_{k-1}$. After the m-th round (the last round), the recipient of the last message, say Bob, measures his work register B_k, possibly controlled by his input register Y, to produce the binary output of the protocol.

The length of a message is the number of qubits in the message register for that round. The *communication complexity* of the protocol is the sum of the length of the m messages in it. We say the protocol computes the function f with success probability p if for all inputs x, y, the probability that the protocol outputs $f(x, y)$ is at least p. The goal of the two parties is to compute the function while minimising the communication between themselves. The *quantum communication complexity* of f is the minimum communication complexity of a quantum protocol that computes f with success probability at least $2/3$.

We analyse the *conditional information loss* of two-party protocols, a notion introduced by Jain, Radhakrishnan, and Sen [10]. We define this notion following the convention and notation given above. In particular, we assume that Alice sends the messages in the odd rounds and Bob sends the messages in the even rounds. Suppose the input registers XY are initialised to a pair of inputs drawn from a joint distribution over $\{0,1\}^n \times \{0,1\}^n$. We identify the registers with the jointly distributed input random variables. Suppose Z is some random variable correlated with XY. The *conditional information loss* $\mathrm{IL}(\Pi \,|\, XYZ)$ of a two-party protocol as above is defined as

$$\mathrm{IL}(\Pi \,|\, XYZ) \quad := \quad \sum_{i \in [m],\ i \text{ odd}} \mathrm{I}(X : B_i Y \,|\, Z) + \sum_{i \in [m],\ i \text{ even}} \mathrm{I}(Y : A_i X \,|\, Z) \ ,$$

where the registers are implicitly assumed to be in the state given by the protocol. Since Alice sends the messages in the odd rounds, and Bob in the even rounds, this quantity measures the cumulative information about the inputs leaked to the other party, over the course of the entire protocol.

2.4 Quantum query complexity

For a thorough introduction to the quantum query model, see, for example, the lecture notes by de Wolf [7] and the survey by Ambainis [2]. In this work, we study algorithms for computing a bi-variate Boolean function $f(x,y)$ as above, using *two* unitary operators $\mathcal{O}_x$ and $\mathcal{O}_y$ that provide access to the n-bit inputs x and y, respectively. For any $z \in \{0,1\}^n$, the operator $\mathcal{O}_z$ acts as $\mathcal{O}_z|i,b\rangle = |i, b \oplus z_i\rangle$ on the Hilbert space $\mathcal{H}$ spanned by the orthonormal basis $\{|i,b\rangle : i \in [n], b \in \{0,1\}\}$. We call operators of the form $\mathcal{O}_z$ an *oracle*, and each application of such an operator a *query*.

A query algorithm $\mathcal{A}$ with access to two oracles $\mathcal{O}_x$ and $\mathcal{O}_y$ is an alternating sequence of unitary operators $U_0, \mathcal{O}_{z_1}, U_1, \mathcal{O}_{z_2}, U_2, \mathcal{O}_{z_3}, U_3, \ldots, \mathcal{O}_{z_t}, U_t$, where the operators U_i act on a Hilbert space of the form $\mathcal{H} \otimes \mathcal{W}$ and are independent of the inputs x, y, and the $z_i \in \{x, y\}$. The computation starts in a fixed state $|\bar{0}\rangle \in \mathcal{H} \otimes \mathcal{W}$, followed by the sequence of unitary operators to get the final state $U_t \mathcal{O}_{z_t} \cdots U_3 \mathcal{O}_{z_3} U_2 \mathcal{O}_{z_2} U_1 \mathcal{O}_{z_1} U_0 |\bar{0}\rangle$. Finally, we measure the first qubit in the standard basis to obtain the output $\mathcal{A}(x,y)$ of the algorithm. We say the algorithm computes f with success probability p if for all inputs x, y, we have $\mathcal{A}(x,y) = f(x,y)$ with probability at least p.

As in the standard quantum query model, we focus on the number of applications of the operators $\mathcal{O}_x$ and $\mathcal{O}_y$ in an algorithm, and ignore the cost of implementing unitary operators that are independent of x and y. The *query complexity* of an algorithm is the number of queries made by the algorithm (t in the definition above). The *quantum query complexity* of a function f is the minimum query complexity of any quantum algorithm that computes f with probability at least $2/3$.

3 Set Disjointness on a Line

3.1 The problem and results

The Set Disjointness problem $\mathsf{L}_{n,d}$ on a line was introduced recently by Le Gall and Magniez [12] in the context of distributed computing. It is a communication problem involving $d+1$ communicating parties, $\mathsf{A}_0, \mathsf{A}_1, \ldots, \mathsf{A}_d$, arranged on the vertices of a path of length d. The edges of the path denote two-way quantum communication channels between the players. Parties A_0 and A_d receive n-bit inputs $x, y \in \{0,1\}^n$, respectively. The communication protocol proceeds in rounds. In each round, parties A_{i-1} and A_i may exchange b qubits in each direction, for each $i \in [d]$, i.e., the *bandwidth* of each communication channel is b. The goal of the parties is to determine if the sets x and y intersect or not. I.e., they would like to compute the Set Disjointness function $\mathsf{D}_n(x, y) := \bigvee_{i=1}^{n}(x_i \wedge y_i)$.

We are interested in the number of rounds required to solve $\mathsf{L}_{n,d}$. We readily get a quantum protocol Π_d for this problem with $\mathrm{O}(\sqrt{nd}\,)$ rounds by following an observation due to Zalka [22] on black-box algorithms that make "parallel" queries. Let Π denote the optimal two-party quantum communication protocol for Set Disjointness due to Aaronson and Ambainis [1]. In Π_d, we partition the n-bit inputs into d parts of length n/d each. Parties A_0 and A_d then simulate Π on each of the d corresponding pairs of inputs independently. The protocol Π runs in $\sqrt{n/d}$ rounds with $\mathrm{O}(1)$ qubits of communication per instance of length n/d, per round. So the total communication to or from A_0 due to one round of the d runs of Π is $\mathrm{O}(d)$. Since $\mathrm{O}(d)$ qubits can be transmitted across the path of length d in $\mathrm{O}(d)$ rounds of the multi-party protocol, the protocol Π_d simulates the d parallel runs of Π in $\mathrm{O}(\sqrt{nd}\,)$ rounds. Since Π finds an intersection with probability at least $3/4$ whenever there is one, and does not err when there is no intersection, the protocol Π_d also has the same correctness probability.

Le Gall and Magniez observed that a lower bound of $\Omega(\sqrt{n}/b)$ for the number of rounds follows from the $\Omega(\sqrt{n}\,)$ lower bound due to Razborov [17] on the quantum communication complexity of Set Disjointness in the two-party communication model. This is because two parties, Alice and Bob, may use any r-round protocol for $\mathsf{L}_{n,d}$ to solve Set Disjointness with $2rb$ qubits of communication: Alice simulates A_0 and Bob simulates the actions of the remaining parties $\mathsf{A}_1, \mathsf{A}_2, \ldots, \mathsf{A}_d$. An $\Omega(d)$ lower bound is also immediate due to the need for communication between A_0 and A_d.

Le Gall and Magniez devised a more intricate simulation of a protocol for $\mathsf{L}_{n,d}$ by two parties, thereby obtaining a two-party protocol for Set Disjointness. Using this, they obtained a round lower bound of $\widetilde{\Omega}(\sqrt{nd}\,)$ for $\mathsf{L}_{n,d}$ when the bandwidth b of each communication channel (in each round) and the local memory of the players $\mathsf{A}_1, \mathsf{A}_2, \ldots, \mathsf{A}_{d-1}$ are both $\mathrm{O}(\log n)$ qubits. We show that a similar simulation leads to an unconditional round lower bound of $\Omega(nd^2/b)^{1/3}$ by studying the *conditional information loss* of the resulting two-party protocol, a quantity introduced and analysed by Jain, Radhakrishnan, and Sen [10].

▶ **Theorem 3.** *Any quantum communication protocol with error probability at most* $1/3$ *for the Set Disjointness problem* $\mathsf{L}_{n,d}$ *on the line requires* $\Omega(\sqrt[3]{nd^2/b}\,)$ *rounds.*

This bound dominates the straightforward bound of $\Omega(\sqrt{n}/b)$ mentioned above when $d \geq \sqrt[4]{n/b}$, i.e., when $d \geq \sqrt[4]{n/\log n}$ when $b = \log n$. However, we do not know if either bound is achievable in their respective parameter regimes. We study the optimality via a related query model in Section 4.

Using the reduction from $\mathsf{L}_{n,d}$ to the problem of computing the diameter described in the proof of Theorem 1.3 in Ref. [12], we get a new lower bound for quantum distributed protocols for the diameter problem in the CONGEST model.

▶ **Corollary 4.** *Any quantum distributed protocol for computing the diameter* δ *of* n*-node networks with congestion* $\mathrm{O}(\log n)$ *requires* $\widetilde{\Omega}(\sqrt[3]{n\delta^2}\,)$ *rounds.*

3.2 Overview of the proof

We begin by giving an overview of the proof of Theorem 3. It rests on a simulation of a protocol for $\mathsf{L}_{n,d}$ by a two-party protocol for Set Disjointness similar to one designed by Le Gall and Magniez [12, Theorem 6.1]. (In fact, the simulation works for any multi-party protocol over the path of length d that computes some bi-variate function $g(x, y)$ of the inputs given to A_0 and A_d.) The idea underlying the simulation is the following. Suppose we have a protocol Π_d for the problem $\mathsf{L}_{n,d}$. In the two-party protocol Π, Alice begins by holding the registers used by parties $\mathsf{A}_0, \mathsf{A}_1, \ldots, \mathsf{A}_{d-1}$. She then simulates all the actions – local operations and communication – of the parties $\mathsf{A}_0, \mathsf{A}_1, \ldots, \mathsf{A}_{d-1}$ from the first round in Π_d, except for the communication between A_{d-1} and A_d. This is possible because these actions do not depend on the input y held by A_d. She can continue simulating the actions of $\mathsf{A}_0, \mathsf{A}_1, \ldots, \mathsf{A}_{d-2}$ from the second round, except the communication between A_{d-2} and A_{d-1}, as these do not depend on the message from A_d from the first round in Π_d. Continuing this way, Alice can simulate the actions of $\mathsf{A}_0, \mathsf{A}_1, \ldots, \mathsf{A}_{d-i}$ from round i of Π_d, except the communication between A_{d-i} and A_{d-i+1}, for all $i \in [d]$, all in one round of Π. These actions constitute Alice's local operations in the first round of Π.

Alice then sends Bob the local memory used by parties $\mathsf{A}_1, \ldots, \mathsf{A}_{d-1}$ in Π_d, along with the qubits sent by A_{i-1} to A_i in round i, for each $i \in [d]$. (Alice retains the input x and the memory used by party A_0.) This constitutes the first message from Alice to Bob in Π.

Given the first message, Bob can simulate the remaining actions of $\mathsf{A}_1, \mathsf{A}_2, \ldots, \mathsf{A}_d$ from the first d rounds of Π_d, except for the communication from A_1 to A_0. These constitute his local operations in the second round of Π. He then sends Alice the qubits sent by A_1 to A_0 in round d of Π_d along with the local memory used by the parties A_i, for $i \in [d-1]$. (Bob retains the input y and the local memory used by party A_d.) This constitutes the second message in Π.

In effect, the simulation implements the first d rounds of Π_d in two rounds of Π (see Figure 2). The same idea allows Alice and Bob to simulate the rest of the protocol Π_d while implementing each successive block of d rounds of Π_d in two rounds of Π, with communication per round of the order of $d(b + s)$, where b is the bandwidth of the communication channels in Π_d, and s is a bound on the number of qubits of local memory used by any of the parties $\mathsf{A}_1, \ldots, \mathsf{A}_{d-1}$. Building on the detailed description of protocols on the line in Section 3.3, we describe the simulation formally in Section 3.4, and show the following.

► **Lemma 5.** *Given any r-round quantum protocol Π_d for $\mathsf{L}_{n,d}$ over communication channels with bandwidth b in which each party uses local memory at most s, there is a two-party quantum protocol Π for Set Disjointness D_n that has $2\lceil r/d \rceil$ rounds, total communication of order $r(b + s)$, and has the same probability of success.*

The communication required by a k-round bounded-error two-party protocol for Set Disjointness is $\Omega(n/(k \log^8 k))$ [3, Theorem A]. This gives us the lower bound of $\widetilde{\Omega}(\sqrt{nd}\,)$ due to Le Gall and Magniez on the number of rounds r in Π_d, when $b + s$ is of order $\log n$.

In fact, we may derive an unconditional lower bound on the number of rounds from the same reduction, one that holds without any restriction on the local memory used by the parties in Π_d. This is because the state of the registers of any party A_i, with $i \in [d-1]$, in an r-round protocol has support on a fixed subspace of dimension at most $(2b)^r$, independent of the inputs, at any moment in the protocol. This follows from an argument due to Yao [21], by considering a two party protocol obtained by grouping all parties except A_i together. So the state of party A_i at any point in the protocol can be mapped to one over $r \log(2b)$ qubits. Using this for the bound s on the local memory, bandwidth $b \in \mathrm{O}(\log n)$, and the same reasoning as before, we get a lower bound of $\widetilde{\Omega}(nd)^{1/3}$ on the number of rounds r in Π_d.

We refine the analysis further to obtain Theorem 3, by appealing to an information-theoretic argument. The key insight is that regardless of the size of the local memory maintained by the parties A_i, for $i \in [d-1]$, the new *information* they get about either input x or y in one round is bounded by b, the length of the message from A_0 or A_d, respectively. Thus, the total information contained in the memory and messages of these parties about the inputs may be bounded by rb at any point in the protocol (see Lemma 8). This carries over to the information contained in the messages between Alice and Bob in the two-party protocol Π derived from Π_d. The conditional information loss of the two-party protocol may then be bounded by rbm, where m is the number of rounds in Π (for suitable distributions over the inputs).

▶ **Lemma 6.** *Let XYZ be jointly distributed random variables such that $X, Y \in \{0,1\}^n$, and X and Y are independent given Z. The conditional information loss of the two-party protocol Π for Set Disjointness D_n mentioned in Lemma 5 is bounded as $\mathrm{IL}(\Pi \,|\, XYZ) \in \mathrm{O}(r^2 b/d)$.*

We derive this as Corollary 9 in Section 3.5.

We now appeal to the following result due to Jain *et al.* [10] on the conditional information loss of bounded-round protocols for Set Disjointness. This result is implicit in the proof of the $\Omega(n/m^2)$ lower bound on the communication required by m-round quantum protocols for Set Disjointness.

▶ **Theorem 7** (Jain, Radhakrishnan, Sen [10])**.** *There is a choice of distribution for XYZ such that $X, Y \in \{0,1\}^n$, the random variables X and Y are independent given Z, and for any bounded-error two-party quantum communication protocol Γ for Set Disjointness D_n with m rounds, the conditional information loss $\mathrm{IL}(\Gamma \,|\, XYZ)$ is at least $\Omega(n/m)$.*

Since the number of rounds m in the two-party protocol Π is at most $2\lceil r/d \rceil$, we conclude the $\Omega(nd^2/b)^{1/3}$ lower bound stated in Theorem 3.

3.3 Formal description of protocols on the line

In order to establish the lemmas stated in Section 3.2, we introduce some conventions and notation associated with multi-party protocols on the line of the sort we study for $\mathsf{L}_{n,d}$. By using unitary implementations of measurements, we assume that all the local operations in the protocol, except the final measurement to obtain the outcome of the protocol, are unitary. We also assume that the parties do not share any entangled state at the beginning of the protocol, and that the input registers X with A_0 and Y with A_d are read-only. I.e., the input registers may only be used as control registers during the protocol, and are retained by the respective parties throughout.

For ease of exposition, we use subscripts on the registers held by all the parties to implicitly specify the state of the register and the party which last modified the state of the register. At the beginning of round $t+1$, for $t \in \{0, 1, \ldots, r-1\}$, party A_0 holds registers $X A_{0,t}\, L_{1,t}$, party A_d holds registers $R_{d-1,t}\, A_{d,t}\, Y$, and for $i \in [d-1]$, party A_i holds registers $R_{i-1,t}\, A_{i,t}\, L_{i+1,t}$. The registers $L_{i,t}$ and $R_{i,t}$, for $i \in [0,d]$ and $t \in [0,r]$, all have b qubits. Except in the first round, the first subscript at the beginning of the round, say i, indicates that party A_i held the register in the previous round, and sent the register to the neighbour that holds it in the current round.

At the beginning of the first round, registers X and Y are initialized to the input to the protocol. The qubits in the remaining registers are all initialized to a fixed pure state, say $|0\rangle$, independent of the inputs.

In round $t+1$, each party A_i applies a unitary operation to the registers they hold. We view the unitary operation as an isometry $U_{i,t+1}$ that maps the registers to another sequence of registers with the same dimensions. The registers $X A_{0,t}\, L_{1,t}$ with A_0 are mapped to $X A_{0,t+1}\, R_{0,t+1}$. The registers $R_{d-1,t}\, A_{d,t}\, Y$ with A_d are mapped to $L_{d,t+1}\, A_{d,t+1}\, Y$.

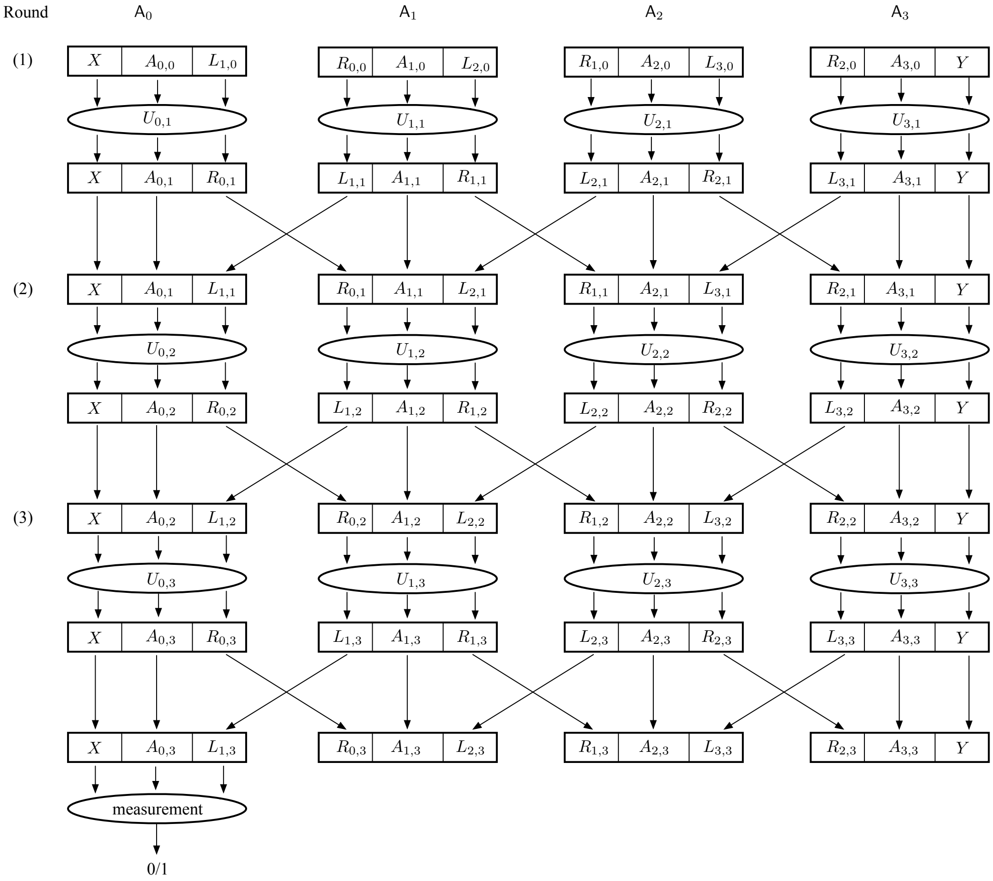


Figure 1 A multi-party communication protocol on the line with 4 parties and 3 rounds, of the type we study for $\mathsf{L}_{n,d}$. For $t \geq 1$, the subscripts i, t on a register indicate that the register was an "output" of the isometry applied by party A_i in round t, and that it is in the corresponding state. For example, the register $R_{1,3}$ was produced by the isometry applied by A_1 in the third round.

For $i \in [d-1]$, the registers $R_{i-1,t}\, A_{i,t}\, L_{i+1,t}$ with A_i are mapped to $L_{i,t+1}\, A_{i,t+1}\, R_{i,t+1}$. So for $t \geq 1$, the subscripts i, t on a register indicate that the register was an "output" of the isometry applied by party A_i in round t, and that it is in the corresponding state.

As the final action in round $t+1$, for $t < r$, if $i > 0$, party A_i sends $L_{i,t+1}$ to the party on the left (i.e., to A_{i-1}) and receives $R_{i-1,t+1}$ from her; and if $i < d$, she sends $R_{i,t+1}$ to the party on the right (i.e., to A_{i+1}) and receives register $L_{i+1,t+1}$ from her.

After the r rounds of the protocol have been completed, party A_0 makes a two-outcome measurement, possibly depending on her input, on the registers $A_{0,r}\, L_{1,r}$. The outcome is the output of the protocol. Figure 1 depicts such a protocol.

3.4 The two-party simulation

We now prove Lemma 5, by giving a formal description of the two-party protocol Π for Set Disjointness D_n derived from a protocol Π_d for $\mathsf{L}_{n,d}$. We use the notation and convention defined in Section 3.3 in our description below. For simplicity, we assume that the number of rounds r in Π_d is a multiple of d, by adding dummy rounds with suitable local operations, if necessary. Since D_n depends non-trivially on *both* inputs, the number of rounds r required to compute the function over a path of length d is at least d. So the addition of dummy rounds may at most double the number of rounds.

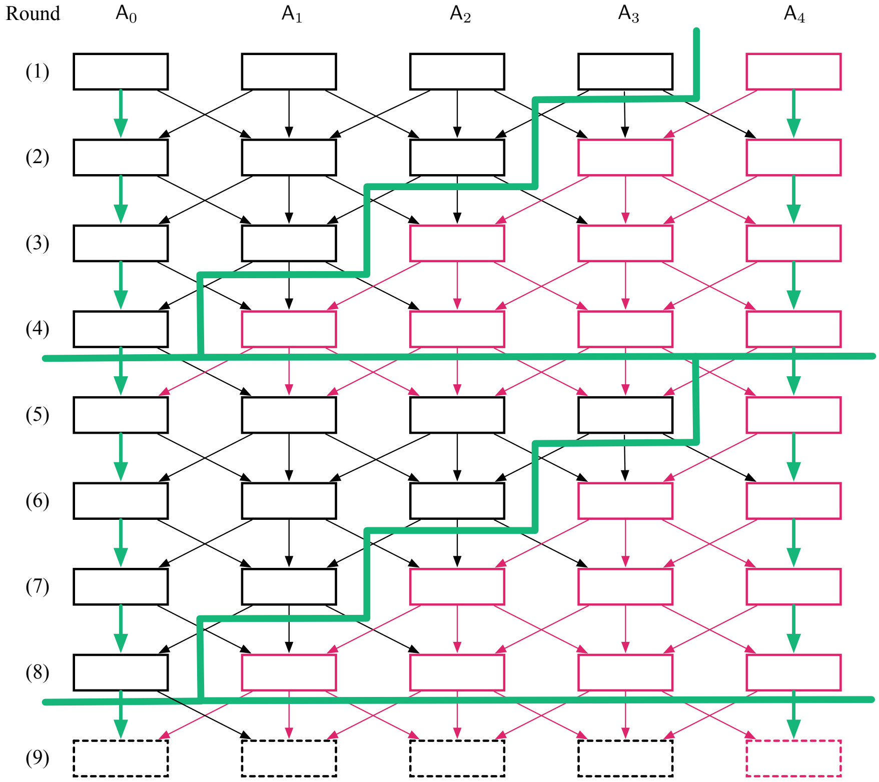


Figure 2 A depiction of the two-party simulation of a multi-party communication protocol of the type we study for $\mathsf{L}_{n,d}$. Here, we have 5 parties and show the simulation of the first 8 rounds of the original protocol. Each round in the two-party protocol is delineated by thick green lines. The black rectangular boxes represent the isometries implemented by Alice, and the black arrows going across the thick green lines represent the communication from her to Bob. The red rectangular boxes represent the isometries implemented by Bob, and the red arrows going across the thick green lines represent the communication from him to Alice. The green arrows indicate that the input register and the local memory of the parties at the extremities are retained by them throughout.

In the protocol Π, Alice initially holds all the registers with parties A_i for $i < d$ at the beginning of the first round, and Bob holds the registers with A_d. All of the registers are initialized as in Π_d. The simulation implements blocks of d successive rounds of Π_d with two rounds in Π, with Alice sending the message in the first of the two rounds and Bob in the second. See Figure 2 for a depiction of the simulation.

Assume that k blocks of d rounds each of Π_d have been implemented with $2k$ rounds in Π, for some $k \in [0, r/d - 1]$. We describe how the $(k+1)$-th block is implemented. Let $t := kd$. We maintain the invariant that at the beginning of the $(2k+1)$-th round in Π, Alice holds the registers $X A_{0,t}\, L_{1,t}$, and the registers $R_{i-1,t}\, A_{i,t}\, L_{i+1,t}$, for all $i \in [d-1]$. Alice's local operations in round $2k+1$ are as follows. For each $j \in \{t+1, t+2, t+3, \ldots, t+d\}$ in increasing order (where j denotes a round in Π_d),

1. Alice applies the isometry $U_{0,j}$ to the registers $XA_{0,j-1}\,L_{1,j-1}$ to get registers $XA_{0,j}\,R_{0,j}$.
2. For each l with $1 \leq l \leq d-(j-t)$ (denoting a party from Π_d), Alice applies the isometry $U_{l,j}$ to the registers $R_{l-1,j-1}\,A_{l,j-1}\,L_{l+1,j-1}$ to get registers $L_{l,j}\,A_{l,j}\,R_{l,j}$.
3. For each l with $1 \leq l \leq d-(j-t)$, Alice swaps registers $R_{l-1,j}$ and $L_{l,j}$.

At this point, Alice has implemented the left upper triangular "space-time slice" of the $(k+1)$-th block of d rounds of Π_d. She holds the registers

$$\begin{gathered} XA_{0,t+d}\,R_{0,t+d} \quad R_{0,t+d-1}\,A_{1,t+d-1}\,R_{1,t+d-1} \quad R_{1,t+d-2}\,A_{2,t+d-2}\,R_{2,t+d-2} \\ R_{2,t+d-3}\,A_{3,t+d-3}\,R_{3,t+d-3} \ \cdots \ R_{d-3,t+2}\,A_{d-2,t+2}\,R_{d-2,t+2} \quad R_{d-2,t+1}\,A_{d-1,t+1}\,R_{d-1,t+1} \ , \end{gathered} \tag{3.1}$$

in the state implicitly specified by the subscripts. (The registers have been grouped into threes, in the order of the parties that hold them in Π_d.) She sends all the registers *except* $XA_{0,t+d}$ to Bob. This concludes the $(2k+1)$-th round of Π.

We also maintain the invariant that at the beginning of the $(2k+2)$-th round of Π, Bob bolds all the registers in Eq. (3.1) except $XA_{0,t+d}$, in addition to the registers $R_{d-1,t}\,A_{d,t}Y$, where $t = kd$. Bob's local operations in round $2k+2$ are as follows. For each $j \in \{t+1, t+2, t+3, \ldots, t+d\}$ in increasing order (where j denotes a round in Π_d that Bob intends to complete),

1. Bob applies the isometry $U_{d,j}$ to the registers $R_{d-1,j-1}\,A_{d,j-1}Y$ to get registers $L_{d,j}\,A_{d,j}Y$.
2. For each l with $d-(j-t-1) \leq l \leq d-1$ (denoting a party from Π_d), Bob applies the isometry $U_{l,j}$ to the registers $R_{l-1,j-1}\,A_{l,j-1}\,L_{l+1,j-1}$ to get registers $L_{l,j}\,A_{l,j}\,R_{l,j}$.
3. For each l with $d-(j-t-1) \leq l \leq d$, Bob swaps registers $R_{l-1,j}$ and $L_{l,j}$.

At this point, Bob holds the registers

$$\begin{gathered} L_{1,t+d} \quad R_{0,t+d}\,A_{1,t+d}\,L_{2,t+d} \quad R_{1,t+d}\,A_{2,t+d}\,L_{3,t+d} \quad R_{2,t+d}\,A_{3,t+d}\,L_{4,t+d} \\ \cdots \ R_{d-2,t+d}\,A_{d-1,t+d}\,L_{d,t+d} \quad R_{d-1,t+d}\,A_{d,t+d}Y \ , \end{gathered} \tag{3.2}$$

in the state implicitly specified by the subscripts. The registers are thus all in the state at the end of the $(kd+d)$-th round in Π_d. Bob sends all the registers *except* $A_{d,t+d}Y$ to Alice. This concludes the $(2k+2)$-th round of Π, and the simulation of the $(k+1)$-th block of rounds of Π_d.

At the end of the simulation of the (r/d)-th block of rounds of Π_d, Alice measures the registers $A_{0,r}\,L_{1,r}$ as in Π_d to obtain the output. This completes the description of the two-party simulation. The correctness of the simulation follows by induction, by observing that Alice and Bob implement all the local operations and communication in Π_d in the correct order and with the correct registers. Lemma 5 thus follows.

3.5 Conditional information loss of the two-party protocol

We are now ready to bound the conditional information loss of the two-party protocol Π derived from the multi-party protocol Π_d. Suppose the input registers X and Y in Π_d (and therefore in Π) are initialised to a pair of n-bit strings drawn from some joint distribution. We use XY to also denote the corresponding random variables. Suppose that Z is a random variable jointly distributed with XY such that X and Y are independent given Z.

We first bound the information contained about any input in the registers held by all other players in Π_d, conditioned on Z. For ease of notation, for $t \geq 0$, we denote by D_t the entire sequence of registers held by the parties A_i, with $i \geq 1$, in the state at the end of the t-th round of Π_d. Similarly, we denote by C_t the entire sequence of registers held by the parties A_i, with $i \leq d-1$, in the state at the end of the t-th round of Π_d.

▶ **Lemma 8.** *For all $t \geq 0$, we have $\mathrm{I}(X : D_t \,|\, Z) \leq 2tb$, and $\mathrm{I}(Y : C_t \,|\, Z) \leq 2tb$.*

Proof. We prove the bound on $\mathrm{I}(X : D_t \,|\, Z)$ by induction. The second bound is obtained similarly.

The base case $t = 0$ is immediate, as the state of the registers of the parties A_i, for $i \in [d-1]$ is independent of the inputs, and the input Y is independent of X given Z.

Assume that the bound holds for $t = j$, with $j \geq 0$. Let G_{j+1} denote the sequence of registers with all the parties A_i, for $i \geq 2$, after the isometry in round $j + 1$ has been applied. Then we have

$$\mathrm{I}(X : L_{1,j+1}\, A_{1,j+1}\, R_{1,j+1} G_{j+1} \,|\, Z) \quad = \quad \mathrm{I}(X : D_j \,|\, Z) \quad \leq \quad 2jb \ ,$$

by the induction hypothesis. Let H_{j+1} denote all the registers of the parties A_i, for $i \geq 2$, after the communication in round $j + 1$. Then $L_{2,j+1}\, H_{j+1}$ and $R_{1,j+1}\, G_{j+1}$ consist of the same set of registers, but in different order. By the properties of entropy and conditional mutual information mentioned below,

$$\begin{aligned}
\mathrm{I}(X : D_{j+1} \,|\, Z) \quad &= \quad \mathrm{I}(X : R_{0,j+1}\, A_{1,j+1}\, L_{2,j+1}\, H_{j+1} \,|\, Z) \\
&= \quad \mathrm{I}(X : R_{0,j+1}\, A_{1,j+1}\, R_{1,j+1}\, G_{j+1} \,|\, Z) \\
&\leq \quad 2\,\mathrm{S}(R_{0,j+1}) + \mathrm{I}(X : A_{1,j+1}\, R_{1,j+1}\, G_{j+1} \,|\, Z) \\
&\leq \quad 2b + \mathrm{I}(X : L_{1,j+1}\, A_{1,j+1}\, R_{1,j+1} G_{j+1} \,|\, Z) \\
&\leq \quad 2b + 2jb \ .
\end{aligned}$$

The first inequality follows from Lemma 2, the second by the property that $\mathrm{S}(B)$ is bounded from above by the number of qubits in the register B and the data processing inequality (Lemma 1), and the final one by the induction hypothesis. ◀

For $k \in [2r/d]$, denote the message registers in the k-th round of the two-party protocol Π in the corresponding state together by M_k. Denote the registers with Alice at the end of the k-th round, in the corresponding state, by E_k, and the registers with Bob at the end of the k-th round, in the corresponding state, by F_k. Next, we observe from the definition of the protocol Π, that for odd numbered rounds $2k-1$, the state given by register D_{kd} is obtained by an isometry on the registers $M_{2k-1}F_{2k-2}$. The registers $M_{2k-1}F_{2k-2}$ (in the state implicitly specified by their definition) are precisely the registers Bob holds at the end of round $2k - 1$ of Π. Moreover, for even numbered rounds $2k$, the state given by the registers $E_{2k-1}M_{2k}$ is precisely the state given by the register C_{kd} in Π_d. The registers $E_{2k-1}M_{2k}$ are precisely the registers Alice holds at the end of round $2k$ of Π. Therefore, by Lemma 8 and the definition of conditional information loss, we have:

▶ **Corollary 9.** *For all $k \in [r/d]$, we have*

$$\begin{aligned}
\mathrm{I}(X : M_{2k-1}F_{2k-2} \,|\, Z) \quad &= \quad \mathrm{I}(X : D_{kd} \,|\, Z) \quad \leq \quad 2kdb \ , \qquad \textit{and} \\
\mathrm{I}(Y : E_{2k-1}M_{2k} \,|\, Z) \quad &= \quad \mathrm{I}(Y : C_{kd} \,|\, Z) \quad \leq \quad 2kdb \ .
\end{aligned}$$

Consequently, the conditional information loss of Π is bounded as $\mathrm{IL}(\Pi \,|\, XYZ) \leq 4r^2b/d$.

4 Two-oracle query algorithms with a switching delay

In this section, we define a new model of query complexity, two-oracle query complexity with a "switching delay", motivated by the study of Set Disjointness on a Line $\mathsf{L}_{n,d}$. The lower bound technique we use to establish Theorem 3 extends to the analogue of Set Disjointness D_n

in this model, with a switching delay of d queries. As a consequence, it yields the same lower bound on *query* complexity. Furthermore, we design a quantum algorithm that matches this bound up to a polylogarithmic factor, thus showing that the lower bound is, in a sense, optimal. We may also interpret this result as showing a limitation of the lower bound technique.

4.1 The new query model

Turning to the definition of the query model, we consider query algorithms for computing bi-variate functions $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$. We define the quantum version of the model; the classical versions may be defined analogously. The inputs x, y to the algorithm are provided indirectly, through oracles $\mathcal{O}_x$ and $\mathcal{O}_y$, as defined in Section 2.4. The query algorithm is defined in the standard manner, as an alternating sequence of unitary operators independent of the inputs x, y, and queries $\mathcal{O}_x$ or $\mathcal{O}_y$, applied to a fixed initial state (that is also independent of the inputs). Thus, the sequence of queries to the inputs is pre-determined. However, we define the complexity of the algorithm differently. In addition to the queries, we charge the algorithm for switching between a query to x and a query to y. We include a cost of d in the complexity whenever the algorithm switches between a query to x and a query to y. This cost parallels the cost of accessing the inputs in the distributed computing scenario in which the inputs are physically separated by distance d.

We may simplify the above model as follows, at the expense of increasing the complexity by a factor of at most 2. In the simplified model, we require that the queries be made in *rounds*. In each round, the algorithm makes d queries, but exclusively to one of the inputs x or y. Further, the algorithm alternates between the two oracles $\mathcal{O}_x$ and $\mathcal{O}_y$ in successive rounds. The complexity of the algorithm is now defined in the standard manner, as the total number of queries in the algorithm. Thus the complexity equals d times the number of rounds.

It is straightforward to verify that any algorithm with complexity q in the first model has complexity at most $2q$ in the second model, for computing any function f that depends on both inputs (i.e., when the algorithm in the first model queries both oracles). Furthermore, any algorithm with complexity q in the second model has complexity at most $2q$ in the first model. The two models are thus equivalent up to a factor of two in complexity.

The second model is also relevant in a "semi-parallel" scenario, where a sequence of d queries are made to x independently of the answers to d other queries made to y during the same time steps. Up to a factor of 2 in complexity, this semi-parallel model can be simulated by the second model above. We thus adopt the second model in the definition below.

▶ **Definition 10.** *A* two-oracle delay-d quantum query algorithm *is a query algorithm $\mathcal{A}$ with (predetermined) access to two oracles $\mathcal{O}_1, \mathcal{O}_2$, which may be decomposed into some number of contiguous sequences of unitary operators called rounds such that each round contains d queries to the same oracle, and the algorithm alternates between the two oracles in successive rounds. The* round complexity *of $\mathcal{A}$ is the number r of rounds in a decomposition of $\mathcal{A}$ as above. The* delay-d query complexity *of $\mathcal{A}$ is $d \times r$.*

We define the *quantum two-oracle delay-d round complexity* of a bi-variate function f as the minimum round-complexity of any two-oracle delay-d quantum query algorithm computing f with probability of error at most $1/3$, given oracles $\mathcal{O}_x, \mathcal{O}_y$ for the inputs x, y. We define the *quantum two-oracle delay-d query complexity* of f similarly. We may assume that $d \leq n$, as otherwise, an algorithm can learn x and y in two rounds.

Adapting the tools developed in Section 3 we get the following lower bound.

▶ **Theorem 11.** *Let $d \leq n$. The quantum two-oracle delay-d round complexity of Set Disjointness* D_n *is* $\Omega(\sqrt{n}/d)$ *and* $\Omega(\sqrt[3]{n/(d \log n)}\,)$. *The quantum two-oracle delay-d query complexity is* $\Omega(\sqrt{n}\,)$ *and* $\Omega(\sqrt[3]{nd^2/\log n}\,)$.

Note that the first expression for either bound dominates when $d^4 \in \mathrm{O}(n \log^2 n)$.

We briefly sketch the proof of Theorem 11. The query lower bound follows from the one on rounds. The $\Omega(\sqrt{n}/d)$ lower bound on rounds follows by observing that Set Disjointness D_n simplifies to the unordered search problem (OR function on n bits) in the standard quantum query model when we set y to be the all 1s string. For the second lower bound, we view a query to an oracle $\mathcal{O}_x$ or $\mathcal{O}_y$ as the exchange of of $2(\log n + 1)$ qubits between the algorithm and the oracle. So we can use any r-round algorithm for computing f in the two-oracle delay-d query model to derive a two-party communication protocol for computing f also with r rounds. The two parties run the query algorithm, sending all its registers to the corresponding player, whenever the algorithm switches between queries to x and queries to y. In each round, the state of the algorithm (therefore the corresponding message) accumulates at most $2d(\log n + 1)$ qubits of *additional* information about either input. This is a consequence of the same kind of reasoning as in Lemma 8. Thus the conditional information loss of the resulting two-party protocol may be bounded by $2r^2 d(\log n + 1)$. By Theorem 7, this is $\Omega(n/r)$, so we get the $\Omega(\sqrt[3]{n/(d \log n)}\,)$ lower bound for the number of rounds stated in Theorem 11.

4.2 Algorithm for Set Disjointness

Finally, we present an algorithm in the two-oracle model that matches the lower bounds stated in Theorem 11, up to polylogarithmic factors.

▶ **Theorem 12.** *Let $d \leq n$. The quantum two-oracle delay-d round and query complexity of Set Disjointness* D_n *are*

- $\mathrm{O}(\sqrt{n \log n}/d)$ *and* $\mathrm{O}(\sqrt{n \log n}\,)$, *respectively, when* $d^4 \leq n \log^3 n$; *and*
- $\mathrm{O}(\sqrt[3]{n/d}\,)$ *and* $\mathrm{O}(\sqrt[3]{nd^2}\,)$, *respectively, when* $d^4 \geq n \log^3 n$.

Proof. We present a quantum two-oracle delay-d query algorithm with a parameter $t \in [n]$, which gives the round and query bounds for suitable choices of t depending on how large d is as compared with n.

The quantum algorithm runs in two stages. First, it searches for a subset $I \subseteq [n]$ of size t such that it contains an index $i \in [n]$ with $x_i = y_i = 1$. If it succeeds in finding such a subset I, in the second stage, the algorithm looks for an index $i \in I$ such that $x_i = y_i = 1$. For this, it sequentially runs through the indices in I and checks if the requisite condition is satisfied. The second stage can thus be implemented in $\mathrm{O}(\max\{1, t/d\})$ rounds. The choice of t is such that the number of rounds in the first stage always dominates, and gives us the stated bounds.

We describe the first stage next. In order to identify a subset I containing an index i as above, if there is any, we implement a search algorithm based on a quantum walk on the Johnson Graph $J(n,t)$, following the framework due to Magniez, Nayak, Roland, and Santha [13]. The vertices of $J(n,t)$ are t-subsets of $[n]$. There is an edge between two vertices I, I' in $J(n,t)$ iff I and I' differ in exactly 2 elements: $(I \setminus I') \cup (I' \setminus I) = \{i,j\}$ for distinct elements $i, j \in [n]$.

The three building blocks of such an algorithm are as follows.

Set-up: Construct the following starting superposition:

$$\frac{1}{\binom{n}{t}^{1/2}} \sum_{I \subseteq [n] \,:\, |I| = t} |(i, x_i) : i \in I\rangle \enspace .$$

Checking: Check whether $x_i = y_i = 1$ for some $i \in I$:

$$|(i, x_i) : i \in I\rangle \quad \mapsto \quad \begin{cases} -|(i, x_i) : i \in I\rangle, & \text{if } x_i = y_i = 1 \text{ for some } i \in I \enspace ; \\ |(i, x_i) : i \in I\rangle, & \text{otherwise.} \end{cases}$$

Update: Replace some index $j \in I$ by an index $k \notin I$, and update the corresponding bit x_j to x_k:

$$|(i, x_i) : i \in I\rangle |j\rangle |k\rangle \quad \mapsto \quad |(i, x_i) : i \in (I \setminus \{j\}) \cup \{k\}\rangle |k\rangle |j\rangle \enspace .$$

Let ε be the probability that a uniformly random t-subset of $[n]$ contains an index i such that $x_i = y_i = 1$, given that such an element i exists. We have $\varepsilon \in \Omega(t/n)$. Then, according to Theorem 1.4 in Ref. [13], there is an algorithm based on quantum walk that finds a subset I such that $x_i = y_i = 1$ for some $i \in I$, if there is any such subset, with constant probability $> 1/2$. The algorithm uses one instance of Set-up, and $\mathrm{O}(\sqrt{1/\varepsilon}\,)$ alternations of one instance of Checking with a sequence of $\mathrm{O}(\sqrt{t}\,)$ instances of Update, interspersed with other unitary operations that are independent of the inputs x, y. (The spectral gap of the Johnson graph needed in the analysis of the algorithm may be derived from the results in Ref. [11], for example.)

Note that Set-up uses t queries to x, and thus can be implemented in $\max(1, 2\lceil t/d \rceil)$ rounds. Update only requires 2 queries to x. Thus a sequence of $\sqrt{t}$ sequential Update operations can be implemented in order $\max(1, 2\sqrt{t}/d)$ rounds. We would like to use the Grover algorithm for unordered search to implement the checking step. The Grover algorithm incurs non-zero probability of error in general, while the algorithm due to Magniez *et al.* assumes that the checking step is perfect. We therefore use an algorithm for unordered search with small error due to Buhrman, Cleve, de Wolf, and Zalka [4] to implement Checking with error at most $c\sqrt{t/n}$ for a suitable positive constant c with order $\sqrt{t \log(n/t)}$ queries to y. Using standard arguments, this only increases the error of the quantum walk algorithm by a small constant, say $1/10$. In effect, Checking (with the stated error) can be implemented in order $\max(1, \sqrt{t \log n}/d)$ rounds. Thus the bound on the round complexity of the quantum walk algorithm is of the order of

$$\max\left\{1, \frac{t}{d}\right\} + \sqrt{\frac{n}{t}} \left(\max\left\{1, \frac{\sqrt{t \log n}}{d}\right\} + \max\left\{1, \frac{\sqrt{t}}{d}\right\} \right) \enspace . \tag{4.1}$$

In order to derive the bounds stated in the theorem, we optimise over t. We consider intervals of values for t such that each of the expressions involving maximisation in Eq. (4.1) simplifies to one of the terms. The intervals are given by partitioning $[n]$ at the points $d, d^2/\log n, d^2$. (Note that d need not be smaller than $d^2/\log n$.) We optimise the number of rounds within each interval, which in turn gives us a relation between d and n for which the rounds are minimised.

We first consider $d \le \log n$, so that $d^2/\log n \le d$, and t in the intervals

$$[1, d^2/\log n], \quad [d^2/\log n, d], \quad [d, d^2], \quad \text{and } [d^2, n] \enspace .$$

We optimise the number of rounds with t in each of these intervals, to find that the number of rounds is $\mathrm{O}(\sqrt{n \log n}/d)$ when $t \coloneqq d$. The optimal values of t in the other intervals also give the same bound, but we stay with $t = d$ so as to minimise the rounds in the second stage of the algorithm.

Next we consider $d \ge \log n$, so that $d \le d^2/\log n \le d^2$. We again optimise over t in four intervals, and get the following bounds:

1. $t \in [1, d]$: $\mathrm{O}(\sqrt{n/d}\,)$ when $t := d$.
2. $t \in [d, d^2/\log n]$: $\mathrm{O}(\sqrt[3]{n/d}\,)$ when $t := \sqrt[3]{nd^2}$, *provided* $d^4 \geq n \log^3 n$. If $d^4 \leq n \log^3 n$, we get $\mathrm{O}(\sqrt{n \log n}/d)$ when $t := d^2/\log n$.
3. $t \in [d^2/\log n, d^2]$: $\mathrm{O}(\sqrt{n \log n}/d)$ when $t := d^2/\log n$ *provided* $d^4 \leq n \log^3 n$; If $d^4 \geq n \log^3 n$, we get $\mathrm{O}(d/\log n)$ with the same value of t.
4. $t \in [d^2, n]$: $\mathrm{O}(\sqrt{n \log n}/d)$ when $t := d^2$ *provided* $d^4 \leq n \log n$; If $d^4 \geq n \log n$, we get $\mathrm{O}(d)$ with the same value of t.

Since $\sqrt{n/d} \geq \sqrt{n \log n}/d$ when $d \geq \log n$, $\sqrt{n \log n}/d \leq d$ when $d^4 \geq n \log n$, and $(n/d)^{1/3} \leq d/\log n$ when $d^4 \geq n \log^3 n$, we conclude the bounds on round complexity stated in the theorem:

- $\mathrm{O}(\sqrt{n \log n}/d)$ with $t := d$ when $d \leq \log n$, or with $t := d^2/\log n$ when $\log^4 n \leq d^4 \leq n \log^3 n$, and
- $\mathrm{O}(\sqrt[3]{n/d}\,)$ with $t := \sqrt[3]{nd^2}$ when $d^4 \geq n \log^3 n$.

The bounds on query complexity follow. ◀

Note that in the range of parameters such that $n \log^2 n \leq d^4 \leq n \log^3 n$, the upper bound $\sqrt{n \log n}/d$ is at most $\sqrt{\log n}$ times the lower bound $\sqrt[3]{n/d \log n}$. So the bounds in Theorems 11 and 12 are indeed within polylogarithmic factors of each other for all values of d, n (such that $d \leq n$).

5 Conclusion

In this work, we studied a fundamental problem, Set Disjointness, in two concrete computational models. Set Disjointness on the Line $\mathsf{L}_{n,d}$ reveals new subtleties in distributed computation with quantum resources. It again puts the spotlight on the "double counting" of information in conditional information loss. One may think that the more sophisticated notion of *quantum information cost* introduced by Touchette [18], along with the results due to Braverman *et al.* [3], might help us overcome this drawback. Indeed, quantum information cost helps us overcome the limitation(s) of the former quantity in the case of Set Disjointness in the standard two-party communication model. Surprisingly, these techniques do not seem to help in obtaining a better lower bound for $\mathsf{L}_{n,d}$. We believe that new ideas may be needed to characterise the its asymptotic round complexity.

The two-oracle query model we introduce gives us a different perspective on Set Disjointness on a Line. It hints at the possibility of a better communication protocol for $\mathsf{L}_{n,d}$, although we may also interpret the optimal algorithm in this model as highlighting the limitation of the information loss technique. More generally, the new query model is tailored towards the study of distributed algorithms on the line and could shed light on protocols for other similar problems. Moreover, the model could also be of relevance in other distributed computation scenarios.

References

1 Scott Aaronson and Andris Ambainis. Quantum search of spatial regions. In *Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2003)*, pages 200–209. IEEE Computer Society, October 2003.

2 Andris Ambainis. Understanding quantum algorithms via query complexity. In *Proceedings of the International Congress of Mathematicians (ICM 2018)*, pages 3265–3285. World Scientific, 2019.

3 Mark Braverman, Ankit Garg, Young Kun Ko, Jieming Mao, and Dave Touchette. Near-optimal bounds on the bounded-round quantum communication complexity of Disjointness. *SIAM Journal on Computing*, 47(6):2277–2314, 2018.

4 Harry Buhrman, Richard Cleve, Ronald de Wolf, and Christof Zalka. Bounds for small-error and zero-error quantum algorithms. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science (FOCS 1999)*, pages 358–368, USA, October 1999. IEEE Computer Society.

5 Harry Buhrman and Ronald de Wolf. Complexity measures and decision tree complexity: a survey. *Theoretical Computer Science*, 288(1):21–43, 2002.

6 Arkadev Chattopadhyay and Toniann Pitassi. The story of Set Disjointness. *SIGACT News*, 41(3):59–85, September 2010.

7 Ronald de Wolf. Quantum computing: Lecture notes, 2019. `arXiv:1907.09415`.

8 Silvio Frischknecht, Stephan Holzer, and Roger Wattenhofer. Networks cannot compute their diameter in sublinear time. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2012)*, pages 1150–1162, 2012.

9 Stephan Holzer and Roger Wattenhofer. Optimal distributed all pairs shortest paths and applications. In *Proceedings of the 2012 ACM Symposium on Principles of Distributed Computing (PODC 2012)*, pages 355–364, 2012.

10 Rahul Jain, Jaikumar Radhakrishnan, and Pranab Sen. A lower bound for the bounded round quantum communication complexity of Set Disjointness. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2003)*, pages 220–229. IEEE Computer Society Press, 2003.

11 Donald E. Knuth. Combinatorial matrices. Manuscript available at `http://www-cs-faculty.stanford.edu/~knuth/preprints.html#unpub`, 1993.

12 François Le Gall and Frédéric Magniez. Sublinear-time quantum computation of the diameter in CONGEST networks. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing (PODC 2018)*, pages 337–346. Association for Computing Machinery, 2018.

13 Frédéric Magniez, Ashwin Nayak, Jérémie Roland, and Miklos Santha. Search via quantum walk. *SIAM Journal on Computing*, 40:142–164, 2011.

14 Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information.* Cambridge University Press, Cambridge, UK, 2000.

15 David Peleg. *Distributed Computing: A Locality-Sensitive Approach.* Society for Industrial and Applied Mathematics, USA, 2000.

16 David Peleg, Liam Roditty, and Elad Tal. Distributed algorithms for network diameter and girth. In *Proceedings of the 39th Internatinal Colloquium on Automata, Languages, and Programming (ICALP 2012)*, pages 660–672, 2012.

17 Alexander Razborov. Quantum communication complexity of symmetric predicates. *Izvestiya: Mathematics*, 67(1):145–159, 2003. Russian version in *Izvestiya Rossiiskoi Academii Nauk (seriya matematicheskaya) 67* (2003), 1, 159–176.

18 Dave Touchette. Quantum information complexity. In *Proceedings of the Forty-seventh Annual ACM Symposium on Theory of Computing (STOC 2015)*, pages 317–326, New York, NY, USA, 2015. ACM.

19 John Watrous. *The Theory of Quantum Information.* Cambridge University Press, May 2018.

20 Andrew Chi-Chih Yao. Some complexity questions related to distributive computing (preliminary report). In *Proceedings of the Eleventh Annual ACM Symposium on Theory of Computing (STOC 1979)*, pages 209–213, New York, NY, USA, 1979. Association for Computing Machinery.

21 Andrew Chi-Chih Yao. Quantum circuit complexity. In *Proceedings of the 34th Annual IEEE Symposium on Foundations of Computer Science (FOCS 1993)*, pages 352–361. IEEE Computer Society Press, 1993.

22 Christof Zalka. Grover's quantum searching algorithm is optimal. *Physical Review A*, 60:2746–2751, October 1999.

Can Verifiable Delay Functions Be Based on Random Oracles?

Mohammad Mahmoody
University of Virginia, Charlottesville, VA, USA
mohammad@virginia.edu

Caleb Smith
University of Virginia, Charlottesville, VA, USA
caleb@virginia.edu

David J. Wu
University of Virginia, Charlottesville, VA, USA
dwu4@virginia.edu

Abstract

Boneh, Bonneau, Bünz, and Fisch (CRYPTO 2018) recently introduced the notion of a *verifiable delay function* (VDF). VDFs are functions that take a long *sequential* time T to compute, but whose outputs $y \coloneqq \mathsf{Eval}(x)$ can be efficiently verified (possibly given a proof π) in time $t \ll T$ (e.g., $t = \mathrm{poly}(\lambda, \log T)$ where λ is the security parameter). The first security requirement on a VDF, called *uniqueness*, is that no polynomial-time algorithm can find a convincing proof π' that verifies for an input x and a different output $y' \neq y$. The second security requirement, called *sequentiality*, is that no polynomial-time algorithm running in time $\sigma < T$ for some parameter σ (e.g., $\sigma = T^{1/10}$) can compute y, even with $\mathrm{poly}(T, \lambda)$ many parallel processors. Starting from the work of Boneh et al., there are now multiple constructions of VDFs from various algebraic assumptions.

In this work, we study whether VDFs can be constructed from ideal hash functions in a black-box way, as modeled in the random oracle model (ROM). In the ROM, we measure the running time by the number of oracle queries and the sequentiality by the number of *rounds* of oracle queries. We rule out two classes of constructions of VDFs in the ROM:

- We show that VDFs satisfying *perfect* uniqueness (i.e., VDFs where no different convincing solution $y' \neq y$ exists) cannot be constructed in the ROM. More formally, we give an attacker that finds the solution y in $\approx t$ *rounds* of queries, asking only $\mathrm{poly}(T)$ queries in total.
- We also rule out *tight* verifiable delay functions in the ROM. Tight verifiable delay functions, recently studied by Döttling, Garg, Malavolta, and Vasudevan (ePrint Report 2019), require sequentiality for $\sigma \approx T - T^{\rho}$ for some constant $0 < \rho < 1$. More generally, our lower bound also applies to proofs of sequential work (i.e., VDFs without the uniqueness property), even in the private verification setting, and sequentiality $\sigma > T - {}^{T}\!/_{2t}$ for a concrete verification time t.

2012 ACM Subject Classification Theory of computation → Cryptographic primitives

Keywords and phrases verifiable delay function, lower bound, random oracle model

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.83

Category Track A: Algorithms, Complexity and Games

Related Version The full version of this paper is available at `https://eprint.iacr.org/2019/663`.

Funding *Mohammad Mahmoody*: Supported by NSF awards CCF-1350939, CCF-1910681, CNS-1936799 and a University of Virginia SEAS Research Innovation Award.
David J. Wu: Supported by NSF CNS-1917414 and a University of Virginia SEAS Research Innovation Award.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 83; pp. 83:1–83:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

A verifiable delay function (VDF) [5] with domain $\mathcal{X}$ and range $\mathcal{Y}$ is a function that takes long *sequential* time T to compute, but whose output can be efficiently verified in time $t \ll T$ (e.g., $t = \text{poly}(\lambda, \log T)$ where λ is a security parameter). More precisely, there exists an evaluation algorithm Eval that on input $x \in \mathcal{X}$ computes a value $y \in \mathcal{Y}$ and a proof π in time T. In addition, there is a verification algorithm Verify that takes as input a domain element $x \in \mathcal{X}$, a value $y \in \mathcal{Y}$, and a proof π and either accepts or rejects in time t. In some cases, a VDF might also have a setup algorithm Setup which generates a set of public parameters pp that is provided as input to Eval and Verify.[1] Typically, we require that the setup is also fast: namely, Setup runs in time $s = \text{poly}(\lambda, \log T)$ as well. The two main security requirements for a VDF are (1) *uniqueness* which says that for all inputs $x \in \mathcal{X}$, no adversary running in time $\text{poly}(\lambda, T)$ can find $y' \neq \mathsf{Eval}(x)$ and a proof π' such that $\mathsf{Verify}(x, y', \pi') = 1$; and (2) *sequentiality* with some parameter $\sigma < T$, which says that no adversary running in *sequential time* σ can compute $y = \mathsf{Eval}(x)$. The sequential time σ allows the adversary to have up to $\text{poly}(\lambda, T)$ parallel processors.

Verifiable delay functions have recently received extensive study, and have found numerous applications to building randomness beacons [5, 13] and cryptographic timestamping schemes [19]. Driven by these exciting applications, a sequence of recent works have developed constructions of verifiable delay functions from various algebraic assumptions [14, 24, 28, 29]. However, existing constructions still leave much to be desired in terms of concrete efficiency, and today, there are significant community-driven initiatives to construct, implement, and optimize more concretely-efficient VDFs [16]. One of the bottlenecks in existing constructions of VDFs is their reliance on structured algebraic assumptions (e.g., groups of unknown order [4, 26] or isogenies over pairing groups [14]).

A natural question to ask is whether we can construct VDFs generically from *unstructured* primitives, such as one-way functions, collision-resistant hash functions, or stronger forms of hash functions. In this work, we study whether black-box constructions of VDFs are possible starting from hash functions or other symmetric primitives. Specifically, we consider black-box constructions of VDFs from ideal hash functions (modeled as a random oracle). Similar to previous work (cf. [2, 21]) in the *parallel* random oracle model (ROM), we measure the running time of the adversary by the number of oracle queries it makes and the sequentiality of the adversary by the number of *rounds* of oracle queries it makes. However, we simply refer to the parallel ROM as the ROM.

1.1 Our Results

In this work, we rule out the existence of VDFs with *perfect* uniqueness (i.e., VDFs where for any $x \in \mathcal{X}$, there does not exist any (y', π) such that $\mathsf{Verify}(x, y', \pi) = 1$ and $y' \neq \mathsf{Eval}(x)$) in the random oracle model. Specifically, we construct an adversary that asks $O(t)$ rounds of queries and a total number of $\text{poly}(T)$ queries and breaks the uniqueness of VDFs with respect to *some* oracle.

A natural class of VDFs with perfect uniqueness is the class of *permutation* VDFs where the function $\mathsf{Eval}(\cdot)$ implements a permutation on the domain (specifically, $\mathcal{X} = \mathcal{Y}$), and verification consists of inverting y and checking if it is x or not. Recently, Abusalah et al. [1]

[1] Ideally, the public parameters can be sampled by a public-coin process [5, 24, 29]. Otherwise, we require a trusted setup to generate the public parameters [14, 28].

constructed permutation VDFs in the ROM, but they additionally relied on the assumptions used in the sloth functions of Lenstra and Wesolowski [20]. Our work shows that relying on some kind of structured assumption is *necessary* to achieve permutation VDFs.

We next show that in the "tight" regime of sequentiality, as recently studied in the concurrent work of Döttling et al. [10] (i.e., when the sequentiality parameter σ is quite close to T), even *proofs of sequential work* (PoSW) [7,12,22,25] cannot be based on random oracles. In particular, our result essentially applies to settings where $\sigma \gg T \cdot (1 - 1/t)$ where t is the verification time. A proof of sequential work is a relaxation of a VDF without the uniqueness or the public-verifiability properties. Thus, our lower bound for ruling out tight PoSW also rules out tight VDFs in the ROM. We note, however, that since (even publicly-verifiable) PoSW satisfying weaker notions of sequentiality (e.g., $\sigma = T/2$) are known to exist in the ROM [22], it is not clear whether this lower bound for PoSW can be extended to rule out (non-tight) VDFs, and we leave this as an intriguing open question.

We note that both of our lower bounds are proven in settings that have already been studied in previous (or concurrent) work. Namely, by ruling out permutation VFDs in the ROM, our first main result complements the previous work on permutation VDFs [1] by showing that the random oracle alone is not sufficient to realize such strong VDFs. Moreover, our second result shows that when it comes to the tight regime of sequentiality (studied in the concurrent work of [10]), VDFs as well as more relaxed notions like proofs of sequential work cannot be based on symmetric primitives in a black-box way.

1.1.1 Our Techniques

At a technical level, the proofs of our lower bounds start from the techniques of Mahmoody, Moran, and Vadhan [21] for ruling out time-lock puzzles in the random oracle model. In fact, for a special case of perfectly-unique VDFs where the VDF is a *permutation* on its domain $\mathcal{X} = \mathcal{Y}$, which we refer to as a "permutation VDF" (cf., [1,18,20]), we can use the impossibility result of [21] as a black-box by reducing the task of constructing time-lock puzzles in the ROM to constructing permutation-VDFs in the ROM.

For the more general case of perfectly-unique VDFs (that are not necessarily permutations) we cannot use the result of [21] as a black-box, but we can still adapt the ideas from their work that are reminiscent of similar techniques also used in [6,23,27]. Namely, our attacker will sample full executions of the evaluation function Eval in its head, while respecting answers to queries that it has already learned from the real oracle. At the end of each simulated execution, it will ask all previously-unasked queries in a *single round* to the oracle (and use those values in subsequent simulated executions). We show that using just $O(t)$ rounds of this form, we can argue that in most of these rounds, the adversary does not hit any "new query" in the verification process. Consequently, in most of the executions it is *consistent* with the verification procedure with respect to *some* oracle O', and thus by the *perfect uniqueness* property, the answer in those executions should be the correct one. Finally, by taking a majority vote over the executions, we obtain the correct answer with high probability. Observe that this argument critically relies on *perfect uniqueness.* The main open question remaining is whether a similar lower bound for *computational uniqueness* holds for VDF in the ROM or not. In this setting, the security requirement is that no *efficient* adversary can find a different value $y' \neq \mathsf{Eval}(x)$ with a proof π' that passes verification. (See Section 3.1.)

We then adapt this technique to additionally rule out *tight* proofs of sequential work in the ROM. Roughly, a scheme satisfies tight sequentiality if no adversary running in sequential time $\sigma = T - T^{\rho}$ for constant $\rho < 1$ can compute (y, π) such that $\mathsf{Verify}(x, y, \pi) = 1$. More generally, we consider a setting of parameters where $\sigma > T \cdot (1 - {}^{1}/_{2t})$, where t denotes the number of queries made by Verify. In this setting, we can construct an algorithm that simulates the answers to *some* but *not* all of the oracle queries made by the evaluation algorithm. The algorithm answers the remaining queries based on the real oracle evaluations. To simplify the description, in the following, assume that the setup algorithm is essentially nonexistent, and that the public parameter is fixed and publicly known ahead of time. Since the scheme is assumed to be *tightly* sequential, as long as the algorithm simulates sufficiently-many queries (e.g., $T/2t$ queries), it is possible to reduce the number of rounds of queries made to the real oracle (i.e., from T to $T - T/2t$). Moreover, if the number of "simulated responses" is small enough and if the set of queries for which the algorithm simulates is chosen at random, then there is a good chance that none of the simulated queries are asked during verification. If both of these properties hold simultaneously, then the algorithm successfully computes a response that verifies, thus breaking tight sequentiality. We provide the formal analysis in Section 3.2. However, the formal version of this argument needs to also incorporate the query complexity of the setup algorithm as well, leading to a weaker attack that only applies when $\sigma > T \cdot (1 - 1/2(s+t))$. However, in Section 4, we describe how to extend our lower bounds on tight proofs of sequential work (and correspondingly, tight VDFs) to additionally rule out tight proofs of sequential work with an *expensive* setup phase as well as tight proofs of sequential work in the random *permutation* model. In particular, we show how to leverage preprocessing to essentially eliminate the dependency of the attack's online phase on the query complexity of the setup.

1.2 Related Work

Verifiable delay functions are closely related to the notion of (publicly-verifiable) proofs of sequential work (PoSW) [1, 8, 11, 22]. The main difference between VDFs and PoSWs is *uniqueness*. More specifically, a VDF ensures that for every input x, an adversary running in time $\text{poly}(\lambda, T)$ can only find at most *one* output y (accompanied with a possibly non-unique proof π) that the verifier would accept (and if it does, the verifier is also convinced that the prover performed T sequential work). In contrast, a PoSW does not provide any guarantees on uniqueness. In particular, for every input x, there might be many possible pairs (y, π) that the verifier would accept, and as a result, in this setting there is no longer a need to distinguish between the output y and the proof π. Even more generally, proofs of work need not be publicly-verifiable [12], and one could only require sequentiality against adversaries who do not know a secret verification key generated during the setup. We emphasize that the uniqueness property in VDFs is important both for applications as well as constructions. Indeed, publicly-verifiable proofs of sequential work can be constructed in the random oracle model [8, 11, 22], while our work rules out a broad class of VDFs in the same model.

Time-lock puzzles. Time-lock puzzle [25] are closely related to VDFs as they are also based on the notion of sequentiality. In a time-lock puzzle, a puzzle generator can generate a puzzle x *together* with a solution y in time $t \ll T$, but computing y from x still requires sequential time T. The main difference between VDFs and time-lock puzzles is that time-lock puzzles might require knowledge of a *secret key* for efficient verification (in time t). In contrast, VDFs are publicly-verifiable (in time t). However, similar to VDFs, the output of a time-lock puzzle is *unique*. Mahmoody et al. [21] leverage this very uniqueness property *and* the fact that the solution is known ahead of the time to the verifier (because it is sampled during the puzzle generation) to show an impossibility result for time-lock puzzles in the random oracle model. While VDFs also require unique solutions, these solutions might not be known when we directly sample an input.

Concurrent work. In an independent and concurrent work, Döttling et al. [10] introduce and provide an in-depth study of tight verifiable delay functions. Their work both provides a positive construction of tight VDFs (from algebraic assumptions) as well as a negative result on the existence of *tight* VDFs in the random oracle model. In this work, we show that the lower bound on tight VDFs also extends to (even *privately*-verifiable) proofs of sequential work. At the same time, we note that (even publicly-verifiable) proofs of sequential work do exist in the *non-tight* regime (e.g., $\sigma = T/2$) in the ROM [22]. Thus, whether or not this lower bound in the random oracle model on tight VDFs can be extended to arbitrary (*non-tight*) VDFs or not still remains an intriguing open question.

2 Preliminaries

Throughout this work, we use λ to denote the security parameter. For an integer $n \in \mathbb{N}$, we write $[n]$ to denote the set $\{1, 2, \ldots, n\}$. We write $\text{poly}(\lambda)$ to denote a quantity that is bounded by a fixed polynomial in λ and $\text{negl}(\lambda)$ to denote a function that is $\lambda^{-\omega(1)}$. For a distribution D, we write $x \leftarrow D$ to denote that x is drawn from D. For a randomized algorithm Alg, we write $y \leftarrow \mathsf{Alg}(x)$ to denote the process of computing y by running Alg on input x with (implicitly-defined) randomness r of the appropriate length (based on the length of x). For a finite set $\mathcal{S}$, we write $x \overset{\$}{\leftarrow} \mathcal{S}$ to denote that x is sampled uniformly at random from $\mathcal{S}$. We say that an algorithm is efficient if it runs in probabilistic polynomial time in the length of its input. We now review the definition of a verifiable delay function.

▶ **Definition 1** (Verifiable Delay Function [5]). *A* verifiable delay function *is a tuple of algorithms* $\Pi_{\mathsf{VDF}} = (\mathsf{Setup}, \mathsf{Eval}, \mathsf{Verify})$ *with the following properties:*

- $\mathsf{Setup}(1^\lambda, T) \to \mathsf{pp}$*: On input the the security parameter* λ *and the time bound* T*, the setup algorithm outputs the public parameters* pp*. The public parameter determines a (uniformly)* samplable *input space* $\mathcal{X}_{\mathsf{pp}}$ *and an output space* $\mathcal{Y}_{\mathsf{pp}}$*. When the context is clear, we simply denote them as* $\mathcal{X}$ *and* $\mathcal{Y}$*.*
- $\mathsf{Eval}(\mathsf{pp}, x) \to (y, \pi)$*: On input the public parameters* pp *and an element* $x \in \mathcal{X}$*, the evaluation algorithm outputs a value* $y \in \mathcal{Y}$ *and a (possibly empty) proof* π*. Moreover, in case* Eval *is randomized, the first output* y *should be determined by* x *and* pp *(and be independent of the randomness used). We will typically refer to* y *as the "output" of the VDF on* x*, and since* y *is unique, when the context is clear, we simply write* $y = \mathsf{Eval}(\mathsf{pp}, x)$ *to denote the output of the VDF on* x*.*
- $\mathsf{Verify}(\mathsf{pp}, x, y, \pi) \to \{0, 1\}$*: On input the public parameters* pp*, an element* $x \in \mathcal{X}$*, a value* $y \in \mathcal{Y}$*, and a proof string* $\pi \in \{0, 1\}^*$*, the verification algorithm outputs a bit (*1 *means accept and* 0 *means reject).*

Moreover, the algorithms must satisfy the following efficiency requirements:

- *The setup algorithm* Setup *runs in time* $\text{poly}(\lambda, \log T)$*.*
- *The evaluation algorithm* Eval *runs in time* T*.*
- *The verification algorithm* Verify *runs in time* $\text{poly}(\lambda, \log T)$*.*

For simplicity of notation, in the following sections, we sometimes write s (resp., t) to denote the running time of Setup (resp., Verify). This additionally allow us to state our results more generally while also explicitly stating how our results depend on the precise bounds on the running time of Setup and Verify.

Completeness. We now define the completeness requirement on VDFs.

▶ **Definition 2** (Completeness of VDF). *A VDF* $\Pi_{\mathsf{VDF}} = (\mathsf{Setup}, \mathsf{Eval}, \mathsf{Verify})$ *has completeness error* γ *if for all* $\lambda \in \mathbb{N}$ *and* $T \in \mathbb{N}$*,*

$$\Pr\left[\mathsf{Verify}(\mathsf{pp}, x, y, \pi) = 1 \;:\; \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, T), x \xleftarrow{\$} \mathcal{X}_{\mathsf{pp}} \\ (y, \pi) \leftarrow \mathsf{Eval}(\mathsf{pp}, x) \end{array}\right] \geq 1 - \gamma.$$

Unless stated otherwise, we assume $\gamma = 0$. For our first lower bound, we need perfect completeness $\gamma = 0$, but our second lower bound in the tight regime directly extends to more generalized settings where the completeness error γ is $\operatorname{negl}(\lambda)$ (or even a small constant).

Security. The two main security requirements we require on a VDF are *uniqueness* and *sequentiality*. We define these below.

▶ **Definition 3** (Uniqueness of VDF). *A VDF* $\Pi_{\mathsf{VDF}} = (\mathsf{Setup}, \mathsf{Eval}, \mathsf{Verify})$ *satisfies* uniqueness *for a class* $\mathcal{A}$ *of adversaries with error* $\varepsilon(\lambda)$*, if for all adversaries* $\mathsf{Adv} \in \mathcal{A}$*, we have that*

$$\Pr\left[y \neq \mathsf{Eval}(\mathsf{pp}, x) \wedge \mathsf{Verify}(\mathsf{pp}, x, y, \pi) \;:\; \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, T) \\ (x, y, \pi) \leftarrow \mathsf{Adv}(1^\lambda, 1^T, \mathsf{pp}) \end{array} = 1\right] \leq \varepsilon(\lambda).$$

We say that Π_{VDF} *satisfies* statistical uniqueness *if* $\mathcal{A}$ *is the set of all (computationally unbounded) adversaries and* $\varepsilon(\lambda)$ *is negligible. We say* Π_{VDF} *satisfies* perfect uniqueness *if we further require* $\varepsilon(\lambda) = 0$*. We say that* Π_{VDF} *is* computationally unique *if* $\mathcal{A}$ *is the class of* $\operatorname{poly}(\lambda, T)$*-time adversaries and* $\varepsilon(\lambda)$ *is negligible.*

▶ **Definition 4** (Sequentiality of VDF). *A VDF* $\Pi_{\mathsf{VDF}} = (\mathsf{Setup}, \mathsf{Eval}, \mathsf{Verify})$ *is* σ*-sequential (where* σ *may be a function of* λ*,* T *and* t*) if for all adversaries* $\mathsf{Adv} = (\mathsf{Adv}_0, \mathsf{Adv}_1)$*, where* $\mathsf{Adv}_0, \mathsf{Adv}_1$ *both run in total time* $\operatorname{poly}(\lambda, T)$ *and* Adv_1 *runs in* parallel *time at most* σ*, we have that*

$$\Pr\left[y = \mathsf{Eval}(\mathsf{pp}, x) \;:\; \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, T) \\ \mathsf{st}_{\mathsf{Adv}} \leftarrow \mathsf{Adv}_0(1^\lambda, 1^T, \mathsf{pp}) \\ x \xleftarrow{\$} \mathcal{X}, y \leftarrow \mathsf{Adv}_1(\mathsf{st}_{\mathsf{Adv}}, x) \end{array}\right] = \operatorname{negl}(\lambda).$$

We can view Adv_0 *as a "preprocessing" algorithm that precomputes some initial state* $\mathsf{st}_{\mathsf{Adv}}$ *based on the public parameters and* Adv_1 *as the "online" adversarial evaluation algorithm.*

▶ **Definition 5** (Decodable VDF [5]). *Let* t *be a function of* λ *and* T*. A VDF* $\Pi_{\mathsf{VDF}} = (\mathsf{Setup}, \mathsf{Eval}, \mathsf{Verify})$ *is* t-decodable *if there is no extra proof (i.e.,* $\pi = \perp$*) and there is a* decoder Dec *with the following properties:*

- Dec *runs in time* t*.*
- *For all* $x \in \mathcal{X}$*, if* $y = \mathsf{Eval}(\mathsf{pp}, x)$*, then* $\mathsf{Dec}(\mathsf{pp}, y) = x$*.*

We say that Π_{VDF} *is* strongly *decodable, if for all* $y' \neq y = \mathsf{Eval}(\mathsf{pp}, x)$*, it holds that* $\mathsf{Dec}(\mathsf{pp}, y) \neq x$*. Finally, we say a VDF is* efficiently *(strongly) decodable if it is (strongly)* t*-decodable for* $t = \operatorname{poly}(\lambda, \log T)$*.*

▶ Remark 6 (Strongly Decodable VDFs and Perfect Uniqueness). Strong decodability (Definition 5) implies *perfect* uniqueness (Definition 3). However, the reverse is not true in general.

▶ **Definition 7** (Random Oracle Model (ROM))**.** *A random oracle $\mathcal{O}$ implements a truly random function from $\{0,1\}^*$ to range $\mathcal{R}$.*[2] *Equivalently, one can use "lazy evaluation" to simulate the behavior of a random oracle as follows:*

- *If the oracle has not been queried on $x \in \{0,1\}^*$, sample $y \xleftarrow{\$} \mathcal{R}$. The oracle returns y and remembers the mapping (x, y).*
- *If the oracle was previously queried on $x \in \{0,1\}^*$, return the previously-chosen value of $y \in \mathcal{R}$ associated with x.*

▶ Remark 8 (Hardness in the Random Oracle Model). Note that constructions with unconditional security in the ROM use the oracle as the *only* source of hardness. In particular, as stated above, the number of rounds of queries to the oracle model the cost of the parallel computation. If one allows other sources of computational hardness, existing constructions of VDFs trivially also exist relative to the ROM (by ignoring the random oracle).

▶ **Definition 9** (VDFs in the ROM)**.** *We define uniqueness and sequentiality of a VDF in the ROM by allowing the components* Setup, Eval, Verify *of a VDF to be oracle-aided algorithms in the ROM and adjusting their notion of time and parallel time (in Definitions 3 and 4 according to Definition 7 and Remark 8). In particular, for sequentiality, we measure the running time of the adversary by the number of* rounds *of oracle queries the adversary makes (this is to model the capabilities of a* parallel *adversary). Furthermore, for the uniqueness property, we require that the probability of the adversary succeeding is taken over the random coins of* Setup *and of the adversary, but* not *over the choice of oracle.*

3 Lower Bounds for VDFs in the Random Oracle Model

In this section, we first show that perfectly unique VDFs (Definition 3) are impossible in the random oracle model. Then, as a corollary, we obtain barriers for strongly decodable VDFs as well. In particular, if a VDF in the ROM is perfectly unique, it means that for *every* sampled random oracle $O \leftarrow \mathcal{O}$, perfect uniqueness holds. Moreover, our result shows that a recent construction of [1] for reversible VDFs that is of the form of a permutation function and uses the sloth functions from [20] cannot be modified to only rely on a random oracle.

We then turn our attention to the case of tight VDFs (and more generally, tight proofs of sequential work) and show that they cannot be constructed in the ROM as well.

3.1 Case of Perfectly Unique VDFs

In this section, we give a lower bound for perfectly unique VDFs in the random oracle model. Specifically, we prove the following theorem:

▶ **Theorem 10** (Lower Bounds for Perfectly Unique VDFs in the ROM)**.** *Suppose $\Pi_{\mathsf{VDF}} =$ (*Setup, Eval, Verify*) is a VDF in the ROM with perfect uniqueness and perfect completeness in which (for a concrete choice of λ),* Setup *runs in time s,* Eval *runs in time T, and* Verify *runs in time t. Then, there is an adversary* Adv *that breaks sequentiality (Definition 4) by asking a total of $2T \cdot (s + t)$ queries in $2(s + t)$ rounds of queries.*

[2] There are multiple possible ways to model a random oracle in the literature. We describe three possibilities here. Sometimes, the range $\mathcal{R}$ is $\{0,1\}^\lambda$ where λ is a security parameter; other times, it is simply $\{0,1\}$, and sometimes it is a "length preserving" mapping that maps each input x to a string of the same length.

Before proving Theorem 10, we observe that this result already rules out the possibility of constructing strongly decodable VDFs (which are forced to be perfectly unique; see Remark 6) in the ROM. In fact, a special case of this theorem for the class of "permutation VDFs" can be derived from the impossibility result of [21] for time-lock puzzles [25].[3] We first define this class of special VDFs below and prove their impossibility in the ROM as a warm up.

3.1.1 Warm Up: The Special Case of Permutation VDFs

Permutation VDFs. As a special case of strongly decodable VDFs, one can further restrict the mapping from $\mathcal{X}$ to $\mathcal{Y}$ to be a *permutation* (instead of just being an injective function). Indeed, the recent construction of [1] has this exact property as they construct decodable/reversible VDFs where the evaluation function is a permutation on its domain. Indeed, [1] uses random oracles together with the sloth function from [20] to construct their permutation VDF, and thus it was left open whether random oracle would suffice for permutation VDFs. Our result rules this possibility out.

▶ **Proposition 11.** *Let Π_{VDF} be a permutation VDF in the ROM with a decoder* Dec *that runs in time t, and a setup algorithm* Setup *that runs in time s. Then, there is an adversary that breaks sequentiality (Definition 4) in $O(s+t)$ rounds of queries and $O(T \cdot (s+t))$ queries.*

Proof. Previously, Mahmoody et al. [21] showed an impossibility result for time-lock puzzles in the ROM. To prove the claim, we show how to construct a time-lock puzzle from a permutation VDF. The result then follows from the lower bound of [21]. The construction is as follows. The puzzle-generator would first run the setup algorithm Setup of the VDF to get the public parameter pp. Then, it samples $y \xleftarrow{\$} \mathcal{X} = \mathcal{Y}$ (here, we use the fact that $\mathcal{X}$ is efficiently-samplable, and that $\mathcal{X} = \mathcal{Y}$) and sets $x \leftarrow \mathsf{Dec}(\mathsf{pp}, y)$. It outputs x as the puzzle (and keeps y as the solution). Since Setup and Dec for a VDF are both efficient (i.e., run in time $\mathrm{poly}(\lambda, t)$), the puzzle-generator is also efficient.

We can now use the result of [21] which shows that any time-lock puzzle in the ROM where the puzzle-generation algorithm makes k queries and the puzzle-solving algorithm makes T queries can be broken by an adversary making $O(k)$ rounds of queries and a total of $O(k \cdot T)$ queries. For the time-lock puzzle based on the permutation VDF, $k = s + t$, where s is the number of queries made by the Setup algorithm and t is the number of queries made by the Dec algorithm. ◀

3.1.2 Proof of Theorem 10 in the General (Perfectly Unique) Case

We now give the proof of Theorem 10. It follows the ideas from [21] for ruling out time-lock puzzles in the ROM, but this time, we cannot simply reduce the problem to the setting of time-lock puzzles, and we need to go into the proof and extend it to our setting. We begin with an informal overview of the proof before providing the formal analysis.

Proof overview. We begin with an informal description of the main ideas behind the lower bound. Our goal is to construct an adversary that can efficiently find an output that passes verification, while asking fewer than T *rounds* of queries to the random oracle. To do so, we consider an algorithm that implements the honest evaluation algorithm, but instead of

[3] In a time-lock puzzle, there is a puzzle-generation algorithm that runs in time t and samples a puzzle x together with a solution y, and an evaluation algorithm that runs in sequential time T that takes an input x and outputs the solution y.

Algorithm 1 The adversary Adv that breaks sequentiality of the VDF.

```
1. At the beginning of the game, the adversary Adv receives the public parameters pp and a
   challenge $x \in \mathcal{X}$ from the sequentiality challenger.
2. Initialize a query set $Q_{\mathsf{Adv}} \leftarrow \varnothing$ and a set of query-answer pairs $P_{\mathsf{Adv}} \leftarrow \varnothing$.
3. Let $d = 2(s+t)+1$.
4. For $i \in [d]$, do the following:
   a. Initialize $P^{(i)}_{\mathsf{Adv}} \leftarrow \varnothing$ and $Q^{(i)}_{\mathsf{Adv}} \leftarrow \varnothing$.
   b. Execute $(y_i, \pi_i) \leftarrow \mathsf{Eval}(\mathsf{pp}, x)$ where the random oracle queries (made by Eval) are
      answered using the following procedure. On every oracle query $q$:
      - If $q \in Q_{\mathsf{Adv}}$, then reply with the value $r$ where $(q, r) \in P_{\mathsf{Adv}}$.
      - Otherwise, choose a uniformly random value $r \xleftarrow{\$} \mathcal{R}$ (where $\mathcal{R}$ is the range of the
        random oracle $\mathcal{O}$) and add $(q, r)$ to $P^{(i)}_{\mathsf{Adv}}$ and add $q$ to $Q^{(i)}_{\mathsf{Adv}}$.
   c. If $i < d$, then in *one round*, for all $(q, \star) \in P^{(i)}_{\mathsf{Adv}}$, query the real oracle $\mathcal{O}$ to get $r \leftarrow \mathcal{O}(q)$
      as the answer; and then add $(q, r)$ to $P_{\mathsf{Adv}}$ and add $q$ to $Q_{\mathsf{Adv}}$.
5. Output $\mathrm{Maj}(y_1, \ldots, y_d)$ where Maj denotes the majority operation (which outputs $\perp$ if no
   majority exists).
```

issuing queries to the actual random oracle, the algorithm will instead *sometimes* simulate the outputs from those queries itself (i.e., by sampling from the output distribution of the random oracle). Of course, if one of these "faked" or "simulated" queries was asked to the oracle by another algorithm in the system (e.g., by Setup or Verify), then our algorithm will almost certainly fail. On the other hand, if none of the simulated queries were asked to the oracle, then our attacker wins (because the set of "faked" queries and real queries together form an oracle that is consistent). This means that as long as the number of queries Setup and Verify make are much smaller than T and our algorithm is able to "identify" or "learn" those queries with fewer than T rounds of queries to the random oracle, then the algorithm succeeds in breaking sequentiality of the VDF.

Proof of Theorem 10. Without loss of generality, assume that Eval asks no repeated queries in a single execution. We construct an attacker Adv that breaks sequentiality of the VDF as follows. Our adversary is entirely online (i.e., there is no separate preprocessing step).

We now show that Adv in Algorithm 1 satisfies the properties needed in Theorem 10. Let Q_S be the queries made by the setup algorithm $\mathsf{Setup}(1^\lambda, T)$ to sample pp and Q_V be the queries made by $\mathsf{Verify}(\mathsf{pp}, x, y)$ where $y = \mathsf{Eval}(\mathsf{pp}, x)$ is the true solution.

For $i \in [d]$, we define H_i to be the event where there is a query $q \in Q^{(i)}_{\mathsf{Adv}} \cap (Q_S \cup Q_V)$ during the i^{th} round of emulation that was *not* previously asked by the adversary: $q \notin Q_{\mathsf{Adv}}$ *at that moment*. Equivalently, when q is asked, it holds that $q \in (Q^{(i)}_{\mathsf{Adv}} \cap (Q_S \cup Q_V)) \setminus Q_{\mathsf{Adv}}$.

The following claim shows that H_i cannot happen for too many rounds i.

▷ Claim 12. If $\mathcal{I} = \{i : H_i \text{ holds}\}$, then $|\mathcal{I}| \le s + t$.

Proof. If event H_i occurs as a result of some query q, then at the end of round i, Adv queries the oracle $\mathcal{O}$ on the input q. By construction, since $q \in (Q^{(i)}_{\mathsf{Adv}} \cap (Q_S \cup Q_V)) \setminus Q_{\mathsf{Adv}}$, it must be the case that Adv makes a *new* query on an input that was previously queried by either the Setup or Verify algorithms (but *not* queried in any of the previous rounds). However, since Setup and Verify together ask a combined total of (at most) $s + t$ queries, this event cannot happen more than $s + t$ times. ◁

▷ Claim 13. If H_i does not happen, then $y_i = y$.

Proof. Let $y_i \neq y$ for a round i in which H_i has not happened. This means that the set of oracle query-answer pairs used during Setup, and the i^{th} emulation of Eval by Adv are *consistent.* Namely, there is an oracle O', relative to which, we have $\mathsf{pp} \leftarrow \mathsf{Setup}^{O'}$, $(y', \pi') \leftarrow \mathsf{Eval}^{O'}(\mathsf{pp}, x)$, and $\mathsf{Verify}^{O'}(\mathsf{pp}, x, y, \pi) = 1$. However, this shows that the perfect uniqueness property is violated relative to O', because for input x, there is a "wrong" solution y (i.e., $y \neq y' = \mathsf{Eval}^{O'}(\mathsf{pp}, x)$) together with some proof π for y such that the verification passes $\mathsf{Verify}^{O'}(\mathsf{pp}, x, y, \pi) = 1$. ◁

By the above two claims, it holds that $y_i = y$ for at least $s+t+1$ values of $i \in [2(s+t)+1]$, and thus the majority gives the right answer y for Adv. ◀

3.2 Lower Bound for Tight Proofs of Sequential Work

We can apply similar techniques to rule out *tight* proofs of sequential work [22] in the random oracle model. At a high level, a (publicly-verifiable) proof of sequential work is a VDF without *uniqueness.* Namely, for an input x, there can be many pairs (y, π) that passes verification. In this setting, there is no need to distinguish y and π. While we have constructions of (publicly-verifiable) proofs of sequential work in the ROM, our results show that *tight* proofs of sequential work (see [10] for more discussion on this tightness notion) are impossible in this setting. In particular, the following barrier applies to settings where the sequentiality parameter σ is *very* close to T (e.g., this does not apply to $\sigma = T/2$). The following definition derives publicly-verifiable proofs of sequential work [7, 12, 22, 25] as a relaxation of VDFs.[4]

▶ **Definition 14** (Publicly-Verifiable Proofs of Sequential Work). *A* publicly-verifiable *proof of sequential work is a relaxation of a VDF where the uniqueness property is not needed. Hence, there is no need to distinguish between an output y and a proof π. In particular, $y = \pi$ can be the only (not-necessarily-unique) output of* Eval *that is still sequentially hard to compute.*

Proofs of sequential work in the ROM. While Definition 9 defines the notion of a VDF in the ROM, the same definition extends to the setting of (publicly-verifiable) proofs of sequential work in the ROM as well. Namely, we measure the running time in terms of the total number of oracle queries and the parallel time by the the number of rounds of oracle queries. We now show how to adapt our techniques for ruling our perfectly-unique VDFs in the ROM to also rule out tight proofs of sequential work.

▶ **Theorem 15** (Attacking Proofs of Sequential Work in the ROM). *Suppose $\Pi_{\mathsf{PSW}} =$ (Setup, Eval, Verify) is a publicly-verifiable proof of sequential work in the ROM in which (for a concrete choice of λ),* Setup *runs in time s,* Eval *runs in time T, and* Verify *runs in time t. Then, for any $1 < G < T$ there is an adversary* Adv *that asks a total of at most $T - G$ queries and breaks sequentiality (Definition 4) with probability at least $1 - (s+t) \cdot G/T$.*

▶ **Corollary 16** (Ruling Out Tight Proofs of Sequential Work in the ROM). *Let λ be a security parameter and T be the time bound parameter. For any choice of $s, t = \operatorname{poly}(\lambda, \log T)$, there does not exist a proof of sequential work in the ROM with sequentiality $\sigma = T \cdot (1 - 1/2(s+t))$. More generally, for any constant $0 < \rho < 1$, there does not exist a proof of sequential work in the ROM with sequentiality $\sigma = T - T^{\rho}$.*

[4] Definition 14 is even more general as it allows a setup phase.

Proof. The first statement follows by instantiating Theorem 15 with $G = {}^{T}\!/_{2(s+t)}$. For this setting of parameters, Theorem 15 implies an adversary that breaks sequentiality with probability at least $1/2$ and making only $T - G = T \cdot (1 - {}^{1}\!/_{2(s+t)})$ queries. For the more general statement, we take T to be a sufficiently large polynomial in the security parameter λ such that $s, t < T^{1-\rho}/4$. Note that this is always possible since $s, t = \mathrm{poly}(\lambda, \log T)$; that is, both s, t are poly-logarithmic in T and $0 < \rho < 1$ is constant. Then, we can again set $G = T - \sigma = T^{\rho}$ and appeal to Theorem 15 to obtain an adversary that makes $T - G = \sigma$ queries and breaks sequentiality with probability at least $1/2$. ◀

▶ Remark 17 (Secretly-Verifiable Tight Proofs of Sequential Work). We note that Theorem 15 also holds for secretly-verifiable proofs of sequential work as well. The proof for the secretly-verifiable setting is identical to that for the publicly-verifiable setting. For simplicity of notation we write the proof for the publicly-verifiable version (Definition 4) which uses pp as both the evaluation and the verification key.

▶ Remark 18 (Other Extensions). The extension from Remark 17 on ruling out secretly-verifiable tight proofs of sequential work relies on the same proof as that of Theorem 15. In Section 4, we show two extensions of this result by adapting the proof of Theorem 15. These include extending the lower bound to: (1) the setting where the Setup algorithm is "slow" (i.e., runs in time proportional to T); and (2) the random permutation model (rather than the random oracle model). We refer to Section 4 for the details on those extensions.

Proof of Theorem 15. Again, without loss of generality, we assume that Eval asks no repeated queries in a single execution. The attacker's algorithm Adv is defined as follows:

Algorithm 2 The adversary Adv that breaks sequentiality for the proof of sequential work.

```
1. At the beginning of the game, the adversary Adv receives the public parameters pp and a
   challenge $x \in \mathcal{X}$ from the sequentiality challenger.
2. Pick a random set $\mathcal{S} \subseteq [T]$ of size $T - G$.
3. Execute $(y, \pi) \leftarrow \mathsf{Eval}(\mathsf{pp}, x)$ where the $i^{\text{th}}$ oracle query $q_i$ is answered as follows:
     - If $i \in \mathcal{S}$, compute the response $r_i \leftarrow \mathcal{O}(q_i)$ from the true oracle $\mathcal{O}$.
     - Otherwise sample a uniformly random $r_i \leftarrow \mathcal{R}$ as the response for the query $q_i$.
4. Output $(y, \pi)$.
```

To analyze Algorithm 2, we compare the output of the attacker's experiment (Real) with the output in an "ideal" experiment (Ideal) where all of the oracle queries are answered using the real oracle. We define these two experiments below:

1. Sample $\mathsf{pp} \leftarrow \mathsf{Setup}(1^{\lambda}, T)$.
2. Sample $x \xleftarrow{\$} \mathcal{X}$.
3. In Experiment Real, run Algorithm 2 to obtain a pair (y, π). In Experiment Ideal, run Algorithm 2, except use the real oracle $\mathcal{O}$ to answer *all* of the oracle queries made by Eval (in Step 3 of Algorithm 2).
4. The output of the experiment is 1 if $\mathsf{Verify}(\mathsf{pp}, x, y, \pi) = 1$ and 0 otherwise.

Figure 1 The Real and Ideal experiments.

In the following, we write $\Pr_{\mathsf{real}}[\cdot]$ (resp., $\Pr_{\mathsf{ideal}}[\cdot]$) to denote the probability of an event E in the Real (resp., Ideal) experiment.

Events. Let Q_V be the set of oracle queries made by Verify and Q_S be the set of oracle queries made by Setup. Let Q_{Adv} be the set of oracle queries q_i that appears in Step 3 of Algorithm 2 where $i \notin \mathcal{S}$. Namely, these are the set of oracle queries q_i that Adv answers with uniformly random values in Real. We now define the following two events:

- Let W be the event that $\mathsf{Verify}(\mathsf{pp}, x, y, \pi) = 1$ when (y, π) is the output of the adversary (i.e., W is the event that the adversary wins and the experiment outputs 1).
- Let B be the "bad" event where $(Q_V \cup Q_S) \cap Q_{\mathsf{Adv}} \neq \varnothing$. Namely, this is the event that adversary makes up an answer to a query that is asked either by the setup algorithm or the verification algorithm.

With these definitions, the following claim trivially holds in the ideal experiment (by perfect completeness of the underlying proof of sequential work[5]), as all of the oracle queries q_i are computed using the real oracle $\mathcal{O}(q_i)$.

▷ Claim 19. $\Pr_{\mathsf{ideal}}[W] = 1$.

The following lemma states that until event B happens, the two experiments are identical.

▶ **Lemma 20.** $\Pr_{\mathsf{real}}[B] = \Pr_{\mathsf{ideal}}[B]$. *Moreover, conditioned on the event B not happening, the two experiments are identically distributed. In particular, for any event like W, it hold that* $\Pr_{\mathsf{real}}[W \vee B] = \Pr_{\mathsf{ideal}}[W \vee B]$.

Proof. Here, we make a crucial use of the fact that the oracle $\mathcal{O}$ is random. To prove the lemma, we run the two games *in parallel* using the *same* randomness for any query that is asked by any party, step by step. Namely, we start by executing the evaluation algorithm identically as much as possible until event B happens. More formally, we run both experiments by using fresh randomness to answer any new query asked during the execution, and we will *stop* the execution as soon as event B happens. Since until the event B happens both games proceed *identically* (in a perfect sense) and *consistently* according to their own distribution, it means that until event B happens, the two games have the same exact distributions. ◀

We now observe that the probability of the event B is small in the ideal game, and conclude that it is indeed small in both games.

▷ Claim 21. $\Pr_{\mathsf{ideal}}[B] \leq (s + t) \cdot \frac{G}{T}$.

Proof. In this game, the set $\mathcal{S}$ is independent of all other components in the experiment, so we can choose $\mathcal{S}$ *at the end* of the experiment (*after* Q_S and Q_V have been determined). By definition of Q_{Adv}, we have that $|Q_{\mathsf{Adv}}| \leq G$. This means that for any query $q \in Q_S \cup Q_V$ that is also queried by $\mathsf{Eval}(\mathsf{pp}, x)$, the probability that $q \in Q_{\mathsf{Adv}}$ is at most G/T. The claim now follows by a union bound. ◁

The above claims complete the proof of Theorem 15, as we now can conclude that the probability of W in both experiments is "close":

$$\left| \Pr_{\mathsf{real}}[W] - \Pr_{\mathsf{ideal}}[W] \right| \leq \Pr_{\mathsf{ideal}}[B].$$

We already know that $\Pr_{\mathsf{ideal}}[W] = 1$, therefore, we conclude that

$$\Pr_{\mathsf{real}}[W] \geq \Pr_{\mathsf{ideal}}[W] - \Pr_{\mathsf{ideal}}[B] \geq 1 - (s + t) \cdot \frac{G}{T}.$$ ◀

[5] Note that this argument extends also to the setting where we have completeness error γ (i.e., completeness holds with probability $1 - \gamma$ over the choice of the public parameters). This modification introduces a γ loss in the adversary's advantage in the real game.

4 Extensions to the Lower Bounds

In this section, we briefly discuss several extensions of our lower bounds on perfectly unique VDFs and tight proofs of sequential work by extending the proofs of Theorems 10 and 15.

4.1 Handling Expensive Setup Phase

Our lower bounds in Theorems 10 and 15 construct an adversary whose running time depends on both t (the verification time) as well as s (the setup time). When these bounds t, s are only $\text{poly}(\lambda, \log T)$, the running time of our attacker is much smaller compared to T, and thus, breaks the sequentiality of the scheme. However, one might argue that since the setup is executed only once, it is reasonable to consider a scenario where s is potentially as large as T. In this case, *both* of our lower bounds in Theorems 10 and 15 become meaningless.

Case of expensive *public* setup. If the setup algorithm is public coin (i.e, the randomness to Setup is publicly known), then both Theorems 10 and 15 directly extend to the setting where $s = \text{poly}(\lambda, T)$. Specifically, in this setting, the setup queries can be discovered publicly (i.e., by emulating an execution of Setup). Thus, the adversary can learn the oracle's values on the set of queries Q_S made by Setup. In the online phase, the adversary carries out its attack by *incorporating* the knowledge of Q_S when answering oracle queries. In other words, if during the emulation of the Eval algorithm, the adversary encounters a query $q \in Q_S$, it will use the known answer (saved as part of the state $\mathsf{st}_{\mathsf{Adv}}$) instead of asking it from the oracle or guessing its answer.

Case of expensive *private* setup for Theorem 15. When the setup algorithm uses *private* randomness, the above argument for extending Theorems 10 and 15 no longer applies. Nonetheless, we can still show how the proof of Theorem 15 (for ruling our tight proofs of sequential work) can be extended to this setting as well.

▶ **Theorem 22** (Attacking Tight Proofs of Sequential Work with Expensive Setup in the ROM)**.** *Suppose* $\Pi_{\mathsf{PSW}} = (\mathsf{Setup}, \mathsf{Eval}, \mathsf{Verify})$ *is a publicly-verifiable proof of sequential work in the ROM in which (for a concrete choice of* λ, T*),* Setup *runs in time* s*,* Eval *runs in time* T*,* Verify *runs in time* t*, and completeness error is at most* γ *(see Definition 2). Then, for any* $\varepsilon < 1$ *and* $1 < G < T$*, there is an adversary* $\mathsf{Adv} = (\mathsf{Adv}_0, \mathsf{Adv}_1)$ *where* Adv_0 *runs in time* $\text{poly}(s, T, t, 1/\varepsilon)$ *and* Adv_1 *makes at most* $T - G$ *queries (and run in total time* $\text{poly}(T)$*) and breaks sequentiality (Definition 4) with probability at least* $1 - \varepsilon - t \cdot G/T - \gamma$*.*

Before proving Theorem 22, we first derive a corollary, formally stating the range of tight sequentiality for which we rule out PoSWs in the ROM.

▶ **Corollary 23** (Ruling Out Tight Proofs of Sequential Work with Expensive Setup in the ROM)**.** *Let* λ *be a security parameter and* T *be the time bound parameter. For any choice of* $s = \text{poly}(\lambda, T), t = \text{poly}(\lambda, \log T)$*, and completeness error* $\gamma = \text{negl}(\lambda)$*, there does not exist a proof of sequential work in the ROM with sequentiality* $\sigma = T \cdot (1 - {}^1/_{2t})$ *and completeness error* γ*. In particular, for any constant* $0 < \rho < 1$*, there does not exist a proof of sequential work in the ROM with sequentiality* $\sigma = T - T^\rho$ *and completeness error* γ*.*

Proof. The proof is identical to that of Corollary 16, except here, we use the attack algorithm from Theorem 22 that relies on a preprocessing step with time complexity $\text{poly}(\lambda, T)$. The only difference is that we now also need to make sure $\varepsilon + \gamma < 1/2$, which is easily satisfied whenever $\gamma = \text{negl}(\lambda)$. For example, take $\varepsilon = 1/3$. ◀

Algorithm 3 The adversary Adv_0 that precomputes the ε-heavy queries of the Setup algorithm.

```
1. Initialize a set $Q \leftarrow \varnothing$.
2. Repeat the following procedure $k = s/\varepsilon = \mathrm{poly}(\lambda, T)$ times:
   a. Sample $x \overset{\$}{\leftarrow} \mathcal{X}$.
   b. Compute $(y, \pi) \leftarrow \mathsf{Eval}(\mathsf{pp}, x)$. Whenever Eval makes a query $q$ to the oracle $\mathcal{O}$, add
      the pair $(q, \mathcal{O}(q))$ to $Q$.
3. Output $\mathsf{st}_{\mathsf{Adv}} = Q$.
```

We now give the proof of Theorem 22.

Proof of Theorem 22. Here, Adv_0 will perform a precomputation and compute the so-called "ε-heavy" queries [3,17] of the setup algorithm. More precisely, we aim to find queries that are asked by the setup and have noticeable chance of being queried again during the evaluation. In particular, Algorithm Adv_0 outputs a set Q consisting of input-output pairs of the oracle $\mathcal{O}$. The online algorithm Adv_1 then follows Algorithm 2 from the proof of Theorem 15, except whenever Eval makes a query q where $q \in Q$, Adv_1 replies with the precomputed value of $\mathcal{O}(q)$ in Q. We now describe Adv_0:

We define Adv_1 as in Algorithm 2, except in Step 3, if the query q_i is contained in $\mathsf{st}_{\mathsf{Adv}} = Q$, then the adversary always replies with the precomputed value of $\mathcal{O}(q_i)$ in Q. Otherwise, it uses the same procedure as in Algorithm 3. The rest of the analysis proceeds similar to that in the proof of Theorem 15. Namely, let Q_S denote the set of oracle queries made by Setup and Q_V be the set of queries made by Verify in the real/ideal experiments. We define the events W and B exactly as in the proof of Theorem 15. We now show an analog of Claim 21:

▷ Claim 24. $\Pr_{\mathsf{ideal}}[B] \leq \varepsilon + t \cdot \frac{G}{T}$.

Proof. Recall that the event B occurs if $(Q_V \cup Q_S) \cap Q_{\mathsf{Adv}} \neq \varnothing$. We consider the following two setting. (All probabilities are stated in the ideal experiment.)

- Consider the probability that $Q_V \cap Q_{\mathsf{Adv}} \neq \varnothing$. By the same argument as in the proof of Claim 21, this event occurs with probability at most $|Q_V| \cdot G/T \leq t \cdot G/T$ over the randomness of $\mathcal{S}$.
- Consider the probability that $Q_S \cap Q_{\mathsf{Adv}} \neq \varnothing$. Take any query $q \in Q_S$ and consider the event H_q that $q \in Q_{\mathsf{Adv}}$ and $q \notin \mathsf{st}_{\mathsf{Adv}}$. By construction of Algorithm 3, for H_q to happen, the first k iterations of Adv_0 should *not* query q during Eval, and yet q *is* queried in the next (actual) execution of Eval. However, it is easy to show that for any Bernoulli variable (of arbitrary probability α) the probability of missing it k times and hitting it on the $(k+1)^{\text{st}}$ iteration is at most $1/k$. Since Setup makes at most s queries, $|Q_S| \leq s$, so by a union bound,

$$\Pr[Q_S \cap Q_{\mathsf{Adv}} \neq \varnothing] \leq \frac{s}{k} = \varepsilon.$$

The claim now follows by a union bound. ◁

The rest of the proof of Theorem 22 proceeds identically to the proof of Theorem 15 and noting that the probability of W in the ideal game is $1 - \gamma$. ◀

4.2 Extending to the Random Permutation Oracle Model

Random oracles can be used to instantiate all symmetric primitives (including the ideal cipher model [9,15]) with one exception: the random permutation oracle R that implements a random permutation on $\{0,1\}^n$ for all $n \in \mathbb{N}$. Indeed there are impossibility results showing that such a construction does not exist [23,27]. We can extend the proof of Theorem 15 to rule out constructions of tight proofs of sequential work in the random permutation oracle as well by developing a preprocessing attack and then using standard techniques based on the fact that a random permutation oracle and a random oracle over a large domain is statistically indistinguishable to any query-bounded algorithm [17].

Case of large input domains. More formally, let n be such that $(T+s+t)^2/2^n \leq \varepsilon$. Then, first suppose the only domain used by the three algorithms $(\mathsf{Setup}, \mathsf{Eval}, \mathsf{Verify})$ is $\{0,1\}^n$. In this case, the probability that a random oracle used by these three algorithms $\mathsf{Setup}, \mathsf{Eval}, \mathsf{Verify}$ has *any* collision during the course of their execution is at most $(T+s+t)^2/2^n \leq \varepsilon$. However, whenever there are no collisions, there is no way to distinguish between random permutations or random oracles. Therefore, our attack in Theorem 15 would automatically work (as is) up to a loss of ε in the success probability. This argument also works if the three algorithms $(\mathsf{Setup}, \mathsf{Eval}, \mathsf{Verify})$ ask their queries from input domains with *different* size as long as for any domain $\{0,1\}^n$ that they query, n satisfies $(T+s+t)^2/2^n \leq \varepsilon$.

General case. The above argument fails when any of the algorithms $(\mathsf{Setup}, \mathsf{Eval}, \mathsf{Verify})$ ask their queries from any domain $\{0,1\}^n$ where n is small (and as such, $(T+s+t)^2/2^n > \varepsilon$). However, the *total* number of queries over *all* such domains is at most

$$\tau = |\{0,1\}^1| + |\{0,1\}^2| + \cdots + |\{0,1\}^n|,$$

where $(T+s+t)^2/2^n = \varepsilon$, which means

$$\tau = 2 + 4 + \cdots + \frac{(T+s+t)^2}{\varepsilon} < \frac{2(T+s+t)^2}{\varepsilon} = \mathrm{poly}(\lambda, T).$$

Therefore, a preprocessing adversary Adv_0 can ask all of these $\tau = \mathrm{poly}(\lambda, T)$ queries from the real oracle $\mathcal{O}$ and send them together with their answers to the online adversary Adv_1 who will then use the answers to these queries whenever needed without asking them from $\mathcal{O}$ or guessing their answers. In this case, the analysis of our attack is identical to the aforementioned case with inputs drawn from a large domain.

5 Conclusion and Open Questions

In this work, we initiated a formal study of the assumptions behind VDFs and provided new lower bounds on basing VDFs with perfect uniqueness in the random oracle model as well as stronger lower bounds in the tight security regime in which the sequentiality guarantees a very close running time to the honest execution. The second lower bound applies not only to VDFs but also to relaxations of it such as sequential proofs of work. While our first lower bound captures existing notions [1], they do not extend to the full range of VDF notions.

The main open question remaining is whether we can extend our first lower bound to rule out VDFs satisfying *computational uniqueness* in the ROM, and ideally do so allowing negligible completeness error as well. Alternatively, the fact that our current lower bound critically relies on the perfect uniqueness property may suggest new approaches to basing

VDFs on weaker assumptions. Namely, any such approach must either rely on non-black-box techniques or leverage imperfect soundness in a critical manner. Both of these possibilities represent intriguing avenues for further research.

References

1 Hamza Abusalah, Chethan Kamath, Karen Klein, Krzysztof Pietrzak, and Michael Walter. Reversible proofs of sequential work. In *EUROCRYPT*, pages 277–291, 2019.

2 Joël Alwen and Vladimir Serbinenko. High parallel complexity graphs and memory-hard functions. In *STOC*, pages 595–603, 2015.

3 Boaz Barak and Mohammad Mahmoody. Merkle's key agreement protocol is optimal: An $o(n^2)$ attack on any key agreement from random oracles. *J. Cryptology*, 30(3):699–734, 2017.

4 Ingrid Biehl, Johannes A. Buchmann, Safuat Hamdy, and Andreas Meyer. A signature scheme based on the intractability of computing roots. *Des. Codes Cryptogr.*, 25(3):223–236, 2002.

5 Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In *CRYPTO*, pages 757–788, 2018.

6 Zvika Brakerski, Jonathan Katz, Gil Segev, and Arkady Yerukhimovich. Limits on the power of zero-knowledge proofs in cryptographic constructions. In *TCC*, pages 559–578, 2011.

7 Jin-yi Cai, Richard J. Lipton, Robert Sedgewick, and Andrew Chi-Chih Yao. Towards uncheatable benchmarks. In *Structure in Complexity Theory Conference*, pages 2–11, 1993.

8 Bram Cohen and Krzysztof Pietrzak. Simple proofs of sequential work. In *EUROCRYPT*, pages 451–467, 2018.

9 Jean-Sébastien Coron, Jacques Patarin, and Yannick Seurin. The random oracle model and the ideal cipher model are equivalent. In *CRYPTO*, pages 1–20, 2008.

10 Nico Döttling, Sanjam Garg, Giulio Malavolta, and Prashant Nalini Vasudevan. Tight verifiable delay functions. *IACR Cryptology ePrint Archive*, 2019:659, 2019.

11 Nico Döttling, Russell W. F. Lai, and Giulio Malavolta. Incremental proofs of sequential work. In *EUROCRYPT*, pages 292–323, 2019.

12 Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *CRYPTO*, pages 139–147, 1992.

13 Naomi Ephraim, Cody Freitag, Ilan Komargodski, and Rafael Pass. Continuous verifiable delay functions. *IACR Cryptology ePrint Archive*, 2019:619, 2019.

14 Luca De Feo, Simon Masson, Christophe Petit, and Antonio Sanso. Verifiable delay functions from supersingular isogenies and pairings. In *ASIACRYPT*, pages 248–277, 2019.

15 Thomas Holenstein, Robin Künzler, and Stefano Tessaro. The equivalence of the random oracle model and the ideal cipher model, revisited. In *STOC*, pages 89–98, 2011.

16 Matt Howard and Bram Cohen. Chia network announces 2nd VDF competition with $100,000 in total prize money, 2019.

17 Russell Impagliazzo and Steven Rudich. Limits on the provable consequences of one-way permutations. In *STOC*, pages 44–61, 1989.

18 Eleftherios Kokoris-Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. In *USENIX Security Symposium*, pages 279–296, 2016.

19 Esteban Landerreche, Marc Stevens, and Christian Schaffner. Non-interactive cryptographic timestamping based on verifiable delay functions. *IACR Cryptology ePrint Archive*, 2019:197, 2019.

20 Arjen K. Lenstra and Benjamin Wesolowski. A random zoo: sloth, unicorn, and trx. *IACR Cryptology ePrint Archive*, 2015:366, 2015.

21 Mohammad Mahmoody, Tal Moran, and Salil P. Vadhan. Time-lock puzzles in the random oracle model. In *CRYPTO*, pages 39–50, 2011.

22 Mohammad Mahmoody, Tal Moran, and Salil P. Vadhan. Publicly verifiable proofs of sequential work. In *ITCS*, pages 373–388, 2013.

23 Takahiro Matsuda and Kanta Matsuura. On black-box separations among injective one-way functions. In *TCC*, pages 597–614, 2011.

24 Krzysztof Pietrzak. Simple verifiable delay functions. In *ITCS*, pages 60:1–60:15, 2019.

25 R. L. Rivest, A. Shamir, and D. A. Wagner. Time-lock puzzles and timed-release crypto, 1996.

26 Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

27 Steven Rudich. *Limits on the Provable Consequences of One-way Functions.* PhD thesis, EECS Department, University of California, Berkeley, 1988. URL: `http://www2.eecs.berkeley.edu/Pubs/TechRpts/1988/6060.html`.

28 Barak Shani. A note on isogeny-based hybrid verifiable delay functions. *IACR Cryptology ePrint Archive*, 2019:205, 2019.

29 Benjamin Wesolowski. Efficient verifiable delay functions. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 379–407. Springer, 2019.

On the Two-Dimensional Knapsack Problem for Convex Polygons

Arturo Merino
Technische Universität Berlin, Germany
merino@math.tu-berlin.de

Andreas Wiese
Universidad de Chile, Santiago, Chile
awiese@dii.uchile.cl

Abstract

We study the two-dimensional geometric knapsack problem for convex polygons. Given a set of weighted convex polygons and a square knapsack, the goal is to select the most profitable subset of the given polygons that fits non-overlappingly into the knapsack. We allow to rotate the polygons by arbitrary angles. We present a quasi-polynomial time $O(1)$-approximation algorithm for the general case and a polynomial time $O(1)$-approximation algorithm if all input polygons are triangles, both assuming polynomially bounded integral input data. Also, we give a quasi-polynomial time algorithm that computes a solution of optimal weight under resource augmentation, i.e., we allow to increase the size of the knapsack by a factor of $1 + \delta$ for some $\delta > 0$ but compare ourselves with the optimal solution for the original knapsack. To the best of our knowledge, these are the first results for two-dimensional geometric knapsack in which the input objects are more general than axis-parallel rectangles or circles and in which the input polygons can be rotated by arbitrary angles.

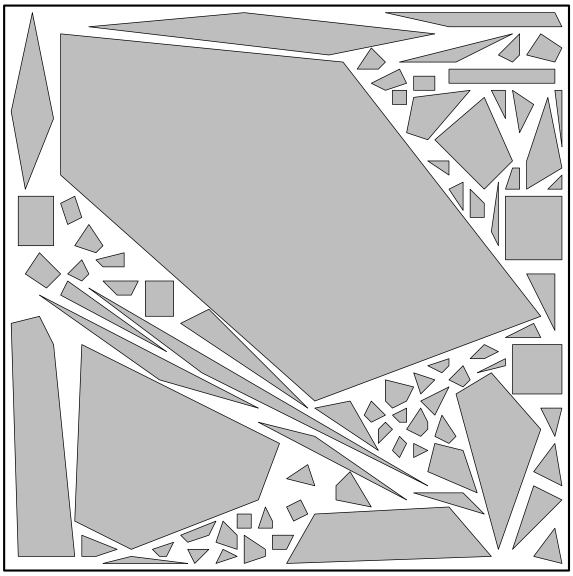
2012 ACM Subject Classification Theory of computation → Packing and covering problems

Keywords and phrases Approximation algorithms, geometric knapsack problem, polygons, rotation

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.84

Category Track A: Algorithms, Complexity and Games

Funding *Arturo Merino*: Partially supported by DFG Project 413902284 and ANID Becas Chile 2019-72200522.
Andreas Wiese: Partially supported by FONDECYT Regular grant 1170223.



47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 84; pp. 84:1–84:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

In the two-dimensional geometric knapsack problem (2DKP) we are given a square knapsack $K := [0, N] \times [0, N]$ for some integer N and a set of n convex polygons $\mathcal{P}$ where each polygon $P_i \in \mathcal{P}$ has a weight $w_i > 0$; we write $w(\mathcal{P}') := \sum_{P_i \in \mathcal{P}'} w_i$ for any set $\mathcal{P}' \subseteq \mathcal{P}$. The goal is to select a subset $\mathcal{P}' \subseteq \mathcal{P}$ of maximum total weight $w(\mathcal{P}')$ such that the polygons in $\mathcal{P}'$ fit non-overlapping into K if we translate and rotate them suitably (by arbitrary angles). 2DKP is a natural packing problem, the reader may think of cutting items out of a piece of raw material like metal or wood, cutting cookings out of dough, or, in three dimensions, loading cargo into a ship or a truck. In particular, in these applications the respective items can have various kinds of shapes. Also note that 2DKP is a natural geometric generalization of the classical one-dimensional knapsack problem.

Our understanding of 2DKP highly depends on the type of input objects. If all polygons are axis-parallel squares there is a $(1 + \epsilon)$-approximation with a running time of the form $O_\epsilon(1)n^{O(1)}$ (i.e., an EPTAS) [6], and there can be no FPTAS (unless $\mathsf{P} = \mathsf{NP}$) since the problem is strongly NP-hard [11]. For axis-parallel rectangles there is a polynomial time $(17/9 + \epsilon) < 1.89$-approximation algorithm and a $(3/2 + \epsilon)$-approximation if the items can be rotated by exactly 90 degrees [5]. If the input data is quasi-polynomially bounded there is even a $(1 + \epsilon)$-approximation in quasi-polynomial time [2], with and without the possibility to rotate items by 90 degrees. For circles a $(1 + \epsilon)$-approximation is known under resource augmentation in one dimension if the weight of each circle equals its area [12].

To the best of our knowledge, there is no result known for 2DKP for shapes different than axis-parallel rectangles and circles. Also, there is no result known in which input polygons are allowed to be rotated by angles different than 90 degrees. However, in the applications of 2DKP the items might have shapes that are more complicated than rectangles or circles. Also, it makes sense to allow rotations by arbitrary angles, e.g., when cutting items out of some material. In this paper, we present the first results for 2DKP in these settings.

1.1 Our contribution

We study 2DKP for arbitrary convex polygons, allowing to rotate them by arbitrary angles. Note that due to the latter, it might be that some optimal solution places the vertices of the polygons on irrational coordinates, even if all input numbers are integers. Our first results are a quasi-polynomial time $O(1)$-approximation algorithm for general convex polygons and a polynomial time $O(1)$-approximation algorithm for triangles.

By rotation we can assume for each input polygon that the line segment defining its diameter is horizontal. We identify three different types of polygons for which we employ different strategies for packing them, see Figure 1a). First, we consider the *easy* polygons which are the polygons whose bounding boxes fit into the knapsack without rotation. We pack these polygons such that their bounding boxes do not intersect. Using area arguments and the Steinberg's algorithm [13] we obtain a $O(1)$-approximation for the easy polygons. Then we consider the *medium* polygons which are the polygons whose bounding boxes easily fit into the knapsack if we can rotate them by 45 degrees. We use a special type of packing in which the bounding boxes are rotated by 45 degrees and then stacked on top of each other, see Figure 1b). More precisely, we group the polygons by the widths of their bounding boxes and to each group we assign two rectangular containers in the packing. We compute the essentially optimal solution of this type by solving a generalization of one-dimensional knapsack for each group. Our key structural insight for medium polygons is that such a solution is $O(1)$-approximate. To this end, we prove that in OPT the medium polygons of

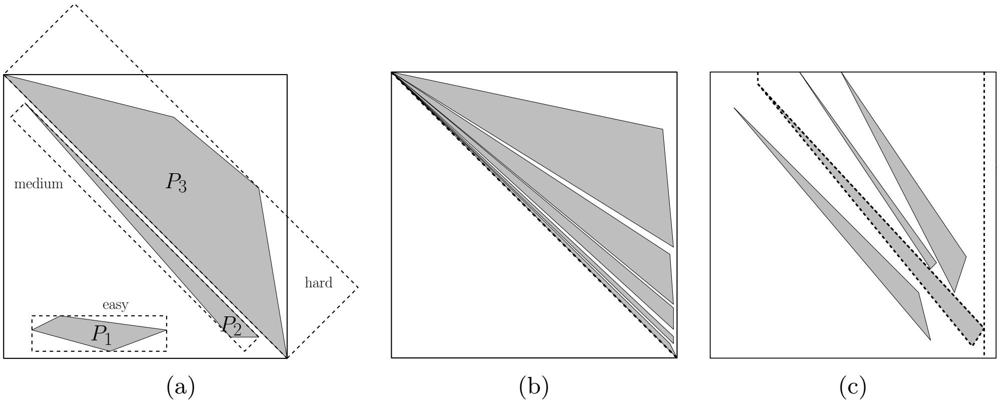


Figure 1 (a) An easy, a medium, and a hard polygon and their bounding boxes (b): Triangles packed in a top-left packing (c) The geometric DP subdivides the knapsack along the dashed lines and then recurses within each resulting area.

each group occupy an area that is by at most a constant factor bigger than the corresponding containers, and that a constant fraction of these polygons fit into the containers. In particular, we show that medium polygons with very wide bounding boxes lie in a very small hexagonical area close to the diagonal of the knapsack. Our routines for easy and medium polygons run in polynomial time.

It remains to pack the *hard* polygons whose bounding boxes just fit into the knapsack or do not fit at all, even under rotation. Note that this does not imply that the polygon itself does not fit. Our key insight is that there can be only $O(\log N)$ such polygons in the optimal solution, at most $O(1)$ from each group. Therefore, we can guess these polygons in quasi-polynomial time, assuming that N is quasi-polynomially bounded. However, in contrast to other packing problems, it is not trivial to check whether a set of given polygons fits into the knapsack since we can rotate them by arbitrary angles and we cannot enumerate all possibilities for the angles. However, we show that by losing a constant factor in the approximation guarantee we can assume that the placement of each hard polygon comes from a precomputable polynomial size set and hence we can guess the placements of the $O(\log N)$ hard polygons in quasi-polynomial time.

▶ **Theorem 1.** *There is a $O(1)$-approximation algorithm for 2DKP with a running time of $(nN)^{(\log nN)^{O(1)}}$.*

If all hard polygons are triangles we present even a *polynomial* time $O(1)$-approximation algorithm. We split the triangles in OPT into two types, for one type we show that a constant fraction of it can be packed in what we call *top-left-packings*, see Figure 1b). In these packings, the triangles are sorted by the lengths of their longest edges and placed on top of each other in a triangular area. We devise a dynamic program (DP) that essentially computes the most profitable top-left-packing. For proving that this yields a $O(1)$-approximation, we need some careful arguments for rearranging a subset of the triangles with large weight to obtain a packing that our DP can compute. We observe that essentially all hard polygons in OPT must intersect the horizontal line that contains the mid-point of the knapsack. Our key insight is that if we pack a triangle in a top-left-packing then it intersects this line to a similar extent as in OPT. Then we derive a sufficient condition when a set of triangles fits in a top-left-packing, based on by how much they overlap this line.

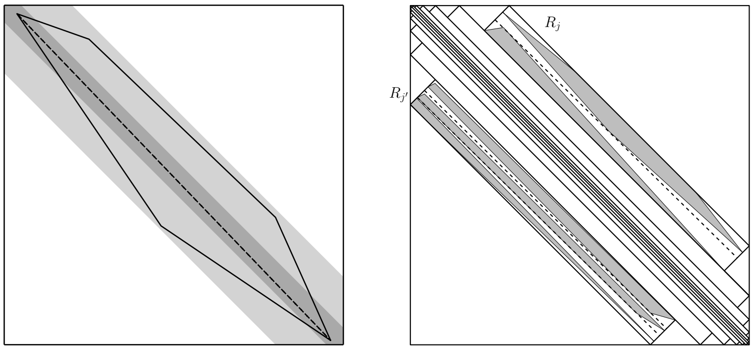


Figure 2 Left: Assume that the polygon (black line segments) is a medium polygon contained in the set $\mathcal{P}_j$. Then the diagonal (dashed) line segment must lie into the dark-gray area and the whole polygon must be contained in the light-gray area. Right: The containers for the medium polygons of the different groups. Within each container, the polygons are stacked on top of each other such that their respective bounding boxes do not intersect.

For the other type of triangles we use a geometric dynamic program. In this DP we recursively subdivide the knapsack into subareas in which we search for the optimal solution recursively, see Figure 1c). In the process we guess the placements of some triangles from OPT. Again, by losing a constant factor we can assume that for each triangle in OPT there are only a polynomial number of possible placements. By exploiting structural properties of this type of triangles we ensure that the number of needed DP-cells is bounded by a polynomial. A key difficulty is that we sometimes split the knapsack into two parts on which we recurse independently. Then we need to ensure that we do not select some (possibly high weight) triangle in both parts. To this end, we globally select at most one triangle from each of the $O(\log N)$ groups (losing a constant factor) and when we recurse, we guess for each subproblem from which of the $O(\log N)$ groups it contains a triangle in OPT. This yields only $2^{O(\log N)} = N^{O(1)}$ guesses.

▶ **Theorem 2.** *There is a $O(1)$-approximation algorithm for 2DKP with a running time of $(nN)^{O(1)}$ if all input polygons are triangles.*

Then we study the setting of resource augmentation, i.e., we compute a solution that fits into a larger knapsack of size $(1+\delta)N \times (1+\delta)N$ for some constant $\delta > 0$ and compare ourselves with a solution that fits into the original knapsack of size $N \times N$. We show that then the optimal solution can contain only *constantly* many hard polygons and hence we can guess them in *polynomial* time.

▶ **Theorem 3.** *There is a $O(1)$-approximation algorithm for 2DKP under $(1+\delta)$-resource augmentation with a running time of $n^{O_\delta(1)}$.*

Finally, we present a quasi-polynomial time algorithm that computes a solution of weight at least $w(\mathrm{OPT})$ (i.e., we do not lose any factor in the approximation guarantee) that is feasible under resource augmentation. This algorithm does not use the above classification of polygons into easy, medium, and hard polygons. Instead, we prove that if we can increase the size of the knapsack slightly we can ensure that for the input polygons there are only $(\log n)^{O_\delta(1)}$ different shapes by enlarging the polygons suitably. Also, we show that we need to allow only a polynomial number of possible placements and rotations for each input polygon, *without* sacrificing any polygons from OPT. Then we use a technique from [1] implying that there is a balanced separator for the polygons in OPT with only $(\log n)^{O_\delta(1)}$ edges and which intersects polygons from OPT with only very small area. We guess the separator, guess how many polygons of each type are placed inside and outside the separator, and then recurse on each of these parts. Some polygons are intersected by the balanced separator. However, we ensure that they have very small area in total and hence we can place them into the additional space of the knapsack that we gain due to the resource augmentation. This generalizes a result in [2] for axis-parallel rectangles.

▶ **Theorem 4.** *There is an algorithm for 2DKP under* $(1+\delta)$*-resource augmentation with a running time of* $n^{O_\delta(\log n)^{O(1)}}$ *that computes a solution of weight at least* $w(\mathrm{OPT})$*.*

In our approximation algorithms, we focus on a clean exposition of our methodology for obtaining $O(1)$-approximations, rather than on optimizing the actual approximation ratio. Due to space constraints, most proofs and details had to be ommited in this extended abstract.

1.2 Other related work

Prior to the results mentioned above, polynomial time $(2+\epsilon)$-approximation algorithms for 2DKP for axis-parallel rectangles were presented by Jansen and Zhang [10, 9]. For the same setting, a PTAS is known under resource augmentation in one dimension [7] and a polynomial time algorithm computing a solution with optimum weight under resource augmentation in both dimensions [6]. Also, there is a PTAS if the weight of each rectangle equals its area [3]. For squares, Jansen and Solis-Oba presented a PTAS [8].

2 Constant factor approximation algorithms

In this section we present our quasi-polynomial time $O(1)$-approximation algorithm for general convex polygons and our polynomial time $O(1)$-approximation algorithm for triangles., assuming polynomially bounded input data. Our strategy is to partition the input polygons $\mathcal{P}$ into three classes, easy, medium, and hard polygons, and then to devise algorithms for each class separately.

Let $K := [0,N] \times [0,N]$ denote the given knapsack. We assume that each input polygon is described by the coordinates of its vertices which we assume to be integral. First, we rotate each polygon in $\mathcal{P}$ such that its longest diagonal (i.e., the line segment that connects the two vertices of largest distance) is horizontal. For each polygon $P_i \in \mathcal{P}$ denote by $(x_{i,1}, y_{i,1}), ..., (x_{i,k_i}, y_{i,k_i})$ the new coordinates of its vertices. Observe that due to the rotation, the resulting coordinates might not be integral, and possibly not even rational. We will take this into account when we define our algorithms. For each $P_i \in \mathcal{P}$ we define its *bounding box* B_i to be the smallest axis-parallel rectangle that contains P_i. Formally, we define $B_i := [\min_\ell x_{i,\ell}, \max_\ell x_{i,\ell}] \times [\min_\ell y_{i,\ell}, \max_\ell y_{i,\ell}]$. For each polygon P_i let $\ell_i := \max_\ell x_{i,\ell} - \min_\ell x_{i,\ell}$ and $h_i := \max_\ell y_{i,\ell} - \min_\ell y_{i,\ell}$. If necessary we will work with suitable estimates of these values later, considering that they might be irrational and hence we cannot compute them exactly.

We first distinguish the input polygons into *easy*, *medium*, and *hard* polygons. We say that a polygon P_i is *easy* if B_i fits into K without rotation, i.e., such that $\ell_i \leq N$ and $h_i \leq N$. Denote by $\mathcal{P}_E \subseteq \mathcal{P}$ the set of easy polygons. Note that the bounding box of a polygon in $\mathcal{P} \setminus \mathcal{P}_E$ might still fit into K if we rotate it suitably. Intuitively, we define the medium polygons to be the polygons P_i whose bounding box B_i fits into K with quite some slack if we rotate B_i properly and the hard polygons are the remaining polygons (in particular those polygons whose bounding box does not fit at all into K).

Formally, for each polygon $P_i \in \mathcal{P}$ we define $h'_i := \sqrt{2}N - \ell_i$. The intuition for h'_i is that a rectangle of width ℓ_i and height h'_i is the highest rectangle of width ℓ_i that still fits into K.

▶ **Lemma 5.** *Let $P_i \in \mathcal{P}$. A rectangle of width ℓ_i and height h'_i fits into K (if we rotate it by 45°) but a rectangle of width ℓ_i and of height larger than h'_i does not fit into K.*

Hence, if h_i is much smaller than h'_i then B_i fits into K with quite some slack. Therefore, we define that a polygon $P_i \in \mathcal{P} \setminus \mathcal{P}_E$ is *medium* if $h_i \leq h'_i/8$ and *hard* otherwise. Denote by $\mathcal{P}_M \subseteq \mathcal{P}$ and $\mathcal{P}_H \subseteq \mathcal{P}$ the medium and hard polygons, respectively. We will present $O(1)$-approximation algorithms for each of the sets $\mathcal{P}_E, \mathcal{P}_M, \mathcal{P}_H$ separately. The best of the computed sets will then yield a $O(1)$-approximation overall.

For the easy polygons, we construct a polynomial time $O(1)$-approximation algorithm in which we select polygons such that we can pack their bounding boxes as non-overlapping rectangles using Steinberg's algorithm [4], see Section 2.1. The approximation ratio follows from area arguments.

▶ **Lemma 6.** *There is a polynomial time algorithm that computes a solution $\mathcal{P}'_E \subseteq \mathcal{P}_E$ with $w(\mathrm{OPT} \cap \mathcal{P}_E) \leq O(w(\mathcal{P}'_E))$.*

For the medium polygons, we obtain a $O(1)$-approximation algorithm using a different packing strategy, see Section 2.2.

▶ **Lemma 7.** *There is an algorithm with a running time of $n^{O(1)}$ that computes a solution $\mathcal{P}'_M \subseteq \mathcal{P}_M$ with $w(\mathrm{OPT} \cap \mathcal{P}_M) \leq O(w(\mathcal{P}'_M))$.*

The most difficult polygons are the hard polygons. First, we show that in quasi-polynomial time we can obtain a $O(1)$-approximation for them, see Section 2.3.

▶ **Lemma 8.** *There is an algorithm with a running time of $(nN)^{(\log nN)^{O(1)}}$ that computes a solution $\mathcal{P}'_H \subseteq \mathcal{P}_H$ with $w(\mathrm{OPT} \cap \mathcal{P}_H) \leq O(w(\mathcal{P}'_H))$.*

Combining Lemmas 6, 7, and 8 yields Theorem 1. If all polygons are triangles, we obtain a $O(1)$-approximation even in polynomial time. The following lemma is proved in Section 2.3.1 and together with Lemmas 6 and 7 implies Theorem 2.

▶ **Lemma 9.** *If all input polygons are triangles, then there is an algorithm with a running time of $(nN)^{O(1)}$ that computes a solution $\mathcal{P}'_H \subseteq \mathcal{P}_H$ with $w(\mathrm{OPT} \cap \mathcal{P}_H) \leq O(w(\mathcal{P}'_H))$.*

Orthogonal to the characterization into easy, medium and hard polygons, we subdivide the polygons in $\mathcal{P}$ further into classes according to the respective values ℓ_i. More precisely, we do this according to their difference between ℓ_i and the diameter of K, i.e., $\sqrt{2}N$. Formally, for each $j \in \mathbb{Z}$ we define $\mathcal{P}_j := \{P_i \in \mathcal{P} | \ell_i \in [\sqrt{2}N - 2^j, \sqrt{2}N - 2^{j-1})\}$ and additionally $\mathcal{P}_{-\infty} := \{P_i \in \mathcal{P} | \ell_i = \sqrt{2}N\}$. Note that for each polygon $P_i \in \mathcal{P}$ we can compute the group $\mathcal{P}_j$ even though ℓ_i might be irrational.

2.1 Easy polygons

We present a $O(1)$-approximation algorithm for the polygons in $\mathcal{P}_E$. First, we show that the area of each polygon is at least half of the area of its bounding box. We will use this later for defining lower bounds using area arguments.

▶ **Lemma 10.** *For each $P_i \in \mathcal{P}$ it holds that* $\mathrm{area}(P_i) \geq \frac{1}{2}\mathrm{area}(B_i)$.

On the other hand, it is known that we can pack any set of axis-parallel rectangles into K, as long as their total area is at most $\mathrm{area}(K)/2$ and each single rectangle fits into K.

▶ **Theorem 11** ([13]). *Let $r_1, ..., r_k$ be a set of axis-parallel rectangles such that $\sum_{i=1}^{k} \text{area}(r_i) \leq \text{area}(K)/2$ and each individual rectangle r_i fits into K. Then there is a polynomial time algorithm that packs $r_1, ..., r_k$ into K.*

We first compute (essentially) the most profitable set of polygons from $\mathcal{P}_E$ whose total area is at most $\text{area}(K)$ via a reduction to one-dimensional knapsack.

▶ **Lemma 12.** *In time $(\frac{n}{\epsilon})^{O(1)}$ we can compute a set of polygons $\mathcal{P}' \subseteq \mathcal{P}_E$ such that $w(\mathcal{P}') \geq (1-\epsilon)w(\text{OPT} \cap \mathcal{P}_E)$ and $\sum_{P_i \in \mathcal{P}_E} \text{area}(P_i) \leq \text{area}(K)$.*

The idea is now to partition $\mathcal{P}'$ into at most 7 sets $\mathcal{P}'_1, ..., \mathcal{P}'_7$. Hence, one of these sets must contain at least a profit of $w(\mathcal{P}')/7$. We define this partition such that each set $\mathcal{P}'_j$ contains only one polygon or its polygons have a total area of at most $\text{area}(K)/4$.

▶ **Lemma 13.** *Given a set $\mathcal{P}' \subseteq \mathcal{P}_E$ with $\sum\limits_{P_i \in \mathcal{P}_E} \text{area}(P_i) \leq \text{area}(K)$. In polynomial time we can compute a set $\mathcal{P}'' \subseteq \mathcal{P}'$ with $w(\mathcal{P}'') \geq \frac{1}{7}w(\mathcal{P}')$ and additionally $\sum\limits_{P_i \in \mathcal{P}''} \text{area}(P_i) \leq \text{area}(K)/4$ or $|\mathcal{P}''| = 1$.*

If $|\mathcal{P}''| = 1$ we simply pack the single polygon in $\mathcal{P}''$ into the knapsack. Otherwise, using Lemmas 10 and 12 and Theorem 11 we know that we can pack the bounding boxes of the polygons in $\mathcal{P}''$ into K. Note that their heights and widths might be irrational. Therefore, we slightly increase them such that these values become rational, before applying Theorem 11 to compute the actual packing. If as a result the total area of the bounding boxes exceeds $\text{area}(K)/2$ we partition them into two sets where each set satisfies that the total area of the bounding boxes is at most $\text{area}(K)/2$ or it contains only one polygon and we keep the more profitable of these two sets (hence losing a factor of 2 in the approximation ratio). This yields a $O(1)$-approximation algorithm for the easy polygons and thus proves Lemma 6.

2.2 Medium polygons

We describe a $O(1)$-approximation algorithm for the polygons in $\mathcal{P}_M$. In its solution, for each $j \in \mathbb{Z}$ we will define two rectangular containers R_j, R'_j for polygons in $\mathcal{P}_M \cap \mathcal{P}_j$, each of them having width $\sqrt{2}N - 2^{j-1}$ and height 2^{j-3}, see Figures 2. Let $\mathcal{R} := \cup_j \{R_j, R'_j\}$. First, we show that we can pack all containers in $\mathcal{R}$ into K (if we rotate them by 45°).

▶ **Lemma 14.** *The rectangles in $\mathcal{R}$ can be packed non-overlappingly into K.*

For each $j \in \mathbb{Z}$ we will compute a set of polygons $\mathcal{P}'_j \subseteq \mathcal{P}_M \cap \mathcal{P}_j$ of large weight. Within each container R_j, R'_j we will stack the bounding boxes of the polygons in $\mathcal{P}'_j$ on top of each other and then place the polygons in $\mathcal{P}'_j$ in their respective bounding boxes, see Figure 2. In particular, a set of items $\mathcal{P}''_j \subseteq \mathcal{P}_j$ fits into R_j (or R'_j) using this strategy if and only if $h(\mathcal{P}''_j) := \sum_{P_i \in \mathcal{P}''_j} h_i \leq 2^{j-3}$. Observe that for a polygon $P_i \in \mathcal{P}_j$ with $P_i \in \mathcal{P}_H$ it is not necessarily true that $h_i \leq 2^{j-3}$ and hence for hard polygons this strategy is not suitable. We compute the essentially most profitable set of items $\mathcal{P}'_j$ that fits into R_j and R'_j with the above strategy. For this, we need to solve a variation of one-dimensional knapsack with two knapsacks (instead of one) that represent R_j and R'_j. The value h_i for a polygon P_i might be irrational, therefore we work with a $(1+\epsilon)$-estimate of h_i instead. This costs only a factor $O(1)$ in the approximation guarantee.

▶ **Lemma 15.** *Let $\epsilon > 0$. For each $j \in \mathbb{Z}$ there is an algorithm with a running time of $n^{O(\frac{1}{\epsilon})}$ that computes two disjoint sets $\mathcal{P}'_{j,1}, \mathcal{P}'_{j,2} \subseteq \mathcal{P}_j \cap \mathcal{P}_M$ such that $h(\mathcal{P}'_{j,1}) \leq 2^{j-3}$ and $h(\mathcal{P}'_{j,2}) \leq 2^{j-3}$ and $w(\mathcal{P}^*{}_{j,1} \cup \mathcal{P}^*{}_{j,2}) \leq O(w(\mathcal{P}'_{j,1} \cup \mathcal{P}'_{j,2}))$ for any disjoint sets $\mathcal{P}^*{}_{j,1}, \mathcal{P}^*{}_{j,2} \subseteq \mathcal{P}_j \cap \mathcal{P}_M$ such that $h(\mathcal{P}^*_{j,1}) \leq 2^{j-3}$ and $h(\mathcal{P}^*_{j,2}) \leq 2^{j-3}$.*

For each $j \in \mathbb{Z}$ with $\mathcal{P}_j \cap \mathcal{P}_M \neq \emptyset$ we apply Lemma 15 and obtain sets $\mathcal{P}'_{j,1}, \mathcal{P}'_{j,2}$. We pack $\mathcal{P}'_{j,1}$ into R_j and $\mathcal{P}'_{j,2}$ into R'_j, using that $h(\mathcal{P}'_{j,1}) \leq h(R_j)$ and $h(\mathcal{P}'_{j,2}) \leq h(R'_j)$. Then we pack all containers R_j, R'_j for each $j \in \mathbb{Z}$ into K, using Lemma 14.

Let $\mathcal{P}'_M := \bigcup_j \mathcal{P}'_{j,1} \cup \mathcal{P}'_{j,2}$ denote the selected polygons. We want to show that $\mathcal{P}'_M$ has large weight; more precisely we want to show that $w(\mathrm{OPT} \cap \mathcal{P}_M) \leq O(w(\mathcal{P}'_M))$. First, we show that for each $j \in \mathbb{Z}$ the polygons in $\mathcal{P}_j \cap \mathcal{P}_M \cap \mathrm{OPT}$ have bounded area. To this end, we show that they are contained inside a certain (irregular) hexagon (see Figure 2) which has small area if the polygons $P_i \in \mathcal{P}_j$ are wide, i.e., if ℓ_i is close to $\sqrt{2}N$. The reason is that then P_i must be placed close to the diagonal of the knapsack and on the other hand h_i is relatively small (since P_i is medium), which implies that all of P_i lies close to the diagonal of the knapsack. For any object $O \subseteq \mathbb{R}^2$ we define area(O) to be its area.

▶ **Lemma 16.** *For each j it holds that* $\mathrm{area}(\mathcal{P}_j \cap \mathcal{P}_M) \leq O(\mathrm{area}(R_j \cup R'_j))$*.*

Using this, we can partition $\mathcal{P}_j \cap \mathcal{P}_M \cap \mathrm{OPT}$ into at most $O(1)$ subsets such that for each subset $\mathcal{P}'$ it holds that $h(\mathcal{P}') \leq 2^{j-3}$ and hence $\mathcal{P}'$ fits into R_j (and R'_j) using our packing strategy above. Here we use that each medium polygon $P_i \in \mathcal{P}_j$ satisfies that $h_i \leq 2^{j-3}$.

▶ **Lemma 17.** *For each $j \in \mathbb{Z}$ there are disjoint set $\mathcal{P}^*{}_{j,1}, \mathcal{P}^*{}_{j,2} \subseteq \mathcal{P}_j \cap \mathcal{P}_M \cap \mathrm{OPT}$ with $w(\mathcal{P}_j \cap \mathcal{P}_M \cap \mathrm{OPT}) \leq O(w(\mathcal{P}^*{}_{j,1} \cup \mathcal{P}^*{}_{j,2}))$ such that $h(\mathcal{P}^*_{j,1}) \leq 2^{j-3}$ and $h(\mathcal{P}^*_{j,2}) \leq 2^{j-3}$.*

By combining Lemmas 14, 15 and 17 we obtain the proof of Lemma 7.

2.3 Hard polygons

We first show that for each class $\mathcal{P}_j$ there are at most a constant number of polygons from $\mathcal{P}_j \cap \mathcal{P}_H$ in OPT, and that for only $O(\log N)$ classes $\mathcal{P}_j$ it holds that $\mathcal{P}_j \cap \mathcal{P}_H \neq \emptyset$.

▶ **Lemma 18.** *For each $j \in \mathbb{Z}$ it holds that $|\mathcal{P}_j \cap \mathcal{P}_H \cap \mathrm{OPT}| \leq O(1)$. Also, if $\mathcal{P}_j \cap \mathcal{P}_H \neq \emptyset$ then $j \in \{j_{\min}, ..., j_{\max}\}$ with $j_{\min} := -\lceil \log N \rceil$ and $j_{\max} := 1 + \lceil \log((\sqrt{2}-1)N) \rceil$.*

We describe now a quasi-polynomial time algorithm for hard polygons, i.e., we want to prove Lemma 8. Lemma 18 implies that $|\mathcal{P}_H \cap \mathrm{OPT}| \leq O(\log N)$. Therefore, we can enumerate all possibilities for $\mathcal{P}_H \cap \mathrm{OPT}$ in time $n^{O(\log N)}$. For each for each enumerated set $\mathcal{P}'_H \subseteq \mathcal{P}_H$ we need to check whether it fits into K. We cannot try all possibilities for placing $\mathcal{P}'_H$ into K since we are allowed tof rotate the polygons in $\mathcal{P}'_H$ by arbitrary angles. To this end, we show that there is a subset of $\mathcal{P}_H \cap \mathrm{OPT}$ of large weight which contains only a single polygon or it does not use the complete space of the knapsack but leaves some empty space. We use this empty space to move the polygons slightly and rotate them such that each of them is placed in one out of $(nN)^{O(1)}$ different positions that we can compute beforehand. Hence, we can guess all positions of these polygons in time $(nN)^{O(\log N)}$. We define that a *placement* of a polygon $P_i \in \mathcal{P}$ inside K is a polygon $\tilde{P}_i$ such that $d + \mathrm{rot}_\alpha(P_i) = \tilde{P}_i \subseteq K$ where $d \in \mathbb{R}^2$ and $\mathrm{rot}_\alpha(P_i)$ is the polygon that we obtain when we rotate P_i by an angle α clockwise around its first vertex.

▶ **Lemma 19.** *For each polygon $P_i \in \mathcal{P}_H$ we can compute a set of $(nN)^{O(1)}$ possible placements $\mathcal{L}_i$ in time $(nN)^{O(1)}$ such that there exists a set $\mathcal{P}'_H \subseteq \mathcal{P}_H \cap \mathrm{OPT}$ with $w(\mathcal{P}_H \cap \mathrm{OPT}) \leq O(w(\mathcal{P}'_H))$ which can be packed into K such that each polygon P_i is packed according to a placement in $\mathcal{L}_i$.*

This yields the proof of Lemma 8.

2.3.1 Hard triangles

In this section we present a $O(1)$-approximation algorithm in *polynomial* time for hard polygons assuming that they are all triangles, i.e., we prove Lemma 9. Slightly abusing notation, denote by OPT the set $\mathcal{P}'_H$ obtained by applying Lemma 19. We distinguish the triangles in OPT into two types: *edge-facing* triangles and *corner-facing* triangles. Let $P_i \in \mathrm{OPT} \cap \mathcal{P}_H$, let e_1, e_2 denote the two longest edges of P_i, and let v_i^* the vertex of P_i adjacent to e_1 and e_2. Let $R_i^{(1)}$ and $R_i^{(2)}$ be the two rays that originate at v_i^* and that contain e_1 and e_2, respectively, in the placement of P_i in OPT. We have that $R_i^{(1)} \setminus \{v_i^*\}$ and $R_i^{(2)} \setminus \{v_i^*\}$ intersect at most one edge of the knapsack each. If $R_i^{(1)} \setminus \{v_i^*\}$ and $R_i^{(2)} \setminus \{v_i^*\}$ intersect the same edge of the knapsack then we say that P_i is *edge-facing,* if one of them intersects a horizontal edge and the other one intersects a vertical edge we say that P_i is *corner-facing.* The next lemma shows that there can be only $O(1)$ triangles in $\mathrm{OPT} \cap \mathcal{P}_H$ that are neither edge- nor corner-facing, and therefore we compute a $O(1)$-approximation with respect to the total profit of such triangles by simply selecting the input triangle with maximum weight.

▶ **Lemma 20.** *There can be at most $O(1)$ triangles in $\mathrm{OPT} \cap \mathcal{P}_H$ that are neither edge-facing nor corner-facing.*

Let p_{TL}, p_{TR}, p_{BL}, and p_{BR} denote the top left, top right, bottom left, and bottom right corners of K, respectively, and let $p_M := \binom{N/2}{N/2}$, $p_L := \binom{0}{N/2}$, and $p_R := \binom{N}{N/2}$, see Figure 3. By losing a factor $O(1)$ we assume from now on that that OPT contains at most one hard triangle from each group $\mathcal{P}_j$, using Lemma 18.

Let $\mathrm{OPT}_{\mathrm{EF}} \subseteq \mathrm{OPT} \cap \mathcal{P}_H$ denote the edge-facing hard triangles in OPT and denote by $\mathrm{OPT}_{\mathrm{CF}} \subseteq \mathrm{OPT} \cap \mathcal{P}_H$ the corner-facing hard triangles in OPT. In the remainder of this section we present now $O(1)$-approximation algorithms for edge-facing and for corner-facing triangles in $\mathcal{P}_H$. By selecting the best solution among the two we obtain the proof of Lemma 9.

Edge-facing triangles

We define a special type of solutions called *top-left-packings* that our algorithm will compute. We will show later that there are solutions of this type whose profit is at least a constant fraction of the profit of $\mathrm{OPT}_{\mathrm{EF}}$.

For each $t \in \mathbb{N}$ let $p_t := p_M + \frac{t}{N^2}\binom{1}{0}$. Let $\mathcal{P}' = \{P_{i_1}, ..., P_{i_k}\}$ be a set of triangles that are ordered according to the groups $\mathcal{P}_j$, i.e., such that for any $P_{i_\ell}, P_{i_{\ell+1}} \in \mathcal{P}'$ with $P_{i_\ell} \in \mathcal{P}_j$ and $P_{i_{\ell+1}} \in \mathcal{P}_{j'}$ for some j, j' it holds that $j \leq j'$. We define a placement of $\mathcal{P}'$ that we call a *top-left-packing.* First, we place P_{i_1} such that $v_{i_1}^*$ concides with p_{TL} and one edge of P_{i_1} lies on the diagonal of K that connects p_{TL} and p_0. Note that there is a unique way to place P_{i_1} in this way. Iteratively, suppose that we have packed triangles $\{P_{i_1}, ..., P_{i_\ell}\}$ such that for each triangle $P_{i_{\ell'}}$ in this set its respective vertex $v_{i_{\ell'}}^*$ coincides with p_{TL}, see Figure 1c). Intuitively, we pack $P_{i_{\ell+1}}$ on top of P_{i_ℓ} such that $v_{i_{\ell+1}}^*$ coincides with p_{TL}. Let t be the smallest integer such that the line segment connecting p_t and p_R has empty intersection with each triangle $P_{i_1}, ..., P_{i_\ell}$ according to our placement. We place $P_{i_{\ell+1}}$ such that $v_{i_{\ell+1}}^*$ concides with p_{TL} and one of its edges lies on the line that contains p_{TL} and p_t. There is a unique way to place $P_{i_{\ell+1}}$ in this way. We continue until we placed all triangles in $\mathcal{P}'$. If all of them are placed completely inside K we say that the resulting solution is a *top-left-packing* and that $\mathcal{P}'$ is *top-left-packable.* We define *bottom-right-packing* and *bottom-right-packable* symmetrically, mirroring the above definition along the line that contains p_{BL} and p_{TR}.

In the next lemma we show that there is always a top-left-packable or a bottom-right-packable solution with large profit compared to $\mathcal{P}_H \cap \mathrm{OPT}$ or there is a single triangle with large profit.

▶ **Lemma 21.** *There exists a solution $\mathcal{P}_H^* \subseteq \mathcal{P}_H \cap \mathrm{OPT}_{\mathrm{EF}}$ such that $w(\mathcal{P}_H \cap \mathrm{OPT}_{\mathrm{EF}}) \leq O(w(\mathcal{P}_H^*))$ and*

- *$\mathcal{P}_H^*$ is top-left-packable or bottom-right-packable and for each j we have that $|\mathcal{P}_H^* \cap \mathcal{P}_j| \leq 1$,*
- *or it holds that $|\mathcal{P}_H^*| = 1$.*

Proof sketch. Let L be the line segment connecting p_L with p_R. Essentially, we can assume that the length of the longest edge of each triangle is close to $\sqrt{2}N$ (the length of the diagonal of the knapsack), e.g., $(1-\epsilon)\sqrt{2}N$ for some $\epsilon > 0$. One can show that this holds for all but $O_\epsilon(1)$ hard triangles in OPT. Our key observation is that in *any* packing each hard triangle P_i intersects L by the same amount (up to constant factors). Using this, we partition $\mathrm{OPT}_{\mathrm{EF}}$ into $O(1)$ groups such that in any packing each group intersects L by a smaller amount than $\mathrm{OPT}_{\mathrm{EF}}$ in the original packing. This guarantees that the bottom edge of each triangle fits in the top-left-packing for each group. Using that in the original packing the triangles in $\mathrm{OPT}_{\mathrm{EF}}$ were edge-facing, we show that also the entire triangle fits. ◀

We describe now a polynomial time algorithm that computes the most profitable solution that satisfies the properties of Lemma 21. To find the most profitable solution $\mathcal{P}_H^*$ that satisfies that $|\mathcal{P}_H^*| = 1$ we simply take the triangle with maximum weight, let P_{i^*} be this triangle. We establish now a dynamic program that computes the most profitable top-left-packable solution; computing the most profitable bottom-right-packable solution works analogously. Our DP has a cell corresponding to pairs (j,t) with $j,t \in \mathbb{Z}$. Intuitively, (j,t) represents the subproblem of computing a set $\mathcal{P}'_H \subseteq \mathcal{P}_H$ of maximum weight such that $\mathcal{P}'_H \cap \mathcal{P}_{j'} = \emptyset$ for each $j' < j$ and $|\mathcal{P}'_H \cap \mathcal{P}_{j''}| \leq 1$ for each $j'' \geq j$ and such that $\mathcal{P}'_H$ is top-left-packable inside the triangular area T_t defined by the line that contains p_{TL} and p_t, the top edge of K, and the right edge of K. Given a cell (j,t) we want to compute a solution $DP(j,t)$ associated with (j,t). Intuitively, we guess whether the optimal solution $\mathcal{P}'_H$ to (j,t) contains a triangle from $\mathcal{P}_H \cap \mathcal{P}_j$. Therefore, we try each triangle $P_i \in \mathcal{P}_H \cap \mathcal{P}_j$ and place it inside T_t such that v_i^* concides with p_{TL} and one of its edges lies on the line containing p_{TL} and p_t. Let $t'(P_i)$ denote the smallest integer such that $t'(P_i) \geq t$ and $p_{t'(P_i)}$ is not contained in the resulting placement of P_i inside T_t. We associate with P_i the solution $P_i \cup DP(j+1, t'(P_i))$. Finally, we define $DP(j,t)$ to be the solution of maximum profit among the solutions $P_i \cup DP(j+1, t'(P_i))$ for each $P_i \in \mathcal{P}_H \cap \mathcal{P}_j$ and the solution $DP(j+1,t)$.

We introduce a DP-cell $DP(j,t)$ for each pair $(j,t) \in \mathbb{Z}^2$ where $j_{\min} \leq j \leq j_{\max}$ and $0 \leq t \leq \log_{1+1/n}\left(\frac{N}{2}\right)$. Note that due to Lemma 18 for all other values of j we have that $\mathcal{P}_j \cap \mathcal{P}_H = \emptyset$. Also note that $p_t \notin K$ if $t \geq N^2/2$. This yields at most $(nN)^{O(1)}$ cells in total. Finally, we output the solution $DP(j_{\min}, 0)$.

In the next lemma we prove that our DP computes the optimal top-left-packable solution with the properties of Lemma 21.

▶ **Lemma 22.** *There is an algorithm with a running time of $(nN)^{O(1)}$ that computes the optimal solution $\mathcal{P}' \subseteq \mathcal{P}_H$ such that $\mathcal{P}'$ is top-left-packable or bottom-right-packable and such that for each j we have that $|\mathcal{P}' \cap \mathcal{P}_j| \leq 1$.*

We execute the above DP and its counterpart for bottom-right-packable solutions to obtain a top-left-packable solution $\mathcal{P}'_1$ and a bottom-right-packable solution $\mathcal{P}'_2$. We output the most profitable solution among $\{P_{i^*}\}, \mathcal{P}'_1, \mathcal{P}'_2$. Due to Lemma 21 this yields a solution with weight at least $\Omega(w(\mathcal{P}_H \cap \mathrm{OPT}))$.

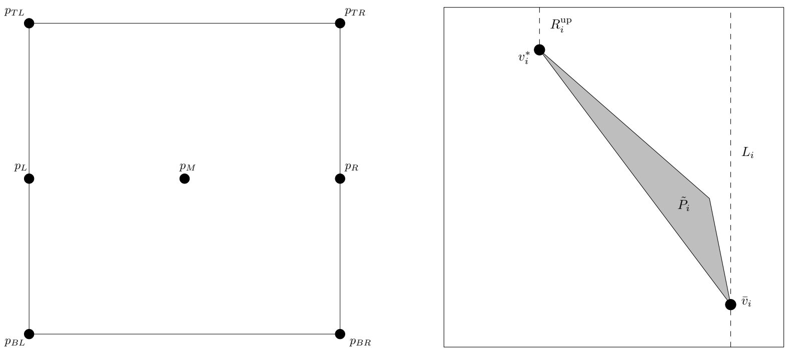


Figure 3 Left: The points $p_{TL}, p_{TR}, p_{BL}, p_{BR}, p_L, p_M, p_R$. Right: A corner-facing triangle, its vertices v_i^* and $\bar{v}_i$, and the lines L_i and R_i^{up}.

▶ **Lemma 23.** *There is an algorithm with a running time of $(nN)^{O(1)}$ that computes a solution $\mathcal{P}'_H \subseteq \mathcal{P}_H$ such that $w(\text{OPT}_{\text{EF}}) \leq O(w(\mathcal{P}'_H))$.*

Corner-facing triangles

We present now a $O(1)$-approximation algorithm for the corner-facing triangles in OPT, i.e., our algorithm computes a solution $\mathcal{P}' \subseteq \mathcal{P}$ of profit at least $\Omega(w(\text{OPT}_{\text{CF}}))$. We first establish some properties for OPT_{CF}. We argue that by losing a constant factor we can assume that each triangle in OPT_{CF} intuitively faces the bottom-right corner.

▶ **Lemma 24.** *By losing a factor 4 we can assume that for each triangle $P_i \in \text{OPT}_{\text{CF}}$ we have that $R_i^{(1)} \setminus \{v_i^*\}$ intersects the bottom edge of the knapsack and and $R_i^{(2)} \setminus \{v_i^*\}$ intersects the right edge of the knapsack, or vice versa.*

In the following lemma we establish a property that will be crucial for our algorithm. For each $P_i \in \text{OPT}_{\text{CF}}$ let R_i^{up} denote the ray originating at v_i^* and going upwards. We establish that we can assume that R_i^{up} does not intersect with any triangle $P_{i'} \in \text{OPT}_{\text{CF}}$, see Figure 3.

▶ **Lemma 25.** *By losing a factor $O(1)$ we can assume that for each $P_i, P_{i'} \in \text{OPT}_{\text{CF}}$ it holds that $R_i^{\text{up}} \cap P_{i'} = \emptyset$.*

Our algorithm is a dynamic program that intuively guesses the placements of the triangles in OPT_{CF} step by step. To this end, each DP-cell corresponds to a subproblem that is defined via a part $K' \subseteq K$ of the knapsack and a subset of the groups $J \subseteq \{j_{\min}, ..., j_{\max}\}$. The goal is to place triangles from $\bigcup_{j \in J} \mathcal{P}_j$ of maximum profit into K'. Formally, each DP-cell is defined by up to two triangles $P_i, P_{i'}$, placements $\tilde{P}_i, \tilde{P}_{i'}$ for them, and a set $J \subseteq \{j_{\min}, ..., j_{\max}\}$; if the cell is defined via exactly one triangle P_i then there is also a value $\text{dir} \in \{\text{left}, \text{mid}\}$. The corresponding region K' is defined as follows: if the cell is defined via zero triangles then the region is the whole knapsack K. Otherwise, let $\bar{v}_i$ denote the right-most vertex of $\tilde{P}_i$, i.e., the vertex of $\tilde{P}_i$ that is closest to the right edge of the knapsack (see Figure 3). Let L_i denote the vertical line that goes through $\bar{v}_i$ (and thus intersects the top and the bottom edge of the knapsack). If the cell is defined via one triangle P_i then observe that $K \setminus (\tilde{P}_i \cup R_i^{\text{up}} \cup L_i)$ has three connected components,

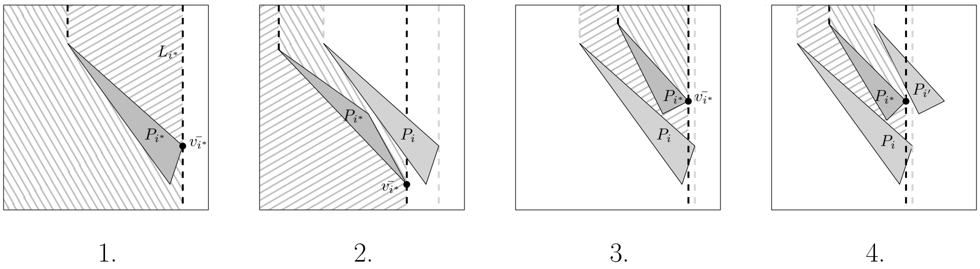


Figure 4 The cases in the transition of the DP for corner-facing triangles (see Lemma 26).

- one on the left, surrounded by R_i^{up}, parts of $\tilde{P}_i$, the left edge of the knapsack, and parts of the top and bottom edge of the knapsack
- one on the right, surrounded by L_i, the right edge of the knapsack, and parts of the top and bottom edge of the knapsack, and
- one in the middle, surrounded by the top edge of the knapsack, $\tilde{P}_i$, and L_i.

If dir = left then the region of the cell equals the left component, if dir = mid then the region of the cell equals the middle component. Assume now that the cell is defined via two triangles $P_i, P_{i'}$. Assume w.l.o.g. that $\bar{v}_i$ is closer to the right edge of the knapsack than $\bar{v}_{i'}$. Then $K \setminus (\tilde{P}_i \cup \tilde{P}_{i'} \cup R_i^{\text{up}} \cup R_{i'}^{\text{up}} \cup L_{i'})$ has one connected component that is surrounded by $\tilde{P}_i, \tilde{P}_{i'}, R_i^{\text{up}}, R_{i'}^{\text{up}}, L_{i'}$ and we define the region of the cell to be this component. Observe that the total number of DP-cells is bounded by $(nN)^{O(1)}$, using that there are only $(nN)^{O(1)}$ possible placements for each triangle.

We describe now a dynamic program that computes the optimal solution to each cell. Assume that we are given a cell C for which we want to compute the optimal solution. We guess the triangle P_{i^*} in the optimal solution to this cell such that $\bar{v}_{i^*}$ is closest to the right edge of the knapsack, and its placement $\tilde{P}_{i^*}$ in the optimal solution to C. Let j^* such that $P_{i^*} \in \mathcal{P}_{j^*}$. We will prove in the next lemma that the optimal solution to C consists of P_{i^*} and the optimal solutions to two other DP-cells, see Figure 4.

▶ **Lemma 26.** *Let C be a DP-cell, let $J \subseteq \{j_{\min}, ..., j_{\max}\}$, and let $P_i \in \mathcal{P}_\ell, P_{i'} \in \mathcal{P}_{\ell'}$ be two triangles with $\ell < \ell'$ and let $\tilde{P}_i, \tilde{P}_{i'}$ be placements for them. Then there are disjoint sets $J', J'' \subseteq J$ such that*

1. *if $C = (J)$, then its optimal solution consists of P_{i^*} and the optimal solutions to the cells $(J', P_{i^*}, \tilde{P}_{i^*}, \text{left})$ and $(J'', P_{i^*}, \tilde{P}_{i^*}, \text{mid})$,*
2. *if $C = (J, P_i, \tilde{P}_i, \text{left})$ then its optimal solution consists of P_{i^*} and the optimal solutions to the cells $(J', P_{i^*}, \tilde{P}_{i^*}, \text{left})$ and $(J'', P_i, \tilde{P}_i, P_{i^*}, \tilde{P}_{i^*})$,*
3. *if $C = (J, P_i, \tilde{P}_i, \text{mid})$ then its optimal solution consists of P_{i^*} and the optimal solutions to the cells $(J', P_{i^*}, \tilde{P}_{i^*}, \text{mid})$ and $(J'', P_i, \tilde{P}_i, P_{i^*}, \tilde{P}_{i^*})$,*
4. *if $C = (J, P_i, \tilde{P}_i, P_{i'}, \tilde{P}_{i'})$ then the optimal solution to C consists of P_{i^*} and the optimal solutions to the cells $(J', P_i, \tilde{P}_i, P_{i^*}, \tilde{P}_{i^*})$ and $(J'', P_{i^*}, \tilde{P}_{i^*}, P_{i'}, \tilde{P}_{i'})$.*

We guess the sets $J', J'' \subseteq J$ according to Lemma 26 and store in C the solution consisting of P_{i^*}, and the solutions stored in the two cells according to the lemma. At the end, the cell $C = (\{j_{\min}, ..., j_{\max}\})$ (whose corresponding region equals to K) contains the optimal solution. By combining Lemmas 23 and 27 we obtain the proof of Lemma 9.

▶ **Lemma 27.** *There is an algorithm with a running time of $(nN)^{O(1)}$ that computes a solution $\mathcal{P}' \subseteq \mathcal{P}$ such that $w(\text{OPT}_{\text{CF}}) \leq O(w(\mathcal{P}'))$.*

2.4 Hard polygons under resource augmentation

Let $\delta > 0$. We consider the setting of $(1+\delta)$-resource augmentation, i.e., we want to compute a solution $\mathcal{P}' \subseteq \mathcal{P}$ that is feasible for a knapsack of size $(1+\delta)N \times (1+\delta)N$ and such that $w(\mathrm{OPT}) \leq O(w(\mathcal{P}'))$ where OPT is the optimal solution for the original knapsack of size $N \times N$. Note that increasing K by a factor of $1+\delta$ is equivalent to shrinking the input polygons by a factor of $1+\delta$.

Given a polygon P defined via coordinates $(x_1, y_1), ..., (x_k, y_k) \in \mathbb{R}^2$ we define $\mathrm{shr}_{1+\delta}(P)$ to be the polygon with coordinates $(\bar{x}_1, \bar{y}_1), ..., (\bar{x}_k, \bar{y}_k) \in \mathbb{R}^2$ where $\bar{x}_{k'} = x_{k'}/(1+\delta)$ and $\bar{y}_{k'} = y_{k'}/(1+\delta)$ for each k'. For each input polygon $P_i \in \mathcal{P}$ we define its shrunk counterpart to be $\bar{P}_i := \mathrm{shr}_{1+\delta}(P_i)$. Based on $\bar{\mathcal{P}}$ we define sets $\bar{\mathcal{P}}_E, \bar{\mathcal{P}}_M, \bar{\mathcal{P}}_H$ and the set $\bar{\mathcal{P}}_j$ for each $j \in \mathbb{Z}$ in the same way as we defined $\mathcal{P}_E, \mathcal{P}_M, \mathcal{P}_H$ and $\mathcal{P}_j$ based on $\mathcal{P}$ above.

For the sets $\bar{\mathcal{P}}_E$ and $\bar{\mathcal{P}}_M$ we use the algorithms due to Lemmas 6 and 7 as before. For the hard polygons $\bar{\mathcal{P}}_H$ we can show that there are only $O_\delta(1)$ groups $\bar{\mathcal{P}}_j$ that are non-empty, using that we obtained them via shrinking the original input polygons. Intuitively, this is true since $\bar{\ell}_i \leq \frac{\sqrt{2}N}{1+\delta}$ for each $\bar{P}_i \in \bar{\mathcal{P}}$ where $\bar{\ell}_i$ denotes the length of the longest diagonal of $\bar{P}_i$, and hence $\bar{\mathcal{P}}_j \cap \bar{\mathcal{P}}_H = \emptyset$ if $j < \log\left(\frac{\delta}{1+\delta}\sqrt{2}N\right)$.

▶ **Lemma 28.** *We have that $\bar{\mathcal{P}}_j = \emptyset$ if $j < \log\left(\frac{\delta}{1+\delta}\sqrt{2}N\right)$. Hence, there are only $\log\left(\frac{1+\delta}{\delta}\right)+1$ values $j \in \mathbb{Z}$ such that $\bar{\mathcal{P}}_j \neq \emptyset$.*

Lemmas 18 and 28 imply that $|\overline{\mathrm{OPT}} \cap \bar{\mathcal{P}}_H| \leq O(\log\left(\frac{1+\delta}{\delta}\right))$ where $\overline{\mathrm{OPT}}$ denotes the optimal solution for the polygons in $\bar{\mathcal{P}}$. Let $\bar{\mathcal{P}}'_H \subseteq \bar{\mathcal{P}}_H$ denote the set due to Lemma 19 when assuming that $\bar{\mathcal{P}}_H$ are the hard polygons in the given instance. Therefore, we guess $\bar{\mathcal{P}}'_H$ in time $n^{O(\log(\frac{1+\delta}{\delta}))}$. Finally, we output the solution of largest weight among $\bar{\mathcal{P}}'_H$ and the solutions due applying to Lemmas 6 and 7 to the input sets $\bar{\mathcal{P}}_E$ and $\bar{\mathcal{P}}_M$, respectively. This yields the proof of Theorem 3.

3 Optimal profit under resource augmentation

In this section we also study the setting of $(1+\delta)$-resource augmentation, i.e., we want to compute a solution $\mathcal{P}'$ which is feasible for an enlarged knapsack of size $(1+\delta)N \times (1+\delta)N$, for any constant $\delta > 0$. We present an algorithm with a running time of $n^{(\log(n)/\delta)^{O(1)}}$ that computes such a solution $\mathcal{P}'$ with $w(\mathcal{P}') \geq \mathrm{OPT}$ where OPT is the optimal solution for the original knapsack of size $N \times N$. In particular, we here do not lose any factor in our approximation guarantee.

First, we prove a set of properties that we can assume "by $(1+\delta)$-resource augmentation" meaning that if we increase the size of K by a factor $1+\delta$ then there exists a solution of weight $w(\mathrm{OPT})$ with the mentioned properties, or that we can modify the input in time $n^{O(1)}$ such that it has these properties and there still exists a solution of weight $w(\mathrm{OPT})$.

3.1 Few types of items

We want to establish that the input polygons have only $(\log(n)/\delta)^{O(1)}$ different shapes. Like in Section 2 for each polygon $P_i \in \mathcal{P}$ denote by B_i its bounding box with width ℓ_i and height h_i. Note that $\ell_i \geq h_i$. The bounding boxes of all polygons $P_i \in \mathcal{P}$ such that $h_i \leq \delta\frac{N}{n}$ have a total height of at most δN. Therefore, we can pack all these polygons into the extra capacity that we gain by increasing the size of K by a factor $1+\delta$ and therefore ignore them in the sequel.

▶ **Lemma 29.** *By $(1+\delta)$-resource augmentation we can assume for each $P_i \in \mathcal{P}$ that $\ell_i \geq h_i \geq \delta N/n$ and that $\text{area}(P_i) = \Omega(\text{area}(K)\delta^2/n^2)$.*

Next, intuitively we stretch the optimal solution OPT by a factor $1+\delta$ which yields a container C_i for each polygon $P_i \in$ OPT which contains P_i and which is slightly bigger than P_i. We define a polygon P'_i such that $P_i \subseteq P'_i \subseteq C_i$ and that globally there are only $(\log(n)/\delta)^{O_\delta(1)}$ different ways P'_i can look like, up to translations and rotations. We refer to those as a set $\mathcal{S}$ of *shapes* of input objects. Hence, due to the resource augmentation we can replace each input polygon P_i by one of the shapes in $\mathcal{S}$.

▶ **Lemma 30.** *By $(1+\delta)$-resource augmentation we can assume that there is a set of shapes $\mathcal{S}$ with $|\mathcal{S}| \leq (\log(n)/\delta)^{O_\delta(1)}$ such that for each $P_i \in \mathcal{P}$ there is a shape $S \in \mathcal{S}$ such that $P_i = S$ and S has only $\Lambda = (1/\delta)^{O(1)}$ many vertices.*

Finally, we ensure that for each polygon $P_i \in \mathcal{P}$ we can restrict ourselves to only $(n/\delta)^{O(1)}$ possible placements in K.

▶ **Lemma 31.** *By $(1+\delta)$-resource augmentation, for each polygon $P_i \in \mathcal{P}$ we can compute a set $\mathcal{L}_i$ of at most $(n/\delta)^{O(1)}$ possible placements for P_i in time $(n/\delta)^{O(1)}$ such that if $P_i \in$ OPT then in OPT the polygon P_i is placed inside K according to one placement $\tilde{P}_i \in \mathcal{L}_i$.*

3.2 Recursive algorithm

We describe our main algorithm. First, we guess how many polygons of each of the shapes in $\mathcal{S}$ are contained in OPT. Since there are only $(\log(n)/\delta)^{O_\delta(1)}$ different shapes in $\mathcal{S}$ we can do this in time $n^{(\log(n)/\delta)^{O_\delta(1)}}$. Once we know how many polygons of each shape we need to select, it is clear which polygons we should take since if for some shape $S_i \in \mathcal{S}$ we need to select n_i polygons with that shape then it is optimal to select the n_i polygons in $\mathcal{P}$ of shape S_i with largest weight. Therefore, in the sequel assume that we are given a set of polygons $\mathcal{P}' \subseteq \mathcal{P}$ and we want to find a packing for them inside K.

Our algorithm is recursive and it generalizes a similar algorithm for the special case of axis-parallel rectangles in [1]. On a high level, we guess a partition of K given by a separator Γ which is a polygon inside K. It has the property that at most $\frac{2}{3}|\text{OPT}|$ of the polygons of OPT lie inside Γ and at most $\frac{2}{3}|\text{OPT}|$ of the polygons of OPT lie outside Γ. We guess how many polygons of each shape are placed inside and outside Γ in OPT. Then we recurse separately inside and outside Γ. For our partition, we are looking for a polygon Γ according to the following definition.

▶ **Definition 32.** *Let $\ell \in \mathbb{N}$ and $\epsilon > 0$. Let $\bar{\mathcal{P}}$ be a set of pairwise non-overlapping polygons in K. A polygon Γ is a* balanced $\hat{\epsilon}$-cheap ℓ-cut *if*

- *Γ has at most ℓ edges,*
- *the polygons contained in Γ have a total area of at most $2/3 \cdot \text{area}(\bar{\mathcal{P}})$,*
- *the polygons contained in the complement of Γ, i.e., in $K \setminus \Gamma$, have a total area of at most $2/3 \cdot \text{area}(\bar{\mathcal{P}})$, and*
- *the polygons intersecting the boundary of Γ have a total area of at most $\hat{\epsilon} \cdot \text{area}(\bar{\mathcal{P}})$.*

In order to restrict the set of balanced cheap cuts to consider, we will allow only polygons Γ such that each of its vertices is contained in a set Q of size $(n/\delta)^{O(1)}$ defined as follows. We fix a triangulation for each placement $P'_i \in \mathcal{L}_i$ of each polygon $P_i \in \mathcal{P}'$. We define a set Q_0 where for each placement $P'_i \in \mathcal{L}_i$ for P_i we add to Q_0 the positions of the vertices of P'_i. Also, we add the four corners of K to Q_0. Let $\mathcal{V}$ denote the set of vertical lines $\{(\bar{x}, \bar{y})|\bar{y} \in \mathbb{R}\}$ such that $\bar{x}$ is the x-coordinate of one point in Q_0. We define a set Q_1 where for each placement $P_i' \in \mathcal{L}_i$ of each $P_i \in \mathcal{P}'$, each edge e of a triangle in the triangulation of P_i', and each vertical line $L \in \mathcal{V}$ we add to Q_1 the intersection of e and L. Also, we add to Q_1 the intersection of each line in $L \in \mathcal{V}$ with the two boundary edges of K. Let Q_2 denote the set of all intersections of pairs of line segments whose respective endpoints are in $Q_0 \cup Q_1$. We define $Q := Q_0 \cup Q_1 \cup Q_2$. A result in [1] implies that there exists a balanced cheap cut whose vertices are all contained in Q.

▶ **Lemma 33** ([1]). *Let $\epsilon > 0$ and let $\mathcal{P}'$ be a set of pairwise non-intersecting polygons in the plane with at most Λ edges each such that $\text{area}(P) < \text{area}(\mathcal{P}')/3$ for each $P \in \mathcal{P}$. Then there exists a balanced $O(\epsilon\Lambda)$-cheap $\Lambda(\frac{1}{\epsilon})^{O(1)}$-cut Γ whose vertices are contained in Q.*

Our algorithm is recursive and places polygons from $\mathcal{P}'$, trying to maximize the total area of the placed polygons. In each recursive call we are given an area $\bar{K} \subseteq K$ and a set of polygons $\bar{\mathcal{P}} \subseteq \mathcal{P}'$. In the main call these parameters are $\bar{K} = K$ and $\bar{\mathcal{P}} = \mathcal{P}'$. If $\bar{\mathcal{P}} = \emptyset$ then we return an empty solution. If there is a polygon $P_i \in \bar{\mathcal{P}}$ with $\text{area}(P_i) \geq \text{area}(\bar{\mathcal{P}})/3$ then we guess a placement $P_i' \in \mathcal{L}_i$ and we recurse on the area $K \setminus P_i'$ and on the set $\bar{\mathcal{P}} \setminus \{P_i\}$. Otherwise, we guess the balanced cheap cut Γ due to Lemma 33 with $\epsilon := \frac{\delta}{\Lambda \log(n/\delta)}$ and for each shape $S \in \mathcal{S}$ we guess how many polygons of $\mathcal{P}'$ with shape S are contained in $\Gamma \cap \bar{K}$, how many are contained in $\bar{K} \setminus \Gamma$, and how many cross the boundary of Γ (i.e., have non-empty intersection with the boundary of Γ). Note that there are only $n^{(\Lambda \log(n/\delta))^{O(1)}}$ possibilities to enumerate. Let $\bar{\mathcal{P}}_{\text{in}}, \bar{\mathcal{P}}_{\text{out}}$, and $\bar{\mathcal{P}}_{\text{cross}}$ denote the respective sets of polygons. Then we recurse on the area $\Gamma \cap \bar{K}$ with input polygons $\bar{\mathcal{P}}_{\text{in}}$ and on the area $\bar{K} \setminus \Gamma$ with input polygons $\bar{\mathcal{P}}_{\text{out}}$. Suppose that the recursive calls return two sets of polygons $\bar{\mathcal{P}}_{\text{in}}' \subseteq \bar{\mathcal{P}}_{\text{in}}$ and $\bar{\mathcal{P}}'_{\text{out}} \subseteq \bar{\mathcal{P}}_{\text{out}}$ that the algorithm managed to place inside the respective areas $\Gamma \cap \bar{K}$ and $\bar{K} \setminus \Gamma$. Then we return the set $\bar{\mathcal{P}}_{\text{in}}' \cup \bar{\mathcal{P}}_{\text{out}}'$ for the guesses of Γ, $\bar{\mathcal{P}}_{\text{in}}, \bar{\mathcal{P}}_{\text{out}}$, and $\bar{\mathcal{P}}_{\text{cross}}$ that maximize $\text{area}(\bar{\mathcal{P}}_{\text{in}}' \cup \bar{\mathcal{P}}_{\text{out}}')$. If we guess the (correct) balanced cheap cut due to Lemma 33 in each iteration then our recursion depth is $O(\log_{3/2}(n^2/\delta^2)) = O(\log(n/\delta))$ since the cuts are balanced and each polygon has an area of at least $\Omega(\text{area}(K)\delta^2/n^2)$ (see Lemma 29). Therefore, if in a recursive call of the algorithm the recursion depth is larger than $O(\log(n/\delta))$ then we return the empty set and do not recurse further. Also, if we guess the correct cut in each node of the recursion tree then we cut polygons whose total area is at most a $\frac{\delta}{\log(n/\delta)}$-fraction of the area of all remaining polygons. Since our recursion depth is $O(\log(n/\delta))$, our algorithm outputs a packing for a set of polygons in $\mathcal{P}'$ with area at least $(1 - \frac{\delta}{\log(n/\delta)})^{O(\log(n/\delta))} \bar{w}(\bar{\mathcal{P}}) = (1 - O(\delta))\text{area}(\bar{\mathcal{P}})$. This implies the following lemma.

▶ **Lemma 34.** *Assume that there is a non-overlapping packing for $\mathcal{P}'$ in K. There is an algorithm with a running time of $n^{(\Lambda \log(n/\delta))^{O(1)}}$ that computes a placement of a set of polygons $\bar{\mathcal{P}}' \subseteq \mathcal{P}'$ inside K such that $\text{area}(\bar{\mathcal{P}}') \geq (1 - O(\delta))\text{area}(\mathcal{P}')$.*

It remains to pack the polygons in $\tilde{\mathcal{P}}' := \mathcal{P}' \setminus \bar{\mathcal{P}}'$. The total area of their bounding boxes is bounded by $\sum_{P_i \in \tilde{\mathcal{P}}'} B_i \leq 2\text{area}(\tilde{\mathcal{P}}') \leq O(\delta)\text{area}(\mathcal{P}') \leq O(\delta)\text{area}(K)$. Therefore, we can pack them into additional space that we gain via increasing the size of K by a factor $1 + O(\delta)$, using the Next-Fit-Decreasing-Height algorithm [4].

▶ **Theorem 35.** *There is an algorithm with a running time of $n^{(\log(n)/\delta))^{O(1)}}$ that computes a set $\mathcal{P}'$ with $w(\mathcal{P}') \geq \text{OPT}$ such that $\mathcal{P}'$ fits into K under $(1 + \delta)$-resource augmentation.*

References

1 Anna Adamaszek and Andreas Wiese. A QPTAS for maximum weight independent set of polygons with polylogarithmically many vertices. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2014)*, pages 645–656. SIAM, 2014.

2 Anna Adamaszek and Andreas Wiese. A quasi-PTAS for the two-dimensional geometric knapsack problem. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2015)*, pages 1491–1505. SIAM, 2015.

3 Nikhil Bansal, Alberto Caprara, Klaus Jansen, Lars Prädel, and Maxim Sviridenko. A structural lemma in 2-dimensional packing, and its implications on approximability. In *Algorithms and Computation (ISAAC 2009)*, volume 5878 of *LNCS*, pages 77–86. Springer, 2009.

4 Edward G Coffman, Jr, Michael R Garey, David S Johnson, and Robert Endre Tarjan. Performance bounds for level-oriented two-dimensional packing algorithms. *SIAM Journal on Computing*, 9:808–826, 1980.

5 Waldo Gálvez, Fabrizio Grandoni, Sandy Heydrich, Salvatore Ingala, Arindam Khan, and Andreas Wiese. Approximating geometric knapsack via l-packings. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 260–271, 2017. doi:10.1109/FOCS.2017.32.

6 Sandy Heydrich and Andreas Wiese. Faster approximation schemes for the two-dimensional knapsack problem. *ACM Trans. Algorithms*, 15(4):47:1–47:28, 2019. doi:10.1145/3338512.

7 Klaus Jansen and Roberto Solis-Oba. New approximability results for 2-dimensional packing problems. In *Mathematical Foundations of Computer Science (MFCS 2007)*, volume 4708 of *LNCS*, pages 103–114. Springer, 2007.

8 Klaus Jansen and Roberto Solis-Oba. A polynomial time approximation scheme for the square packing problem. In *Integer Programming and Combinatorial Optimization (IPCO 2008)*, volume 5035 of *LNCS*, pages 184–198. Springer, 2008.

9 Klaus Jansen and Guochuan Zhang. Maximizing the number of packed rectangles. In *Algorithm Theory (SWAT 2004)*, volume 3111 of *LNCS*, pages 362–371. Springer, 2004.

10 Klaus Jansen and Guochuan Zhang. On rectangle packing: maximizing benefits. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2004)*, pages 204–213. SIAM, 2004. URL: http://dl.acm.org/citation.cfm?id=982792.982822.

11 Joseph YT Leung, Tommy W Tam, CS Wong, Gilbert H Young, and Francis YL Chin. Packing squares into a square. *Journal of Parallel and Distributed Computing*, 10(3):271–275, 1990.

12 Carla Negri Lintzmayer, Flávio Keidi Miyazawa, and Eduardo Candido Xavier. Two-dimensional knapsack for circles. In *Latin American Symposium on Theoretical Informatics*, pages 741–754. Springer, 2018.

13 A Steinberg. A strip-packing algorithm with absolute performance bound 2. *SIAM Journal on Computing*, 26(2):401–409, 1997.

Proportionally Fair Clustering Revisited

Evi Micha
University of Toronto, Canada
emicha@cs.toronto.edu

Nisarg Shah
University of Toronto, Canada
nisarg@cs.toronto.edu

Abstract

In this work, we study fairness in centroid clustering. In this problem, k cluster centers must be placed given n points in a metric space, and the cost to each point is its distance to the nearest cluster center. Recent work of Chen et al. [8] introduces the notion of a *proportionally fair* clustering, in which no group of at least n/k points can find a new cluster center which provides lower cost to each member of the group. They propose a *greedy capture* algorithm which provides a $1 + \sqrt{2}$ approximation of proportional fairness for any metric space, and derive generalization bounds for learning proportionally fair clustering from samples in the case where a cluster center can only be placed at one of finitely many given locations in the metric space.

We focus on the case where cluster centers can be placed anywhere in the (usually infinite) metric space. In case of the L^2 distance metric over $\mathbb{R}^t$, we show that the approximation ratio of greedy capture improves to 2. We also show that this is due to a special property of the L^2 distance; for the L^1 and L^∞ distances, the approximation ratio remains $1 + \sqrt{2}$. We provide universal lower bounds which apply to all algorithms.

We also consider metric spaces defined on graphs. For trees, we show that an exact proportionally fair clustering always exists and provide an efficient algorithm to find one. The corresponding question for general graph remains an interesting open question.

Finally, we show that for the L^2 distance, checking whether a proportionally fair clustering exists and implementing greedy capture over an infinite metric space are NP-hard problems, but (approximately) solvable in special cases. We also derive generalization bounds which show that an approximately proportionally fair clustering for a large number of points can be learned from a small number of samples. Our work advances the understanding of proportional fairness in clustering, and points out many avenues for future work.

2012 ACM Subject Classification Theory of computation → Algorithmic mechanism design; Theory of computation → Facility location and clustering

Keywords and phrases Fairness, Clustering, Facility location

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.85

Category Track A: Algorithms, Complexity and Games

Related Version A full version of the paper is available at `http://www.cs.toronto.edu/~nisarg/papers/fair_clustering.pdf`.

Funding This work is partially funded by a Discovery Grant from the Natural Sciences and Engineering Research Council (NSERC).

Acknowledgements We thank anonymous reviewers for suggesting Theorem 11.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 85; pp. 85:1–85:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Machine learning algorithms are increasingly being used for decision making in applications where they affect human lives; popular examples include resume screening, evaluation of loan applications, bail decisions, etc. [27]. This has led to growing concern as to whether these algorithms, which may view humans as "data points", treat them fairly [3, 25]. Consequently, research on designing fair machine learning algorithms is proliferating [24, 22].

Much of this literature focuses on fairness in classification [36, 35, 19], but the study of fairness in other settings such as regression [1] and clustering [8] is also on the rise. In this paper, we focus on fairness in clustering, specifically, in centroid clustering. In this problem, we are given a set $\mathcal{N}$ of data points in a metric space, and a set $\mathcal{M}$ of possible locations for cluster centers in the same metric space. Given $k \in \mathbb{N}$, the task is to select a set $X \subseteq \mathcal{M}$ consisting of $|X| = k$ cluster centers and assign each data point to a cluster center – usually the closest – with the goal that data points are close to their assigned cluster centers. Clustering has diverse applications in market research, pattern recognition, data analysis, image segmentation, and facility location. In applications like image segmentation or market research, often the goal is to simply identify different clusters of points among the data. However, in facility location [14, 21, 15], where data points may represent locations of houses in the neighborhood and cluster centers may represent locations where public facilities (such as parks) will be built, it is of paramount importance that the facilities be distributed to fairly serve the population.

Adapting an example given by Chen et al. [8], imagine that there is a dense urban area with a population of 10,000, and far from it, there are 10 small communities with a population of 100 each. The communities are closer to each other compared to how far they are from the urban area, but still well distinguished. With $k = 11$, a standard clustering algorithm such as k-means would identify the urban area as one cluster, and each small community as one cluster. However, building just one park that serves 10,000 people in the urban area, while each community of 100 people gets its own park violates the principle of *equal entitlement* [23]; this principle would suggest that when allocating 11 parks among a total of 11,000 people, the urban area consisting of 10,000 people should be allocated their proportional share of 10 parks, and one park should serve the 10 smaller communities consisting of 1,000 people altogether.

This notion of what a group deserves – *group fairness* – has been extensively studied in machine learning, and a variety of definitions have been proposed [7, 16, 24, 19, 32]. Borrowing from a long line of literature on fair resource allocation [34, 30, 12, 10], Chen et al. [8] proposed a novel definition of fairness in clustering that perfectly fits our motivation. Given a metric d over a set $\mathcal{N}$ of n points and a set $\mathcal{M}$ of feasible cluster centers, they say that a k-clustering X satisfies *proportional fairness* if there is no group of points $S \subseteq \mathcal{N}$ with $|S| \geq n/k$ and a new cluster location $y \in \mathcal{M}$ such that $d(i, y) < \min_{x \in X} d(i, x)$ for each member $i \in S$. Most fairness definitions in machine learning protect groups of individuals that are pre-defined based on certain *protected attributes* [4, 5, 9, 28] or that are sufficiently large [20]. In contrast, proportional fairness guarantees fairness to arbitrarily defined groups of all possible sizes. This may be helpful given recent observations that protecting groups defined based on individual attributes may allow an algorithm to circumvent fairness [20], or that information about which groups to protect may not be known in advance [17]. For references to other related work, we direct the reader to the work of Chen et al. [8].

1.1 Our Contribution

We build upon the work of Chen et al. [8]. While their work considers the metric d and the set of feasible cluster center locations $\mathcal{M}$ to be arbitrary (and $|\mathcal{M}|$ to be typically finite), we focus on the case where the metric consists of usual distance functions such as L^1, L^2, or L^∞ over $\mathbb{R}^t$, and cluster centers can be placed anywhere in the infinite metric space (i.e. $\mathcal{M} = \mathbb{R}^t$). While this change is seemingly simple, the infinite cardinality of $\mathcal{M}$ requires new algorithmic tools and generalization bounds, which we provide in this work. In some cases, we show that this in fact allows us to provide stronger approximation guarantees.

In Section 3, we analyze the greedy capture algorithm introduced by Chen et al. in the case where $d \in \{L^1, L^2, L^\infty\}$ and $\mathcal{M} = \mathbb{R}^t$. Chen et al. show that the algorithm provides $1 + \sqrt{2} \approx 2.414$ approximation to proportional fairness for all metric spaces. We show that for $d = L^2$ and $\mathcal{M} = \mathbb{R}^t$, it actually provides a better 2-approximation. We prove this via a refinement of the result of Chen et al.: we express the approximation ratio obtained by the algorithm in terms of a new characteristic of the metric that we term *Apollonius radius*, and show that this radius is small for the L^2 distance, allowing us to achieve a better approximation ratio. However, we show that for L^1 and L^∞, the approximation ratio of greedy capture is no better than $1 + \sqrt{2}$.

In Section 4, we provide universal lower bounds which apply to all algorithms. Specifically, we show that for $d = L^2$ and $\mathcal{M} = \mathbb{R}^t$, no algorithm achieves better than $2/\sqrt{3} \approx 1.155$ approximation ratio, whereas for $d \in \{L^1, L^\infty\}$ and $\mathcal{M} = \mathbb{R}^t$, we get a lower bound of 1.4. Note that these lower bounds are existential, and not computational.

In Section 5, we consider the case where $\mathcal{M}$ is the set of nodes of an unweighted graph, and d measures the shortest distance between two nodes on the graph. When the graph is a tree, we show that an exact proportionally fair clustering necessarily exists, and provide an efficient algorithm to find one. When the graph is arbitrary, but $k \geq n/2$ clusters need to be placed, we show that a proportionally fair clustering again necessarily exists and can be computed efficiently. Whether an exact proportionally fair clustering exists for all graphs remains an interesting open question.

Next, in Section 6, we show that for $d = L^2$ and $\mathcal{M} = \mathbb{R}^t$ for $t \geq 2$, checking whether a proportional clustering exists is NP-hard. When t is large, even implementing the greedy capture algorithm is NP-hard. However, this problem becomes efficiently solvable when t is constant, and when t is large, using a PTAS for an important sub-routine of greedy capture, we can efficiently compute a $2 \cdot (1 + \epsilon)$-proportionally fair clustering for any fixed $\epsilon > 0$.

Finally, in Section 7, we consider the problem of generalization: would a clustering that is proportionally fair with respect to samples drawn from $\mathcal{N}$ remain (approximately) proportionally fair with respect to the entire set $\mathcal{N}$? Chen et al. provide a positive answer for the case when $\mathcal{M}$ is finite. Using the framework of VC dimension, we show that, when $d = L^2$, the answer remains positive even when $\mathcal{M} = \mathbb{R}^t$.

2 Preliminaries

Let $\mathcal{N}$ be a set of n data points (or agents), which lie in a metric space $(\mathcal{X}, d)$, where $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a distance function satisfying the triangle inequality. For most of this work, we consider the case where $\mathcal{X} = \mathbb{R}^t$ for some $t \in \mathbb{N}$, but in Section 5, we consider the case where $\mathcal{X}$ is the set of nodes of a graph. We also focus on special distance functions such as the Euclidean distance (L^2), the Manhattan distance (L^1), and the L^∞ distance. Let $\mathcal{M} \subseteq \mathcal{X}$ be a the set of locations where cluster centers can be placed. In this work, we focus on the case where $\mathcal{M} = \mathcal{X}$. Let $k \in \mathbb{N}$. We denote by $[\mathcal{M}]^k$ the set of all subsets of M of size k. A k-clustering is a set $X \in [\mathcal{M}]^k$ containing k locations for cluster centers. We refer to each $x \in X$ as an *open* cluster center.

The cost to agent $i \in \mathcal{N}$ induced by a cluster center $x \in \mathcal{M}$ is the distance $d(i,x)$, and the cost to agent $i \in \mathcal{N}$ induced by a k-clustering X is the minimum distance from i to any cluster center, i.e., $d(i,X) \triangleq \min_{x\in X} d(i,x)$. Agent i is interested in minimizing her cost.

A set of points $S \subseteq \mathcal{N}$ containing at least $\lceil n/k \rceil$ is entitled at least one cluster center. If they can find a new cluster center that is better for each of them, we consider it a violation of fairness. Formally:

▶ **Definition 1.** *Given $\rho \geq 1$, we say a k-clustering $X \in [\mathcal{M}]^k$ is* ρ-proportionally fair *if there is no $S \subseteq \mathcal{N}$ with $|S| \geq \lceil n/k \rceil$ and $y \in \mathcal{M}$ such that $\rho \cdot d(i,y) < d(i,X)$ for all $i \in S$. If such a coalition S and point y exist, we refer to S as a* blocking coalition *and y as the center that they* deviate *to. When $\rho = 1$, we simply call this proportional fairness.*

The reason to define ρ-proportionally fair solutions for $\rho > 1$ is because (exactly) proportionally fair solutions may not exist [8].

In Section 5, we consider the problem of proportional fairness on graphs. Specifically, given an undirected graph $G = (V,E)$, we assume that $\mathcal{N} \subseteq V$, $\mathcal{M} = V$ (i.e. every node in the graph is a feasible cluster center), and the distance between two nodes $u, v \in V$, denoted by $d(u,v)$, is the length of the shortest path connecting them. Notice that d satisfies the triangle inequality.

3 Greedy Capture

In this section, we study the greedy capture algorithm defined by Chen et al. [8]. Put succinctly, the algorithm starts with $X = \emptyset$. It grows a ball at every location in $\mathcal{M}$ at the same rate. As soon as a ball contains at least $\lceil n/k \rceil$ points, the corresponding center is added to X and all the points covered by the ball are removed. As balls continue growing, balls centered at previously added locations in X also continue growing with them and any new points covered by such balls are immediately removed. We refer the reader to the work of Chen et al. for full description of the algorithm. They show that for any metric space, greedy capture is guaranteed to find a $(1+\sqrt{2})$-proportionally fair clustering.

We begin by providing a refined analysis of greedy capture by expressing the approximation ratio in terms of a characteristic of the metric we call the Apollonius radius.

▶ **Definition 2.** *Given $\rho \geq 1$, the ρ-Apollonius radius of a metric $(\mathcal{X}, d)$ is defined as $A_{\mathcal{X},d}(\rho) = \sup_{x,y\in\mathcal{X}} \Delta(\rho,x,y)/d(x,y)$, where $\Delta(\rho,x,y)$ is the radius of the smallest ball centered at some point in $\mathcal{X}$ that contains the entire set $\{p \in \mathcal{X} : \rho \cdot d(p,y) \leq d(p,x)\}$.*

The reason that we term it the Apollonius radius is because the renowned Greek geometer Apollonius of Perga was famously interested in the set $\{p \in \mathbb{R}^t : \rho \cdot d(p,y) \leq d(p,x)\}$ for $d = L^2$, and showed that this set is a ball already when $\rho > 1$. This special structure of L^2 is what will allow us to achieve a better approximation guarantee for it.

▶ **Theorem 3.** *For any metric $(\mathcal{X}, d)$ and $\mathcal{M} = \mathcal{X}$, greedy capture finds a ρ-proportionally fair clustering, where $\rho \geq 1$ is the smallest positive number satisfying $A_{\mathcal{X},d}(\rho) \cdot \frac{\rho+1}{\rho} \leq 1$.*

Proof. Let X be the clustering returned by the algorithm. Suppose that X is not ρ-proportionally fair for some ρ. Then, there exist $S \subseteq \mathcal{N}$ with $|S| \geq \lceil n/k \rceil$ and $y \in \mathcal{M}$ such that $\rho \cdot d(j,y) < d(j,X)$ for all $j \in S$.

Note that this implies $y \notin X$. Let i be the first point in S that was covered during the execution of greedy capture; suppose it was covered by a ball located at $x \in X$.

Note that for each $j \in S$, we have $\rho \cdot d(j,y) < d(j,X) \le d(j,x)$. Hence, $S \subseteq \{p \in \mathcal{X} : \rho \cdot d(p,y) < d(p,x)\}$. Hence, by definition of ρ-Apollonius radius, there exists a ball of radius at most $A_{\mathcal{X},d}(\rho) \cdot d(x,y)$ that contains all points in S. Since i was the first point in S covered by the ball centered at x, we must have $d(i,x) \le A_{\mathcal{X},d}(\rho) \cdot d(x,y)$, otherwise greedy capture would have selected the ball of radius at most $A_{\mathcal{X},d}(\rho) \cdot d(x,y)$ covering all points of S prior to i being covered by the ball centered at x. Further, since $i \in S$, we also have $d(i,y) < d(i,x)/\rho$. Using the triangle inequality, we get

$$d(x,y) \le d(i,x) + d(i,y) < d(i,x) \cdot \frac{\rho+1}{\rho} \le A_{\mathcal{X},d}(\rho) \cdot \frac{\rho+1}{\rho} \cdot d(x,y) \Rightarrow A_{\mathcal{X},d}(\rho) \cdot \frac{\rho+1}{\rho} > 1.$$

We have proved that if X is not ρ-proportionally fair, then $A_{\mathcal{X},d}(\rho) \cdot \frac{\rho+1}{\rho} > 1$. Hence, whenever $A_{\mathcal{X},d}(\rho) \cdot \frac{\rho+1}{\rho} \le 1$, we have that X is ρ-proportionally fair. ◀

Let us argue that Theorem 3 is in fact a refinement of the $(1+\sqrt{2})$-approximation proved by Chen et al. [8] that holds for all metrics.

▶ **Theorem 4.** *For any metric $(\mathcal{X}, d)$, the ρ-Apollonius radius is $A_{\mathcal{X},d}(\rho) \le \frac{1}{\rho-1}$. Hence, greedy capture finds a $(1+\sqrt{2})$-proportionally fair clustering for every metric.*

Proof. Consider the set $\{p \in \mathcal{X} : \rho \cdot d(p,y) \le d(p,x)\}$. For any point p in this set, we have that $\rho \cdot d(p,y) \le d(p,x) \le d(p,y) + d(x,y)$ by the triangle inequality. Hence, $d(p,y) \le d(x,y) \cdot \frac{1}{\rho-1}$ for all p in the set. Thus, a ball centered at $y \in \mathcal{M}$ with radius $\frac{d(x,y)}{\rho-1}$ certainly covers the entire set. Hence, $A_{\mathcal{X},d}(\rho) \le \frac{1}{\rho-1}$. Next, for $\rho = 1+\sqrt{2}$, we have

$$A_{\mathcal{X},d}(\rho) \cdot \frac{\rho+1}{\rho} \le \frac{\rho+1}{\rho \cdot (\rho-1)} = 1.$$

Hence, by Theorem 3, greedy capture finds a $(1+\sqrt{2})$-proportionally fair clustering. ◀

Next, we show that for $d = L^2$, the ρ-Apollonius radius is slightly better, leading to a better 2-approximation guarantee for greedy capture.

▶ **Theorem 5.** *For the metric space $(\mathbb{R}^t, L^2)$, where $t \in \mathbb{N}$, the ρ-Apollonius radius is $A_{\mathbb{R}^t,L^2}(\rho) \le \frac{\rho}{\rho^2-1}$, and hence, greedy capture finds a 2-proportionally fair clustering.*

Proof. For the L^2 norm in a Euclidean space, it is well-known that given $x, y \in \mathbb{R}^t$ and $\rho > 1$, the set of points $\{p \in \mathbb{R}^t : \rho \cdot d(p,y) \le d(p,x)\}$ is a ball of radius $d(x,y) \cdot \frac{\rho}{\rho^2-1}$. This is a simple algebraic exercise; its two-dimensional variant was known to Apollonius himself, after whom the result is named (the derivation is widely available online, e.g., see [11]). This immediately implies that $A_{\mathbb{R}^t,L^2}(\rho) \le \frac{\rho}{\rho^2-1}$.

Now, we have that

$$A_{\mathcal{X},d}(\rho) \cdot \frac{\rho+1}{\rho} \le \frac{\rho}{\rho^2-1} \cdot \frac{\rho+1}{\rho} = \frac{1}{\rho-1}.$$

This quantity is at most 1 when ρ is at least 2. Hence, by Theorem 3, greedy capture finds a 2-proportionally fair clustering for this metric. ◀

The obvious next question then is whether this refinement also provides an improved approximation bound for other distance metrics. Unfortunately, for two other prominent distance metrics, L^1 and L^∞, the answer is no. We show this by providing a direct counterexample where greedy capture finds a clustering that is no better than $(1+\sqrt{2})$-proportionally fair. The proof of the next result appears in the full version of the paper.

▶ **Theorem 6.** *For the metric space $(\mathbb{R}^t, d)$ where $t \ge 2$ and $d \in \{L^1, L^\infty\}$, and $\mathcal{M} = \mathbb{R}^t$, there exists an example in which the clustering produced by greedy capture is not ρ-proportionally fair for $\rho < 1+\sqrt{2}$.*

4 Universal Lower Bounds

In this section, we show lower bounds on approximation to proportional fairness that apply to all algorithms, as opposed to the lower bounds in the previous section that apply only to greedy capture. Chen et al. [8] show that when $\mathcal{N}$, $\mathcal{M}$, and the metric are arbitrary, ρ-proportional fairness cannot be guaranteed for $\rho < 2$. They also consider the special case where $\mathcal{N} = \mathcal{M}$, and prove a slightly weaker lower bound of 1.5. One question that they do not address is whether greedy capture provides better than $(1 + \sqrt{2})$-approximation in this special case; in the full version, we show that this is *not* the case.

In this section, we turn our attention to the case of our interest: $\mathcal{M} = \mathcal{X} = \mathbb{R}^t$ and $d \in \{L^1, L^2, L^{\text{inf}}\}$. When $t = 1$, it is easy to notice that an exactly proportionally fair clustering always exists.[1] When $t \geq 2$, we provide a lower bound of $2/\sqrt{3}$ for $d = L^2$ and a lower bound of 1.4 for $d \in \{L^1, L^\infty\}$.

▶ **Theorem 7.** *For the metric space $(\mathbb{R}^t, L^2)$ where $t \geq 2$ and $\mathcal{M} = \mathbb{R}^t$, there is an example in which no clustering is ρ-proportionally fair for $\rho < 2/\sqrt{3} \approx 1.155$.*

Proof. Once again, we set $t = 2$ without loss of generality. Consider an instance in which $|\mathcal{N}| = 6$ and $k = 3$. Suppose $|\mathcal{N}|$ consists of two isomorphic sets of 3 points each, where each set of 3 points forms an equilateral triangle of length 1 and the two sets are sufficiently far from each other. Then, by the pigeonhole principle, under any clustering X, at least one set of 3 points, say $\{p_1, p_2, p_3\}$, must derive their costs from a single cluster center x.

Let a denote the circumcenter of their triangle. Then, $d(a, p_1) = d(a, p_2) = d(a, p_3) = 1/\sqrt{3}$. Hence, $d(a, p_1) + d(a, p_2) + d(a, p_3) = \sqrt{3}$. Notice that in an equilateral triangle, the circumcenter is also the Fermat point, which minimizes the sum of distances from the three vertices. Hence, for the cluster center x, we have $d(x, p_1) + d(x, p_2) + d(x, p_3) \geq \sqrt{3}$. Without loss of generality, assume $d(x, p_1) \geq d(x, p_2) \geq d(x, p_3)$. Then, $d(x, p_1) + d(x, p_2) \geq 2/\sqrt{3}$.

Now, p_1 and p_2 can deviate, choose a location y on the line joining p_1 and p_2 such that $d(y, p_1)/d(y, p_2) = d(x, p_1)/d(x, p_2)$. Since $d(y, p_1) + d(y, p_2) = d(p_1, p_2) = 1$, this reduces the cost to each point by a factor of $2/\sqrt{3}$. Hence, the clustering is not ρ-proportionally fair for $\rho < 2/\sqrt{3}$. ◀

Note that the lower bound of 1.155 is significantly lower than the upper bound of 2 obtained by greedy capture for L^2 as shown in Theorem 5. Closing the gap is an interesting open question. Next, we show a lower bound for L^1 and L^∞. The proof appears in the full version of the paper.

▶ **Theorem 8.** *For the metric space $(\mathbb{R}^t, d)$, where $t \geq 2$ and $d \in \{L^1, L^\infty\}$, and $\mathcal{M} = \mathbb{R}^t$, there is an example in which no clustering is ρ-proportionally fair for $\rho < 1.4$.*

5 Clustering in Graphs

In this section, we consider the special case where the metric space $(\mathcal{X}, d)$ is induced by an undirected graph $G = (V, E)$. Specifically, we let $\mathcal{X} = V$ be the set of nodes of the graph, and assume that $d(x, y)$ measures the length of the shortest path between nodes x and y. As in the previous sections, we restrict our attention to the $\mathcal{M} = \mathcal{X}$ case, i.e., when every point of the metric space is a feasible cluster center.

[1] For instance, opening a cluster at every n/k-th data point from left to right is proportionally fair.

Graphs are an important special case since clustering and facility location in graphs and networks are very well studied. However, while objectives such as truthfulness [2] and social welfare maximization (or social cost minimization) [13] have received significant attention, fairness has not.

We study proportional fairness for this setting. Our first result shows that when the graph $G = (V, E)$ is a tree, an exact proportionally fair clustering always exists, and can be computed by an efficient algorithm. Intuitively, the algorithm works as follows. We first root the tree at an arbitrary node r to obtain a rooted tree (G, r). We denote with h the height of the rooted tree, and with level(x) the height of node x relative to the root r (with level$(r) = 1$). Let ST(x) denote the subtree of node x (i.e. the set of nodes v such that with level$(v) \geq$ level(x) and the unique path from v to r contains x). The algorithm starts from the highest level (the leaves), opens a center every time it finds a node whose subtree contains at least $\lceil n/k \rceil$ nodes, and deletes this subtree from the graph. At the end, the cost to each node is still defined using the closest node at which a center is opened by the algorithm.

Algorithm 1 Proportionally Fair Clustering for Trees.

```
Root the tree $G$ at an arbitrary node $r$
$X \leftarrow \emptyset$
$G^h \leftarrow G$
for $\ell = h$ to 1 do
    $G^{\ell-1} \leftarrow G^\ell$
    for every $x \in V$ with level$(x) = \ell$ and $|\mathrm{ST}(x)| \geq \lceil n/k \rceil$ do
        $X \leftarrow X \cup \{x\}$
        $G^{\ell-1} \leftarrow G^{\ell-1} \setminus \mathrm{ST}(x)$
if $G^0 \neq \emptyset$ then
    $X \leftarrow X \cup \{r\}$
return $X$
```

▶ **Theorem 9.** *Let $G = (V, E)$ be an undirected tree, (V, d) be the metric induced by G, $\mathcal{N} \subseteq V$, $\mathcal{M} = V$, and $k \in \mathbb{N}$. Then, Algorithm 1 yields a proportionally fair clustering.*

Proof. Let X be the clustering returned by Algorithm 1. First, we notice that X contains at most k centers. This is because every time the algorithm opens a center in the for loop, it deletes at least $\lceil n/k \rceil$ nodes from the graph. Hence, the for loop can add at most k centers. If it adds exactly k centers, then the remaining graph must be empty, and the if condition is not executed. If it adds at most $k - 1$ centers, then even if the root node is added later, there would be at most k centers.

Next, suppose for contradiction that X is not proportionally fair. Hence, there exists a set $S \subseteq V$ with $|S| \geq \lceil n/k \rceil$ and $y \in V$ such that $d(i, y) < d(i, X)$ for all $i \in S$.

For each node $i \in V$, define $p(i)$ to be its closest ancestor in X (i.e. $p(i) \in X$ and $i \in \mathrm{ST}(p(i))$, and $p(i)$ is the node of maximum level satisfying these two conditions). Note that because the algorithm always opens a center at the root node, $p(i)$ is well-defined for each node i.

Further, note that for each $i \in S$, $d(i, X) \leq d(i, p(i))$. And for nodes $j \notin \mathrm{ST}(p(i))$, $d(i, j) > d(i, p(i))$. Hence, the cost to i can only reduce if the deviating center is in $\mathrm{ST}(p(i))$. We now consider two cases.

- **Case 1: $\exists i, i' \in S : p(i) \neq p(i')$.** First, suppose that $p(i)$ and $p(i')$ are siblings (i.e. $\mathrm{ST}(p(i)) \cap \mathrm{ST}(p(i')) = \emptyset$). As i can only improve if the deviating center is in $\mathrm{ST}(p(i))$ and i' can only improve if the deviating center is in $\mathrm{ST}(p(i'))$, we obtain that no deviating center y can reduce the cost to i and i' simultaneously, which is a contradiction.

Next, suppose that $p(i) \in \mathrm{ST}(p(i'))$. Then, we must have $y \in \mathrm{ST}(p(i))$, otherwise the cost to i would not reduce. But then, $d(i', p(i)) \leq d(i', y)$. Hence, the cost to i' does not reduce due to y, which is also a contradiction.

The remaining case of $p(i') \in \mathrm{ST}(p(i))$ is symmetric to the last case.

- **Case 2: $\forall i \in S, p(i) = p^*$.** Let $O = X \cap \mathrm{ST}(p^*) \setminus \{p^*\}$ be the set of open centers in $\mathrm{ST}(p^*)$ except p^* itself. Note that by definition of p^*, we have that if $i \in S$, then $i \notin \mathrm{ST}(o)$ for any $o \in O$.

  This implies that if $y \in \mathrm{ST}(o)$ for some $o \in O$, then for every point $i \in S$, we have $d(i, y) \geq d(i, o) \geq d(i, X)$, meaning that y would not reduce the cost to any point in S. Hence, $y \in \mathrm{ST}(p^*) \setminus \cup_{o \in O} \mathrm{ST}(o)$.

  In other words, if center p^* was opened in the iteration with index ℓ (let $\ell = 0$ if p^* is the root node that was opened outside of the for loop), then $S \cup \{y\} \in G^\ell$ (any point from S or y could not have been deleted in any previous iteration). However, for y to reduce the cost to each $i \in S$, we must have $S \subseteq \mathrm{ST}(y)$. However, then, y is a node of higher level than p^* that still contains at least $|S| \geq \lceil n/k \rceil$ points, so it must have been removed in a previous iteration. This is the desired contradiction.

This concludes the proof. ◀

We remark that Theorem 9 identifies the broadest class of interesting metrics to date for which an exact proportionally fair clustering is known to always exist. This raises an immediate question: what about graphs that are not trees? We can consider the universal lower bound for the $(\mathbb{R}^2, L^1)$ metric from Theorem 8. If we construct a very dense grid graph (in which the shortest path distance mimics the L^1 distance in the plane) in the relevant region of $\mathbb{R}^2$ from that example, we can derive the same lower bound of 1.4 on the approximation ratio to proportional fairness for graphs. Whether better lower bounds exist is an open question. Similarly, the $1 + \sqrt{2}$ upper bound of greedy capture holds for all metrics, including those induced by a graph. However, whether its approximation ratio or running time can be improved for graphs, possibly with special properties such as planarity, remains to be seen.

It is worthwhile noting the special case of $\mathcal{N} = \mathcal{M} = V$, i.e., when every node of the graph is also a data point (besides being a feasible cluster center location). In this case, we do not know whether an exact proportionally fair clustering always exists, and leave this as an interesting open question. That said, we do note that if G is connected and we want to place a large number of clusters $k \geq n/2$, then it can be shown that a proportionally fair clustering exists. This is because a dominating set[2] of any size $k \geq n/2$ is guaranteed to exist in a graph with n nodes [26] and can be computed efficiently [18]. If nodes in such a set are chosen as the cluster centers, then every node in the graph already has cost at most 1. So to deviate, all nodes in the blocking coalition must achieve cost 0. However, since the blocking coalition must contain at least $\lceil n/k \rceil \geq 2$ nodes, this is impossible. Thus, the questions of existence and computation of a proportionally fair clustering in general graphs with $\mathcal{N} = \mathcal{M} = V$ become trivial when $k \geq n/2$, but remain open when $k < n/2$.

[2] A set of nodes is called a dominating set if every node in the graph is either in this set or adjacent to a node in this set.

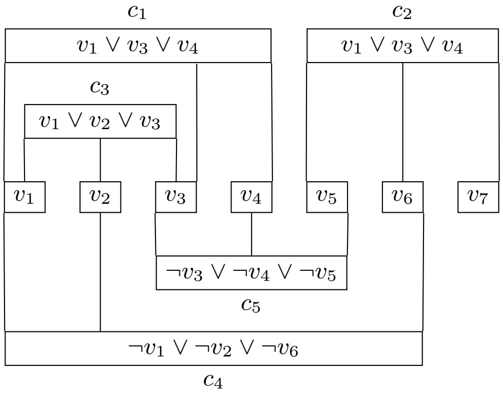


Figure 1 An example of labelling clauses and variables.

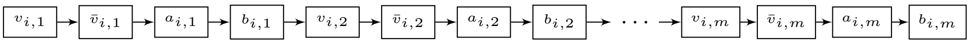


Figure 2 The variable gadget corresponding to variable v_i. Directed edges indicate the closest neighbour of each node.

6 Computational Aspects

In this section, we consider computational aspects of two problems: the problem of checking whether a proportionally fair clustering exists, and the problem of implementing the greedy capture algorithm when $\mathcal{M} = \mathbb{R}^t$. We begin by considering the former problem. The full proof appears in the full version of the paper, but we provide a sketch here.

▶ **Theorem 10.** *Given $\mathcal{X} = \mathbb{R}^2$, finite $N \subset \mathcal{X}$, finite $M \subset \mathcal{X}$, $k \in \mathbb{N}$, and $d = L^2$, checking whether a proportionally fair clustering exists is NP-hard.*

Proof Sketch. We first show a reduction which creates an instance of proportionally fair clustering with $n/k = 2$, and later show how to extend this to the case where n/k is any even integer. We use a polynomial-time reduction from the planar monotone rectilinear 3-SAT problem, where each clause c_j consists of only positive or only negative literals, each variable v_i is represented by a rectangle on the x-axis and each positive (resp. negative) clause is represented by a rectangle above (resp. below) the x-axis with three vertical lines or legs to its three variables. The graph that connects clauses to literals is planar. Figure 1 shows what a planar monotone rectilinear 3-SAT instance looks like.

Let I be an instance of a planar monotone rectilinear 3-SAT which consists of l boolean variables and m monotone clauses. Given I, we construct an instance I' of proportionally fair clustering with $|\mathcal{N}| = \Theta(lm^2)$ and $\mathcal{M} = \Theta(lm^2)$ such that I is satisfiable if and only if there exists a proportionally fair clustering in I'.

First, for each variable v_i, we construct a variable gadget which contains points $v_{i,j}$, $\bar{v}_{i,j}$, $a_{i,j}$ and $b_{i,j}$ for $j \in [m]$. They are all on the line, and belong to both $\mathcal{N}$ and $\mathcal{M}$. This gadget is shown in Figure 2. The point $v_{i,j}$ (resp. $\bar{v}_{i,j}$) corresponds to the positive (resp. negative) literal of v_i, specifically reserved for clause c_j (whether or not it appears in that clause). We set the distances in a way that the closest node to any node is the node on its right, while the closest node to the last node $b_{i,m}$ is its previous node $a_{i,m}$.

All variable gadgets are located on the x-axis in such a way that the gadget of variable v_i is on the left of the gadget of variable v_{i+1}, and from left to right, the distances between two adjacent variable gadgets are slightly decreasing. (The exact construction is provided in the full proof.)

Next, we construct two clause gadgets, a basic and an auxiliary one, for each clause. The auxiliary gadget of each clause is sufficiently far from all other gadgets in the construction. It consists of 3 points and 3 feasible centers such that at least two centers must be placed within each such gadget, otherwise, ρ-proportional fairness is violated for all $\rho < 1.214$. The purpose of this gadget is to make fewer than a proportional number of centers available for the rest of the construction.

In the basic gadget of a clause, we add points that form a rectangle with three legs. The interaction of this gadget with variable gadgets is shown in Figure 3. Let y_j be the y-coordinate where we place all the points that consist the virtual rectangle of c_j. We choose the values of $y_i, i \in [m]$, such that each point in a basic gadget has as closest neighbour a point in the same gadget. Let c_j be a positive clause which contains the positive literals of variables v_q, v_s and v_t with $q < s < t$ (respectively, if the clause is negative). We add a virtual rectangle in the interval $[(v_{q,j}, y_j), (v_{t,j}, y_j)]$ and add the three virtual legs that are vertical at x-coordinates $v_{q,j}$, $v_{s,j}$, and $v_{t,j}$, respectively. First, we place $2m(t-s)$ points in the interval $[(\bar{v}_{s,j}, 0), (v_{t,j}, 0)]$. Denote these points as $r^1_{j,k}$, $k \in [1, 2m(t-s)]$, where the x-coordinate of $r^1_{j,k}$ is less than the x-coordinate of $r^1_{j,k+1}$, and we set the distances such that $r^1_{j,k}$ has as its closest neighbor $r^1_{j,k+1}$. Second, we place in a similar way $2m(s-q)$ points in the interval $[(v_{q,j}, y_j), (a_{s,j-1}, y_j)]$ (or $[(v_{qj}, y_j), (a_{s-1,m}, y_j)]$ if $j = 1$). We denote these points as $r^2_{j,k}$, $k \in [1, 2m(s-q)]$, where the x-coordinate of point $r^2_{j,k}$ is larger than the x-coordinate of point $r^2_{j,k+1}$, and we set the distances such that $r^2_{j,k}$ has as its closest neighbor $r^2_{j,k+1}$.

It remains to construct the legs of each clause. First, we place $2n_j$ points (see the full version for the exact value of n_j), denoted by $l^1_{j,k}$, $k \in [1, 2n_j]$, with x-coordinate equal to $v_{q,j}$ and y-coordinate less than y_j. Specifically, we locate $l^1_{j,1}$ in a position such that the leftmost point in the rectangle of c_j has as its closest neighbor $l^1_{j,1}$, the closest neighbor of every $l^1_{j,k}$, $k \in [1, 2n_j - 1]$, is $l^1_{j,k+1}$, and the closest neighbor of $l^1_{j,2n_j}$ is $v_{q,j}$. For the remaining legs, we add points at exactly the same y-coordinates of the points in the first leg, but with x-coordinates equal to $v_{s,j}$ and $v_{t,j}$. Denote the points of the middle and the right leg as $l^2_{j,k}$ and $l^3_{j,k}$, $k \in [1, 2tn_j]$, respectively.

Lastly, for each clause we add one more point, denoted by o_j in a location such that it is the circumcenter of the triangle with nodes $r^1_{j,1}$, $r^2_{j,1}$, and $l^2_{j,1}$, and these are the (tied) closest neighbors of o_j. Figure 3 shows this entire construction for an example instance which consists of 2 positive clauses (only one of which is shown in the figure) and 3 variables.

In this construction, note that each clause gadget (the union of basic and auxiliary gadgets) has an even number of points equal to $2n'_j + 4$ (for some n'_j; see the full version for details), and each variable gadget has $4m$ of points. Hence, we choose $k = 2ml + \sum_{j=1}^{m}(n'_j + 2m)$, so that $n/k = 2$. Now, we are ready to prove that I is satisfiable if and only if there exists a proportionally fair clustering in I'.

Note that in each variable gadget we need at least $2m$ cluster centers in order for the clustering to be proportionally fair. This is because every pair of adjacent points can deviate if neither of them is a cluster center. There are only two ways to place exactly $2m$ centers: we can either open centers at $v_{i,j}$ and $a_{i,j}$ for all $j \in [4m]$, or open centers at $\bar{v}_{i,j}$ and $b_{i,j}$ for $j \in [4m]$. The first choice corresponds to an assignment where x_i is set to true, while the second corresponds to an assignment where x_i is set to false.

As we mentioned earlier, the auxiliary gadget of each clause needs at least two cluster centers placed within it, otherwise a proportionally fair clustering cannot exist. This leaves n'_j centers for every basic clause gadget. However, each basic gadget needs at least n'_j centers. To see this, notice that from point $r^2_{j,1}$ to the last point of the left leg, we need to add one center at every other point. Similarly, from point $r^1_{j,1}$ to the last point of the right leg, we

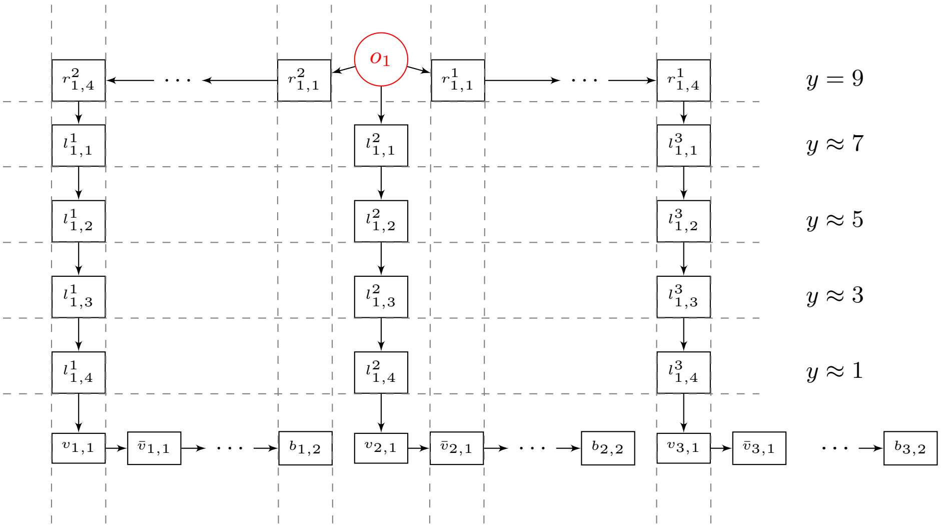


Figure 3 The rectangle and the legs of clause $c_1 = v_1 \vee v_2 \vee v_3$ in an instance with $m = 2$ and $l = 3$. A directed edge from one node points to the closest neighbor of the node.

also need a center at every other point. The same holds for the middle leg. As these are $2n'_j$ points in total, this requires at least n'_j centers in an alternating pattern. Notice that at least one of $r^1_{j,1}$, $r^2_{j,1}$ and $l^2_{j,1}$ must also be an open center, otherwise o_j can deviate with one of them. This is possible only if at least one of the corresponding variable nodes is an open center. The reason is that the last point of at least one leg is not an open center and this point should not want to deviate with its closest node (which corresponds to the literal of the clause). This happens if and only if this node is a center, and so the clause is satisfied.

This shows that I is satisfiable if and only if I' admits a proportionally fair clustering. ◀

Next, we consider implementing the greedy capture algorithm when $\mathcal{M} = \mathbb{R}^t$. Note that because the naive description of greedy capture requires simultaneously growing a ball from each point in $\mathcal{M}$, it is easy to implement when $\mathcal{M}$ is finite (as shown by Chen et al. [8]) but difficult when $\mathcal{M}$ is infinite.

To avoid this issue, one may consider restricting the set of feasible cluster center locations to a finite set $\mathcal{M}'$, and hope that running greedy capture (or any algorithm for that matter) on this finite set still yields a clustering that is approximately fair with respect to the original infinite $\mathcal{M}$. To that end, we show that restricting to $\mathcal{M}' = \mathcal{N}$ (i.e. choosing cluster centers from the set of data points) can only worsen the approximation factor of an algorithm by a factor of at most 2. The proof appears in the full version.

▶ **Theorem 11.** *For any metric space $(\mathcal{X}, d)$, $k \in \mathbb{N}$, $\mathcal{N}, \mathcal{M} \subseteq \mathcal{X}$, and $\rho \geq 1$, any ρ-proportionally fair k-clustering with respect to $(\mathcal{N}, \mathcal{M}' = \mathcal{N})$ is 2ρ-proportionally fair with respect to $(\mathcal{N}, \mathcal{M})$.*

For the metric $(\mathbb{R}^t, L^2)$, where $t \in \mathbb{N}$, recall that greedy capture produces a 2-proportionally fair clustering when $\mathcal{M} = \mathbb{R}^t$. Hence, Theorem 11 implies that running it on $\mathcal{M}' = \mathcal{N}$, for finite $\mathcal{N}$, would yield a 4-proportionally fair clustering in polynomial time. However, for this special case, we can in fact efficiently achieve $2 + \epsilon$ approximation for any constant $\epsilon > 0$, as we discuss below.

Let us consider the first cluster center added by greedy capture when $\mathcal{M} = \mathbb{R}^t$. It is the center of a smallest ball that contains at least n/k of the n given points. The problem of finding the smallest ball containing at least p of n given points is a well-studied problem in computational geometry. This is known to be NP-hard [31], which is what makes implementing greedy capture hard when $\mathcal{M} = \mathbb{R}^t$.

However, when t is constant, the problem is known to be solvable efficiently [31]; we show that in this case, greedy capture can also be implemented efficiently, and we can achieve the 2-approximation from Theorem 5.

Further, even when t is large, the problem admits a PTAS [31]. We show that this PTAS can be used to approximately implement greedy capture, and the approximation ratio only slightly worsens. The proof of the next result appears in the full version.

▶ **Theorem 12.** *Let $t \in \mathbb{N}$, finite $\mathcal{N} \subset \mathbb{R}^t$, and $k \in \mathbb{N}$ be given as input. Suppose $\mathcal{M} = \mathbb{R}^t$ and $d = L^2$. Then, the following hold.*

1. *The clustering returned by greedy capture algorithm cannot be computed in polynomial time unless $P = NP$.*
2. *If t is constant, then it can be computed in polynomial time.*
3. *Even if t is not constant, for any constant $\epsilon > 0$, there exists a polynomial-time algorithm which finds a $(2 + \epsilon)$-proportionally fair clustering.*

7 Learning Fair Clustering

A key concern in machine learning is generalization. In our context, the question is whether a clustering that is (approximately) proportionally fair with respect to random samples taken from an underlying population would remain (approximately) proportionally fair with respect to the whole population. A positive answer to this question could be useful in two ways.

First, sometimes we may have access only to samples from an underlying population. In this case, we can rest assured that by computing a clustering that is fair with respect to the samples, it is also fair with respect to the population. Second, even if the entire population is known, it may be very large. As we noticed in Section 6, finding a proportionally fair clustering or even running the greedy capture algorithm is NP-hard; thus, these tasks may be infeasible for a large population. However, it may be possible to do so on a smaller sample taken from the population, which is where the generalization guarantee can be useful.

Chen et al. [8] show that generalization indeed holds for proportional fairness. Specifically, they define the following relaxation of ρ-proportional fairness.

▶ **Definition 13.** *We say that a k-clustering X is ρ-proportionally fair to $(1 + \epsilon)$-deviations with respect to $\mathcal{N}$ if for all $S \subseteq \mathcal{N}$ with $|S| \geq |\mathcal{N}| \cdot (1 + \epsilon)/k$ and all $y \in \mathcal{M}$, there exists at least one $i \in S$ such that $\rho \cdot d(i, y) \geq d(i, X)$.*

Chen et al. show that if $N \subseteq \mathcal{N}$ is a uniformly random sample of size $|N| = \Omega\left(\frac{k^3}{\epsilon} \ln \frac{|\mathcal{M}|}{\delta}\right)$, and if X is ρ-proportionally fair with respect to N, then X is ρ-proportionally fair to $(1+\epsilon)$-deviations with respect to $\mathcal{N}$ with probability at least $1 - \delta$.

Unfortunately, this bound depends on $|\mathcal{M}|$, and breaks down when $|\mathcal{M}|$ is infinite, which is the focus of our work. We establish a stronger guarantee that does not depend on $|\mathcal{M}|$ by utilizing the framework of VC dimension [33] for binary classifiers. First, we show that there is a natural family of binary classifiers associated with a given clustering.

▶ **Definition 14.** *Given a set of points $\mathcal{N}$, a k-clustering $X \in [\mathcal{M}]^k$, and $y \in \mathcal{M}$, define the binary classifier $h_{X,y} : \mathcal{N} \to \{0, 1\}$ such that $h_{X,y}(i) = 1$ if and only if $\rho \cdot d(i, y) < d(i, X)$. Define the "error" of this classifier on a set of points $S \subseteq \mathcal{N}$ as $err_S(h_{X,y}) = (1/|S|) \cdot \sum_{i \in S} h_{X,y}(i)$.*

Intuitively, $h_{X,y}(i) = 1$ if and only if i can be part of a coalition that complains about the unfairness of X by demonstrating y as a location that provides them ρ-improvement. The use of the term "error" may be confusing. Unlike in traditional classification context, where there is a true classifier and the error is measured in terms of the fraction of points on which a given classifier differs from the true classifier, in our case the "error" is simply the fraction of points that can deviate. One can equivalently think of the "true classifier" as the one that outputs 0 on every point.

Note that X is ρ-proportionally fair to $(1+\epsilon)$-deviations with respect to $\mathcal{N}$ if and only if $\mathrm{err}_{\mathcal{N}}(X, y) \leq \frac{1+\epsilon}{k}$ for all $y \in \mathcal{M}$. Our goal is to show that given a sufficiently large random sample $N \subseteq \mathcal{N}$, if we have a clustering X that is ρ-proportionally fair with respect to N, then it is ρ-proportionally fair to $(1+\epsilon)$-deviations with respect to $\mathcal{N}$ with high probability. However, note that we have no control over what X or y are. This is where the stronger "uniform convergence" guarantee – this establishes that $|\mathrm{err}_N(h_{X,y}) - \mathrm{err}_{\mathcal{N}}(h_{X,y})|$ is bounded for all X, y – becomes useful. Let us begin by introducing the VC dimension and a well-known uniform convergence guarantee that depends on the VC dimension.

▶ **Definition 15** (VC Dimension). *Let $\mathcal{N}$ be a set of points. Let $\mathcal{H}$ be a family of binary classifiers over $\mathcal{N}$. We say that $\mathcal{H}$* shatters *$S \subseteq \mathcal{N}$ if for every labeling $\ell : S \to \{0, 1\}$, there exists a classifier $h \in \mathcal{H}$ such that $h(i) = \ell(i)$ for all $i \in S$. The VC dimension of $\mathcal{H}$, denoted* $\dim^{\mathrm{VC}}(\mathcal{H})$*, is the size of the largest S that can be shattered by $\mathcal{H}$.*

▶ **Proposition 16** ([29]). *Let $\mathcal{H}$ be a family of binary classifiers over a set of points $\mathcal{N}$. If $N \subseteq \mathcal{N}$ is a uniformly random sample with $|N| \geq \Omega\left(\frac{\dim^{\mathrm{VC}}(\mathcal{H})+\ln(1/\delta)}{\epsilon^2}\right)$, then with probability at least $1 - \delta$, $|err_N(h) - err_{\mathcal{N}}(h)| \leq \epsilon$ for all $h \in \mathcal{H}$.*

We show that the family of classifiers $\{h_{X,y} | X \in [\mathcal{M}]^k, y \in \mathcal{M}\}$ has finite VC dimension when $\mathcal{M} = \mathbb{R}^t$ with finite t. This, along with Proposition 16, gives us the desired result.

▶ **Theorem 17.** *Fix $\epsilon, \delta > 0$, $\rho \geq 1$, $k, t \in \mathbb{N}$, and metric $(\mathbb{R}^t, d)$ where $d = L^2$. Let $\mathcal{N}$ be a set of points and $\mathcal{M} = \mathbb{R}^t$ be the set of feasible cluster centers. Let $N \subseteq \mathcal{N}$ be sampled uniformly at random with $|N| \geq \Omega\left(\frac{k^2\cdot(tk\ln k+\ln(1/\delta))}{\epsilon^2}\right)$. Then, with probability at least $1 - \delta$, every k-clustering $X \in [\mathcal{M}]^k$ that is ρ-proportionally fair with respect to N is ρ-proportionally fair to $(1 + \epsilon)$-deviations with respect to $\mathcal{N}$.*

Proof. Given a pair of points x and y, note that the set of points i such that $\rho d(i, y) \geq d(i, x)$ is a half-space in $\mathbb{R}^t$ when $\rho = 1$ and a ball in $\mathbb{R}^t$ when $\rho > 1$. Hence, given $X \in [\mathcal{M}]^k$, the set of points i satisfying $\rho d(i, y) \geq d(i, X)$ is the union of k half-spaces or balls in $\mathbb{R}^t$. It is known that the VC dimension of unions of k half-spaces or balls in $\mathbb{R}^t$ is $O(tk \ln k)$ [6].

Substituting this bound in Proposition 16, we get that $|N| \geq \Omega\left(\frac{k^2\cdot(tk\ln k+\ln(1/\delta))}{\epsilon^2}\right)$ is sufficient to ensure that with probability at least $1 - \delta$, $\mathrm{err}_{\mathcal{N}}(h_{X,y}) \leq \mathrm{err}_N(h_{X,y}) + \epsilon/k$ for all X, y. In particular, when X is ρ-proportionally fair with respect to N, this ensures that with probability at least $1 - \delta$, X is ρ-proportionally fair to $(1 + \epsilon)$-deviations with respect to $\mathcal{N}$. ◀

While we do not formally consider the case of infinite set of points ($|\mathcal{N}| = \infty$) in this work, one can define $\mathcal{N}$ as a distribution over infinitely many points, and ask whether a probability mass of at least $1/k$ has a beneficial deviation. Note that Theorem 17 applies to this case as well because it does not depend on $|\mathcal{N}|$. On the other hand, note that Theorem 17 applies only for L^2 distance metric; deriving generalization bounds for other distance metrics remains an interesting challenge for future work.

8 Discussion

In this work, we advanced the study of proportionally fair clustering in a metric space, and focused on the case where the set of possible cluster center $\mathcal{M}$ is the entire (usually infinite) metric space. Our work leaves a number of open questions.

The most immediate question is to bridge the gap between our lower and upper bounds on the approximation ratio to proportional fairness from Sections 3 and 4. In particular, we believe that for L^2, the lower bound of $2/\sqrt{3}$ from Theorem 7 may be achievable. This specific number is reminiscent of Jung's theorem, which states that for L^2 distance in $\mathbb{R}^2$ (which is where the lower bound stems from), any set of points with diameter at most 1 is contained in a ball of radius at most $1/\sqrt{3}$. This could be useful in closing the gap for L^2. Section 5 leaves open an important question which is whether a proportionally fair clustering exists for all graphs when $\mathcal{N} = \mathcal{M}$ is the set of all nodes of the graph. Our hardness results from Section 6 and learnability results from Section 7 only apply to the L^2 distance because they use results and techniques from the literature that are only available for L^2. Deriving similar results for other distance metrics would be very interesting. While Chen et al. [8] show that achieving bounded approximation to proportional fairness is incompatible with achieving bounded approximation to three classic objective functions (k-center, k-means, and k-median), it would be interesting to study the tradeoff between fairness and these objectives in special cases (such as graphs).

In this paper, our focus was on the case where the set of possible cluster centers $\mathcal{M}$ is the entire metric space. In the case where the metric is induced by a graph $G = (V, E)$ (Section 5), it is also interesting to consider the case where $\mathcal{M} \subset V$. Does a proportionally fair clustering always exist for trees when $\mathcal{M} \subset V$, or for general graphs when $\mathcal{N} = \mathcal{M} \subset V$? These questions remain open.

More broadly, we find proportional fairness to be a very elegant fairness solution concept for clustering. Adapting this idea of proportional fairness to other machine learning settings such as regression or classification can lead to many avenues for future work.

References

1 Alekh Agarwal, Miroslav Dudík, and Zhiwei Steven Wu. Fair regression: Quantitative definitions and reduction-based algorithms. In *Proceedings of the 36thInternational Conference on Machine Learning (ICML)*, volume 97, pages 120–129, 2019.

2 Noga Alon, Michal Feldman, Ariel D. Procaccia, and Moshe Tennenholtz. Strategyproof approximation of the minimax on networks. *Mathematics of Operations Research*, 35(3):513–526, 2010.

3 Julia Angwin, Jeff Larson, Surya Mattu, and Lauren Kirchner. Machine bias: There's software used across the country to predict future criminals. *And it's biased against blacks. ProPublica*, 23, 2016.

4 Arturs Backurs, Piotr Indyk, Krzysztof Onak, Baruch Schieber, Ali Vakilian, and Tal Wagner. Scalable fair clustering. In *Proceedings of the 36thInternational Conference on Machine Learning (ICML)*, pages 405–413, 2019.

5 Suman Bera, Deeparnab Chakrabarty, Nicolas Flores, and Maryam Negahbani. Fair algorithms for clustering. In *Proceedings of the 32nd Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 4955–4966, 2019.

6 Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth. Learnability and the vapnik-chervonenkis dimension. *Journal of the Association for Computing Machinery*, 36(4):929–965, 1989.

7 Toon Calders, Faisal Kamiran, and Mykola Pechenizkiy. Building classifiers with independency constraints. In *Proceedings of the IEEE International Conference on Data Mining Workshops (ICDM)*, pages 13–18, 2009.

8 Xingyu Chen, Brandon Fain, Charles Lyu, and Kamesh Munagala. Proportionally fair clustering. In *Proceedings of the 36thInternational Conference on Machine Learning (ICML)*, pages 1032–1041, 2019.

9 Flavio Chierichetti, Ravi Kumar, Silvio Lattanzi, and Sergei Vassilvitskii. Fair clustering through fairlets. In *Proceedings of the 31st Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 5036–5044, 2017.

10 Vincent Conitzer, Vincent Freeman, Nisarg Shah, and Jennifer Wortman Vaughan. Group fairness for the allocation of indivisible goods. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI)*, pages 1853–1860, 2019.

11 Joseph Cox and Michael B Partensky. Spatial localization problem and the circle of Apollonius. arXiv, 2007. `arXiv:physics/0701146`.

12 Brandon Fain, Kamesh Munagala, and Nisarg Shah. Fair allocation of indivisible public goods. In *Proceedings of the 19th ACM Conference on Economics and Computation (EC)*, pages 575–592, 2018.

13 Michal Feldman and Yoav Wilf. Strategyproof facility location and the least squares objective. In *Proceedings of the fourteenth ACM conference on Electronic Commerce (EC)*, pages 873–890, 2013.

14 Dimitris Fotakis. Incremental algorithms for facility location and k-median. *Theoretical Computer Science*, 361(2-3):275–313, 2006.

15 Anupam Gupta, Guru Guruganesh, and Melanie Schmidt. Approximation algorithms for aversion k-clustering via local k-median. In *Proceedings of the 43rd International Colloquium on Automata, Languages and Programming (ICALP)*, pages 66:1–66:13, 2016.

16 Moritz Hardt, Eric Price, and Nathan Srebro. Equality of opportunity in supervised learning. In *Proceedings of the 30th Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 3315–3323, 2016.

17 Tatsunori B Hashimoto, Megha Srivastava, Hongseok Namkoong, and Percy Liang. Fairness without demographics in repeated loss minimization. arXiv, 2018. `arXiv:1806.08010`.

18 Stephen T. Hedetniemi, David P. Jacobs, and K. E. Kennedy. A theorem of ore and self-stabilizing algorithms for disjoint minimal dominating sets. *Theoretical Computer Science*, 593:132–138, 2015.

19 Safwan Hossain, Andjela Mladenovic, and Nisarg Shah. Designing fairly fair classifiers via economic fairness notions. In *Proceedings of the 29th International World Wide Web Conference (WWW)*, pages 1559–1569, 2020.

20 Michael Kearns, Seth Neel, Aaron Roth, and Zhiwei Steven Wu. Preventing fairness gerrymandering: Auditing and learning for subgroup fairness. In *Proceedings of the 35thInternational Conference on Machine Learning (ICML)*, pages 2569–2577, 2018.

21 Ke Liao and Diansheng Guo. A clustering-based approach to the capacitated facility location problem 1. *Transactions in GIS*, 12(3):323–339, 2008.

22 Ninareh Mehrabi, Fred Morstatter, Nripsuta Saxena, Kristina Lerman, and Aram Galstyan. A survey on bias and fairness in machine learning. arXiv, 2019. `arXiv:1908.09635`.

23 Hervé Moulin. *Fair Division and Collective Welfare*. MIT Press, 2003.

24 Arvind Narayanan. Translation tutorial: 21 fairness definitions and their politics. In *Proceedings of the Conference on Fairness, Accountability, and Transparency (FAT)*, 2018.

25 Cathy O'Neil. *Weapons of math destruction: How big data increases inequality and threatens democracy*. Broadway Books, 2016.

26 Oystein Ore. Theory of graphs. *American Mathematical Society Colloquium. Providence, R.I.*, 38, 1962.

27 David C. Parkes and Rakesh V. Vohra. Algorithmic and economic perspectives on fairness. arXiv, 2019. `arXiv:1909.05282`.

28 Clemens Rösner and Melanie Schmidt. Privacy preserving clustering with constraints. In *Proceedings of the 45th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 96:1—96:14, 2018.

29 Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms.* Cambridge University Press, 2014.

30 Lloyd Shapley and Herbert Scarf. On cores and indivisibility. *Journal of Mathematical Economics*, 1(1):23–37, 1974.

31 Vladimir Shenmaier. The problem of a minimal ball enclosing k points. *Journal of Applied and Industrial Mathematics*, 7(3):444–448, 2013.

32 Berk Ustun, Yang Liu, and David C. Parkes. Fairness without harm: Decoupled classifiers with preference guarantees. In *Proceedings of the 36thInternational Conference on Machine Learning (ICML)*, pages 6373–6382, 2019.

33 Vladimir Vapnik and Alexey Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16(2):264—280, 1971.

34 Hal Varian. Equity, envy and efficiency. *Journal of Economic Theory*, 9:63–91, 1974.

35 Muhammad Bilal Zafar, Isabel Valera, Manuel Gomez-Rodriguez, and Krishna P. Gummadi. Fairness constraints: A flexible approach for fair classification. *Journal of Machine Learning Research*, 20(75):1–42, 2019.

36 Rich Zemel, Yu Wu, Kevin Swersky, Toni Pitassi, and Cynthia Dwork. Learning fair representations. In *Proceedings of the 30thInternational Conference on Machine Learning (ICML)*, pages 325–333, 2013.

Breaking the Barrier of 2 for the Storage Allocation Problem

Tobias Mömke
Saarland University, Saarland Informatics Campus, Saarbrücken, Germany
moemke@cs.uni-saarland.de

Andreas Wiese
Department of Industrial Engineering, Universidad de Chile, Santiago, Chile
awiese@dii.uchile.cl

Abstract

Packing problems are an important class of optimization problems. The probably most well-known problem if this type is knapsack and many generalizations of it have been studied in the literature like Two-dimensional Geometric Knapsack (2DKP) and Unsplittable Flow on a Path (UFP). For the latter two problems, recently the first polynomial time approximation algorithms with better approximation ratios than 2 were presented [Gálvez et al., FOCS 2017][Grandoni et al., STOC 2018]. In this paper we break the barrier of 2 for the Storage Allocation Problem (SAP), a problem which combines properties of 2DKP and UFP. In SAP, we are given a path with capacitated edges and a set of tasks where each task has a start vertex, an end vertex, a size, and a profit. We seek to select the most profitable set of tasks that we can draw as non-overlapping rectangles underneath the capacity profile of the edges where the height of each rectangle equals the size of the corresponding task.

The problem SAP appears naturally in settings of allocating resources like memory, bandwidth, etc. where each request needs a contiguous portion of the resource. The best known polynomial time approximation algorithm for SAP has an approximation ratio of $2+\varepsilon$ [Mömke and Wiese, ICALP 2015] and no better quasi-polynomial time algorithm is known. We present a polynomial time $(63/32+\varepsilon) < 1.969$-approximation algorithm for the important case of uniform edge capacities and a quasi-polynomial time $(1.997+\varepsilon)$-approximation algorithm for non-uniform quasi-polynomially bounded edge capacities. Key to our results are building blocks consisting of *stair-blocks, jammed tasks,* and *boxes* that we use to construct profitable solutions and which allow us to compute solutions of these types efficiently. Finally, using our techniques we show that under slight resource augmentation we can obtain even approximation ratios of $3/2+\varepsilon$ in polynomial time and $1+\varepsilon$ in quasi-polynomial time, both for arbitrary edge capacities.

2012 ACM Subject Classification Theory of computation → Dynamic programming; Theory of computation → Packing and covering problems; Theory of computation → Rounding techniques

Keywords and phrases Approximation Algorithms, Resource Allocation, Dynamic Programming

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.86

Category Track A: Algorithms, Complexity and Games

Related Version A full version of the paper is available at [28], `https://arxiv.org/abs/1911.10871`.

Funding *Tobias Mömke*: This work was partially supported by the ERC Advanced Investigators Grant 695614 (POWVER), and by DFG Grant 389792660 as part of TRR 248 (CPEC).
Andreas Wiese: Partially supported by FONDECYT Regular grant 1170223.

1 Introduction

Packing problems play an important role in combinatorial optimization. The most basic packing problem is knapsack where we are given a knapsack of a certain capacity, a set of items with different sizes and profits, and we are looking for a subset of items of maximum profit that fit into the knapsack. Many generalizations of it have been studied. For example,

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 86; pp. 86:1–86:19
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

in the Two-dimensional Geometric Knapsack problem (2DKP) the items are axis-parallel rectangles and we seek to find the most profitable subset of them that fit non-overlappingly into a given rectangular knapsack. Another generalization of the knapsack problem is called Unsplittable Flow on a Path (UFP). We are given a path with capacities on its edges and each item can be interpreted as a commodity of flow that needs to send a given amount of flow from its start vertex to its end vertex in case that we select it. If the path consists of a single edge then UFP is identical to knapsack.

In this paper, we study the Storage Allocation Problem (SAP) which combines properties of 2DKP and UFP: We are given a path (V, E) where each edge $e \in E$ has a capacity $u_e \in \mathbb{N}$, and a set of tasks T where each task $i \in T$ is specified by a size $d_i \in \mathbb{N}$, a profit $w_i \in \mathbb{N}$, a start vertex $s_i \in V$, and an end vertex $t_i \in V$. Let $P(i)$ denote the path between s_i and t_i for each $i \in T$. The goal is to select a subset of tasks $T' \subseteq T$ and define a height level $h(i) \geq 0$ for each task $i \in T'$ such that the resulting rectangle $[s_i, t_i) \times [h(i), h(i) + d_i)$ lies within the profile of the edge capacities, and we require that the rectangles of the tasks in T' are pairwise non-overlapping. Formally, for each task $i \in T'$ we require that $h(i) + d_i \leq u_e$ for each edge $e \in P(i)$ and additionally for any two tasks $i, i' \in T'$ we require that if $P(i) \cap P(i') \neq \emptyset$, then $[h(i), h(i) + d_i) \cap [h(i'), h(i') + d_{i'}) = \emptyset$. Note that since we can choose $h(i)$ we can define the vertical position of the rectangle of each task i but not its horizontal position. Again, if E has only one edge then the problem is identical to knapsack.

The problem SAP is motivated by settings in which tasks need a contiguous portion of an available resource, e.g., a consecutive portion of the computer memory or a frequency bandwidth. Observe that in contrast to UFP, in many applications of SAP the instances have uniform edge capacities, e.g., if the available memory or frequency spectrum does not change over time. From a mathematical perspective, SAP and UFP are closely related. Every feasible SAP-solution T' satisfies $\sum_{i \in T': e \in P(i)} d_i \leq u_e$ on each edge e. This is exactly the condition when a solution to UFP is feasible (UFP and SAP have the same type of input). In SAP we require additionally that we can represent the tasks in T' as non-overlapping rectangles. Also, if all edges have the same capacity then SAP can be seen as a variant of 2DKP in which the horizontal coordinate of each item i is fixed and we can choose only the vertical coordinate.

For quite some time, the best known polynomial time approximation ratios for 2DKP and UFP had been $2 + \varepsilon$ [24, 2]. Recently, the barrier of 2 was broken for both problems and algorithms with strictly better approximation ratios have been presented [16, 20]. For SAP, the best known approximation ratio is still $2 + \varepsilon$ [27], even for uniform edge capacities and if we allow quasi-polynomial running time. In contrast, for 2DKP and UFP, better quasi-polynomial time algorithms had been known earlier [3, 9, 1].

1.1 Our contribution

In this paper, we break the barrier of 2 for SAP and present a polynomial time $(63/32 + \varepsilon) < 1.969$-approximation algorithm for uniform edge capacities and a quasi-polynomial time $(1.997 + \varepsilon)$-approximation algorithm for non-uniform edge capacities in a quasi-polynomial range. Key to our results is to identify suitable building blocks to construct profitable near-optimal solutions such that we can design algorithms that find profitable solutions of this type. We call a task *small* if its demand is small compared to the capacity of the edges on its path and *large* otherwise. One can show that each edge can be used by only relatively few large tasks which allows for a dynamic program that finds the best solution with large tasks only. However, there can be many small tasks using an edge and hence this approach fails for small tasks. We therefore consider *boxable solutions* in which the tasks

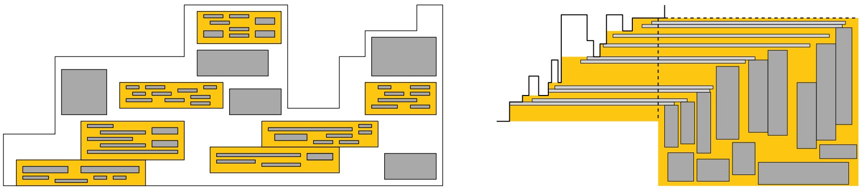


Figure 1 Left: a boxable solution in which the (gray) tasks are assigned into the (orange) boxes. Right: A stair-block into which small tasks (light gray) and large tasks (dark gray) are assigned. All small tasks need to cross the vertical dashed line and all large tasks need to be placed on the right of it underneath the dashed horizontal line. Therefore, the orange area denotes the space that is effectively usable for the tasks that we assign into the stair-block.

are assigned into rectangular boxes such that each edge is used by only $(\log n)^{O(1)}$ of these boxes, see Fig. 1. Using the latter property, we present a quasi-polynomial time algorithm that essentially finds the optimal boxable solution. Furthermore, for many types of instances we prove that there exist boxable solutions with high profit.

There are, however, instances for which it is not clear how to construct boxable solutions that yield a better approximation ratio than 2. This is where our second building block comes into play which are *stair-blocks*. Intuitively, a stair-block is an area into which we assign small and large tasks such that the small tasks are jammed between the large tasks and the capacity profile of the edges, see Fig. 1. We prove the crucial insight that if we fail to construct a good boxable solution then this is because a lot of profit of the optimum is due to small tasks in stair-blocks. We therefore devise a second algorithm that computes solutions for such instances, yielding an approximation ratio better than 2. The algorithm is based on a configuration-LP with a variable for each possible set of large tasks in each stair-block and additionally variables for placing the small tasks in the remaining space. We separate it via the dual LP in which the separation problem turns out to be a variation of SAP with large tasks only. Then we sample the set of large tasks according to the probabilities implied by the LP solution. As a result, there are some small tasks that we cannot pick anymore since they would overlap the sampled large tasks. For some small tasks this will happen with very large probability so most likely we will lose their profit. This is problematic if they represent a large fraction of the profit of the LP. We therefore introduce additional constraints that imply that if the latter happens then we can use another rounding routine for small tasks only that yields enough profit.

▶ **Theorem 1.** *There is a quasi-polynomial time* $(1.997+\varepsilon)$*-approximation algorithm for SAP if the edge capacities are quasi-polynomially bounded integers.*

Recall that in many applications of SAP the instances have uniform edge capacities. For our polynomial time algorithm for this setting the above building blocks are not sufficient since for example in our boxable solutions above an edge can be used by more than constantly many boxes and hence we cannot enumerate all possibilities for those in polynomial time. We therefore identify types of boxable solutions that are more structured and that allow us to find profitable solutions of these types in polynomial time. The first such type are boxable solutions in which each edge is used by only constantly many boxes. A major difficulty is here that for a small task there are possibly several boxes that we can assign it to and if we assign it to the wrong box then it occupies space that we should have used for other tasks instead (in our quasi-polynomial time algorithm above we use a method to address this which inherently needs quasi-polynomial time). We solve this issue by guessing the boxes in a suitable hierarchical order which is *not* the canonical linear order given by their

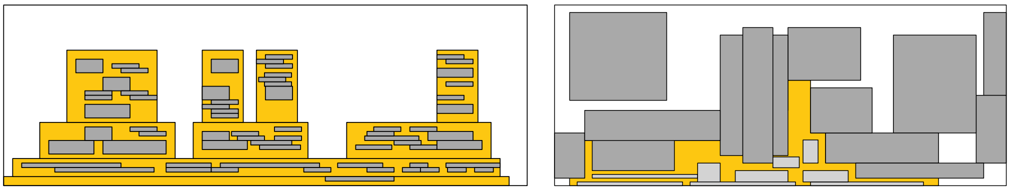


Figure 2 Left: a laminar boxable solution that consists of boxes of geometrically increasing sizes whose paths form a laminar family. Right: a jammed solution in which a set of small tasks (light gray) that are placed underneath some large tasks (dark gray). The small tasks are relatively large compared to the (orange) space underneath the large tasks.

respective leftmost edges and we assign the tasks into the boxes in the guessed order. With a double-counting argument we show that with our strategy we obtain a solution which has essentially at least the profit of the large tasks in the optimal boxable solution of the first type plus half of the profit of the small tasks. Our second special type of boxable solutions is the case in which the paths of the boxes form a laminar family and the sizes of the boxes are geometrically increasing, see Fig. 2. Even though there can be $\Omega(\log n)$ such boxes using an edge, we devise an algorithm with polynomial running time for this kind of solutions. It is a dynamic program inspired by [20] that guesses the boxes in the order given by the laminar family and assigns the tasks into them. Finally, there can be small tasks such that in the optimal solution the large tasks take away so much space that with respect to the remaining space those small tasks actually become relatively large. We say that a solution consisting of such small and large tasks forms a *jammed solution* which is our third type of building block, see Fig. 2. We extend an algorithm in [27] for instances with large tasks only to compute essentially the most profitable jammed solution. Our key technical lemma shows that for any instance there exists a profitable solution that uses only the building blocks above and we provide a polynomial time algorithm that finds such a solution.

▶ **Theorem 2.** *There is a polynomial time* $(63/32 + \varepsilon) < 1.969$*-approximation algorithm for SAP for uniform edge capacities.*

We would like to note that we did not attempt to optimize our approximation ratios up to the third decimal place but instead we focus on a clean exposition of our results (which are already quite complicated). Finally, we study the setting of $(1 + \eta)$-resource augmentation where we can increase the capacity of each edge by a factor of $1 + \eta$ for an arbitrarily small constant $\eta > 0$ while the compared optimal solution cannot do this. In this case we obtain even better approximation ratios and improve the factor of 2 for arbitrary edge capacities even with a polynomial time algorithm. Key for these results is to show that using the resource augmentation we can reduce the general case to the case of a constant range of edge capacities and then establish that there are essentially optimal boxable solutions in which each edge is used by a constant number of boxes. Using our algorithmic tools from above this implies the following theorem.

▶ **Theorem 3.** *In the setting of* $(1 + \eta)$*-resource augmentation there exists a polynomial time* $(3/2 + \varepsilon)$*-approximation algorithm and a quasi-polynomial time* $(1 + \varepsilon)$*-approximation algorithm for SAP with arbitrary edge capacities.*

Due to space constraints, this extended abstract intends to give only an overview of our methodology. For a complete presentation of our results we refer to [28].

1.2 Other related work

Previous to the mentioned $(2+\varepsilon)$-approximation algorithm for SAP [27], Bar-Noy et al. [6] found a 7-approximation algorithm if all edges have the same capacities which was improved by Bar-Yehuda et al. to a $(2+\varepsilon)$-approximation [7]. Bar-Yehuda et al. [8] presented the first constant factor approximation algorithm for SAP for arbitrary capacities, having an approximation ratio of $9+\varepsilon$. A related problem is the dynamic storage allocation problem (DSA) where in the input we are given a set of tasks like in SAP and we all need to pack all of them as non-overlapping rectangles, minimizing the maximum height of a packed item. The best known approximation ratio for DSA is a $(2+\varepsilon)$-approximation which in particular uses a $(1+\varepsilon)$-approximation if all tasks are sufficiently small [11]. This improves earlier results [25, 26, 17, 18].

For 2DKP for squares there is an EPTAS [21] which improves earlier PTASs [22, 23]. For rectangles, there was a $(2+\varepsilon)$-approximation known [24, 23] which was improved to a $(17/9+\varepsilon)$-approximation [16]. There is a PTAS if the profit of each item is proportional to its area [5]. Also, there is a QPTAS for quasi-polynomially bounded input data [1]. For UFP there is a long line of work on the case of uniform edge capacities [29, 6, 12], the no-bottleneck-assumption [13, 15], and the general case [3, 4, 14, 10, 2] which culminated in a QPTAS [3, 9], PTASs for several special cases [19, 9], a $(2+\varepsilon)$- approximation [2], which was improved to a $(5/3+\varepsilon)$-approximation [20].

2 Overview

In this section we present an overview of our methodology for our algorithms. Let $\varepsilon > 0$ and assume that $1/\varepsilon \in \mathbb{N}$. First, we classify tasks into large and small tasks. For each task $i \in T$ let $b(i) := \min_{e\in P(i)} u_e$ denote the *bottleneck capacity* of i. For constants $\mu, \delta > 0$ we define that a task i is *large* if $d_i > \delta \cdot b(i)$ and *small* if $d_i \leq \mu \cdot b(i)$. The constants δ, μ are chosen to be the values δ_{i^*} and μ_{i^*} due to the following lemma, which in particular ensures that the tasks i with $\mu \cdot b(i) < d_i \leq \delta \cdot b(i)$ contribute only a marginal amount to the optimal solution OPT whose weight we denote by opt.

▶ **Lemma 4.** *We can compute a set $(\mu_1, \delta_1), \dots, (\mu_{1/\varepsilon}, \delta_{1/\varepsilon})$ such that for each tuple (μ_k, δ_k) we have $\varepsilon^{O\left((1/\varepsilon)^{1/\varepsilon}\right)} \leq \mu_k \leq \varepsilon^{10}\delta_k^{1/\varepsilon}$, $\delta_i \leq \varepsilon$ and for one tuple $(\mu_{k^*}, \delta_{k^*})$ it holds that $w(\mathrm{OPT} \cap \{i \in T \mid \mu_{k^*} \cdot b(i) < d_i \leq \delta_{k^*} \cdot b(i)\}) \leq \varepsilon \cdot \mathrm{opt}$.*

Let T_L and T_S denote the sets of large and small input tasks, respectively. For each edge e let $T_e \subseteq T$ denote the set of tasks $i \in T$ for which $e \in P(i)$. We will show later that for many instances there are profitable solutions that are *boxable* which intuitively means that the tasks can be assigned into rectangular boxes such that each edge is used by only few boxes. A *box* B is defined by a start vertex s_B, an end vertex t_B, and a size d_B. We define $P(B)$ to be the path of B which is the path between s_B and t_B. A set of tasks $T' \subseteq T$ *fits* into B if

- for each $i \in T'$ we have that $P(i) \subseteq P(B)$, and
- there is a value $h(i) \in [0, d_B)$ for each $i \in T'$ such that (T', h) is feasible if each edge $e \in P(B)$ has capacity d_B, and
- $|T'| = 1$ or we have $d_i \leq \varepsilon^8 \cdot d_B$ for each $i \in T'$.

We say that a set of boxes $\mathcal{B}$ and a height level assignment $h : \mathcal{B} \to \mathbb{N}$ forms a feasible solution $(\mathcal{B}, h)$ if the boxes in $\mathcal{B}$ interpreted as tasks form a feasible solution with h (see Fig. 1), i.e., if the set $(T(\mathcal{B}), h')$ is feasible where $T(\mathcal{B})$ contains a task $i(B)$ for each $B \in \mathcal{B}$ such that $P(i(B)) = P(B)$, $d_i = d_{i(B)}$ and $h'(i(B)) = h(B)$.

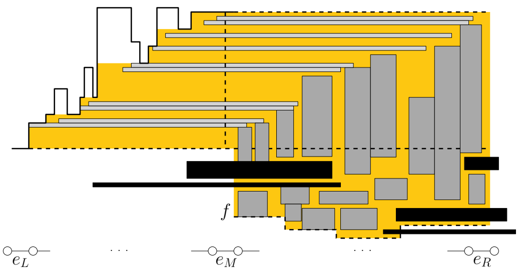


Figure 3 A stair-block $\mathcal{SB} = (e_L, e_M, e_R, f, T'_L, h')$. The black tasks are the tasks in T'_L. The orange area denotes the area that is effectively usable for the tasks that we assign to $\mathcal{SB}$. The light and dark gray tasks are small and large tasks, respectively, that together fit into $\mathcal{SB}$.

▶ **Definition 5.** *A solution (T', h') is a β-*boxable solution *if there exists a set of boxes $\mathcal{B} = \{B_1, \dots, B_{|\mathcal{B}|}\}$ and a partition $T' = T'_1 \dot\cup \dots \dot\cup T'_{|\mathcal{B}|}$ such that*

- *for each $j \in [|\mathcal{B}|]$, T'_j fits into the box B_j and if $T'_j \cap T_L \neq \emptyset$ then $|T'_j| = 1$,*
- *each edge $e \in E$ is used by the paths of at most β boxes in $\mathcal{B}$,*
- *there is a height level $h'(B)$ for each box $B \in \mathcal{B}$ such that $(\mathcal{B}, h')$ is feasible.*

In the following lemma we present an algorithm that essentially computes the optimal β-boxable solution. We will use it later with $\beta = (\log n)^{O(1)}$. Assume in the sequel that we are given a SAP-instance where $u_e \leq n^{(\log n)^c}$ for some $c \in \mathbb{N}$ for each $e \in E$.

▶ **Lemma 6.** *Let $\beta \in \mathbb{N}$ and let $(T_{\text{box}}, h_{\text{box}})$ be a β-boxable solution. There is an algorithm with running time $n^{(\beta \log n / aw\varepsilon)^{O(c)}}$ that computes a β-boxable solution (T', h') with $w(T') \geq w(T_{\text{box}})/(1+\varepsilon)$.*

Our second type of solutions are composed by *stair-blocks* (see Fig. 1 and Fig. 3). Intuitively, a stair-block is an area underneath the capacity profile defined by a function $f\colon E \to \mathbb{N}_0$ and three edges e_L, e_M, e_R, where e_M lies between e_L and e_R. The corresponding area contains all points above each edge between e_L and e_M whose y-coordinate is at least u_{e_L} and all points above each edge e between e_M and e_R whose y-coordinate is in $[f_e, u_{e_M})$. Additionally, there are some tasks $T'_L \subseteq T_L \cap (T_{e_M} \cup T_{e_R})$ and a function $h' : T'_L \to \mathbb{N}_0$ that assigns height levels to them where the intuition is that those tasks are given in advance and fixed. We require that each of them intersects the mentioned area below u_{e_L}, i.e., for each $i \in T'_L$ we have that $h'(i) + d_i \leq u_{e_L}$ and there is an edge $e \in P(i) \cap P_{e_M, e_R} \setminus \{e_M\}$ such that $h'(i) + d_i > f_e$ where P_{e_M, e_R} is the path that starts with e_M and ends with e_R. Also, we require that $f(e) = u_{e_L}$ for $e = e_M$ and each edge e on the left of e_M.

Given a stair-block, we will assign tasks T'' into the mentioned area such that we require that all small tasks in T'' use e_M and for each large tasks $i \in T''$ we require that $P(i) \subseteq P_{e_M, e_R}$. Due to the former condition, not all points with x-coordinate between e_L and e_M are actually usable for tasks assigned to $\mathcal{SB}$ and the usable ones form a staircase shape (see Fig. 1). Formally, we say that a solution (T'', h'') *fits into a stair-block* $\mathcal{SB} = (e_L, e_M, e_R, f, T'_L, h')$ if $P(i) \subseteq P_{e_M, e_R}$ and $f_e \leq h''(i) \leq u_{e_M} - d_i$ for each $i \in T'' \cap T_L$ and each $e \in P(i)$, $h''(i') \geq u_{e_L}$ and $i' \in T_{e_M}$ for each $i' \in T'' \cap T_S$, and additionally $(T'_L \cup T'', h' \cup h'')$ forms a feasible solution. Also, we require that $h''(i) < d_i$ for each $i \in T'' \cap T_L$ which is a technical condition that we need later in order to be able to compute a profitable stair solution efficiently. A set of tasks T'' fits into a stair-block $\mathcal{SB}$, if there is a function h'' such that the solution (T'', h'') fits into $\mathcal{SB}$. We will need later that the function f is simple and to this end we say that a stair-block $\mathcal{SB} = (e_L, e_M, e_R, f, T'_L, h')$ is a γ-*stair-block* if f is a a step-function with at most γ steps. Note that it can happen that e_R lies on the left of e_L and then we define P_{e_M,e_R} to be the path that starts with e_R and ends with e_M (one may imagine that Fig. 1 is mirrored).

We seek solutions that consist of stair-blocks and large tasks that are compatible with each other. To this end, for a stair-block $\mathcal{SB} = (e_L, e_M, e_R, f, T'_L, h')$ we define $P(\mathcal{SB})$ to be the path starting with the edge on the right of e_L and ending with e_R. A large task i with height $h(i)$ is *compatible with* $\mathcal{SB}$ if $i \notin T'_L$ and intuitively i does not intersect the area of the stair-block, i.e., if $h(i) \geq u_{e_M}$ or $h(i) + d_i \leq f_e$ for each $e \in P(i) \cap P(\mathcal{SB})$. We say that a task $i \in T_L$ with height $h(i)$ is *part of* $\mathcal{SB}$ if $i \in T'_L$ and $h(i) = h'(i)$. We say that stair-blocks $\mathcal{SB} = (e_L, e_M, e_R, f, T'_L, h')$ and $\overline{\mathcal{SB}} = (\bar{e}_L, \bar{e}_M, \bar{e}_R, \bar{f}, \bar{T}'_L, \bar{h}')$ are *compatible* if for each task $i \in \bar{T}'_L \cap T'_L$ we have $h'(i) = \bar{h}'(i)$, each task $i \in \bar{T}'{}_L \setminus T'_L$ is compatible with $\mathcal{SB}$, each task $i \in T'_L \setminus \bar{T}'_L$ is compatible with $\overline{\mathcal{SB}}$, and there is no task $i \in T$ that fits into both $\mathcal{SB}$ and $\overline{\mathcal{SB}}$ (for suitable heights $h''(i)$ and $\bar{h}''(i)$). Intuitively, a stair-solution consists of a set of stair-blocks and a set of large tasks T^0_L that are all compatible with each other.

▶ **Definition 7.** *A solution* (T'', h'') *is a* γ-stair-solution *if there exists a set of* γ*-stair-blocks* $\{\mathcal{SB}_1, \ldots, \mathcal{SB}_k\}$ *and partitions* $T'' \cap T_L = T^0_L \dot\cup T^1_L \dot\cup \ldots \dot\cup T^k_L$ *and* $T'' \cap T_S = T^1_S \dot\cup \ldots \dot\cup T^k_S$ *such that for each* $j \in [k]$*, the tasks* $T^j_L \cup T^j_S$ *fit into* $\mathcal{SB}_j$*, for any* $j, j' \in [|\mathcal{SB}|]$ *the stair-blocks* SB_j *and* $SB_{j'}$ *are compatible, for each stair-block* $\mathcal{SB}_j$ *and each task* $i \in T^0_L$ *with height* $h''(i)$*, the task* i *is compatible with* $\mathcal{SB}_j$ *or part of* $\mathcal{SB}_j$*, and each edge is contained in the path* $P(i)$ *of at most* γ *tasks* $i \in T^0_L$ *and in the path* $P(\mathcal{SB}_j)$ *of at most* γ *stair-blocks* $\mathcal{SB}_j$*.*

Our main structural lemma is that there exists a boxable solution or a stair solution whose profit is large enough so that we can get an approximation ratio better than 2.

▶ **Lemma 8** (Structural lemma). *There exists a* $(\log n/\delta^2)^{O(c+1)}$*-boxable solution* T_{box} *such that* $w(T_{\text{box}}) \geq \text{opt}/(1.997 + \varepsilon)$ *or there exists a* $(\log n/O(\delta))^{O(c+1)}$*-stair-solution* T_{stair} *with* $w(T_S \cap T_{\text{stair}}) \geq \frac{1}{\alpha} w(T_L \cap T_{\text{stair}})$ *for some value* $\alpha \geq 1$ *such that* $w(T_{\text{stair}} \cap T_L) + \frac{1}{8(\alpha+1)} w(T_{\text{stair}} \cap T_S) \geq \text{opt}/(1.997 + \varepsilon)$*.*

If the first case of Lemma 8 applies then the algorithm due to Lemma 6 yields a $(1.997+\varepsilon)$-approximation. In the second case the following algorithm yields a $(1.997+\varepsilon)$-approximation which completes the proof of Theorem 1.

▶ **Lemma 9.** *Let* $(T_{\text{stair}}, h_{\text{stair}})$ *be a* γ*-stair solution with* $w(T_S \cap T_{\text{stair}}) \geq \frac{1}{\alpha} w(T_L \cap T_{\text{stair}})$ *for some value* $\alpha \geq 1$*. There is an algorithm with running time* $(n \cdot \max_e u_e)^{O_\delta(\gamma^2 \log(\max_e u_e))}$ *that computes a stair solution* (T', h') *with* $w(T') \geq (1 - O(\varepsilon))(w(T_{\text{stair}} \cap T_L) + \frac{1}{8(\alpha+1)} w(T_{\text{stair}} \cap T_S))$*.*

2.1 Uniform edge capacities

Assume now that all edge capacities are identical, i.e., that there exists a value U such that $u_e = U$ for each edge $e \in E$ but that not necessarily $U \leq n^{(\log n)^c}$. For this case we want to design a *polynomial* time $(63/32 + \varepsilon)$-approximation algorithm. The above building blocks are not sufficient since the corresponding algorithms need quasi-polynomial time. Therefore, first we consider special cases of boxable solutions for which we design polynomial time algorithms. We begin with such an algorithm for β-boxable solutions for constant β that intuitively collects all the profit from the large tasks in the optimal β-boxable solution and half of the profit of its small tasks.

▶ **Lemma 10.** *Let* $\beta \in \mathbb{N}$ *and let* $(T_{\text{box}}, h_{\text{box}})$ *be a* β*-boxable solution. There is an algorithm with running time* $n^{O(\beta^3/\varepsilon)}$ *that computes a solution* (T', h') *with* $w(T') \geq w(T_{\text{box}} \cap T_L) + (1/2 - \varepsilon) w(T_{\text{box}} \cap T_S)$*.*

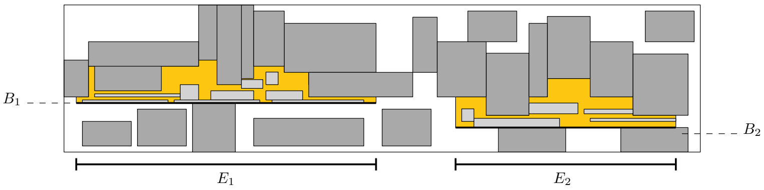


Figure 4 A jammed solution with two subpaths E_1, E_2 corresponding horizontal line segments $E_1 \times B_1$ and $E_1 \times B_2$, and small tasks (light gray) that are jammed in the respective orange areas between the large tasks (dark gray).

Next, we define laminar boxable solutions which are boxable solutions in which the paths of the boxes form a laminar family and the sizes of the boxes are geometrically increasing through the levels (see Fig. 2). A set of boxes $\mathcal{B} = \{B_1, \dots, B_{|\mathcal{B}|}\}$ with a height assignment $h : \mathcal{B} \to \mathbb{N}$ is a *laminar set of boxes* if

- the paths of the boxes form a laminar family, i.e., for any two boxes $B_k, B_{k'}$ we have that $P(B_k) \subseteq P(B_{k'})$, $P(B_{k'}) \subseteq P(B_k)$, or $P(B_k) \cap P(B_{k'}) = \emptyset$,
- there is a box $B^* \in \mathcal{B}$ with $P(B) \subseteq P(B^*)$ for each $B \in \mathcal{B}$,
- for each box $B \in \mathcal{B}$ we have that $d_B = (1+\varepsilon)^k$ for some $k \in \mathbb{N}_0$,
- for each box $B \in \mathcal{B}$ with $d_B = (1+\varepsilon)^k$ for some integer $k \geq 1$ there is a box $B' \in \mathcal{B}$ with $P(B) \subseteq P(B')$, $d_{B'} = (1+\varepsilon)^{k-1}$, and $h(B_k) = h(B_{k-1}) + d_{B_{k-1}}$.

We define $P(\mathcal{B}) := P(B^*)$. A *β-laminar boxable solution* is now a boxable solution whose boxes can be partitioned into sets $\mathcal{B} = \{\mathcal{B}_0, \mathcal{B}_1, \dots, \mathcal{B}_{|\mathcal{B}|-1}\}$ such that the boxes in the sets $\mathcal{B}_1, \dots, \mathcal{B}_{|\mathcal{B}|-1}$ are laminar sets of boxes whose respective paths $P(\mathcal{B}_j)$ are pairwise disjoint and each edge is used by at most β boxes from $\mathcal{B}_0$. Also, each box in $\mathcal{B}_0$ contains exactly one large task and each box in $\mathcal{B}_1, \dots, \mathcal{B}_{|\mathcal{B}|-1}$ contains only small tasks. We design a polynomial time algorithm for finding profitable laminar boxable solutions.

▶ **Lemma 11.** *Let $(T_{\text{lam}}, h_{\text{lam}})$ be a β-laminar boxable solution. There is an algorithm with a running time of $n^{O(\beta+1/\varepsilon^2)}$ that computes a β-laminar boxable solution (T', h') with $w(T') \geq w(T_{\text{lam}} \cap T_L) + w(T_{\text{lam}} \cap T_S)/(2+\varepsilon)$.*

The next class of solutions are pile boxable solutions. A set of boxes $\mathcal{B} = \{B_1, \dots, B_{|\mathcal{B}|}\}$ with a height assignment $h : \mathcal{B} \to \mathbb{N}$ is called a *β-pile of boxes* if $|\mathcal{B}| \leq \beta$, $P(B_k) \supseteq P(B_{k+1})$, $h(B_k) = (k-1)U/|\mathcal{B}|$ and $d_{B_k} = U/|\mathcal{B}|$ for each k. We define $P(\mathcal{B}) := P(B_1)$. A *β-pile boxable solution* is, similarly as above, a boxable solution whose boxes can be partitioned into sets of boxes $\mathcal{B} = \{\mathcal{B}_0, \mathcal{B}_1, \dots, \mathcal{B}_{|\mathcal{B}|-1}\}$ such that the boxes in the sets $\mathcal{B}_1, \dots, \mathcal{B}_{|\mathcal{B}|-1}$ are β-piles of boxes whose respective paths $P(\mathcal{B}_j)$ are pairwise disjoint and each edge is used by at most β boxes in $\mathcal{B}_0$. For β-pile boxable solutions we design a polynomial time algorithm that finds essentially the optimal solution of this type.

▶ **Lemma 12.** *Let $(T_{\text{pile}}, h_{\text{pile}})$ be a β-pile boxable solution. There is an algorithm with a running time of $n^{O(\beta+1/\delta)}$ that computes a β-pile boxable solution (T', h') with $w(T') \geq w(T_{\text{pile}})/(1+\varepsilon)$.*

Finally, we define jammed solutions (which are *not* defined via boxes). Intuitively, they consist of large and small tasks such that the small tasks are placed in areas between some horizontal line segments and the large tasks such that the small tasks are relatively large compared to the free space on each edge in these areas (see Fig. 2 and Fig. 4). Formally, given a solution (T', h') where we define $T'_L := T' \cap T_L$, let $E' \subseteq E$ be a subpath, and let $B \geq 0$ such that intuitively no task $i \in T'_L$ crosses the line segment $E' \times B$, i.e., for each task $i \in T'_L$ we have that $E' \cap P(i) = \emptyset$ or $h'(i) \geq B$ or $h'(i) + d_i \leq B$. The reader may imagine that we draw the line segment $E' \times B$ in the solution given by the large tasks T'_L and that we are interested in small tasks that are drawn above $E' \times B$. For each edge $e \in E'$ let $u'_e := \min_{i \in T'_L : e \in P(i) \wedge h(i) \geq B} h(i) - B$ and define $u'_e := U - B$ if there is no task $i \in T'_L$ with $e \in P(i)$ and $h(i) \geq B$. A task $i \in T' \cap T_S$ is a δ'*-jammed tasks for* (T'_L, E', B, h') if $P(i) \subseteq E'$, $B \leq h'(i) \leq h'(i) + d_i \leq u'_e$ for each edge $e \in P(i)$, and there exists an edge $e' \in P(i)$ such that $d_i > \delta' u'_{e'}$, i.e., intuitively i is relatively large for the edge capacities u'.

▶ **Definition 13.** *A solution solution (T', h') is a δ'-*jammed-solution *if there are pairwise disjoint subpaths $E_1, \ldots, E_k \subseteq E$, values $B_1, \ldots, B_k$, and a partition $T'_S := T' \cap T_S = T'_{S,1} \dot\cup \ldots \dot\cup T'_{S,k}$ such that $T'_{S,\ell}$ is a set of δ'-jammed tasks for $(T'_L, E_\ell, B_\ell, h')$ for each $\ell \in [k]$ with $T'_L := T' \cap T_L$.*

▶ **Lemma 14.** *Let $(T_{\text{jam}}, h_{\text{jam}})$ be a δ'-jammed solution. There is an algorithm with a running time of $n^{O_\varepsilon(1/(\delta \cdot \delta')^3)}$ that computes a $O(\delta')$-jammed solution (T', h') with $w(T') \geq w(T_{\text{lam}})/(1+\varepsilon)$.*

Our key structural lemma for the case of uniform edge capacities shows that for each instance there exists a solution of one of the above types for which the respective algorithm finds a solution of profit at least opt$/(63/32+\varepsilon)$. Then Theorem 2 follows from combining Lemmas 10, 11, 12, 14, and 15.

▶ **Lemma 15** (Structural lemma, uniform capacities)**.** *Given a SAP-instance (T, E) where $u_e = U$ for each edge $e \in E$ and some value U. There exists at least one of the following solutions*

- *a $O_\varepsilon(1)$-boxable solution $(T_{\text{box}}, h_{\text{box}})$ such that* $w(T_{\text{box}} \cap T_L) + w(T_{\text{box}} \cap T_S)/2 \geq \text{OPT}/(63/32+\varepsilon)$
- *a laminar boxable solution $(T_{\text{lam}}, h_{\text{lam}})$ with* $w(T_{\text{lam}} \cap T_L) + w(T_{\text{lam}} \cap T_S)/2 \geq \text{OPT}/(63/32+\varepsilon)$
- *a $O_\varepsilon(1)$-pile boxable solution $(T_{\text{pile}}, h_{\text{pile}})$ with $w(T_{\text{pile}}) \geq \text{OPT}/(63/32+\varepsilon)$*
- *a jammed-solution $(T_{\text{jam}}, h_{\text{jam}})$ with $w(T_{\text{jam}}) \geq \text{OPT}/(63/32+\varepsilon)$.*

2.2 Resource augmentation

We consider now again the case of arbitrary edge capacities but under $(1+\eta)$-resource augmentation. First, we show that due to the latter we can reduce the general case to the case of a constant range of edge capacities.

▶ **Lemma 16.** *If there is an α-approximation algorithm with a running time of $n^{O(f(\eta,M))}$ for the case of $(1+\eta)$-resource augmentation where $\eta < 1$ and $u_e \leq M u_{e'}$ for any two edges e, e' then there is an $\alpha(1+\varepsilon)$-approximation algorithm with a running time of $n^{O(f(\eta,1/(\varepsilon\eta)))}$ for the case of $(1+O(\eta))$-resource augmentation.*

Next, we show that if we are given an instance with a constant range of edge capacities, under $(1+\eta)$-resource augmentation we can guarantee that there is an $(1+\varepsilon)$-approximative $O_{\varepsilon,\eta}(1)$-boxable solution. Then Theorem 3 follows by combining Lemmas 6, 10, 16, and 17 with the $(1+\varepsilon)$-approximation algorithm for sufficiently small tasks in [27].

▶ **Lemma 17.** *Given an instance where $u_e \leq M u_{e'}$ for any two edges e, e' with optimal solution (T^*, h^*). If we increase the edge capacities by a factor of $1+\eta$, there is a $O_{\varepsilon,\eta}(1)$-boxable solution (T', h') such that $w(T') \geq w(T^*)/(1+\varepsilon)$.*

3 Structural lemma for uniform capacities

In the remaining part of this subsection, we sketch the proof of Lemma 15. Consider an optimal solution (OPT, h). We define $\mathrm{OPT}_L := \mathrm{OPT} \cap T_L$ and $\mathrm{OPT}_S := \mathrm{OPT} \cap T_S$.

Recall that our goal is to improve the approximation ratio of 2. Observe that OPT_L alone is a $1/\delta$-boxable solution and hence if $w(\mathrm{OPT}_L) \geq \frac{1}{(63/32+\varepsilon)}\mathrm{opt}$ then we obtain a $1/\delta$-boxable solution with the desired properties. Similarly, the set OPT_S yields a pile boxable solution with exactly 1 box B with $P(B) = E$ and $d_B = U$. Therefore, if $w(\mathrm{OPT}_S) \geq \frac{1}{(63/32+\varepsilon)}\mathrm{opt}$ then we are done. The reader may imagine that $w(\mathrm{OPT}_S) = w(\mathrm{OPT}_L) = \frac{1}{2}\mathrm{opt}$. We split the small tasks in OPT_S into three sets $\mathrm{OPT}_{S,\text{top}}, \mathrm{OPT}_{S,\text{mid}}, \mathrm{OPT}_{S,\text{bottom}}$. Intuitively, we draw a strip of height δU at the bottom of the capacity profile and a strip of height δU at the top of the capacity profile and assign to $\mathrm{OPT}_{S,\text{top}}$ all tasks in OPT_S whose top edge lies in the top strip, we assign to $\mathrm{OPT}_{S,\text{bottom}}$ all tasks in OPT_S whose bottom edge lies in the bottom strip, and we assign to $\mathrm{OPT}_{S,\text{mid}}$ all other small tasks. Formally, we define $\mathrm{OPT}_{S,\text{top}} := \{i \in \mathrm{OPT}_S \mid h(i) + d_i > (1-\delta)U\}$, $\mathrm{OPT}_{S,\text{bottom}} := \{i \in \mathrm{OPT}_S \mid h(i) < \delta U\}$, and $\mathrm{OPT}_{S,\text{mid}} := \mathrm{OPT}_S \setminus (\mathrm{OPT}_{S,\text{top}} \cup \mathrm{OPT}_{S,\text{bottom}})$.

Small tasks at bottom and large tasks. We define solutions that consists of OPT_L and subsets of $\mathrm{OPT}_{S,\text{bottom}}$. For each edge e let $u'_e := \min_{i \in T'_L : e \in P(i)} h(i)$ and define $u'_e := U$ if there is no task $i \in T'_L$ with $e \in P(i)$. One may think of u' as a pseudo-capacity profile for which $\mathrm{OPT}_{S,\text{bottom}}$ is a feasible solution. We the following lemma and obtain sets $\mathrm{OPT}_{S,\text{bottom},L}$, $\mathrm{OPT}_{S,\text{bottom},S}$ with $w(\mathrm{OPT}_{S,\text{bottom},L} \cup \mathrm{OPT}_{S,\text{bottom},S}) \geq (1-\varepsilon)w(\mathrm{OPT}_{S,\text{bottom}})$.

▶ **Lemma 18.** *Given a solution (T', h') for a SAP instance where $\max_e u_e \leq n^{(\log n)^c}$ for some constant c. Then there are sets $T'_L \subseteq T'$ and $T'_S \subseteq T'$ with $w(T'_L \cup T'_S) \geq (1-O(\varepsilon))w(T')$ and there is an $\eta = O_{\varepsilon,\delta}(1)$ such that*

1. *for each edge e it holds that $|T'_L \cap T_e| \leq (\log n)^{O_{\varepsilon,\delta}(c)}$, and*
2. *for each task $i \in T'_L$ there is an edge $e \in P(i)$ with $d_i \geq \eta u_e$,*
3. *there is a boxable solution for T'_S in which each edge e is used by at most $(\log n)^{O_\varepsilon(c)}$ boxes,*
4. *these boxes form groups of laminar sets of boxes.*

We have that for each task $i \in \mathrm{OPT}_{S,\text{bottom},L}$ there is an edge $e \in P(i)$ with $d_i > \delta u'_e$, therefore $\mathrm{OPT}_{S,\text{bottom},L}$ is a set of δ-jammed tasks for $(\mathrm{OPT}_L, E, 0, h)$ and hence $\mathrm{OPT}_L \cup \mathrm{OPT}_{S,\text{bottom},L}$ forms a δ-jammed-solution. Also, $\mathrm{OPT}_L \cup \mathrm{OPT}_{S,\text{bottom},S}$ forms a laminar boxable solution. Hence, if $w(\mathrm{OPT}_{S,\text{bottom},S})$ or $w(\mathrm{OPT}_{S,\text{bottom},L})$ is sufficiently large then we are done. Therefore, the reader may imagine that $w(\mathrm{OPT}_{S,\text{bottom},S}) = w(\mathrm{OPT}_{S,\text{bottom},L}) = 0$.

Small tasks at top and large tasks. We mirror OPT along the y-axis and do a symmetric construction with $\mathrm{OPT}_{S,\text{top}}$: we apply Lemma 18 which yields sets $\mathrm{OPT}_{S,\text{top},L}, \mathrm{OPT}_{S,\text{top},S}$ and a δ-jammed solution $\mathrm{OPT}_L \cup \mathrm{OPT}_{S,\text{top},L}$ and a laminar boxable solution $\mathrm{OPT}_L \cup \mathrm{OPT}_{S,\text{top},S}$. Like before, the reader may imagine that $w(\mathrm{OPT}_{S,\text{top},S}) = w(\mathrm{OPT}_{S,\text{top},L}) = 0$ and hence $w(\mathrm{OPT}_{S,\text{mid}}) = w(\mathrm{OPT}_S) = \frac{1}{2}\mathrm{opt}$.

Small and large tasks in the middle. Next, we split the large tasks OPT_L into three sets $\mathrm{OPT}_{L,\text{top}}$, $\mathrm{OPT}_{L,\text{mid}}$, and $\mathrm{OPT}_{L,\text{bottom}}$. Intuitively, $\mathrm{OPT}_{L,\text{top}}$ consists of all tasks in OPT_L whose top edge lies in the strip of height δU at the top of the capacity profile, $\mathrm{OPT}_{L,\text{bottom}}$ consists of all tasks in OPT_L whose bottom edge lies in the strip of height δU at the bottom of the capacity profile, and $\mathrm{OPT}_{L,\mathrm{mid}}$ contains all remaining tasks in OPT_L. Formally, $\mathrm{OPT}_{L,\mathrm{top}} := \{i \in \mathrm{OPT}_L | h(i) + d_i > (1-\delta)U\}$, $\mathrm{OPT}_{L,\mathrm{bottom}} := \{i \in \mathrm{OPT}_L | h(i) < \delta U\}$, and $\mathrm{OPT}_{L,\mathrm{mid}} := \mathrm{OPT}_L \setminus (\mathrm{OPT}_{L,\mathrm{top}} \cup \mathrm{OPT}_{L,\mathrm{bottom}})$.

Observe that no task in $\mathrm{OPT} \setminus (\mathrm{OPT}_{L,\mathrm{bottom}} \cup \mathrm{OPT}_{S,\mathrm{bottom}})$ touches the rectangle $E \times [0, \delta U)$. Using this, we define a boxable solution T_{box} that will consist essentially of all tasks in the set $\mathrm{OPT} \setminus (\mathrm{OPT}_{L,\mathrm{bottom}} \cup \mathrm{OPT}_{S,\mathrm{bottom}})$. Intuitively, we will use the free space $E \times [0, \delta U)$ in order to untangle the interaction between the large and small tasks. More precisely, T_{box} will be a $O_\delta(1)$-boxable solution in which all small tasks are assigned into boxes of height $\Theta_\delta(U)$ each. More formally, we apply the following lemma to $\mathrm{OPT} \setminus (\mathrm{OPT}_{L,\mathrm{bottom}} \cup \mathrm{OPT}_{S,\mathrm{bottom}})$ and denote by $(T'_{\mathrm{box}}, h'_{\mathrm{box}})$ the resulting solution.

▶ **Lemma 19.** *Given a solution (T', h') such that no task touches the rectangle $E \times [0, \delta U)$. Then there exists a $O_\delta(1)$-boxable solution (T', h_{box}).*

We apply Lemma 19 to the solution obtained by taking $\mathrm{OPT} \setminus (\mathrm{OPT}_{L,\mathrm{top}} \cup \mathrm{OPT}_{S,\mathrm{top}})$ and shifting each task up by δU units. Let $(T''_{\mathrm{box}}, h''_{\mathrm{box}})$ denote the resulting solution. Assume w.l.o.g. that $w(\mathrm{OPT}_{L,\mathrm{top}}) \le w(\mathrm{OPT}_{L,\mathrm{bottom}})$. Observe that if $w(\mathrm{OPT}_{L,\mathrm{mid}}) \ge \gamma\mathrm{opt}$ then

$$\begin{aligned} w(T' \cap T_L) &\ge w(\mathrm{OPT}_{L,\mathrm{bottom}}) + w(\mathrm{OPT}_{L,\mathrm{mid}}) \\ &\ge \frac{1}{2} w(\mathrm{OPT}_{L,\mathrm{top}} \cup \mathrm{OPT}_{L,\mathrm{bottom}}) + w(\mathrm{OPT}_{L,\mathrm{mid}}) \\ &\ge \frac{1+\gamma}{2} w(\mathrm{OPT}_L) \end{aligned}$$

and $w(T' \cap T_S) \ge w(\mathrm{OPT}_S \setminus \mathrm{OPT}_{S,\mathrm{bottom}})$. Therefore, the reader may imagine now that $w(\mathrm{OPT}_{L,\mathrm{mid}}) = 0$.

Types of points in the middle

We distinguish points in the rectangle $E \times [0, U]$ into different types and identify each task $i \in \mathrm{OPT}$ with a rectangle $R_i := P(i) \times [h(i), h(i) + d_i]$. For each point p let ℓ_p denote the maximally long horizontal line segment in $E \times [0, U]$ that contains p and that does not touch the relative interior of a task in $\mathrm{OPT}_{L,\mathrm{top}} \cup \mathrm{OPT}_{L,\mathrm{bottom}}$. We say that p is a *top-point* if no endpoint of ℓ_p touches a task in $\mathrm{OPT}_{L,\mathrm{bottom}}$, p is a *sandwich-point* if one of the end-points of ℓ_p touches a task in $\mathrm{OPT}_{L,\mathrm{top}}$ and the other end-point touches task in $\mathrm{OPT}_{L,\mathrm{bottom}}$, and p is a *bottom-point* if no endpoint of ℓ_p touches a task in $\mathrm{OPT}_{L,\mathrm{top}}$ and at least one endpoint of ℓ_p touches a task in $\mathrm{OPT}_{L,\mathrm{bottom}}$. Note that here we do not define stair-points since the edges have uniform capacities. Let $\mathcal{C}_{\mathrm{top}}, \mathcal{C}_{\mathrm{sw}}, \mathcal{C}_{\mathrm{bottom}}$ denote the set of connected components of top- sandwich-, and bottom-points, respectively.

Each edge is used by at most three connected components in $\mathcal{C}_{\mathrm{top}} \cup \mathcal{C}_{\mathrm{sw}} \cup \mathcal{C}_{\mathrm{bottom}}$.

▶ **Lemma 20.** *Each edge e can be used by at most one connected component of top-points, by at most one connected component of sandwich-points, and by at most one connected component of bottom-points.*

Let $\mathrm{OPT}_{S,\mathrm{cross}} \subseteq \mathrm{OPT}_{S,\mathrm{mid}}$ denote the tasks in $\mathrm{OPT}_{S,\mathrm{mid}}$ that intersect at least two different connected components, e.g., a connected component of top-points and a connected component of sandwich-points.

▶ **Lemma 21.** *Each edge is used by at most 2 different tasks in $\mathrm{OPT}_{S,\mathrm{cross}}$.*

In particular, $\mathrm{OPT}_L \cup \mathrm{OPT}_{S,\mathrm{cross}}$ forms a $O(1/\delta)$-boxable solution. If $w(\mathrm{OPT}_{S,\mathrm{cross}})$ is sufficiently large we are therefore done. The reader may imagine that $w(\mathrm{OPT}_{S,\mathrm{cross}}) = 0$.

Top points. Consider a connected component C of top-points. Let $\mathrm{OPT}_{S,\mathrm{mid},\mathrm{top}}(C) \subseteq \mathrm{OPT}_{S,\mathrm{mid}}$ denote the tasks in $\mathrm{OPT}_{S,\mathrm{mid}}$ contained in C. We apply Lemma 18 and obtain the sets $\mathrm{OPT}_{S,\mathrm{mid},\mathrm{top},S}(C)$ and $\mathrm{OPT}_{S,\mathrm{mid},\mathrm{top},L}(C)$. Let

$$\mathrm{OPT}_{S,\mathrm{mid},\mathrm{top},S} := \bigcup_{C \in \mathcal{C}_{\mathrm{top}}} \mathrm{OPT}_{S,\mathrm{mid},\mathrm{top},S}(C)$$

and

$$\mathrm{OPT}_{S,\mathrm{mid},\mathrm{top},L} := \bigcup_{C \in \mathcal{C}_{\mathrm{top}}} \mathrm{OPT}_{S,\mathrm{mid},\mathrm{top},L}(C).$$

We obtain that $\mathrm{OPT}_{L,\mathrm{top}} \cup \mathrm{OPT}_{L,\mathrm{bottom}} \cup \mathrm{OPT}_{S,\mathrm{mid},\mathrm{top},L}$ is a δ-jammed solution and $\mathrm{OPT}_{L,\mathrm{top}} \cup \mathrm{OPT}_{L,\mathrm{bottom}} \cup \mathrm{OPT}_{S,\mathrm{mid},\mathrm{top},S}$ is a laminar boxable solution. Intuitively, if $w(\mathrm{OPT}_{S,\mathrm{mid},\mathrm{top},L})$ or $w(\mathrm{OPT}_{S,\mathrm{mid},\mathrm{top},S})$ is sufficiently large then we are done. The reader may therefore imagine that both quantities are zero.

Bottom points. We do a symmetric operation for all connected components C of bottom-points, obtaining respective sets $\mathrm{OPT}_{S,\mathrm{mid},\mathrm{bottom},S}, \mathrm{OPT}_{S,\mathrm{mid},\mathrm{bottom},L}$, a δ-jammed solution

$$\mathrm{OPT}_{L,\mathrm{top}} \cup \mathrm{OPT}_{L,\mathrm{bottom}} \cup \mathrm{OPT}_{S,\mathrm{mid},\mathrm{bottom},L},$$

and a laminar boxable solution

$$\mathrm{OPT}_{L,\mathrm{top}} \cup \mathrm{OPT}_{L,\mathrm{bottom}} \cup \mathrm{OPT}_{S,\mathrm{mid},\mathrm{bottom},S}.$$

Like before, the reader may imagine that $w(\mathrm{OPT}_{S,\mathrm{mid},\mathrm{bottom},S}) = w(\mathrm{OPT}_{S,\mathrm{mid},\mathrm{bottom},L}) = 0$.

Sandwich points. Finally, let $\mathrm{OPT}_{S,\mathrm{mid},\mathrm{sw}}$ denote all points in $\mathrm{OPT}_{S,\mathrm{mid}}$ that are contained in a connected component in $\mathcal{C}_{\mathrm{sw}}$. We assume first that $w(\mathrm{OPT}_{L,\mathrm{top}}) \geq w(\mathrm{OPT}_{L,\mathrm{bottom}})$. We define a solution consisting of some tasks in $\mathrm{OPT}_{S,\mathrm{mid},\mathrm{sw}}$ and additionally all tasks in $\mathrm{OPT}_{L,\mathrm{top}}$. Let $C \in \mathcal{C}_{\mathrm{sw}}$ and denote by $\mathrm{OPT}_{S,\mathrm{mid}}(C)$ the set of tasks in $\mathrm{OPT}_{S,\mathrm{mid}}$ that are contained in C. Let E' denote the maximally long subpath between two vertices v, v' such that for each $x \in [v, v']$ the vertical line through x, i.e., $\{x\} \times \mathbb{R}$, has non-empty intersection with C. Note that the rectangle $E' \times [0, \delta U]$ has empty intersection with each tasks in $\mathrm{OPT}_{L,\mathrm{top}} \cup \mathrm{OPT}_{S,\mathrm{mid}}$. Intuitively, we use this free space in order to push all tasks in $\mathrm{OPT}_{S,\mathrm{mid}}(C)$ down by δU units. Then they all fit into $O_{\varepsilon}(1)$ boxes that have non-empty intersection with the tasks in $\mathrm{OPT}_{L,\mathrm{top}}$.

▶ **Lemma 22.** *Given $C \in \mathcal{C}_{\mathrm{sw}}$. There is a pile of boxes $\mathcal{B} = \{B_1, \ldots, B_{1/\delta}\}$ (with a height assignment $h \colon \mathcal{B} \to \mathbb{N}_0$) such that $P(B) \subseteq E'$ and there are pairwise disjoint sets $T_1, \ldots, T_{|\mathcal{B}|} \subseteq \mathrm{OPT}_{S,\mathrm{mid}}(C)$ such that for each j, the tasks in T_j fit into B_j. The weight of tasks in the boxes is at least $\sum_j w(T_j) \geq (1-\varepsilon) w(\mathrm{OPT}_{S,\mathrm{mid}}(C))$.*

Let $\mathrm{OPT}_{S,\mathrm{mid},\mathrm{sw}} := \bigcup_{C \in \mathcal{C}_{\mathrm{sw}}} \mathrm{OPT}_{S,\mathrm{mid},S}(C)$. We apply Lemma 22 to each component $C \in \mathcal{C}_{\mathrm{sw}}$ and hence we obtain a pile boxable solution whose profit is at least

$$(1-\varepsilon) w\left(\mathrm{OPT}_{L,\mathrm{top}} \cup \mathrm{OPT}_{S,\mathrm{mid},\mathrm{sw}}\right).$$

In a similar way we can construct a pile boxable solution of profit at least

$$(1-\varepsilon) w\left(\mathrm{OPT}_{L,\mathrm{bottom}} \cup \mathrm{OPT}_{S,\mathrm{mid},\mathrm{sw}}\right).$$

▶ **Lemma 23.** *There is a pile boxable solution* $(T'_{\mathrm{sw}}, h'_{\mathrm{sw}})$ *with profit at least*

$$(1-\varepsilon)w\left(\mathrm{OPT}_{L,\mathrm{top}} \cup \mathrm{OPT}_{S,\mathrm{mid},\mathrm{sw}} \cup \mathrm{OPT}_{S,\mathrm{cross}}\right)$$

and a pile boxable solution $(T''_{\mathrm{sw}}, h''_{\mathrm{sw}})$ *with profit at least*

$$(1-\varepsilon)w\left(\mathrm{OPT}_{L,\mathrm{bottom}} \cup \mathrm{OPT}_{S,\mathrm{mid},\mathrm{sw}} \cup \mathrm{OPT}_{S,\mathrm{cross}}\right).$$

Intuitively, since $w(\mathrm{OPT}_{L,\mathrm{mid}}) = 0$ we have $w(\mathrm{OPT}_{L,\mathrm{top}}) \geq \mathrm{opt}/4$ or $w(\mathrm{OPT}_{L,\mathrm{bottom}}) \geq \mathrm{opt}/4$. Also, since $w(\mathrm{OPT}_{S,\mathrm{mid},\mathrm{top}}) = w(\mathrm{OPT}_{S,\mathrm{mid},\mathrm{bottom}}) = w(\mathrm{OPT}_{S,\mathrm{cross}}) = 0$ and $w(\mathrm{OPT}_{S,\mathrm{mid}}) = \mathrm{opt}/2$ we have that $w(\mathrm{OPT}_{S,\mathrm{mid},\mathrm{sw}}) = \mathrm{opt}/2$. Hence, (T'_{sw}, h') or (T''_{sw}, h'') satisfies the claim of the lemma. Formally, our candidate solutions are

$$\begin{aligned}
\mathrm{OPT}^{(1)} &:= \mathrm{OPT}_S\,,\\
\mathrm{OPT}^{(2)} &:= \mathrm{OPT}_L \cup \mathrm{OPT}_{S,\mathrm{bottom},S}\,,\\
\mathrm{OPT}^{(3)} &:= \mathrm{OPT}_L \cup \mathrm{OPT}_{S,\mathrm{bottom},L}\,,\\
\mathrm{OPT}^{(4)} &:= \mathrm{OPT}_L \cup \mathrm{OPT}_{S,\mathrm{top},S}\\
\mathrm{OPT}^{(5)} &:= \mathrm{OPT}_L \cup \mathrm{OPT}_{S,\mathrm{top},L}\,,\\
\mathrm{OPT}^{(6)} &:= T'_{\mathrm{box}}\,,\\
\mathrm{OPT}^{(7)} &:= T''_{\mathrm{box}}\,,\\
\mathrm{OPT}^{(8)} &:= \mathrm{OPT}_L \cup \mathrm{OPT}_{S,\mathrm{cross}}\,,\\
\mathrm{OPT}^{(7)} &:= \mathrm{OPT}_{L,\mathrm{top}} \cup \mathrm{OPT}_{L,\mathrm{bottom}} \cup \mathrm{OPT}_{S,\mathrm{mid},\mathrm{top},S}\,,\\
\mathrm{OPT}^{(8)} &:= \mathrm{OPT}_{L,\mathrm{top}} \cup \mathrm{OPT}_{L,\mathrm{bottom}} \cup \mathrm{OPT}_{S,\mathrm{mid},\mathrm{top},L}\,,\\
\mathrm{OPT}^{(9)} &:= \mathrm{OPT}_{L,\mathrm{top}} \cup \mathrm{OPT}_{L,\mathrm{bottom}} \cup \mathrm{OPT}_{S,\mathrm{mid},\mathrm{bottom},S}\,,\\
\mathrm{OPT}^{(10)} &:= \mathrm{OPT}_{L,\mathrm{top}} \cup \mathrm{OPT}_{L,\mathrm{bottom}} \cup \mathrm{OPT}_{S,\mathrm{mid},\mathrm{bottom},L}\,,\\
\mathrm{OPT}^{(11)} &:= T'_{\mathrm{sw}}\,,\ \text{and}\\
\mathrm{OPT}^{(12)} &:= T''_{\mathrm{sw}}\,.
\end{aligned}$$

The sets $\mathrm{OPT}^{(1)}, \mathrm{OPT}^{(9)}$, and $\mathrm{OPT}^{(10)}$ are pile boxable solutions, $\mathrm{OPT}^{(2)}, \mathrm{OPT}^{(4)}, \mathrm{OPT}^{(7)}$, and $\mathrm{OPT}^{(9)}$ are laminar boxable solutions, $\mathrm{OPT}^{(3)}, \mathrm{OPT}^{(5)}, \mathrm{OPT}^{(8)}$, and $\mathrm{OPT}^{(10)}$ are δ-jammed solutions, and finally $\mathrm{OPT}^{(7)}$ and $\mathrm{OPT}^{(8)}$ are $O_\delta(1)$-boxable solutions.

Our insights above determine constraints of a linear program which provides an upper bound on the approximation ratio. Using LP duality, we are able to prove that the obtained value is at most 63/32, which completes the proof of Lemma 15.

4 Structural lemma for arbitrary capacities

In this section we prove Lemma 8. We first limit the number of large tasks per edge that can appear in a feasible solution.

▶ **Lemma 24.** *For each edge* e *and each feasible solution* (T_{SOL}, h_{SOL}) *it holds that* $|T_{SOL} \cap T_L \cap T_e| \leq (\log u_e)/\delta^2$.

Consider an optimal solution (OPT, h). Define $\mathrm{OPT}_L := \mathrm{OPT} \cap T_L$ and $\mathrm{OPT}_S := \mathrm{OPT} \cap T_S$. Since we assumed the maximum edge capacity to be quasi-polynomially bounded, Lemma 24 shows that each edge can be used by at most $1/\delta^2(\log n)^{O(1)}$ large tasks in OPT.

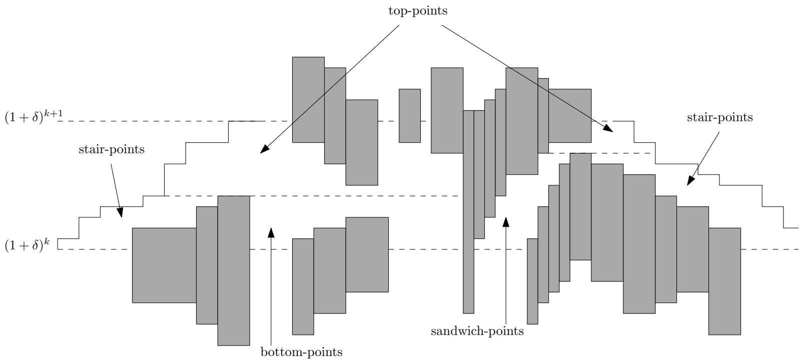


Figure 5 The different types of points within a corridor C_k. Note that the small tasks are not shown in the figure.

Therefore, the tasks in OPT_L alone form a boxable solution. Since our goal is to improve the approximation ratio of 2 in [27], we are done if $w(\mathrm{OPT}_L) \geq (\frac{1}{2}+\gamma)\mathrm{OPT}$ for some $\gamma > 0$. The reader may therefore imagine that $w(\mathrm{OPT}_L) \leq \mathrm{OPT}/2$ and hence that $w(\mathrm{OPT}_S) \geq \mathrm{OPT}/2$.

We partition the large tasks OPT_L into two groups. We define $\mathrm{OPT}_{L,\downarrow} := \{i \in \mathrm{OPT}_L \mid h(i) < d_i\}$ and $\mathrm{OPT}_{L,\uparrow} := \{i \in \mathrm{OPT}_L \mid h(i) \geq d_i\}$. In the next lemma we show that there is a boxable solution that contains essentially all tasks in $\mathrm{OPT}_S \cup \mathrm{OPT}_{L,\uparrow}$.

▶ **Lemma 25.** *For an arbitrary $0 < \varepsilon \leq 1/3$ there exists a boxable solution with profit at least $(1-\varepsilon)w(\mathrm{OPT}_S \cup \mathrm{OPT}_{L,\uparrow})$.*

Intuitively, if now $w(\mathrm{OPT}_{L,\uparrow}) \geq \gamma\mathrm{OPT}$ for some $\gamma > 0$ then $w(\mathrm{OPT}_S \cup \mathrm{OPT}_{L,\uparrow}) \geq (\frac{1}{2}+\gamma)\mathrm{OPT}$ and we are done, due to Lemma 6 and Lemma 25. The reader therefore may imagine that $w(\mathrm{OPT}_{L,\uparrow}) = 0$ and $w(\mathrm{OPT}_S) = \mathrm{OPT}/2$ and hence also $w(\mathrm{OPT}_{L,\downarrow}) = \mathrm{OPT}/2$.

Next, we define solutions that either consist of $\mathrm{OPT}_{L,\downarrow}$ or of a subset of $\mathrm{OPT}_{L,\downarrow}$ and additionally some small tasks. We will prove that one of the constructed sets or $\mathrm{OPT}_S \cup \mathrm{OPT}_{L,\uparrow}$ has large profit. In the sequel, we will identify the vertices $\{v_1,\ldots,v_{|V|}\}$ of (V,E) with the coordinates $1,\ldots,|V|$ and a path P between vertices $v_i, v_{i'}$ with the closed interval $[i,i']$. For each task $i \in \mathrm{OPT}$ define its rectangle $R_i := P(i) \times [h(i), h(i)+d_i]$. We will identify a task i with its rectangle R_i.

We define $(\log n)^{O_\delta(1)}$ corridors. We draw a horizontal line $\ell^{(k)}$ with y-coordinate $y = (1+\delta)^k$ for each $k \in \mathbb{N}$. For each $k \in \mathbb{N}$ we define the area $\mathbb{R} \times [\ell^{(k)}, \ell^{(k+1)})$ to be the *corridor* C_k. Consider a task $i \in \mathrm{OPT}_L$. Observe that for each task $i \in \mathrm{OPT}_L$ the rectangle R_i has to be intersected by at least one line $\ell^{(k)}$ since $d_i > \delta \cdot b(i)$. Also, observe that for each edge e and each corridor C_k there can be at most two tasks $i, i' \in \mathrm{OPT}_L$ whose respective paths $P(i), P(i')$ use e and whose respective rectangles $R_i, R_{i'}$ intersect C_k. If there are two such task i, i' then for one of them its rectangle must intersect $\ell^{(k)}$ and for the other its rectangle must intersect $\ell^{(k+1)}$.

Let C_k be a corridor. For a task $i \in \mathrm{OPT}_L$ with $R_i \cap C_k \neq \emptyset$ we say that i is a *top-large-task for* C_k if $h(i) \in [(1+\delta)^k, (1+\delta)^{k+1})$, a *bottom-large-task for* C_k if $h(i)+d_i \in [(1+\delta)^k, (1+\delta)^{k+1})$, and a *cross-large-task for* C_k if $h(i) < (1+\delta)^k$ and $h(i)+d_i \geq (1+\delta)^{k+1}$. We partition the area of C_k that is not used by large tasks into connected components of points. For each point p, let ℓ_p denote the maximally long horizontal line segment that contains p and that neither crosses a large task nor the capacity profile. We say that p is a *top-point* if each endpoint of ℓ_p touches a top-large-task or the capacity profile; p is a *sandwich-point* if one of the end-points of ℓ_p touches a top-large-task and the other end-point touches a bottom-large-task; p is a *stair-point* if one end-point of ℓ_p touches a bottom-large-task and the other end-point touches the capacity profile; and p is a *bottom-point*, if both end-points of ℓ_p touch a bottom-large-task, see Fig. 5. In each corridor C_k this yields connected components of top-, bottom-, sandwich-, and stair-points. In the remaining proof (see [28]) we show that for each of these types there exists a $(\log n/\delta)^{O_\varepsilon(c)}$-boxable solution or a $(\log n/\delta)^{O_\varepsilon(c)}$-stair solution that contains all tasks in $\mathrm{OPT}_{L,\downarrow}$ and a constant fraction of the small tasks in all connected components of the respective type, or of the small tasks that overlap more than one type of points. Hence, we obtain an approximation ratio strictly better than 2, unless essentially all small tasks lie in connected components of sandwich points. For this case, intuitively we prove that there is a $(\log n/\delta)^{O_\varepsilon(c)}$-boxable solution that contains all profit from the small tasks and a 1/4-fraction of the profit of the tasks in $\mathrm{OPT}_{L,\downarrow}$. We show that the best of our solutions yields an approximation ratio of $1.997+\varepsilon$ which proves Lemma 8.

5 Compute stair solution

In this section we prove Lemma 9. To this end, we first show how to compute a solution for a single stair-block $\mathcal{SB}$. Then we devise a dynamic program that intuitively sweeps the path from left to right, guesses the stair-blocks and the other large tasks, and then uses the subroutine for a single stair-block to assign tasks into each stair-block. Suppose that we are given a γ-stair-block $\mathcal{SB} = (e_L, e_M, e_R, f, T'_L, h')$. Let $\bar{T}$ denote the set of tasks $i \in T$ such that $\{i\}$ fits into $\mathcal{SB}$. In the sequel, we prove the following lemma.

▶ **Lemma 26.** *Let $T^* \subseteq \bar{T}$ be an unknown set of tasks that fits into $\mathcal{SB}$ such that $w(T_S \cap T^*) \geq \frac{1}{\alpha} w(T_L \cap T^*)$ for some value $\alpha \geq 1$. There is a $(n \cdot \max_e u_e)^{O_\delta(\log(\max_e u_e))}$ time algorithm that computes a set $\hat{T}$ that fits into $\mathcal{SB}$ such that*

$$w(\hat{T}) \geq (1 - O(\varepsilon)) \left(w(T_L \cap T^*) + \frac{1}{8(\alpha+1)} \cdot w(T_S \cap T^*) \right).$$

First, we guess $w(T_L \cap T^*)$ up to a factor $1+\varepsilon$, i.e., we guess a value W such that $w(T_L \cap T^*) \in [W, (1+\varepsilon)W)$. One can show that $(n/\varepsilon)^{O(1)}$ many guesses for W suffice. Our algorithm is based on a linear program that uses configurations for the sets of large tasks that fit into $\mathcal{SB}$. Formally, we define a pair $C = (\bar{T}', \bar{h}')$ to be a *configuration* if $\bar{T}' \subseteq \bar{T}$, $w(\bar{T}') \in [W, (1+\varepsilon)W)$, $\bar{h}'$ is a function $\bar{h}' : \bar{T}' \to \mathbb{N}$ such that $\bar{h}'(i) < d_i$ for each task $i \in \bar{T}'$, and $(\bar{T}', \bar{h}')$ fits into $\mathcal{SB}$. Let $\mathcal{C}$ denote the set of all configurations. We introduce a variable y_C for each configuration $C \in \mathcal{C}$. Intuitively, $y_C = 1$ indicates that the computed solution contains exactly the set of large tasks in C, each of them drawn at the height level determined by C. For each small task $j \in \bar{T}_S := \bar{T} \cap T_S$ and each $t \in \{0, \ldots, b(j) - d_j\}$ we introduce a variable $x_{j,t}$ indicating whether j is contained in the solution and drawn at height t. Note that we do not need variables $x_{j,t}$ for $t > b(j) - d_j$ since the upper edge of j has to have a height of at most $b(j)$.

We add constraints that ensure that the rectangles corresponding to the selected tasks do not overlap. To this end, for each small tasks $j \in \bar{T}_S$ and each possible height $t \in \{0, \ldots, b(j) - d_j\}$ we define a “rectangle” $p(j,t) = \{(e,t') \mid e \in P(j) \text{ and } t \leq t' < t + d_j\}$. For a pair (e,t) the reader may imagine that it represents the point whose x-coordinate is the mid-point of the edge e and whose y-coordinate is t. Similarly, for a configuration $C = (\bar{T}', \bar{h}') \in \mathcal{C}$ we define the “points” covered by C to be $p(C) := \{(e,t') \mid \exists i \in \bar{T}' : e \in P(i) \text{ and } \bar{h}'(i) \leq t' < \bar{h}'(i) + d_i\}$ and $w_C := w(\bar{T}')$.

Denote by $\mathrm{LP}_{\mathcal{SB}}$ the linear program below where for convenience we assume that all non-existing variables are set to zero.

$$\max \textstyle\sum_{C\in\mathcal{C}} y_C w_C + \sum_{j\in\bar{T}_S,t} x_{j,t} w_j \tag{1}$$

$$\text{s.t.} \quad \textstyle\sum_{C:\,(e,t)\in p(C)} y_C \tag{2}$$

$$+ \sum_{j\in\bar{T}_S,t':\,(e,t)\in p(j,t')} x_{j,t'} \leq 1 \quad \text{for all } e\in P_{e_M,e_R}, t\geq 0 \tag{3}$$

$$\textstyle\sum_{C:\,(e,t)\in p(C)} y_C + \sum_{t':\,t'\leq t} x_{j,t'} \leq 1 \quad \text{for all } e\in P_{e_M,e_R}, t\geq 0, j\in \bar{T}_S\cap T_e \tag{4}$$

$$\textstyle\sum_{C\in\mathcal{C}} y_C = 1 \tag{5}$$

$$\textstyle\sum_{t\geq 0} x_{j,t} \leq 1 \quad \text{for all } j\in\bar{T}_S \tag{6}$$

$$x_{j,t}, y_C \geq 0 \quad \text{for all } j\in\bar{T}_S, t\in\mathbb{N}, t\leq b(j)-d_j, C\in\mathcal{C}$$

The first set of constraints (3) expresses intuitively that no two rectangles overlap. Then (4) strengthens this condition by stating that if a configuration C covers a point (e,t) then no small task j using e can be selected such that it covers a point (e,t') with $t'\leq t$ (note that if C covers (e,t) then it also covers each point (e,t') with $t'\leq t$). Constraint (5) ensures that we select exactly one configuration. A task j still cannot be drawn at two positions simultaneously, which we ensure with (6). In the LP above, we introduce constraints (3) and (4) for each $t\geq 0$, however, it is sufficient to state those for each t such that $t\in\mathbb{N}$ since $d_i\in\mathbb{N}$ for each task $i\in T$ and we introduce the variables $x_{j,t}$ only for values t with $t\in\mathbb{N}$. This yields an equivalent formulation with only $(n\max_e u_e)^{O(1)}$ constraints (apart from the non-negativity constraints). The number of variables in $\mathrm{LP}_{\mathcal{SB}}$ is exponential. However, we can solve it in polynomial time via a suitable separation oracle for the dual.

▶ **Lemma 27.** *There is an algorithm with running time $(n\max_e u_e)^{O_\delta(\log(\max_e u_e))}$ that computes an optimal solution to $LP_{\mathcal{SB}}$.*

5.1 The rounding algorithm

Let (x^*,y^*) denote the optimal solution to $\mathrm{LP}_{\mathcal{SB}}$. We round (x^*,y^*) via randomized rounding. First, we sample a configuration $\hat{C}$ using the distribution determined by y^*, i.e., for each configuration $C\in\mathcal{C}$, we obtain $\hat{C}=C$ with probability y_C. Define $\bar{y}_{\hat{C}}:=1$ and $\bar{y}_C:=0$ for each $C\in\mathcal{C}\setminus\{\hat{C}\}$. Then we construct a new solution x for the small tasks where intuitively we remove all pairs (j,t) that overlap with $\hat{C}$, i.e., such that $p(j,t)\cap p(\hat{C})\neq\emptyset$. For each pair of the latter type we define $x_{j,t}:=0$ and we define $x_{j,t}:=x^*_{j,t}$ for all other pairs (j,t). Observe that x is a solution to the LP that is obtained by taking $\mathrm{LP}_{\mathcal{SB}}$ and removing all variables y_C and constraint (5). Denote by $\mathrm{LP}'_{\mathcal{SB}}$ the resulting LP. We can round it via randomized rounding with alteration, using that for two pairs $(j,t),(j',t')$ with $j,j'\in T_S$ the corresponding rectangles $p(j,t),p(j',t')$ overlap if and only if they overlap on a "vertical line segment above e_M", i.e., on $\cup_{t''\in[t,t+d_j)\cap[t',t'+d_{j'})}(e_M,t'')$.

▶ **Lemma 28.** *Given a solution x to $LP'_{\mathcal{SB}}$. In polynomial time we can compute an integral solution $\bar{x}$ to $LP'_{\mathcal{SB}}$ with expected value $\sum_{j\in\bar{T}_S,t}\bar{x}_{j,t}w_j \geq \frac{1}{4}\sum_{j\in\bar{T}_S,t}x_{j,t}w_j$ such that the support of $\bar{x}$ is contained in the support of x.*

Let $(\bar{x},\bar{y})$ denote the resulting solution. Secondly, we compute the optimal solution to $\mathrm{LP}'_{\mathcal{SB}}$ (hence ignoring the configurations of large tasks) and round it via Lemma 28, let $(\bar{x}',\bar{y}')$ denote the resulting solution. In the sequel we prove that the most profitable solution among $(\bar{x},\bar{y})$ and $(\bar{x}',\bar{y}')$ satisfies the claim of Lemma 9. Since we sampled $\hat{C}$ according to the distribution given by y^*, we have that $\mathbb{E}[w_{\hat{C}}] = \sum_{C\in\mathcal{C}} y^*_C w_C$. Recall that we discarded all pairs (j,t) such that $p(j,t)$ overlaps with $p(\hat{C})$. Hence, there are some pairs (j,t) that are discarded with very high probability. We call such a pair problematic where formally we say that a pair (j,t) with $j \in \bar{T}_S$ and $t \in \mathbb{N}$ is *problematic* if $\sum_{C\in\mathcal{C}:p(C)\cap p(j,t)\neq\emptyset} y^*_C > 1-\eta$ for some value $\eta > 0$ to be defined later. Let $T_{S!}$ denote the set of all problematic pairs. In the following lemma we prove that their contribution to the profit of (x^*, y^*) is only small and hence we can afford to ignore them, unless $(\bar{x}', \bar{y}')$ already has enough profit. Here we crucially need constraint (4). We define $\text{opt}_{LP} := \sum_{C\in\mathcal{C}} y^*_C w_C + \sum_{j\in\bar{T}_S,t} x^*_{j,t} w_j$.

▶ **Lemma 29.** *We have that* $\sum_{(j,t)\in T_{S!}} x^*_{j,t} w_j \le 4\eta \mathit{opt}_{LP}$ *or the profit of* $(\bar{x}', \bar{y}')$ *is at least* $\mathit{opt}_{LP} \ge w(T_L \cap T^*) + w(T_S \cap T^*)$.

Assume now that the first case of Lemma 29 applies. We argue that then the problematic pairs contribute at most half of the profit of all pairs (for all small tasks) and hence we can ignore the problematic pairs.

▶ **Lemma 30.** *Assume that* $\eta \le \frac{1}{8}\frac{1/\alpha - O(\varepsilon)}{1+1/\alpha-O(\varepsilon)}$. *Then* $\sum_{(j,t)\notin T_{S!}} x^*_{j,t} w_j \ge \frac{1}{2}\sum_{j\in\bar{T}_S,t} x^*_{j,t} w_j$.

Proof sketch. Let us pretend that $\sum_{C\in\mathcal{C}} y^*_C w_C = w(T_L \cap T^*)$. Then $\sum_{j\in\bar{T}_S,t} x^*_{j,t} w_j \ge w(T_S \cap T^*)$ since (x^*, y^*) is the optimal fractional solution. Therefore, $\sum_{j\in\bar{T}_S,t} x^*_{j,t} w_j \ge \frac{1}{\alpha}\sum_{C\in\mathcal{C}} y^*_C w_C$ and $(1+\frac{1}{\alpha})\sum_{j\in\bar{T}_S,t} x^*_{j,t} w_j \ge \frac{1}{\alpha}\left(\sum_{C\in\mathcal{C}} y^*_C w_C + \sum_{j\in\bar{T}_S,t} x^*_{j,t} w_j\right) = \frac{1}{\alpha}\text{opt}_{LP}$. This implies that $\sum_{j\in\bar{T}_S,t} x^*_{j,t} w_j \ge \text{opt}_{LP}/(\alpha+1)$. Since $\sum_{(j,t)\in T_{S!}} x^*_{j,t} w_j \le 4\eta\text{opt}_{LP} \le \frac{\text{opt}_{LP}}{2(\alpha+1)} \le \frac{1}{2}\sum_{j\in\bar{T}_S,t} x^*_{j,t} w_j$ the claim follows. ◀

Each non-problematic pair is discarded only with probability at most $1-\eta$. Therefore, the expected profit of the auxiliary solution x is at least an η-fraction of the profit of the non-problematic pairs. Due to Lemma 30 we can neglect the profit due to problematic pairs. For $\eta := \frac{1-O(\varepsilon)}{8(\alpha+1)}$ the claim of Lemma 26 follows from some simple calculation.

Arbitrary stair-solutions

In order to compute a profitable γ-stair-solution we device a DP that intuitively sweeps the path from left to right and guesses the stair-blocks in the optimal stair-solution. For each stair-block we invoke the algorithm above. Since each edge can be used by at most γ stair-blocks and large tasks we obtain a running time of $n^{(\gamma\log n)^{O(c/\delta)}}$. Since we require the stair-blocks to be compatible, there can be no task that can be assigned to more than one stair-block, even if the subproblems for each stair-block is solved independently. Recall that for a large task $i \in T_L$ we required that $h(i) < d_i$ for its computed height $h(i)$ and hence we cannot assign it into two stair-blocks, even if it would fit into the respective areas of the stair-blocks. We defer the details to the full version of the paper [28].

References

1 Anna Adamaszek and Andreas Wiese. A quasi-PTAS for the two-dimensional geometric knapsack problem. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2015)*, pages 1491–1505. SIAM, 2015.

2 Aris Anagnostopoulos, Fabrizio Grandoni, Stefano Leonardi, and Andreas Wiese. A mazing $2+\varepsilon$ approximation for unsplittable flow on a path. In *SODA*, 2014.

3 N. Bansal, A. Chakrabarti, A. Epstein, and B. Schieber. A quasi-PTAS for unsplittable flow on line graphs. In *STOC*, pages 721–729. ACM, 2006.

4 N. Bansal, Z. Friggstad, R. Khandekar, and R. Salavatipour. A logarithmic approximation for unsplittable flow on line graphs. In *SODA*, pages 702–709, 2009.

5 Nikhil Bansal, Alberto Caprara, Klaus Jansen, Lars Prädel, and Maxim Sviridenko. A structural lemma in 2-dimensional packing, and its implications on approximability. In *Algorithms and Computation*, pages 77–86. Springer, 2009.

6 Amotz Bar-Noy, Reuven Bar-Yehuda, Ari Freund, Joseph Naor, and Baruch Schieber. A unified approach to approximating resource allocation and scheduling. *Journal of the ACM (JACM)*, 48(5):1069–1090, 2001.

7 Reuven Bar-Yehuda, Michael Beder, Yuval Cohen, and Dror Rawitz. Resource allocation in bounded degree trees. *Algorithmica*, 54(1):89–106, 2009.

8 Reuven Bar-Yehuda, Michael Beder, and Dror Rawitz. A constant factor approximation algorithm for the storage allocation problem. *Algorithmica*, 77(4):1105–1127, 2017.

9 Jatin Batra, Naveen Garg, Amit Kumar, Tobias Mömke, and Andreas Wiese. New approximation schemes for unsplittable flow on a path. In *SODA*, pages 47–58, 2015. doi:10.1137/1.9781611973730.5.

10 Paul Bonsma, Jens Schulz, and Andreas Wiese. A constant-factor approximation algorithm for unsplittable flow on paths. *SIAM Journal on Computing*, 43:767–799, 2014.

11 Adam L Buchsbaum, Howard Karloff, Claire Kenyon, Nick Reingold, and Mikkel Thorup. Opt versus load in dynamic storage allocation. *SIAM Journal on Computing*, 33(3):632–646, 2004.

12 Gruia Călinescu, Amit Chakrabarti, Howard J. Karloff, and Yuval Rabani. An improved approximation algorithm for resource allocation. *ACM Transactions on Algorithms*, 7:48:1–48:7, 2011. doi:10.1145/2000807.2000816.

13 A. Chakrabarti, C. Chekuri, A. Gupta, and A. Kumar. Approximation algorithms for the unsplittable flow problem. *Algorithmica*, 47:53–78, 2007.

14 C. Chekuri, A. Ene, and N. Korula. Unsplittable flow in paths and trees and column-restricted packing integer programs. In *APPROX-RANDOM*, pages 42–55, 2009.

15 C. Chekuri, M. Mydlarz, and F. Shepherd. Multicommodity demand flow in a tree and packing integer programs. *ACM Transactions on Algorithms*, 3, 2007.

16 Waldo Gálvez, Fabrizio Grandoni, Sandy Heydrich, Salvatore Ingala, Arindam Khan, and Andreas Wiese. Approximating geometric knapsack via l-packings. In *58th Annual Symposium on Foundations of Computer Science (FOCS 2017)*, pages 260–271. IEEE, 2017.

17 Jordan Gergov. Approximation algorithms for dynamic storage allocation. In *Algorithms–ESA'96*, pages 52–61. Springer, 1996.

18 Jordan Gergov. Algorithms for compile-time memory optimization. In *Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*, pages 907–908. Society for Industrial and Applied Mathematics, 1999.

19 Fabrizio Grandoni, Tobias Mömke, Andreas Wiese, and Hang Zhou. To augment or not to augment: Solving unsplittable flow on a path by creating slack. In *SODA*, pages 2411–2422, 2017.

20 Fabrizio Grandoni, Tobias Mömke, Andreas Wiese, and Hang Zhou. A $(5/3 + \varepsilon)$-approximation for unsplittable flow on a path: placing small tasks into boxes. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 607–619, 2018. doi:10.1145/3188745.3188894.

21 Sandy Heydrich and Andreas Wiese. Faster approximation schemes for the two-dimensional knapsack problem. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2017)*, pages 79–98, 2017. doi:10.1137/1.9781611974782.6.

22 Klaus Jansen and Roberto Solis-Oba. A polynomial time approximation scheme for the square packing problem. In Andrea Lodi, Alessandro Panconesi, and Giovanni Rinaldi, editors, *Integer Programming and Combinatorial Optimization*, pages 184–198, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

23 Klaus Jansen and Guochuan Zhang. Maximizing the number of packed rectangles. In *Scandinavian Workshop on Algorithm Theory*, pages 362–371. Springer, 2004.

24 Klaus Jansen and Guochuan Zhang. Maximizing the total profit of rectangles packed into a rectangle. *Algorithmica*, 47(3):323–342, 2007.

25 Hal A Kierstead. The linearity of first-fit coloring of interval graphs. *SIAM Journal on Discrete Mathematics*, 1(4):526–530, 1988.

26 Hal A Kierstead. A polynomial time approximation algorithm for dynamic storage allocation. *Discrete Mathematics*, 88(2):231–237, 1991.

27 Tobias Mömke and Andreas Wiese. A $(2+\varepsilon)$-approximation algorithm for the storage allocation problem. In *Proceedings of the 42nd Annual International Colloquium on Automata, Languages and Programming (ICALP 2015)*, volume 9134 of *Lecture Notes in Computer Science*, pages 973–984. Springer, 2015.

28 Tobias Mömke and Andreas Wiese. Breaking the barrier of 2 for the storage allocation problem. *CoRR*, abs/1911.10871, 2019. `arXiv:1911.10871`.

29 C. A. Phillips, R. N. Uma, and J. Wein. Off-line admission control for general scheduling problems. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '00)*, pages 879–888. ACM, 2000.

On the Complexity of Zero Gap MIP*

Hamoon Mousavi
Department of Computer Science, University of Toronto, Canada
hmousavi@cs.toronto.edu

Seyed Sajjad Nezhadi
Department of Computer Science, University of Toronto, Canada
sajjad.nezhadi@mail.utoronto.ca

Henry Yuen
Department of Computer Science and Department of Mathematics, University of Toronto, Canada
hyuen@cs.toronto.edu

Abstract

The class MIP^* is the set of languages decidable by multiprover interactive proofs with quantum entangled provers. It was recently shown by Ji, Natarajan, Vidick, Wright and Yuen that MIP^* is equal to RE, the set of recursively enumerable languages. In particular this shows that the complexity of approximating the quantum value of a non-local game G is equivalent to the complexity of the Halting problem.

In this paper we investigate the complexity of deciding whether the quantum value of a non-local game G is *exactly* 1. This problem corresponds to a complexity class that we call *zero gap* MIP^*, denoted by MIP_0^*, where there is no promise gap between the verifier's acceptance probabilities in the YES and NO cases. We prove that MIP_0^* extends beyond the first level of the arithmetical hierarchy (which includes RE and its complement coRE), and in fact is equal to Π_2^0, the class of languages that can be decided by quantified formulas of the form $\forall y \, \exists z \, R(x,y,z)$.

Combined with the previously known result that MIP_0^{co} (the *commuting operator* variant of MIP_0^*) is equal to coRE, our result further highlights the fascinating connection between various models of quantum multiprover interactive proofs and different classes in computability theory.

2012 ACM Subject Classification Theory of computation → Interactive proof systems; Theory of computation → Computability; Theory of computation → Quantum complexity theory

Keywords and phrases Quantum Complexity, Multiprover Interactive Proofs, Computability Theory

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.87

Category Track A: Algorithms, Complexity and Games

Related Version Arxiv version available at `https://arxiv.org/abs/2002.10490`.

Funding *Hamoon Mousavi*: was supported by the Ontario Graduate Scholarship (OGS).
Henry Yuen: was supported by NSERC Discovery Grant 2019-06636.

Acknowledgements We thank Matt Coudron, Thomas Vidick, and especially William Slofstra for numerous helpful discussions. We also thank the reviewers of ICALP 2020 for suggestions to improve the presentation.

1 Introduction

A two-player *non-local game* is played between a verifier and two cooperating players named Alice and Bob who cannot communicate with each other once the game starts. During the game, the verifier samples a pair of questions (x, y) from a joint distribution μ, sends x to Alice and y to Bob, who respond with answers a and b respectively. The verifier accepts if and only if $D(x, y, a, b) = 1$ for some predicate D. The *quantum value* of a non-local game G, denoted by $\omega_q(G)$, is defined to be the supremum of the verifier's acceptance probability over all possible finite dimensional quantum strategies of Alice and Bob for the game G.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 87; pp. 87:1–87:12
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

What is the complexity of computing the quantum value of non-local games? In [16], Slofstra proved that the problem of determining whether a given game G has $\omega_q(G) = 1$ is *undecidable.* Recently, it was shown that *approximating* $\omega_q(G)$ up to any additive constant is also an uncomputable problem [11]. In particular, there is a computable reduction from Turing machines M to non-local games G_M such that if M halts (when run on an empty input), then $\omega_q(G_M) = 1$, and otherwise $\omega_q(G_M) \leq \frac{1}{2}$. Since determining whether a given Turing machine halts (i.e. the Halting problem) is undecidable, so is the problem of determining whether the quantum value of a non-local game is 1 or at most $\frac{1}{2}$.

Conversely, one can reduce the problem of approximating the quantum value of non-local games to the Halting problem; there is an algorithm that for every non-local game G exhaustively searches over finite-dimensional strategies of increasing dimension to find one that succeeds with probability close to 1 (above 0.99, say). If $\omega_q(G) = 1$ then the algorithm is guaranteed to find such a strategy; otherwise if $\omega_q(G) \leq 1/2$ the algorithm will run forever. In complexity-theoretic terms, this shows that the class MIP^*, the set of languages decidable by multiprover interactive proofs with quantum provers, is equal to RE, the set of recursively enumerable languages (i.e. the class for which the Halting problem is complete).

In this paper, we return to the problem originally investigated by Slofstra [16]: what is the complexity of deciding if $\omega_q(G)$ is *exactly* equal to 1 for nonlocal games G? This corresponds to the complexity class that we call *zero gap* MIP^*, denoted by MIP_0^*. In this model of interactive proofs, in the YES case (i.e. $x \in L$), there is a sequence of finite-dimensional prover strategies that cause the verifier to accept with probability approaching 1. In the NO case (i.e. $x \notin L$), all finite-dimensional prover strategies are rejected with positive probability – but could be arbitrarily close to 0. It is easy to see that $\mathsf{MIP}^* \subseteq \mathsf{MIP}_0^*$ and thus MIP_0^* contains undecidable languages. Furthermore, we know that MIP_0^* *cannot* be equal to MIP^*; the results of [16, 5] imply that coRE, the complement of RE, is also contained in MIP_0^*. Since $\mathsf{RE} \neq \mathsf{coRE}$, this implies that MIP_0^* strictly contains $\mathsf{MIP}^* = \mathsf{RE}$.

What problems can be reduced to the task of exactly computing the quantum value of non-local games, rather than "just" approximating it? We characterize the class MIP_0^* by showing that it is equal to Π_2^0, a class that belongs to the *arithmetical hierarchy* from computability theory. The arithmetical hierarchy is defined by classes of languages decidable via formulas with alternating quantifiers. For example, the class RE is equal to the class Σ_1^0, which is the set of languages L of the form $\{x : \exists y.\, R(x, y) = 1\}$ for some decidable predicate R. The class coRE is equal to Π_1^0, the set of languages of the form $\{x : \forall y.\, R(x, y) = 1\}$. The class Π_2^0 is the set of languages L of the form $\{x : \forall y.\, \exists z.\, R(x, y, z) = 1\}$.

An equivalent definition of the class Π_2^0 is that it is the set of languages L such that there is a Turing machine A that has *oracle access* to the Halting problem, and $x \notin L$ if and only if $A(x) = 1$. It is known that Π_2^0 strictly contains $\Sigma_1^0 = \mathsf{RE}$. This shows that MIP_0^* contains problems that are *harder* (in a computability sense) than the Halting problem.

We specifically show that there exists a computable reduction from Π_2^0 languages to the problem of deciding whether a *three-player* non-local game G has quantum value 1. It is likely that a similar reduction holds for two-player non-local games but we leave this for future work. We also show that the problem of deciding if a non-local game has quantum value 1 can be reduced to a Π_2^0 language, thus establishing the equality $\mathsf{MIP}_0^* = \Pi_2^0$.

This paper, combined with the results of [11] and [16], paints a fascinating landscape about the complexity of quantum multiprover interactive proofs, in which there are four different complexity classes to consider. The first two are MIP^* and MIP_0^*, which we defined already. The second two are $\mathsf{MIP}^{\mathsf{co}}$ and its zero-gap variant $\mathsf{MIP}_0^{\mathsf{co}}$. The class $\mathsf{MIP}^{\mathsf{co}}$ stands for languages that are decidable by quantum multiprover interactive proofs in the *commuting* *operator* model: here, the provers are allowed to use infinite-dimensional quantum strategies, and the measurement operators of Alice only need to commute with those of Bob (rather than be in tensor product).

$$\begin{tikzcd}
\Sigma_2^0 & & \boldsymbol{\Pi_2^0} = \mathsf{MIP}^*_{\boldsymbol{0}} \\
& \Delta_2^0 \arrow[ul] \arrow[ur] & \\
\mathsf{MIP}^* = \Sigma_1^0 \arrow[uu] \arrow[ur] & & \Pi_1^0 = \mathsf{MIP}_0^{\mathsf{co}} \overset{?}{=} \mathsf{MIP}^{\mathsf{co}} \arrow[uu] \arrow[ul] \\
& \Delta_1^0 \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

Figure 1 The computability landscape of quantum multiprover interactive proofs. Arrows denote inclusion. The set Δ_1^0 denotes the set of all decidable languages. The set Σ_1^0 denotes the recursively enumerable languages, and Π_1^0 denotes the set of co-recursively enumerable languages. It is known that $\mathsf{MIP}^{\mathsf{co}} \subseteq \mathsf{MIP}_0^{\mathsf{co}}$, but unknown whether they are equal.

One of the consequences of the fact that $\mathsf{MIP}^* = \mathsf{RE}$ is that $\mathsf{MIP}^{\mathsf{co}} \neq \mathsf{MIP}^*$. This is because $\mathsf{MIP}^{\mathsf{co}} \subseteq \mathsf{coRE}$, due to the fact that the commuting operator value of a non-local game can be upper-bounded using a convergent sequence of semidefinite programs [14, 4]. It is also the case that $\mathsf{MIP}_0^{\mathsf{co}} \subseteq \mathsf{coRE}$, and in fact equality holds due to [16, 3]. It remains an open question to determine if $\mathsf{MIP}^{\mathsf{co}} = \mathsf{MIP}_0^{\mathsf{co}} = \mathsf{coRE}$.

There are a number of curious and counter-intuitive aspects about this landscape of complexity for non-local games. First, if $\mathsf{MIP}^{\mathsf{co}} = \mathsf{coRE}$, then there would be a pleasing symmetry in that $\mathsf{MIP}^* = \mathsf{RE}$ and $\mathsf{MIP}^{\mathsf{co}} = \mathsf{coRE}$ (even though the "co" refer to different things on each side of the equation!). On the other hand, we have that $\mathsf{MIP}_0^* = \Pi_2^0$ and $\mathsf{MIP}_0^{\mathsf{co}} = \mathsf{coRE}$, meaning that – in the zero gap setting – there are *more* languages that can be verified with provers using (a limit of) finite-dimensional strategies than can be decided with provers using infinite-dimensional commuting operator strategies! Of course, in the setting of interactive proofs, giving provers access to more resources can change the complexity of the interactive proof model in unexpected ways.

1.1 Proof overview

We prove the lower bound $\Pi_2^0 \subseteq \mathsf{MIP}_0^*$ by combining two components: first we leverage the result of [11] that $\mathsf{MIP}^* = \mathsf{RE}$ as a black box, which implies that there is a quantum multiprover interactive proof for the Halting problem. Next, we use a *compression theorem* for quantum multiprover interactive proofs that was proved in [5]. A compression theorem, roughly speaking, states that given a verifier V for a quantum multiprover interactive protocol (which can be modeled as a Turing machine with tapes to receive/send messages to the provers), one can compute a much more time-efficient verifier V' whose quantum value is related in some predictable way to the quantum value of V. Several recent results about the complexity of non-local games crucially rely on proving compression theorems with various properties [10, 5, 13, 11].

In more detail, the compression theorem of [5] (which in turn is a refinement of the compression theorem of [10]) states that given a description of a verifier V, one can compute a description of a three-player[1] non-local game G_V (which is a multiprover protocol with only one round of interaction) whose properties are as follows:

1. The time complexity of the verifier in G_V is *polylogarithmic* in the time complexity of V.
2. The quantum value of the protocol executed by V is related to the quantum value of G_V in the following manner:

$$\omega_q(G_V) \geq \frac{1}{2} + \frac{1}{2}\omega_q(V)$$

and furthermore if $\omega_q(V) < 1$ then $\omega_q(G_V) < 1$.

The utilization of the compression theorem of [5] is the reason why the main result of this paper holds for three-player non-local games, rather than two.

We call this compression theorem a "zero gap" compression theorem, because it does not preserve any promise gap on the value of the input verifier V: if the value of V is promised to be either 1 or 1/2, then G_V is only guaranteed to have value either 1 or 3/4. If we iterate this compression procedure, then we get a promise gap that goes to zero. In contrast, the compression theorem used to prove $\mathsf{MIP}^* = \mathsf{RE}$ *is* gap-preserving.

The zero gap compression theorem was used to prove that $\mathsf{coRE} \subseteq \mathsf{MIP}_0^*$ in [5]. At a high level, this is shown by constructing a verifier that recursively calls the zero gap compression procedure on itself. In this paper, we follow this approach, except we also embed an MIP^* protocol for RE inside the verifier that is recursively calling the zero gap compression procedure; this composition of protocols allows the verifier to verify languages in Π_2^0.

1.2 Further remarks

$\mathsf{MIP}^* = \mathsf{RE}$ is equivalent to gap-preserving compression

As mentioned, the key to proving $\mathsf{MIP}^* = \mathsf{RE}$ [11] was establishing a gap-preserving compression theorem for non-local games, albeit for a special case of non-local games satisfying a so-called "normal form" property. In Section 4, we present a relatively simple – but in our opinion quite interesting – observation that $\mathsf{MIP}^* = \mathsf{RE}$ is in some sense, *equivalent* to a gap-preserving compression theorem.

A proof of $\mathsf{MIP}_0^* = \Pi_2^0$ under weaker assumptions?

One might wonder if there might be an elementary way of proving that $\mathsf{MIP}_0^* = \Pi_2^0$, *without* relying on the statement that $\mathsf{MIP}^* = \mathsf{RE}$. For example, the results of [16, 5] show that $\mathsf{coRE} \subseteq \mathsf{MIP}_0^*$ and furthermore [16] shows that $\mathsf{coRE} = \mathsf{MIP}_0^{co}$. These previous "zero-gap results" do not appear to have the same mathematical consequences as $\mathsf{MIP}^* = \mathsf{RE}$ (e.g. yielding a negative answer to Connes' embedding problem if $\mathsf{RE} \subseteq \mathsf{MIP}^*(2)$, the two-player variant of MIP^*), which suggests the intuition that characterizing the complexity of *exactly* computing the quantum (or commuting operator) value of nonlocal games may be fundamentally easier than characterizing the complexity of *approximating* it.

This intuition is not entirely correct: the "zero-gap" statement $\mathsf{MIP}_0^* = \Pi_2^0$ is already enough to yield a negative answer to Tsirelson's problem: there exists a k where k-partite commuting operator correlations cannot be approximated by finite dimensional correlations.

[1] The results of [5] are stated for games with 15 players, but can be improved to hold for 3-player games by using a different error correcting code in the construction.

Put another way, if Tsirelson's problem has a positive answer, then the commuting operator and quantum values of games are always equal, and then $\mathsf{MIP}^*_0 = \mathsf{MIP}^{co}_0 = \mathsf{coRE}$. However, Π^0_2 strictly contains coRE – thus Tsirelson's problem has a negative answer. Furthermore, Tsirelson's problem for $k = 2$ is known to be equivalent to Connes' embedding problem [6, 12, 15].

This suggests that our characterization of the class MIP^*_0 must necessarily involve a nontrivial tool such as $\mathsf{MIP}^* = \mathsf{RE}$.

1.3 Open problems

We list some open problems.

1. Just as the complexity statement $\mathsf{MIP}^* = \mathsf{RE}$ has consequences for questions in pure mathematics (such as the Connes' embedding problem), does the equality $\mathsf{MIP}^*_0 = \Pi^0_2$ have any implications for operator algebras? We believe there may be a connection to model-theoretic approaches to the Connes' embedding problem (see, e.g., [8, 7]).
2. What is the complexity of $\mathsf{MIP}^{\mathsf{co}}$? Is it equal to coRE?
3. Can the reduction from Π^0_2 languages to the problem of deciding whether $\omega_q(G) = 1$ be improved to hold for two-player games G?
4. We showed that, essentially, $\mathsf{MIP}^* = \mathsf{RE}$ implies a gap-preserving compression theorem. Can one show that it also implies in a black-box fashion, a zero gap compression theorem, of the same kind as proved in [5]? This then proves that $\mathsf{MIP}^* = \mathsf{RE}$ directly implies $\mathsf{MIP}^*_0 = \Pi^0_2$.
5. Does $\mathsf{MIP}^*_0 = \Pi^0_2$ imply $\mathsf{MIP}^* = \mathsf{RE}$ in a "black-box" fashion?

2 Preliminaries

We write $\mathbb{N}$ to denote the natural numbers $\{1, 2, 3, \ldots\}$. All logarithms are base 2. For a string $x \in \{0,1\}^*$ let $|x|$ denote the length of x. We let

$$\log^*(n) = \begin{cases} 0, & n \le 1 \\ 1 + \log^*(\log(n)), & n > 1 \end{cases}$$

denote the iterated logarithm function.

2.1 Turing machines and the arithmetical hierarchy

A total Turing machine is one that halts on every input. Fix a string encoding of Turing machines, and for a Turing machine M, let $|M|$ denote the length of the encoding of M.

▶ **Proposition 1** (Universal Turing machine). *There exists a universal constant $C > 0$ and a universal Turing machine $\mathfrak{U}$ that, given an input pair (M, x) where M is an encoding of a Turing machine, computes $M(x)$ in time $C \max(|M|, \mathsf{TIME}(M, x))^2$, where $\mathsf{TIME}(M, x)$ is the number of steps taken by M on input x before it halts.*

▶ **Definition 2.** *The i-th level of the* arithmetical hierarchy *contains 3 classes Σ^0_i, Π^0_i, and Δ^0_i. The class Σ^0_i is the set of languages defined as*

$$L = \{x \in \{0,1\}^* : \exists y_1 \forall y_2 \exists y_3 \cdots Q\, y_i\, R(x, y_1, \cdots, y_i) = 1\}$$

for some total Turing machine R, where Q is the $\forall$ quantifier when i is even and otherwise is the $\exists$ quantifier. The class Π^0_i is the complement of Σ^0_i, and $\Delta^0_i = \Sigma^0_i \cap \Pi^0_i$.

In particular the first level of the arithmetical hierarchy corresponds to the classes $\Sigma^0_1 = \mathsf{RE}$, $\Pi^0_1 = \mathsf{coRE}$, and Δ^0_1 the set of decidable languages $\mathsf{RE} \cap \mathsf{coRE}$.

2.2 Interactive verifiers

In this section, we model multiprover interactive protocols, which is specified by a *verifier V*, as a randomized algorithm. In the protocol, the verifier V interacts with multiple provers, and at the end of the protocol the verifier outputs a bit indicating whether to accept or reject. A verifier can be identified with the interactive protocol it executes, and vice versa.

In more detail, define a *k-input, r-prover verifier V* to be a randomized interactive Turing machine that has k designated input tapes, r communication tapes, a single workspace tape, and a single output tape. An interaction with r provers is executed in the following way: the Turing machine V alternates between computation and communication phases; in the computation phase, the Turing machine behaves like a normal Turing machine with $k+r+2$ tapes, and it may halt and indicate accept or reject on the output tape. It can also pause its computation and go into a communication phase, in which case the contents of each of i-th communication tape is read by the i-th prover, who then edits the i-th communication tape with its answer. After all the provers have finished with their responses, the next computation phase resumes. This is the standard way of modeling interactive Turing machines [1]. In this formulation, a non-local game is simply specified by a 0-input, 2-prover verifier V that has only one communication phase.

Given a k-input, r-prover verifier V, define its *time complexity* with respect to a k-tuple of inputs $(x_1, \ldots, x_k)$ to be the maximum number of time steps taken by the verifier V when it is initialized with $(x_1, \ldots, x_k)$ on its k input tapes, over all possible responses of the r-provers, before it halts. We denote this by $\mathsf{TIME}(V(x_1, \ldots, x_k))$.

We now define, in a somewhat informal level, *finite-dimensional prover strategies* (or simply a *strategy*) $\mathcal{S}$ for the interaction specified by a k-input, r-prover verifier V. This is a specification of the following data:

1. Local dimension $d \in \mathbb{N}$,
2. A state $|\psi\rangle \in (\mathbb{C}^d)^{\otimes r}$, and
3. For every prover i, for every round $t \in \mathbb{N}$, for every string $\pi \in \{0,1\}^*$, a POVM $\{M^a_{i,t,\pi}\}_a$ acting on $\mathbb{C}^d$.

Given a verifier V, a k-tuple $(x_1, \ldots, x_k)$, and a prover strategy $\mathcal{S}$ for V, the interaction proceeds as follows: at the beginning of the protocol, the provers share the state $|\psi\rangle$, and the verifier's input tapes are initialized to $(x_1, \ldots, x_k)$. At round t, the i-th prover performs the measurement $\{M^a_{i,t,\pi}\}_a$ on its local space to obtain an outcome a, where π is the *history* of all the messages seen by prover i in all previous rounds (including the message from the verifier in the t-th round). It then writes outcome a on the i-th communication tape of the verifier. Thus at each round the shared state between the provers depend on the outcomes of their measurements, and evolves probabilistically over time. The *value of strategy $\mathcal{S}$ in the interaction with verifier V on input* $(x_1, \ldots, x_k)$ is defined to be the probability that the verifier halts and accepts. We denote this by $\omega_q(V(x_1, \ldots, x_k), \mathcal{S})$. The *quantum value of verifier V on input* $(x_1, \ldots, x_k)$ is defined to be the supremum of $\omega_q(V(x_1, \ldots, x_k), \mathcal{S})$ over all finite-dimensional strategies $\mathcal{S}$, which we denote by $\omega_q(V(x_1, \ldots, x_k))$.

▶ **Definition 3.** *Let $m, r \in \mathbb{N}$ and let $0 \le s \le c \le 1$. The class $\mathsf{MIP}^*[m, r, c, s]$ is defined to be the set of languages L for which there exists a verifier V and a polynomial $p(n)$ with the following properties:*

1. *V is a 1-input, r-prover verifier that halts after m communication phases.*
2. *For all x, $\mathsf{TIME}(V(x)) \le p(|x|)$.*
3. *If $x \in L$, then $\omega_q(V(x)) \ge c$.*
4. *If $x \notin L$, then $\omega_q(V(x)) < s$.*

We define the class MIP^* to be the union of $\mathsf{MIP}^*[m, r, c, s]$ for all $m, r \in \mathbb{N}$ and $c > s$. We define the class MIP_0^* to be the union of $\mathsf{MIP}^*[m, r, 1, 1]$ over all $m, r \in \mathbb{N}$. In other words, in the YES case (i.e., $x \in L$), there is a sequence of finite-dimensional prover strategies that are accepted with probability approaching 1. In the NO case (i.e., $x \notin L$), there exists a positive $\varepsilon > 0$ (that generally depends on x) such that all finite dimensional strategies are rejected with probability at least ε.

2.3 Compression of quantum multiprover interactive protocols

In this section we formally present the two main ingredients used in our proof: the zero gap compression procedure of [5], and the reduction from the Halting problem to the problem of approximating the quantum value of a quantum multiprover interactive protocol.

First we introduce the definition of λ-boundedness, which specifies how both the description and time complexity of a verifier is bounded by a polynomial with exponent λ.

▶ **Definition 4.** *Let $\lambda \in \mathbb{N}$. A $(k+1)$-input r-prover verifier V is λ-bounded if for all integers $n \in \mathbb{N}$, $x_1, \ldots, x_k \in \{0,1\}^*$, we have $\mathsf{TIME}(V(n, x_1, ..., x_k)) \le \lambda(n \cdot |x_1| \cdots |x_k|)^{\lambda}$.*

Here, we assume that the first input to a verifier V is an integer $n \in \mathbb{N}$ which intuitively specifies a "complexity" parameter.

▶ **Theorem 5** (Zero-gap compression [5, Theorem 6.1])**.** *Let $r \ge 3$ be an integer. There exists a universal constant $C_{comp} \in \mathbb{N}$ such that for every $\lambda \in \mathbb{N}$, there exists a Turing machine $\mathrm{COMPRESS}_\lambda$ with the following properties. Given as input a $(k+1)$-input r-prover verifier V, the Turing machine $\mathrm{COMPRESS}_\lambda$ outputs a $(k+1)$-input r-prover verifier $V^\#$ in time $C_{comp}(|V| \cdot \lambda)^{C_{comp}}$ with the following properties: for all $x_1, \ldots, x_k \in \{0,1\}^*$, we have*

1. *if V is λ-bounded, then $\omega_q(V^\#(n, x_1, ...x_k)) \ge \frac{1}{2} + \frac{1}{2}\omega_q(V(2^n, x_1, ...x_k))$,*
2. *if V is λ-bounded and $\omega_q(V(2^n, x_1, ...x_k)) < 1$, then $\omega_q(V^\#(n, x_1, ...x_k)) < 1$,*
3. *for all integers $n \in \mathbb{N}$, $x_1, \ldots, x_k \in \{0,1\}^*$, we have $\mathsf{TIME}(V^\#(n, x_1, ..., x_k)) \le C_{comp}(\lambda \cdot n \cdot |x_1| \cdots |x_k|)^{C_{comp}}$.*

The zero-gap compression theorem, as presented here, differs from the one presented in [5, Theorem 6.1]. For example, verifiers in [5] are described using so-called "Gate Turing Machines" (GTMs). However, using the same oblivious Turing machine simulation techniques as discussed in the appendix of [5], from a verifier V (as defined in this paper), we can obtain a GTM that specifies the same interactive protocol. Another difference, as remarked in the introduction, is that here the compression result applies to protocols with three or more players, whereas it is stated for protocols with 15 or more players in [5]. However, the results of [5] can be adapted to the case of three players by using a $[[3, 1, 2]]_3$ error detecting code with qutrits (instead of using the 7-qubit Steane code with qubits) [2].

Next we present the main result of [11], which presents a computable reduction from the Halting problem to the problem of approximating the quantum value of a non-local game.

▶ **Theorem 6** ($\mathsf{MIP}^* = \mathsf{RE}$ [11])**.** *There exists a Turing machine H and a universal constant $C_{\mathrm{HALT}} \in \mathbb{N}$ with the following properties. Given as input a Turing machine M, it runs in time $C_{\mathrm{HALT}}|M|^{C_{\mathrm{HALT}}}$ and outputs a 0-input 2-prover verifier $V_{\mathrm{HALT},M}$ such that*

1. *If M halts on empty tape then $\omega_q(V_{\mathrm{HALT},M}) = 1$, and otherwise $\omega_q(V_{\mathrm{HALT},M}) \le \frac{1}{2}$.*
2. $\mathsf{TIME}(V_{\mathrm{HALT},M}) \le C_{\mathrm{HALT}}|M|^{C_{\mathrm{HALT}}}$.

3 $\mathsf{MIP}^*_0 = \Pi^0_2$

We start this section by showing the upper bound $\mathsf{MIP}^*_0 \subseteq \Pi^0_2$.

▶ **Theorem 7.** $\mathsf{MIP}^*_0 \subseteq \Pi^0_2$

Proof. Let $L \in \mathsf{MIP}^*_0$. There exists a 1-input r-prover verifier V such that $x \in L$ iff $\omega_q(V(x)) = 1$ for all $x \in \{0,1\}^*$. Let $\mathbb{S}_{\varepsilon,d}$ be an ε-net for the space of strategies of dimension d; in particular, for every dimension-d strategy $\mathcal{S}$ there exists a strategy $\mathcal{S}' \in \mathbb{S}_{\varepsilon,d}$ such that for all verifiers V we have that $|\omega_q(V,\mathcal{S}) - \omega_q(V,\mathcal{S}')| \leq \varepsilon$ (in other words, the winning probability of the strategies differ by at most ε). Because the set of strategies over a finite dimensional Hilbert space of a fixed dimension is a compact set [9], we can take $\mathbb{S}_{\varepsilon,d}$ to be a finite set. Let $\mathbb{S}_\varepsilon = \bigcup_{d\in\mathbb{N}} \mathbb{S}_{\varepsilon,d}$, and let $\{\mathcal{S}_\varepsilon(1), \mathcal{S}_\varepsilon(2), \ldots\}$ be an enumeration of strategies in $\mathbb{S}_\varepsilon$.

Consider the following total Turing machine T: On input triple (x,m,n) where $x \in \{0,1\}^*, m,n \in \mathbb{N}$. It outputs 1 if and only $\omega_q(V(x), \mathcal{S}_{1/2m}(n)) \geq 1 - 1/m$. Now it is easy to verify that

$$L = \{x : \forall m.\ \exists n.\, T(x,m,n) = 1\},$$

and therefore L is a Π^0_2 language.

To see this, let $x \in L$. Then $\omega_q(V(x)) = 1$, and for any gap (i.e. $\frac{1}{m}$) there exists a strategy S such that $\omega_q(V(x), S) \geq 1 - \frac{1}{2m}$. Choosing $\varepsilon = 1/2m$, then there must also exist a strategy $S' \in \mathbb{S}_{1/2m}$ such that $\omega_q(V(x), S') \geq \omega_q(V(x), S) - \frac{1}{2m} \geq 1 - \frac{1}{m}$. Therefore $\forall m.\ \exists n.\, T(x,m,n) = 1$.

Likewise, if $x \notin L$ then there exists $m \in \mathbb{N}$ for which $\omega_q(V(x)) < 1 - \frac{1}{m}$ and so no strategy can win with probability greater or equal to $1 - \frac{1}{m}$. Therefore $\exists m.\ \forall n.\, T(x,m,n) = 0$. ◀

Now we prove the reverse inclusion. Fix an $L \in \Pi^0_2$ and let R be a total Turing machine such that $L = \{x \in \{0,1\}^* : \forall m. \exists n. R(x,m,n) = 1\}$. To prove $L \in \mathsf{MIP}^*_0$, we construct a 2-input 3-prover verifier V that takes as input $m \in \mathbb{N}$ and $x \in \{0,1\}^*$, and has the key property that $\omega^*(V(m,x)) = 1$ if and only if $\forall m' \geq \log^*(m). \exists n. R(x,m',n) = 1$. Therefore $\omega_q(V(1,x)) = 1$ if and only if $x \in L$.

We first give the explicit description of a 3-input 3-prover verifier V' below. We then use that to construct V. In the description of V', we refer to the Turing machine $R_{x,m}$. For every $x \in \{0,1\}^*$ and $m \in \mathbb{N}$, $R_{x,m}$ is the Turing machine that on the empty tape enumerates over $n \in \mathbb{N}$ and accepts if $R(x,m,n) = 1$, otherwise it loops forever.

Now let V be the 2-input 3-prover verifier that on the input (m,x) runs $V'(m,x,V')$. Informally, $V(m,x)$ first decides $\exists n. R(x, \log^*(m), n) = 1$ by simulating the verifier in $V_{\text{HALT},R_{x,\log^*(m)}}$ from Theorem 6. Recall that the existence of the MIP^* protocol $V_{\text{HALT},R_{x,\log^*(m)}}$ is due to $\mathsf{MIP}^* = \mathsf{RE}$ and the fact that $\exists n. R(x,\log^*(m),n) = 1$ is an RE predicate. Now if $R(x,\log^*(m),n) = 0$ for all n, then $R_{x,\log^*(m)}$ never halts. This in turn implies that V rejects with probability at least 1/2. Otherwise, if $\exists n. R(x,\log^*(m),n) = 1$, V proceeds to run the compression algorithm to obtain $V'^{\#} = \text{COMPRESS}_\lambda(V')$. It then executes $V'^{\#}(m,x,V')$. Informally speaking, due to the compression theorem, the execution of $V'^{\#}(m,x,V')$ has the same effect as recursively executing $V(2^m,x)$. Now the first duty of the verifier $V(2^m,x)$ is to decide $\exists n. R(x, 1+\log^*(m), n) = 1$. So we can apply the above reasoning this time on $V(2^m,x)$ instead of $V(m,x)$. Following this reasoning ad infinitum, we establish that $\omega_q(V(m,x)) = 1$ if and only if $\forall m' \geq \log^*(m). \exists n. R(x,m',n) = 1$. This is made precise in the proof of Theorem 9.

Input: (m, x, W) where $m \in \mathbb{N}$, $x \in \{0,1\}^*$, W is a 3-input 3-prover verifier.
Perform the following steps:

1. Compute $V_{\text{HALT},R_{x,\log^*(m)}} = H(R_{x,\log^*(m)})$ (where H is from Theorem 6).
2. Execute the interactive protocol specified by the verifier $V_{\text{HALT},R_{x,\log^*(m)}}$. If the verifier $V_{\text{HALT},R_{x,\log^*(m)}}$ rejects then reject, otherwise continue.
3. Compute $W^\# = \text{COMPRESS}_\lambda(W)$ (where COMPRESS_λ is from Theorem 5).
4. Execute the interactive protocol specified by the verifier $W^\#(m, x, W)$ and accept if and only if the verifier $W^\#(m, x, W)$ accepts.

Figure 2 Specification of the 3-input 3-prover verifier V'.

Note that Theorem 5 relates $V^\#(m, x)$ to $V(2^m, x)$. That is the reason $\log^*(m)$ (as opposed to m) is appearing in Figure 2. Note that as m increases, $\log^*(m)$ ranges over all nonnegative integers.

In order to apply Theorem 5 to compress V in step 3, we must ensure that the verifier is λ-bounded for some $\lambda \in \mathbb{N}$.

▷ Claim 8. There exists a $\lambda \in \mathbb{N}$ such that V is λ-bounded.

Proof. We bound the running time of V by bounding the running time of each of the steps in its specification. The time to generate $R_{x,\log^*(m)}$, in step 1, is $C((|R| \cdot |x| \cdot m))$ for some constant C. The time to generate the encoding of $V_{\text{HALT},R_{x,\log^*(m)}}$ is $C_{\text{HALT}}(|R| \cdot |x| \cdot m)^{C_{\text{HALT}}}$. This also bounds the running time of $V_{\text{HALT},R_{x,\log^*(m)}}$. Therefore the time to simulate $V_{\text{HALT},R_{x,\log^*(m)}}$ is bounded by $C^2_{\text{HALT}}(|R| \cdot |x| \cdot m)^{2C_{\text{HALT}}}$. The time to simulate $\text{COMPRESS}_\lambda(V)$ is $C^2_{comp}(|V| \cdot \lambda)^{2C_{comp}}$. The time to simulate $V^\#(m, x)$ is bounded by $C^2_{comp}(\lambda \cdot m \cdot |x|)^{2C_{comp}}$. Therefore the running time of $V(m, x)$ is bounded above by

$$2C^2_{\text{HALT}}(|R| \cdot |x| \cdot m)^{2C_{\text{HALT}}} + C(|R| \cdot |x| \cdot m) + C^2_{comp}(|V| \cdot \lambda)^{2C_{comp}} + C^2_{comp}(\lambda \cdot m \cdot |x|)^{2C_{comp}}.$$

The values $C_{comp}, C_{\text{HALT}}, C$, and $|R|$ are all constants so we can choose $\lambda \in \mathbb{N}$ sufficiently large so that $\lambda(m \cdot |x|)^\lambda$ is larger then the quantity above and therefore V is λ-bounded. ◁

Now that we established that V is λ-bounded, we can apply Theorem 5 to get the main theorem of this paper.

▶ **Theorem 9.** *$x \in L$ if and only if $\omega_q(V(1, x)) = 1$*

Proof. First suppose $x \in L$. Then $\forall m. \exists n. R(x, m, n) = 1$. Since the Turing machine $R_{x,m}$ halts for every $m \in \mathbb{N}$, by Theorem 6, $\omega_q(V_{\text{HALT},R_{x,m}}) = 1$. Therefore $\omega_q(V(p, x)) = \omega_q(V^\#(p, x))$, for any $p \in \mathbb{N}$, by construction (step 4). Now, from Theorem 5, we have

$$\omega_q(V(p, x)) \geq \frac{1}{2} + \frac{\omega_q(V(2^p, x))}{2},$$

and by k applications of the theorem, we obtain

$$\omega_q(V(p, x)) \geq \frac{\omega_q(V(\overbrace{2^{2^{\cdots^{2^p}}}}^{k}, x))}{2^k} + \sum_{i=1}^{k} \frac{1}{2^i}.$$

for every k. Taking the limit $k \to \infty$, we have $\omega_q(V(p, x)) = 1$ for all $p \in \mathbb{N}$. In particular $\omega_q(V(1, x)) = 1$.

Now suppose $x \notin L$. Then $\exists m. \forall n. R(x,m,n) = 0$. We prove that $\omega(V(1,x)) < 1$. Let p be the smallest integer for which $R(x, \log^*(p), n) = 0$ for every n. In other words, the Turing machine $R_{x,\log^*(p)}$ does not halt. Therefore by Theorem 6 we have that $\omega_q(V(p,x)) \leq \omega_q(V_{\text{HALT},R_{x,\log^*(p)}}) \leq \frac{1}{2}$.

If $p = 1$, we are done. Suppose $p > 1$. For all $k < p$, the game $V_{\text{HALT},R_{x,\log^*(k)}}$ never rejects since the Turing machine $R_{x,\log^*(k)}$ halts, by the minimality of p. Therefore $\omega_q(V(k,x)) = \omega_q(V^{\#}(k,x))$. So by recursively applying Theorem 5, we have that

$$\omega_q(V(p,x)) < 1 \implies \omega_q(V(1,x)) < 1.$$

Since $\omega_q(V(p,x)) \leq \omega_q(V_{\text{HALT},R_{x,\log^*(p)}}) \leq \frac{1}{2}$ then $\omega_q(V(1,x)) < 1$. ◀

▶ **Corollary 10.** $\Pi_2^0 \subseteq \mathsf{MIP}_0^*$.

Proof. Let $L \in \Pi_2^0$ then $L = \{x \in \{0,1\}^* : \forall m. \exists n. R(x,m,n) = 1\}$. Let U be the 1-input 3-prover verifier, that on input x executes the verifier $V(1,x)$ where $x \in \{0,1\}^*$. By Claim 8, $\mathsf{TIME}(U(x)) = \mathsf{TIME}(V(1,x)) \leq \lambda(1+|x|)^{\lambda}$ and by Theorem 9, $x \in L$ iff $\omega^*(U(x)) = 1$. Thus U is an MIP_0^* protocol for the language L, and $L \in \mathsf{MIP}_0^*$. ◀

This concludes the proof of the main result of this paper.

4 MIP* = RE implies gap-preserving compression

As mentioned in the introduction, the key to proving $\mathsf{MIP}^* = \mathsf{RE}$ in [11] was establishing a gap-preserving compression theorem for non-local games. Here we observe that the reverse holds: $\mathsf{MIP}^* = \mathsf{RE}$ implies a gap-preserving compression theorem.

▶ **Theorem 11.** *If* $\mathsf{MIP}^* = \mathsf{RE}$*, then there exists a Turing machine* COMPRESS*, with the following properties. Given as input a k-input r-prover verifier V,* COMPRESS *outputs a k-input 2-prover verifier $V^{\#}$ in time polynomial in the description length of V, with the following properties:*

1. *if $\omega_q(V(x_1, ..., x_k)) = 1$ then $\omega_q(V^{\#}(x_1, ..., x_k)) = 1$*
2. *if $\omega_q(V(x_1, ..., x_k)) \leq \frac{1}{2}$ then $\omega_q(V^{\#}(x_1, ..., x_k)) \leq \frac{1}{2}$*
3. *The runtime of the verifier $V^{\#}$ is polynomial in the description length of V and its input.*

Proof. COMPRESS is the Turing machine that, when given input a verifier V, it returns the description of the verifier $V^{\#}$ from Figure 3.

In the description of $V^{\#}$, we refer to the Turing machine $T_{V,(x_1,...,x_k)}$. For every k-input r-prover verifier V and $x_1, \ldots, x_k \in \{0,1\}^*$, $T_{V,(x_1,...,x_k)}$ is the Turing machine that on empty tape enumerates over finite-dimensional quantum strategies for $V(x_1, ..., x_k)$ and only accepts if it finds a strategy that wins the game with probability greater than $\frac{1}{2}$. It does this via enumerating over ε-nets (for $\varepsilon = \frac{1}{4}$) for strategies of dimension d for all $d \in \mathbb{N}$, as with the proof of Theorem 7.

By Theorem 6, if the Turing machine $T_{V,(x_1,...,x_k)}$ halts then

$$\omega_q(V_{\text{HALT},T_{V,(x_1,\ldots,x_k)}}) = 1,$$

otherwise $\omega_q(V_{\text{HALT},T_{V,(x_1,\ldots,x_k)}}) \leq \frac{1}{2}$. Also the runtime of $V_{\text{HALT},T_{V,(x_1,\ldots,x_k)}}$ is $p(|V| + |x_1| + ... + |x_n|)$, for some polynomial p.

Then if $\omega_q(V(x_1, ..., x_k)) = 1$ the Turing machine $T_{V,(x_1,...,x_k)}$ finds a strategy that wins with probability greater than $\frac{3}{4}$ and halts. Therefore

$$\omega_q(V^{\#}(x_1, ..., x_k)) = \omega_q(V_{\text{HALT},T_{V,(x_1,\ldots,x_k)}}) = 1.$$

Input: $(x_1, ..., x_k)$, where $x_1, \ldots, x_k \in \{0,1\}^*$
Perform the following steps:

1. Compute $V_{\text{HALT},T_{V,(x_1,\ldots,x_k)}} = H(T_{V,(x_1,\ldots,x_k)})$ (where H is from Theorem 6).
2. Execute the interactive protocol specified by the verifier $V_{\text{HALT},T_{V,(x_1,\ldots,x_k)}}$ and accept if and only if the verifier accepts.

Figure 3 Specification of the compressed verifier $V^{\#}$.

Otherwise, if $\omega_q(V(x_1, ..., x_k)) \leq \frac{1}{2}$ then there is no strategy that wins the game with probability $\frac{1}{2}$ and the Turing machine $T_{V,(x_1,\ldots,x_k)}$ never halts. Therefore

$$\omega_q(V^{\#}(x_1, ..., x_k)) = \omega_q(V_{\text{HALT},T_{V,(x_1,\ldots,x_k)}}) \leq \frac{1}{2}. \qquad ◀$$

Note that in this gap-preserving compression theorem, the time complexity of the verifier $V^{\#}$ is polynomial in the *description length* of V and its input – rather than the *time complexity* of V.

References

1 Michael Ben-Or, Shafi Goldwasser, Joe Kilian, and Avi Wigderson. Multi-prover interactive proofs: how to remove intractability assumptions. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 113–131, 1988.

2 Richard Cleve, Daniel Gottesman, and Hoi-Kwong Lo. How to share a quantum secret. *Physical Review Letters*, 83(3):648, 1999.

3 Matthew Coudron and William Slofstra. Complexity lower bounds for computing the approximately-commuting operator value of non-local games to high precision. *arXiv preprint*, 2019. `arXiv:1905.11635`.

4 Andrew C Doherty, Yeong-Cherng Liang, Ben Toner, and Stephanie Wehner. The quantum moment problem and bounds on entangled multi-prover games. In *2008 23rd Annual IEEE Conference on Computational Complexity*, pages 199–210. IEEE, 2008.

5 Joseph Fitzsimons, Zhengfeng Ji, Thomas Vidick, and Henry Yuen. Quantum proof systems for iterated exponential time, and beyond. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2019, page 473–480, New York, NY, USA, 2019. Association for Computing Machinery. `doi:10.1145/3313276.3316343`.

6 Tobias Fritz. Tsirelson's problem and kirchberg's conjecture. *Reviews in Mathematical Physics*, 24(05):1250012, 2012.

7 Isaac Goldbring. Enforceable operator algebras. *Journal of the Institute of Mathematics of Jussieu*, pages 1–33, 2017.

8 Isaac Goldbring and Bradd Hart. A computability-theoretic reformulation of the connes embedding problem. *arXiv preprint*, 2013. `arXiv:1308.2638`.

9 Gus Gutoski and John Watrous. Toward a general theory of quantum games. In *Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of Computing*, STOC '07, page 565–574, New York, NY, USA, 2007. Association for Computing Machinery. `doi:10.1145/1250790.1250873`.

10 Zhengfeng Ji. Compression of quantum multi-prover interactive proofs. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 289–302, 2017.

11 Zhengfeng Ji, Anand Natarajan, Thomas Vidick, John Wright, and Henry Yuen. $\mathsf{MIP}^* = \mathsf{RE}$. *arXiv preprint*, 2020. `arXiv:2001.04383`.

12 Marius Junge, Miguel Navascues, Carlos Palazuelos, David Perez-Garcia, Volkher B Scholz, and Reinhard F Werner. Connes' embedding problem and tsirelson's problem. *Journal of Mathematical Physics*, 52(1):012102, 2011.

13 Anand Natarajan and John Wright. NEEXP $\subseteq$ MIP*. In *IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 510–518, 2019.

14 Miguel Navascués, Stefano Pironio, and Antonio Acín. A convergent hierarchy of semidefinite programs characterizing the set of quantum correlations. *New Journal of Physics*, 10(7):073013, 2008.

15 Narutaka Ozawa. About the connes embedding conjecture, algebraic approaches. *Jpn. J. Math.*, 8:147–183, 2013.

16 William Slofstra. The set of quantum correlations is not closed. In *Forum of Mathematics, Pi*, volume 7. Cambridge University Press, 2019.

Hypergraph Isomorphism for Groups with Restricted Composition Factors

Daniel Neuen
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany
dneuen@mpi-inf.mpg.de

Abstract

We consider the isomorphism problem for hypergraphs taking as input two hypergraphs over the same set of vertices V and a permutation group Γ over domain V, and asking whether there is a permutation $\gamma \in \Gamma$ that proves the two hypergraphs to be isomorphic. We show that for input groups, all of whose composition factors are isomorphic to a subgroup of the symmetric group on d points, this problem can be solved in time $(n+m)^{\mathcal{O}((\log d)^c)}$ for some absolute constant c where n denotes the number of vertices and m the number of hyperedges. In particular, this gives the currently fastest isomorphism test for hypergraphs in general. The previous best algorithm for the above problem due to Schweitzer and Wiebking (STOC 2019) runs in time $n^{\mathcal{O}(d)}m^{\mathcal{O}(1)}$.

As an application of this result, we obtain, for example, an algorithm testing isomorphism of graphs excluding $K_{3,h}$ as a minor in time $n^{\mathcal{O}((\log h)^c)}$. In particular, this gives an isomorphism test for graphs of Euler genus at most g running in time $n^{\mathcal{O}((\log g)^c)}$.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Mathematics of computing → Graph algorithms; Mathematics of computing → Combinatorial algorithms; Mathematics of computing → Graphs and surfaces

Keywords and phrases graph isomorphism, groups with restricted composition factors, hypergraphs, bounded genus graphs

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.88

Category Track A: Algorithms, Complexity and Games

Related Version A full version of the paper is available at `https://arxiv.org/abs/2002.06997`.

1 Introduction

Luks's algorithm [21] is an important cornerstone of the algorithmic theory of the Graph Isomorphism Problem. With some additional improvements given later [6], it tests in time $n^{\mathcal{O}(d/\log d)}$ whether two given n-vertex graphs of maximum degree d are isomorphic. In the last four decades, Luks's algorithm has developed to a central building block for many algorithms tackling the isomorphism problem. Indeed, Luks's algorithmic framework has been used as a subroutine to design, for example, isomorphism tests for graphs of bounded genus [27], graphs of bounded tree-width [15], graphs excluding some fixed graph as a minor [31], and even graph classes that exclude some fixed graph as a topological minor [13]. Further examples include color-t-bounded graphs [4], defined by Babai et al. in the context of isomorphism testing of strongly regular graphs, unit square graphs [28], and graphs of bounded rank-width [17]. Moreover, Luks's algorithm has also played a role in the area of computational group theory for example for computing normalizers for certain groups [23].

Additionally, Luks's algorithm also forms the basis for Babai's recent quasipolynomial isomorphism test [1] as well as the corresponding canonization algorithm [2]. Indeed, Babai's algorithm follows the recursive framework of Luks's algorithm and attacks the obstacle cases where the recursion performed by Luks's algorithm does not lead to the desired running time. Hence, a natural question to ask is whether it is possible extend the group-theoretic methods added in Babai's algorithm to the setting of bounded degree graphs in order to

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 88; pp. 88:1–88:19
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

obtain improved algorithms also for the isomorphism problem for graphs of small degree. This question was answered in the affirmative by Grohe, Schweitzer and the author of this paper in [14] providing an isomorphism test for graphs of maximum degree d running in time $n^{\mathcal{O}((\log d)^c)}$ for some constant c.

With the large number of applications of Luks's algorithmic framework [21] over the last decades it is natural to ask for improvements of other algorithms that exploit Luks's methods as a subroutine. However, up to this point, the only application of the improved isomorphism test for graphs of small degree is a faster fixed-parameter tractable isomorphism test for graphs of bounded tree-width [15]. The reason for this mainly lies in the fact that most of the algorithms exploiting Luks's framework as a subroutine actually use this framework to solve more general problems than Luks's original algorithm.

To be more precise, Luks's original algorithm [21] attacks the *String Isomorphism Problem*. The input to the String Isomorphism Problem are two strings $\mathfrak{x}, \mathfrak{y} \colon \Omega \to \Sigma$, where Ω is a finite set and Σ a finite alphabet, and a permutation group $\Gamma \leq \mathrm{Sym}(\Omega)$ (given by a set of generators). The task of the String Isomorphism Problem is to decide whether there exists some $\gamma \in \Gamma$ that transforms $\mathfrak{x}$ into $\mathfrak{y}$. In order to solve the isomorphism problem for graphs of bounded degree, Luks [21] provides a polynomial-time algorithm solving the String Isomorphism Problem for all input groups Γ in the class $\widehat{\Gamma}_d$[1], the class of groups all of whose composition factors are isomorphic to a subgroup of S_d (the symmetric group on d points). To give a faster isomorphism test for graph of small degree, Grohe et al. [14] follow the same route of considering the String Isomorphism Problem and provide an algorithm solving the problem in time $n^{\mathcal{O}((\log d)^c)}$ for the class of $\widehat{\Gamma}_d$-groups.

On the other hand, many algorithms exploiting the methods of Luks do so by solving more general problems. Indeed, a common extension is the Hypergraph Isomorphism Problem for $\widehat{\Gamma}_d$-groups. Here, the input consists of two hypergraphs $\mathcal{H}_1 = (V, \mathcal{E}_1)$ and $\mathcal{H}_2 = (V, \mathcal{E}_2)$ over the same set of vertices and a $\widehat{\Gamma}_d$-group $\Gamma \leq \mathrm{Sym}(V)$, and the task is to decide whether there is some $\gamma \in \Gamma$ that transforms $\mathcal{H}_1$ into $\mathcal{H}_2$. As observed by Miller [26], with some small modifications, Luks's algorithm for the String Isomorphism Problem immediately extends to the Hypergraph Isomorphism Problem. To be more precise, by an extension of the arguments of Luks [21], the Hypergraph Isomorphism Problem for $\widehat{\Gamma}_d$-groups can be solved in time $(m+n)^{\mathcal{O}(d)}$ where n denotes the number of vertices and m the number of edges. This fact is exploited by several of the algorithms mentioned above. For example, this includes the isomorphism tests for graphs of bounded genus [27], graphs excluding some fixed graph as a (topological) minor [31], color-t-bounded graphs [4], and unit square graphs [28]. Recently, an improved algorithm for the Hypergraph Isomorphism Problem for $\widehat{\Gamma}_d$-groups running in time $n^{\mathcal{O}(d)}m^{\mathcal{O}(1)}$ was presented by Schweitzer and Wiebking [34]. While the algorithm of Schweitzer and Wiebking significantly improves the running for large numbers of hyperedges, this is irrelevant for the applications described above where the number of hyperedges is typically linearly bounded in the number of vertices of the original input graph.

The main contribution of this paper is to provide an algorithm for the Hypergraph Isomorphism Problem for $\widehat{\Gamma}_d$-groups running in time $(n+m)^{\mathcal{O}((\log d)^c)}$ for some absolute constant c. Besides its potential applications outlined above, the Hypergraph Isomorphism Problem is of course also interesting in its own right. Indeed, there is also a long history of studying the Hypergraph Isomorphism Problem in itself. One of the first results in this direction was an algorithm testing isomorphism of hypergraphs in time $2^{\mathcal{O}(n)}$ [22] where

[1] In [21] this class is denoted by Γ_d. However, in the more recent literature, Γ_d often refers to a more general class of groups.

n denotes the number of vertices. Assuming the hyperedges are not too large, this result was improved by Babai and Codenotti in [5] to a running of $n^{\widetilde{\mathcal{O}}(k^2 \cdot \sqrt{n})}$ where $\widetilde{\mathcal{O}}(\cdot)$ hides polylogarithmic factors and k denotes the maximal size of a hyperedge. Again assuming small hyperedges, another improvement follows from the work of Grohe et al. [14] resulting in an isomorphism test for hypergraphs running in time $n^{\mathcal{O}(k \cdot (\log n)^c)}$ for a constant c.

Results. The main result of this paper is a faster algorithm for the Hypergraph Isomorphism Problem for $\widehat{\Gamma}_d$-groups.

▶ **Theorem 1.** *The Hypergraph Isomorphism Problem for $\widehat{\Gamma}_d$-groups can be solved in time $(n+m)^{\mathcal{O}((\log d)^c)}$ for some absolute constant c where n denotes the number of vertices and m the number of edges of the input hypergraphs.*

An immediate consequence of this result is the fastest algorithm for testing isomorphism of hypergraphs in general (assuming the number of hyperedges is moderately exponentially bounded in the number of vertices, i.e., $m = 2^{\mathcal{O}(n^{1-\varepsilon})}$ for some $\varepsilon > 0$).

▶ **Corollary 2.** *The Hypergraph Isomorphism Problem can be solved in time $(n+m)^{\mathcal{O}((\log n)^c)}$ for some absolute constant c where n denotes the number of vertices and m the number of edges of the input hypergraphs.*

In particular, this result removes the dependence on k, the maximal size of a hyperedge, in the running time. Observe that this improvement is significant if k is large and m is small compared to $\binom{n}{k}$ (which is typical for many applications). Also, the last theorem generalizes the corresponding result for the String Isomorphism Problem for $\widehat{\Gamma}_d$-groups obtained in [14].

For the algorithm, we take a similar route than the algorithm from [14] first normalizing the input to ensure a suitable variant of Babai's Unaffected Stabilizers Theorem [1]. Based on this variant, Grohe et al. are able to extend the Local Certificates Routine, a key subroutine of Babai's quasipolynomial-time algorithm [1], to the setting of $\widehat{\Gamma}_d$-groups. Unfortunately, when going from strings to hypergraphs, there is no simple extension of the Local Certificates Routine even in the normalized setting from [14]. Intuitively speaking, the reason is that the Local Certificates Routine crucially exploits that positions in disjoint sets can be permuted independently of each other which is not the case for hypergraphs.

To circumvent this problem we introduce a novel simplification routine which is repeatedly executed during the Local Certificates Routine. The idea for the simplification is based on the following observation. Suppose that all hyperedges are identical on a window $W \subseteq V$ (i.e., $E_1 \cap W = E_2 \cap W$ for all hyperedges $E_1, E_2 \in \mathcal{E}$). In this case each permutation in $\Gamma_{(V \setminus W)}$ (the subgroup that fixes each position outside of W) is an automorphism as required by the Local Certificates Routine since there are no additional dependencies between W and $V \setminus W$ coming from the hyperedges. The main idea is to always reduce to this simple case. Intuitively speaking, this is achieved as follows. Two hyperedges $E_1, E_2 \subseteq V$ are *W-equivalent* if $E_1 \cap W = E_2 \cap W$. The algorithm creates, for each equivalence class, a copy of the vertex set and associates all hyperedges from the equivalence class with this copy. After this modification, each copy satisfies the requirement that all hyperedges are W-equivalent (actually, to realize this modification, we shall consider a more general problem). This enables us to implement a Local Certificates Routine for hypergraphs which builds the central subroutine of our isomorphism test. Unfortunately, while this settles the original problem, it also creates several additional issues that need to be addressed and which require various extensions of existing techniques making the entire algorithm quite complicated.

With the Hypergraph Isomorphism Problem for $\widehat{\Gamma}_d$-groups being used as a subroutine in a number of algorithms, it is natural to ask for the implications of this result. One of the first examples in this direction is Miller's algorithm for testing isomorphism of graphs of bounded genus [27]. However, Miller's algorithm reduces the isomorphism problem for graphs of genus g to the Hypergraph Isomorphism Problem where the input group is a Γ_d-*tower* rather than a $\widehat{\Gamma}_d$-group where $d = \mathcal{O}(g)$. The class of Γ_d-towers extends the class $\widehat{\Gamma}_d$ by, intuitively speaking, allowing to combine groups that are "almost" $\widehat{\Gamma}_d$-groups along a specified partition of the domain.

Since it is already completely unclear how to extend the algorithm of Grohe et al. [14] to Γ_d-towers it is not possible to use Miller's algorithm directly. To still obtain a faster isomorphism test for graphs of small genus, we strengthen Miller's result and develop a reduction from the isomorphism problem for graphs of genus g to the Hypergraph Isomorphism Problem for $\widehat{\Gamma}_d$-groups where $d \coloneqq 4g + 2$. Actually, our reduction works for any graph class that excludes $K_{3,h}$ as a minor (graphs of genus g exclude $K_{3,4g+3}$ as a minor [32, 18]).

To build the reduction, we introduce *t-CR-bounded graphs* which extend the notion of color-t-bounded graphs introduced in [4]. Intuitively speaking, a vertex-colored graph is t-CR-bounded if the vertex-coloring of the graph can be turned into a discrete coloring (i.e., each vertex has its own color) by repeatedly applying the standard Color Refinement algorithm (see, e.g., [10, 19]) and by splitting color classes of size at most t. We show that the isomorphism problem for t-CR-bounded graphs can be solved in time $n^{\mathcal{O}((\log t)^c)}$ by providing a reduction to the Hypergraph Isomorphism Problem for $\widehat{\Gamma}_t$-groups. While this result may already be interesting in its own right, the main purpose of the isomorphism problem for t-CR-bounded graphs in this paper is to serve as an intermediate problem.

We continue to prove that the isomorphism problem for graph classes that exclude $K_{3,h}$ as a minor is polynomial-time reducible to testing isomorphism of t-CR-bounded graphs where $t \coloneqq h - 1$. This implies the following theorem.

► **Theorem 3.** *The Graph Isomorphism Problem for graphs that exclude $K_{3,h}$ as a minor can be solved in time $n^{\mathcal{O}((\log h)^c)}$ for some absolute constant c where n denotes the number of vertices of the input graphs.*

The previous best algorithm for this setting is due to Ponomarenko [31] who provided a polynomial-time isomorphism test for graph classes that exclude an arbitrary minor. While Ponomarenko does not provide a precise analysis on the running time of his algorithm, the exponent depends at least linearly on h when excluding K_h as a minor. Hence, in the case of excluding $K_{3,h}$ as a minor, the above theorem significantly improves on all previous algorithms.

Finally, exploiting the fact that $K_{3,h}$ has Euler genus linear in h [32, 18], we obtain the following corollary.

► **Corollary 4.** *The Graph Isomorphism Problem for graphs of Euler genus at most g can be solved in time $n^{\mathcal{O}((\log g)^c)}$ for some absolute constant c where n denotes the number of vertices of the input graphs.*

While the isomorphism problem for graphs of bounded genus is also fixed-parameter tractable [20] (again, the author does not analyze the dependence of the running time on g), the algorithm from the previous corollary is guaranteed to be faster as soon as the genus passes a threshold that is polylogarithmic in the number of vertices. Also, I remark that the isomorphism problem for graphs of bounded genus can be solved in logarithmic space [12].

As a another application of the isomorphism problem for t-CR-bounded graphs, we consider the isomorphism problem for hereditarily finite sets (see also [34]). Intuitively speaking, a hereditarily finite set over ground set V is any type of combinatorial object that can be built by iteratively taking sets and tuples over previously created objects. In particular, hereditarily finite sets include graphs, vertex- and arc-colored graphs, hypergraphs and relational structures. We further extend our previous result on the Hypergraph Isomorphism Problem for $\widehat{\Gamma}_d$-groups and show that isomorphism for any pair of hereditarily finite sets with a $\widehat{\Gamma}_d$-group can be tested in time $(n+m)^{\mathcal{O}((\log d)^c)}$ where n denotes the size of the ground set and m the size of the hereditarily finite sets. In case m is only moderately exponential in n (i.e., $m = 2^{\mathcal{O}(n^{1-\varepsilon})}$ for some fixed $\varepsilon > 0$) this improves over a previous algorithm of Schweitzer and Wiebking [34] (although it should be noted that Schweitzer and Wiebking actually solve a slightly more general problem).

Related Work. Another extension of Babai's quasipolynomial-time algorithm has been independently proposed by Daniel Wiebking [37] providing another proof that the isomorphism problem for arbitrary hereditarily finite sets can be solved in quasipolynomial time. However, Wiebking not only solves the isomorphism problem for hereditarily finite sets as defined in this paper, but actually provides an algorithm for an extended setting where also implicitly represented labeling cosets are allowed as atomic objects for the hereditarily finite sets. Also, Wiebking gives a canonization algorithm which extends Babai's recent canonization algorithm for graphs [2].

But on the other hand, the algorithmic framework of Wiebking [37] is not able to exploit any restrictions on the input group. Hence, the two results are incomparable with respect to the power of the algorithms obtained.

This is also highlighted by the applications. While the results of this paper allow the design of an algorithm solving the isomorphism problem for graphs of Euler genus at most g running in time $n^{\mathcal{O}((\log g)^c)}$, Wiebking utilizes his algorithm to build an isomorphism test for graphs of tree-width at most k running in time $n^{\mathcal{O}((\log k)^c)}$.

2 Preliminaries

Graphs. A *graph* is a pair $G = (V(G), E(G))$ with vertex set $V(G)$ and edge set $E(G)$. Unless stated otherwise, all graphs are undirected and simple graphs, i.e., there are no loops or multiedges. In this setting an edge is denoted as vw where $v, w \in V(G)$. The *neighborhood* of a vertex v is denoted $N_G(v) := \{w \in V(G) \mid vw \in E(G)\}$. The *degree* of a vertex $v \in V(G)$, denoted $\deg_G(v)$, is the size of its neighborhood. Also, for a set of vertices $X \subseteq V(G)$, the neighborhood of X is defined as $N_G(X) := \left(\bigcup_{v\in X} N_G(v)\right) \setminus X$. Usually, we omit the index G if it is clear from the context and simply write $N(v)$, $N(X)$ and $\deg(v)$. For $X \subseteq V(G)$ the *induced subgraph on* X is $G[X] := (X, \{vw \mid v, w \in X, vw \in E(G)\})$. Also, $G - X := G[V(G) \setminus X]$ denotes the induced subgraph on the complement of X.

Two graphs G and H are isomorphic if there is a bijection $\varphi\colon V(G) \to V(H)$ such that $vw \in E(G)$ if and only if $\varphi(v)\varphi(w) \in E(H)$. In this case φ is an *isomorphism* from G to H, which is also denoted by $\varphi\colon G \cong H$. Moreover, $\mathrm{Iso}(G, H)$ denotes the set of all isomorphisms from G to H. The automorphism group of G is $\mathrm{Aut}(G) := \mathrm{Iso}(G, G)$. Observe that, if $\mathrm{Iso}(G, H) \neq \emptyset$, it holds that $\mathrm{Iso}(G, H) = \mathrm{Aut}(G)\varphi := \{\gamma\varphi \mid \gamma \in \mathrm{Aut}(G)\}$ for every isomorphism $\varphi \in \mathrm{Iso}(G, H)$. The Graph Isomorphism Problem takes as input two graphs G and H and asks whether they are isomorphic.

A *(vertex-)colored graph* is a tuple $G = (V(G), E(G), \chi_V)$ where $\chi_V \colon V(G) \to C$ is a mapping and C is some finite set of colors. A *vertex- and arc-colored graph* is a tuple $G = (V(G), E(G), \chi_V, \chi_E)$ where $\chi_V \colon V(G) \to C$ is a vertex-coloring and $\chi_E \colon \{(v, w) \mid vw \in E(G)\} \to C$ is an arc-coloring, where C is again some finite set of colors. Note that an uncolored graph can be interpreted as a vertex- and arc-colored graph where each vertex is assigned the same color as well as each (directed) edge is assigned the same color. The *vertex color classes* of a (colored) graph are the sets $\chi_V^{-1}(c)$ where $c \in C$. A vertex-coloring χ_V is *discrete* if all color classes are singletons, i.e., $\chi_V(v) \neq \chi_V(w)$ for all distinct $v, w \in V(G)$.

Permutation Groups. Next, we establish the basic notation for permutation groups required for this work. For a general background on group theory I refer to [33] whereas background on permutation groups can be found in [11].

A *permutation group* acting on a set Ω is a subgroup $\Gamma \leq \mathrm{Sym}(\Omega)$ of the symmetric group. The size of the permutation domain Ω is called the *degree* of Γ and, throughout this work, is denoted by $n = |\Omega|$. If $\Omega = [n]$ then we also write S_n instead of $\mathrm{Sym}(\Omega)$. Also, $\mathrm{Alt}(\Omega)$ denotes the alternating group on the set Ω and, similar to the symmetric group, we write A_n instead of $\mathrm{Alt}(\Omega)$ if $\Omega = [n]$. For $\gamma \in \Gamma$ and $\alpha \in \Omega$ we denote by α^γ the image of α under the permutation γ. The set $\alpha^\Gamma := \{\alpha^\gamma \mid \gamma \in \Gamma\}$ is the *orbit* of α. The group Γ is *transitive* if $\alpha^\Gamma = \Omega$ for some (and therefore every) $\alpha \in \Omega$.

For $\alpha \in \Omega$ the group $\Gamma_\alpha := \{\gamma \in \Gamma \mid \alpha^\gamma = \alpha\} \leq \Gamma$ is the *stabilizer* of α in Γ. The group Γ is *semi-regular* if $\Gamma_\alpha = \{\mathrm{id}\}$ for all $\alpha \in \Omega$ (id denotes the identity element of the group). For $A \subseteq \Omega$ and $\gamma \in \Gamma$ let $A^\gamma := \{\alpha^\gamma \mid \alpha \in A\}$. The *pointwise stabilizer* of A is the subgroup $\Gamma_{(A)} := \{\gamma \in \Gamma \mid \forall \alpha \in A \colon \alpha^\gamma = \alpha\}$. The *setwise stabilizer* of A is the subgroup $\Gamma_A := \{\gamma \in \Gamma \mid A^\gamma = A\}$. For a Γ-invariant set $A \subseteq \Omega$ we denote by $\Gamma[A]$ the induced natural action of Γ on A (i.e., restricting every permutation to the set A).

In order to perform computational tasks for permutation groups efficiently it is infeasible to list all elements of a permutation group. Instead, permutation groups are represented by generating sets of small size. A set $S \subseteq \Gamma$, where $\Gamma \leq \mathrm{Sym}(\Omega)$, is a *generating set* for Γ if every $\gamma \in \Gamma$ can be written as a product $\gamma = s_1 s_2 \dots s_k$ of elements $s_1, \dots, s_k \in S$. Indeed, most algorithms for permutation groups are based on so-called *strong generating sets*, which can be chosen of size quadratic in the degree of the group and can be computed in polynomial time given an arbitrary generating set (see, e.g., [35]). This enables the design of efficient algorithms for many basic computational problems for permutation groups (e.g., membership tests, computing the order of a group, the orbits of a group, and generating sets for pointwise stabilizers). For detailed background on algorithms for permutation groups I refer to [35].

Groups with Restricted Composition Factors. In this work we shall be interested in a particular subclass of permutation groups, namely groups with restricted composition factors. Let Γ be a group. A *subnormal series* is a sequence of subgroups $\Gamma = \Gamma_0 \trianglerighteq \Gamma_1 \trianglerighteq \dots \trianglerighteq \Gamma_k = \{\mathrm{id}\}$. The length of the series is k and the groups Γ_{i-1}/Γ_i are the factor groups of the series, $i \in [k]$. A *composition series* is a strictly decreasing subnormal series of maximal length. For every finite group Γ all composition series have the same family of factor groups considered as a multi-set (cf. [33]). A *composition factor* of a finite group Γ is a factor group of a composition series of Γ.

▶ **Definition 5.** *For $d \geq 2$ let $\widehat{\Gamma}_d$ denote the class of all groups Γ for which every composition factor of Γ is isomorphic to a subgroup of S_d.*

We want to stress the fact there are two similar classes of groups both typically denoted by Γ_d in the literature. One is the class we define as $\widehat{\Gamma}_d$ introduced by Luks [21] while the other one used in [3] in particular allows simple groups of Lie type of bounded dimension as composition factors.

3 Isomorphism for Hypergraphs

In the following we provide a very high-level sketch of the algorithm solving the Hypergraph Isomorphism Problem for $\widehat{\Gamma}_d$-groups. For all details I refer to the full version of this paper [30].

3.1 The Generalized String Isomorphism Problem

For the purpose of building a recursive algorithm, we consider a slightly different problem that crucially allows us to modify instances in a certain way exploited later on.

Let $\Gamma \leq \mathrm{Sym}(\Omega)$ be a group and let $\mathfrak{P}$ be a partition of the set Ω. The partition $\mathfrak{P}$ is Γ-*invariant* if $\mathfrak{P}^\gamma = \mathfrak{P}$ for all $\gamma \in \Gamma$ where $\mathfrak{P}^\gamma := \{P^\gamma \mid P \in \mathfrak{P}\}$. A $\mathfrak{P}$-*string* is a pair $(P, \mathfrak{x})$ where $P \in \mathfrak{P}$ and $\mathfrak{x}\colon P \to \Sigma$ is a string over a finite alphabet Σ. For $\sigma \in \mathrm{Sym}(\Omega)$ the string $\mathfrak{x}^\sigma$ is defined by $\mathfrak{x}^\sigma\colon P^\sigma \to \Sigma\colon \alpha \mapsto \mathfrak{x}(\alpha^{\sigma^{-1}})$. A permutation $\sigma \in \mathrm{Sym}(\Omega)$ is a Γ-*isomorphism* from $(P, \mathfrak{x})$ to a second $\mathfrak{P}$-string $(Q, \mathfrak{y})$ if $\sigma \in \Gamma$ and $(P^\sigma, \mathfrak{x}^\sigma) = (Q, \mathfrak{y})$.

▶ **Definition 6.** *The* Generalized String Isomorphism Problem *takes as input a permutation group* $\Gamma \leq \mathrm{Sym}(\Omega)$*, a* Γ*-invariant partition* $\mathfrak{P}$ *of the set* Ω*, and* $\mathfrak{P}$*-strings* $(P_1, \mathfrak{x}_1), \dots, (P_m, \mathfrak{x}_m)$ *and* $(Q_1, \mathfrak{y}_1), \dots, (Q_m, \mathfrak{y}_m)$*, and asks whether there is some* $\gamma \in \Gamma$ *such that*

$$\{(P_1^\gamma, \mathfrak{x}_1^\gamma), \dots, (P_m^\gamma, \mathfrak{x}_m^\gamma)\} = \{(Q_1, \mathfrak{y}_1), \dots, (Q_m, \mathfrak{y}_m)\}.$$

We usually denote $\mathfrak{X} = \{(P_1, \mathfrak{x}_1), \dots, (P_m, \mathfrak{x}_m)\}$ and $\mathfrak{Y} = \{(Q_1, \mathfrak{y}_1), \dots, (Q_m, \mathfrak{y}_m)\}$. Also, $\mathrm{Iso}_\Gamma(\mathfrak{X}, \mathfrak{Y})$ denotes the set of Γ-isomorphisms from $\mathfrak{X}$ to $\mathfrak{Y}$ and $\mathrm{Aut}_\Gamma(\mathfrak{X}) := \mathrm{Iso}_\Gamma(\mathfrak{X}, \mathfrak{X})$.

It is easy to see that the Hypergraph Isomorphism Problem can be reduced to the Generalized String Isomorphism Problem choosing $\mathfrak{P}$ to be the trivial partition consisting of one block and adding strings $\mathfrak{x}_E\colon V \to \{0,1\}$ for each hyperedge E where $\mathfrak{x}_E(v) = 1$ if and only if $v \in E$.

For the rest of this section we denote by $n := |\Omega|$ the size of the domain, and m denotes the size of $\mathfrak{X}$ and $\mathfrak{Y}$ (we always assume $|\mathfrak{X}| = |\mathfrak{Y}|$, otherwise the problem is trivial). The goal of this section is to provide an algorithm solving the Generalized String Isomorphism Problem for $\widehat{\Gamma}_d$-groups in time $(n+m)^{\mathcal{O}((\log d)^c)}$ for some absolute constant c.

We start by introducing some additional notation. For every $P \in \mathfrak{P}$ define $\mathfrak{X}[[P]] := \{\mathfrak{x}\colon P \to \Sigma \mid (P, \mathfrak{x}) \in \mathfrak{X}\}$ and also let $m_{\mathfrak{X}}(P) := |\mathfrak{X}[[\mathfrak{P}]]|$. We say that $\mathfrak{X}$ is *completely occupied* if $m_{\mathfrak{X}}(P) \geq 1$ for every $P \in \mathfrak{P}$. Also, we say that $\mathfrak{X}$ is *simple* if $m_{\mathfrak{X}}(P) \leq 1$ for every $P \in \mathfrak{P}$.

We will assume throughout this work that all sets of $\mathfrak{P}$-strings encountered are completely occupied, also if not explicitly stated. Note that an instance, which is not completely occupied, can be easily turned into one, that is completely occupied, by introducing additional $\mathfrak{P}$-strings. Also, for a set $A \subseteq \Omega$ and a set of $\mathfrak{P}$-strings $\mathfrak{X}$ define

$$\mathfrak{X}[A] := \{(P \cap A, \mathfrak{x}[A \cap P]) \mid (P, \mathfrak{x}) \in \mathfrak{X}, P \cap A \neq \emptyset\}$$

to be the subinstance induced by A where $\mathfrak{x}[A \cap P]$ denotes the substring induced by $A \cap P$.

3.2 Normalization of the Group Action

The main hurdle to build the algorithm for the Generalized String Isomorphism Problem is an extension of the Local Certificates Routine introduced by Babai for his quasipolynomial time isomorphism test [1]. Similar to [14], our algorithm exploits a variant of the Unaffected Stabilizers Theorem for $\widehat{\Gamma}_d$-group which builds the theoretical foundation for the Local Certificates algorithm.

However, this variant of the Unaffected Stabilizers Theorem for $\widehat{\Gamma}_d$-groups additionally requires the input group to have an *almost d-ary sequence of partitions.*

For a partition $\mathfrak{B}$ of the set Ω and $S \subseteq \Omega$ let $\mathfrak{B}[S] := \{B \cap S \mid B \in \mathfrak{B}, B \cap S \neq \emptyset\}$ denote the induced subpartition of $\mathfrak{B}$ on the set S. Let $\mathfrak{B}'$ be a second partition of Ω. We say that $\mathfrak{B}'$ *refines* $\mathfrak{B}$, denoted $\mathfrak{B}' \preceq \mathfrak{B}$, if for every $B' \in \mathfrak{B}'$ there is some $B \in \mathfrak{B}$ such that $B' \subseteq B$. If additionally $\mathfrak{B}' \neq \mathfrak{B}$ the partition $\mathfrak{B}'$ *strictly refines* $\mathfrak{B}$ which is denoted $\mathfrak{B}' \prec \mathfrak{B}$. Also, for a group $\Gamma \leq \mathrm{Sym}(\Omega)$ such that $\mathfrak{B}$ and $\mathfrak{B}'$ are Γ-invariant, let $\Gamma_B[\mathfrak{B}'[B]] \leq \mathrm{Sym}(\mathfrak{B}'[B])$ denote the induced natural action of Γ_B on $\mathfrak{B}'[B]$ for every $B \in \mathfrak{B}$. Finally, recall that a group $\Gamma \leq \mathrm{Sym}(\Omega)$ is semi-regular if the only element with fixed points is the identity.

▶ **Definition 7** (Almost d-ary Sequences of Partitions). *Let $\Gamma \leq \mathrm{Sym}(\Omega)$ be a group and let $\{\Omega\} = \mathfrak{B}_0 \succ \cdots \succ \mathfrak{B}_k = \{\{\alpha\} \mid \alpha \in \Omega\}$ be a sequence of Γ-invariant partitions. The sequence $\mathfrak{B}_0 \succ \cdots \succ \mathfrak{B}_k$ is* almost d-ary *if for every $i \in [k]$ and $B \in \mathfrak{B}_{i-1}$ it holds that*

1. $|\mathfrak{B}_i[B]| \leq d$, *or*
2. $\Gamma_B[\mathfrak{B}_i[B]]$ *is semi-regular.*

Since not every $\widehat{\Gamma}_d$-group $\Gamma \leq \mathrm{Sym}(\Omega)$ has an almost d-ary sequence of partitions, the first step of the algorithm is to normalize the input group. By a simple adaption of the arguments from [14] (see also [29]) we can obtain the following theorem normalizing the input to the Generalized String Isomorphism Problem.

▶ **Theorem 8.** *Let $(\Gamma, \mathfrak{P}, \mathfrak{X}, \mathfrak{Y})$ be an instance of the Generalized String Isomorphism Problem where $\Gamma \leq \mathrm{Sym}(\Omega)$ is a $\widehat{\Gamma}_d$-group.*

Then there is a set Ω^*, a monomorphism $\varphi\colon \Gamma \to \mathrm{Sym}(\Omega^*)$, an almost d-ary sequence of partitions $\mathfrak{B}_0^* \succ \mathfrak{B}_1^* \succ \cdots \succ \mathfrak{B}_\ell^*$ for Γ^φ, and an instance $(\Gamma^\varphi, \mathfrak{P}^*, \mathfrak{X}^*, \mathfrak{Y}^*)$ of the Generalized String Isomorphism Problem such that the following properties are satisfied:

1. $|\Omega^*| \leq n^{\mathrm{f}_{norm}(d)+1}$ *where* $\mathrm{f}_{norm}(d) = \mathcal{O}(\log d)$,
2. *there is some $i \in [k]$ such that $\mathfrak{P}^* = \mathfrak{B}_i^*$, and*
3. $\gamma \in \mathrm{Iso}_\Gamma(\mathfrak{X}, \mathfrak{Y})$ *if and only if* $\varphi(\gamma) \in \mathrm{Iso}_{\Gamma^\varphi}(\mathfrak{X}^*, \mathfrak{Y}^*)$ *for every* $\gamma \in \Gamma$.

Moreover, given $\Gamma \leq \mathrm{Sym}(\Omega)$, there is an algorithm computing the desired objects in time polynomial in the input size and the size of Ω^*.

Hence, in the following we restrict ourselves to the case where the input group has an almost d-ary sequence of partitions. In particular, this allows us to use the variant of the Unaffected Stabilizers Theorem proved in [14].

3.3 Creating Global Automorphisms from Local Information

The variant of the Unaffected Stabilizers Theorem from [14] provides us with the group-theoretic foundation to extend Babai's Local Certificates Routine [1] to the setting of hypergraphs. However, there is a second problem in generalizing the Local Certificates Routine that needs to be addressed.

Let $\Gamma \leq \mathrm{Sym}(\Omega)$ be a $\widehat{\Gamma}_d$-group, $\mathfrak{P}$ a Γ-invariant partition of Ω, and $\mathfrak{X}, \mathfrak{Y}$ two sets of $\mathfrak{P}$-strings. In a nutshell, the Local Certificates Routine considers a Γ-invariant window $W \subseteq \Omega$ such that $\Gamma[W] \leq \mathrm{Aut}(\mathfrak{X}[W])$ (i.e., the group Γ respects $\mathfrak{X}$ restricted to the window W) and aims at creating automorphisms of the entire structure $\mathfrak{X}$ (from the local information that Γ respects $\mathfrak{X}$ on the window W). In order to create these automorphisms the Local Certificates Routine considers the group $\Gamma_{(\Omega\setminus W)}$ fixing every point outside of W. Intuitively speaking, the Unaffected Stabilizers Theorem (resp. the variant suitable for $\widehat{\Gamma}_d$-groups) guarantees that the group $\Gamma_{(\Omega\setminus W)}$ is large. For the String Isomorphism Problem it is easy to see $\Gamma_{(\Omega\setminus W)}$ consists only of automorphisms of the input string since there are no dependencies between the positions within the window W and outside of W. However, for the Generalized String Isomorphism Problem, this is not true anymore. In other words, in order to compute $\mathrm{Aut}(\mathfrak{X})$, it is not possible to consider $\mathfrak{X}[W]$ and $\mathfrak{X}[\Omega \setminus W]$ independently.

Our solution to this problem is guided by the following simple observation. Suppose that $\mathfrak{X}[W]$ is simple, i.e., $m_{\mathfrak{X}[W]}(P) = 1$ for all $P \in \mathfrak{P}[W]$. In this case it is possible to consider $\mathfrak{X}[W]$ and $\mathfrak{X}[\Omega \setminus W]$ independently as the next lemma indicates.

▶ **Lemma 9.** *Let $\Gamma \leq \mathrm{Sym}(\Omega)$ be a permutation group, let $\mathfrak{P}$ be a Γ-invariant partition of Ω and $\mathfrak{X}$ a set of $\mathfrak{P}$-strings. Also suppose $W \subseteq \Omega$ is a Γ-invariant window such that $\mathfrak{X}[W]$ is simple and $\Gamma[W] \leq \mathrm{Aut}(\mathfrak{X}[W])$. Then $\Gamma_{(\Omega\setminus W)} \leq \mathrm{Aut}(\mathfrak{X})$.*

Proof. Let $\gamma \in \Gamma_{(\Omega\setminus W)}$ and $(P, \mathfrak{x}) \in \mathfrak{X}$. It suffices to show that $(P^\gamma, \mathfrak{x}^\gamma) \in \mathfrak{X}$. First suppose $P \subseteq W$. Then $(P, \mathfrak{x}) \in \mathfrak{X}[W]$ and $(P^\gamma, \mathfrak{x}^\gamma) \in \mathfrak{X}[W]$. Moreover, $P^\gamma \subseteq W$ since W is Γ-invariant. Hence, $(P^\gamma, \mathfrak{x}^\gamma) \in \mathfrak{X}$.

Otherwise $P \cap (\Omega \setminus W) \neq \emptyset$. Since $\alpha^\gamma = \alpha$ for all $\alpha \in \Omega \setminus W$ it follows that $P^\gamma \cap P \neq \emptyset$. Using the fact that $\mathfrak{P}$ is Γ-invariant this implies that $P^\gamma = P$. To complete the proof it is argued that $\mathfrak{x}^\gamma = \mathfrak{x}$. Let $\alpha \in P$. If $\alpha \notin W$ then $\mathfrak{x}^\gamma(\alpha) = \mathfrak{x}^\gamma(\alpha^\gamma) = \mathfrak{x}(\alpha^{\gamma\gamma^{-1}}) = \mathfrak{x}(\alpha)$. So assume $\alpha \in W$. Since $\gamma[W] \in \mathrm{Aut}(\mathfrak{X}[W])$ it holds that $((P \cap W)^\gamma, (\mathfrak{x}[P \cap W])^\gamma) = (P \cap W, (\mathfrak{x}[P \cap W])^\gamma) \in \mathfrak{X}[W]$. But this means $(\mathfrak{x}[P \cap W])^\gamma = \mathfrak{x}[P \cap W]$ since $\mathfrak{X}[W]$ is simple. Hence $\mathfrak{x}^\gamma(\alpha) = \mathfrak{x}(\alpha)$. ◀

Let $W \subseteq \Omega$ be a Γ-invariant set such that $\mathfrak{X}[W] \leq \mathrm{Aut}(\mathfrak{X}[W])$ (this is the case during the Local Certificates Routine). In order to solve the problem described above in general, the basic idea is to modify the instance in such a way that $\mathfrak{X}[W]$ becomes simple. This allows us to apply Lemma 9 and eventually, to extend the Local Certificates Routine to our setting. This modification is one of the key conceptual contributions of this work.

Consider a set $P \in \mathfrak{P}$. In order to "simplify" the instance we define an equivalence relation on the set $\mathfrak{X}[[P]]$ of all strings contained in P. Two $\mathfrak{P}$-strings $(P, \mathfrak{x}_1)$ and $(P, \mathfrak{x}_2)$ are W-equivalent if they are identical on the window W, i.e., $\mathfrak{x}_1[W] = \mathfrak{x}_2[W]$. For each equivalence class we create a new block P' containing exactly the strings from the equivalence class. Since the group Γ respects the induced subinstance $\mathfrak{X}[W]$ it naturally acts on the equivalence classes. This process is visualized in Figure 1 and formalized below.

Consider the natural homomorphism $\psi\colon \Gamma \to \mathrm{Sym}(\mathfrak{X}[W])$. Now let $\Omega' := \bigcup_{P\in\mathfrak{P}} P \times \{\mathfrak{x}[W \cap P] \mid (P, \mathfrak{x}) \in \mathfrak{X}\}$ and $\mathfrak{P}' := \{P \times \{\mathfrak{x}[W \cap P]\} \mid (P, \mathfrak{x}) \in \mathfrak{X}\}$. Also define

$$\mathfrak{X}' := \left\{ \Big(P \times \{\mathfrak{x}[W \cap P]\}, \mathfrak{x}'\colon P \times \{\mathfrak{x}[W \cap P]\} \to \Sigma\colon (\alpha, \mathfrak{x}[W \cap P]) \mapsto \mathfrak{x}(\alpha)\Big) \;\middle|\; (P, \mathfrak{x}) \in \mathfrak{X} \right\}$$

and similarly define $\mathfrak{Y}'$ for the instance $\mathfrak{Y}$. Note that $\mathfrak{X}'$ and $\mathfrak{Y}'$ are sets of $\mathfrak{P}'$-strings. Finally, the group Γ faithfully acts on the set Ω' via $(\alpha, \mathfrak{z})^\gamma = (\alpha^\gamma, \mathfrak{z}^\gamma)$ yielding an injective homomorphism $\psi\colon \Gamma \to \mathrm{Sym}(\Omega')$. Define $\Gamma' := \Gamma^\psi$. It can be easily checked that the updated instance is equivalent to the original instance. Also, $\mathfrak{X}'[W']$ is simple where $W' := \{(\alpha, \mathfrak{z}) \in \Omega' \mid \alpha \in W\}$.

While this simplification allows us to treat $\mathfrak{X}'[W']$ and $\mathfrak{X}'[W' \setminus \Omega']$ independently and thus solves the above problem, it creates several additional issues that need to be addressed. First, this modification may destroy the normalization property (i.e., the existence of an

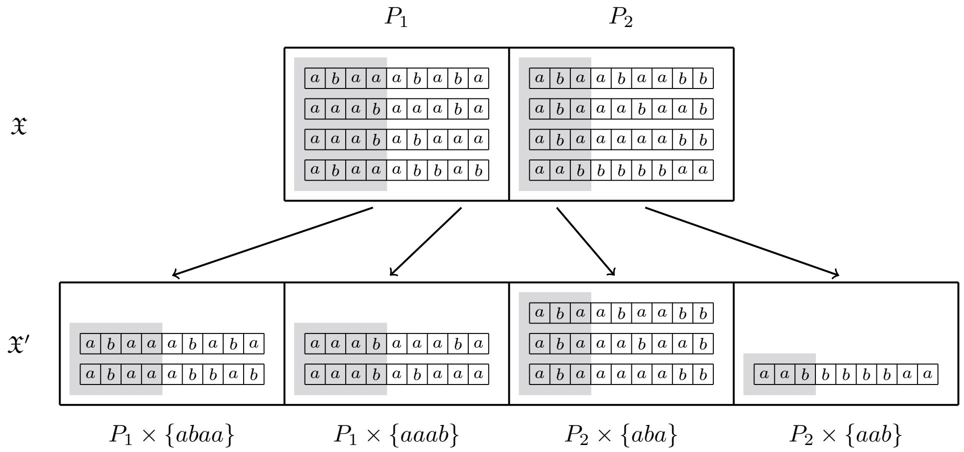


Figure 1 A set $\mathfrak{X}$ of $\mathfrak{P}$-strings is given in the top and the "simplified" instance $\mathfrak{X}'$ is given below. The window W is marked in gray. Note that $\mathfrak{X}'[W']$ is simple where W' denotes the window marked in gray in the bottom part of the figure.

almost d-ary sequence of partitions). As a result, the Local Certificates Routine constantly needs to renormalize the input instances which requires a precise analysis of the increase in size occurring from the renormalization (see Theorem 8). In order to be able to still analyze the progress made by the recursion, we introduce the notion of a *virtual size* of an instance.

▶ **Definition 10.** *Let $\mathfrak{P}$ be a partition of Ω and suppose $\mathfrak{X}$ is a set of $\mathfrak{P}$-strings. The* d-virtual size *of $\mathfrak{X}$ is defined by $s := \sum_{P \in \mathfrak{P}} |P| \cdot (m_{\mathfrak{X}}(P))^{\mathsf{f}_{\mathsf{norm}}(d)+1}$.*

Here, $\mathsf{f}_{\mathsf{norm}}(d) = \mathcal{O}(\log d)$ refers to the normalization cost defined in Theorem 8. Hence, the d-virtual size s of $\mathfrak{X}$ is bounded by $(m+n)^{\mathcal{O}(\log d)}$ and it suffices to analyze the running time of all subroutines in terms of the virtual size of the input instances. Intuitively speaking, the idea of the virtual size is to take into account the cost of all potential renormalization steps which means that, when the algorithm renormalizes an instance, while its actual size might grow significantly, its virtual size does not. This allows us to measure the progress made by the recursive algorithm although instances might grow significantly.

The renormalization of instances also creates another problem. In the *aggregation of local certificates*, the outputs of the Local Certificates Routine are compared with each other requiring the outputs to be isomorphism-invariant. However, the renormalization procedure is not isomorphism-invariant. Our solution to this problem is to run the Local Certificates Routine in parallel on all pairs of test sets compared later on. This way, we can ensure that all instances are normalized in the same way.

This simplification procedure is formalized by the next technical theorem which combines all the requirements outlined above, including the renormalization steps, into a single statement.

▶ **Theorem 11.** *Let $\Gamma \leq \mathrm{Sym}(\Omega)$ be a $\widehat{\Gamma}_d$-group and $\mathfrak{P}$ be a Γ-invariant partition of Ω. Also suppose $\{\Omega\} = \mathfrak{B}_0 \succ \mathfrak{B}_1 \succ \dots \succ \mathfrak{B}_\ell = \{\{\alpha\} \mid \alpha \in \Omega\}$ forms an almost d-ary sequence of Γ-invariant partitions such that $\mathfrak{P} = \mathfrak{B}_i$ for some $i \in [\ell]$. Let $(\mathfrak{X}_i)_{i \in [p]}$ be a list of sets of $\mathfrak{P}$-strings. Also let $W \subseteq \Omega$ be a Γ-invariant set such that $\Gamma[W] \leq \mathrm{Aut}(\mathfrak{X}_i[W])$ for all $i \in [p]$. Moreover, assume that $\mathfrak{X}_i[W] = \mathfrak{X}_j[W]$ for all $i, j \in [p]$.*

Then there is an equivalence relation $\sim$ on the set $[p]$ such that $\mathfrak{X}_i \cong_\Gamma \mathfrak{X}_j$ implies $i \sim j$ for all $i, j \in [p]$, and for each equivalence class $A \subseteq [p]$, we get the following:

A set Ω^*, a group $\Gamma^* \leq \mathrm{Sym}(\Omega)$, elements $\lambda_i \in \Gamma$ for all $i \in A$, a monomorphism $\varphi\colon \Gamma^* \to \Gamma$, a window $W^* \subseteq \Omega^*$, a sequence of partitions $\{\Omega^*\} = \mathfrak{B}_0^* \succ \mathfrak{B}_1^* \succ \dots \succ \mathfrak{B}_k^* = \{\{\alpha\} \mid \alpha \in \Omega^*\}$, and a list of $\mathfrak{P}^*$-strings $(\mathfrak{X}_i^*)_{i\in A}$, such that the following properties are satisfied:

(A) *the sequence $\mathfrak{B}_0^* \succ \dots \succ \mathfrak{B}_k^*$ forms an almost d-ary sequence of Γ^*-invariant partitions,*

(B) *W^* is Γ^*-invariant and $\Gamma^*[W^*] \leq \mathrm{Aut}(\mathfrak{X}_i^*[W^*])$ for all $i \in A$,*

(C) *there is some $i \in [k]$ such that $\mathfrak{P}^* = \mathfrak{B}_i^*$,*

(D) $\mathrm{Iso}_\Gamma(\mathfrak{X}_i, \mathfrak{X}_j) = \lambda_i^{-1}(\mathrm{Iso}_{\Gamma^*}(\mathfrak{X}_i^*, \mathfrak{X}_j^*))^\varphi \lambda_j$ *for all $i, j \in A$,*

(E) *$\mathfrak{X}_i^*[W^*]$ is simple for all $i \in A$,*

(F) *for s the virtual size of $\mathfrak{X}_i$ and s^* the virtual size of $\mathfrak{X}_i^*$ it holds that $s^* \leq s$, and*

(G) *for s the virtual size of $\mathfrak{X}_i[\Omega \setminus W]$ and s^* the virtual size of $\mathfrak{X}_i^*[\Omega^* \setminus W^*]$ it holds that $s^* \leq s$.*

Moreover, there is an algorithm computing the desired objects in time $p^2 \cdot (n+m)^{\mathcal{O}((\log d)^c)}$ for some absolute constant c.

3.4 The Local Certificates Routine

In this subsection the Local Certificates Routine originally introduced in [1] is lifted to the Generalized String Isomorphism Problem for $\widehat{\Gamma}_d$-group which builds the crucial step of extending the group-theoretic techniques of Babai's quasipolynomial time isomorphism test to the setting of this paper.

Let $\Gamma \leq \mathrm{Sym}(\Omega)$ be a permutation group and let $(\Gamma, \mathfrak{P}, \mathfrak{X}, \mathfrak{Y})$ be an instance of the Generalized String Isomorphism Problem. Furthermore let $\varphi\colon \Gamma \to S_k$ be a *giant representation*, i.e., a homomorphism $\varphi\colon \Gamma \to S_k$ such that $\Gamma^\varphi \geq A_k$. For the description of the Local Certificates Routine we extend the notation of set- and point-wise stabilizers for the group Γ to the action on the set $[k]$ defined via the giant representation φ. For a set $T \subseteq [k]$ let $\Gamma_T := \varphi^{-1}((\Gamma^\varphi)_T)$ and $\Gamma_{(T)} := \varphi^{-1}((\Gamma^\varphi)_{(T)})$.

The basic approach of the Local Certificates Routine is to consider *test sets* $T \subseteq [k]$ of logarithmic size.

▶ **Definition 12.** *A test set $T \subseteq [k]$ is* full *if $(\mathrm{Aut}_{\Gamma_T}(\mathfrak{X}))^\varphi[T] \geq \mathrm{Alt}(T)$. A* certificate of fullness *is a subgroup $\Delta \leq \mathrm{Aut}_{\Gamma_T}(\mathfrak{X})$ such that $\Delta^\varphi[T] \geq \mathrm{Alt}(T)$. A* certificate of non-fullness *is a non-giant $\Lambda \leq \mathrm{Sym}(T)$ such that $(\mathrm{Aut}_{\Gamma_T}(\mathfrak{X}))^\varphi[T] \leq \Lambda$.*

The central part of the algorithm is to determine for each test set $T \subseteq [k]$ (of size t approximately logarithmic in d) whether T is full and, depending on the outcome, compute a certificate of fullness or a certificate of non-fullness. Actually, in order to decide isomorphism, non-fullness certificates are also required for pairs of test sets. All of this is achieved by the following theorem.

▶ **Theorem 13.** *Let $\mathfrak{P}$ be a partition of Ω, $\mathfrak{X}$ and $\mathfrak{Y}$ two sets of $\mathfrak{P}$-strings. Let s be the d-virtual size of $\mathfrak{X}$ and $\mathfrak{Y}$. Also let $\Gamma \leq \mathrm{Sym}(\Omega)$ a $\widehat{\Gamma}_d$-group that has an almost d-ary sequence of partitions $\mathfrak{B}_0 \succ \dots \succ \mathfrak{B}_\ell$ such that $\mathfrak{P} = \mathfrak{B}_i$ for some $i \in [\ell]$. Furthermore suppose there is a giant representation $\varphi\colon \Gamma \to S_k$ and let $k \geq t > \max\{8, 2 + \log_2 d\}$. Finally, define $\mathcal{T} = \{(\mathfrak{X}, T), (\mathfrak{Y}, T) \mid T \subseteq [k], |T| = t\}$. For every $(\mathfrak{Z}_1, T_1), (\mathfrak{Z}_2, T_2) \in \mathcal{T}$ one can compute*

(i) *if T_1 is full with respect to $\mathfrak{Z}_1$, a group $\Delta := \Delta(\mathfrak{Z}_1, T_1) \leq \mathrm{Aut}_{\Gamma_{T_1}}(\mathfrak{Z}_1)$ such that $\Delta^\varphi[T_1] \geq \mathrm{Alt}(T_1)$, or*

(ii) *if T_1 is not full with respect to $\mathfrak{Z}_1$, a non-giant group $\Lambda := \Lambda(T_1, \mathfrak{Z}_1, T_2, \mathfrak{Z}_2) \leq \mathrm{Sym}(T_1)$ and a bijection $\lambda := \lambda(T_1, \mathfrak{Z}_1, T_2, \mathfrak{Z}_2)\colon T_1 \to T_2$ such that*

$$\left\{\gamma^\varphi|_{T_1} \;\middle|\; \gamma \in \mathrm{Iso}_\Gamma(\mathfrak{Z}_1, \mathfrak{Z}_2) \wedge T_1^{(\gamma^\varphi)} = T_2\right\} \subseteq \Lambda\lambda.$$

Moreover, the output is isomorphism-invariant, i.e., for every two pairs $(\mathfrak{Z}_1, T_1), (\mathfrak{Z}_2, T_2) \in \mathcal{T}$ *and* $(\mathfrak{Z}'_1, T'_1), (\mathfrak{Z}'_2, T'_2) \in \mathcal{T}$ *and isomorphisms* $\gamma_i \in \mathrm{Iso}_\Gamma((\mathfrak{Z}_i, T_i), (\mathfrak{Z}'_i, T'_i))$, $i \in \{1,2\}$, *it holds that*

(a) *if* T_1 *is full with respect to* $\mathfrak{Z}_1$, *then* $(\Delta(T_1, \mathfrak{Z}_1))^{\gamma_1} = \Delta(T'_1, \mathfrak{Z}'_1)$, *and*

(b) *if* T_1 *is not full with respect to* $\mathfrak{Z}_1$, *then*

$$\varphi(\gamma_1^{-1})\Lambda(T_1, \mathfrak{Z}_1, T_2, \mathfrak{Z}_2)\lambda(T_1, \mathfrak{Z}_1, T_2, \mathfrak{Z}_2)\varphi(\gamma_2) = \Lambda(T'_1, \mathfrak{Z}'_1, T'_2, \mathfrak{Z}'_2)\lambda(T'_1, \mathfrak{Z}'_1, T'_2, \mathfrak{Z}'_2).$$

Moreover, there are numbers $s_1, \dots, s_r \leq s/2$ *such that* $\sum_{i=1}^{r} s_i \leq 4k^{2t} \cdot t! \cdot s$ *and, for each* $i \in [r]$ *using a recursive call to the Generalized String Isomorphism Problem for instances of* d*-virtual size at most* s_i, *and* $k^{\mathcal{O}(t)} \cdot t! \cdot (n+m)^{\mathcal{O}((\log d)^c)}$ *additional computation steps, an algorithm can compute all desired objects.*

The algorithm is similar to the standard Local Certificates Routine implement in Babai's quasipolynomial-time isomorphism test [1]. A main difference is that, in each iteration, the algorithm runs the subroutine implemented in Theorem 11 to "simplify" all instances as described in the previous subsection. This guarantees that, throughout the execution of the algorithm, the prerequisites of Lemma 9 are satisfied allowing for the construction of global automorphisms from local information.

3.5 An Algorithm for the Generalized String Isomorphism Problem

With the Local Certificates Routine for sets of $\mathfrak{P}$-strings presented above it is possible to provide an algorithm for the Hypergraph Isomorphism Problem assuming the input group is equipped with an almost d-ary sequence of partitions. The algorithm mostly follows the same patterns as in [14] giving the corresponding result for the String Isomorphism Problem by replacing the Local Certificates Routine.

▶ **Theorem 14.** *There is an algorithm that, given a partition* $\mathfrak{P}$ *of* Ω, *a* $\widehat{\Gamma}_d$*-group* $\Gamma \leq \mathrm{Sym}(\Omega)$, *two sets of* $\mathfrak{P}$*-strings* $\mathfrak{X}, \mathfrak{Y}$ *and an almost* d*-ary sequence of partitions* $\mathfrak{B}_0 \succ \dots \succ \mathfrak{B}_\ell$ *for* Γ *such that* $\mathfrak{P} = \mathfrak{B}_i$ *for some* $i \in [\ell]$, *computes a representation for* $\mathrm{Iso}_\Gamma(\mathfrak{X}, \mathfrak{Y})$ *in time* $(n+m)^{\mathcal{O}((\log d)^c)}$, *for an absolute constant* c.

Combining Theorem 8 and 14 gives the main technical result of this work.

▶ **Theorem 15.** *The Generalized String Isomorphism Problem for* $\widehat{\Gamma}_d$*-groups can be solved in time* $(n+m)^{\mathcal{O}((\log d)^c)}$ *for some constant* c.

As an immediate consequence we obtain one of the main results of this paper.

▶ **Corollary 16** (Theorem 1 restated)**.** *The Hypergraph Isomorphism Problem for* $\widehat{\Gamma}_d$*-groups can be solved in time* $(n+m)^{\mathcal{O}((\log d)^c)}$ *for some constant* c.

4 Color Refinement and Small Color Classes

In the following two sections we present applications of the improvement obtained for the Hypergraph Isomorphism Problem for $\widehat{\Gamma}_d$-groups. Towards this end, we first introduce the notion of t-CR-bounded graphs and present an isomorphism test for such graphs running in time $n^{\mathcal{O}((\log t)^c)}$. The notion of t-CR-bounded graphs generalizes color-t-bounded graphs considered in [4] in the context of the isomorphism problem for strongly regular graphs. But more importantly, the isomorphism problem for t-CR-bounded graphs can serve as an intermediate problem for designing faster isomorphism tests for important classes of graphs.

For example, there is a simple polynomial-time Turing reduction from the isomorphism problem for graphs of maximum degree d to the isomorphism problem for d-CR-bounded graphs. Hence, our results may be seen as a generalization to the isomorphism test for graphs of small degree presented in [14]. Moreover, in the next section, we present a polynomial-time Turing reduction from the isomorphism problem for graphs of genus at most g to the isomorphism problem for $(4g+2)$-CR-bounded graphs. As a result, isomorphism for graphs of genus at most g can be tested in time $n^{\mathcal{O}((\log g)^c)}$.

The definition of t-CR-bounded graphs builds on the Color Refinement algorithm, a simple combinatorial algorithm that iteratively refines a vertex-coloring in an isomorphism-invariant manner and which forms a fundamental algorithmic tool in the context of the Graph Isomorphism Problem (see, e.g., [4, 7, 8, 24, 25, 36]). We start by formally defining the outcome of the Color Refinement algorithm in the next subsection.

4.1 The Color Refinement Algorithm

Let G be a graph with vertex coloring $\chi_V\colon V(G) \to C_V$ and arc coloring $\chi_E\colon \{(v,w) \mid vw \in E(G)\} \to C_E$. The *Color Refinement algorithm* is a procedure that, given a vertex- and arc-colored graph G, iteratively computes an isomorphism-invariant refinement $\chi_{\mathsf{CR}}[G]$ of the vertex-coloring χ_V.

Let $\chi_1, \chi_2\colon V \to C$ be colorings of vertices where C is some finite set of colors. The coloring χ_1 *refines* χ_2, denoted $\chi_1 \preceq \chi_2$, if $\chi_1(v) = \chi_1(w)$ implies $\chi_2(v) = \chi_2(w)$ for all $v,w \in V$. The colorings χ_1 and χ_2 are *equivalent*, denoted $\chi_1 \equiv \chi_2$, if $\chi_1 \preceq \chi_2$ and $\chi_2 \preceq \chi_1$.

Given a vertex- and arc-colored graph G, the Color Refinement algorithm computes an isomorphism-invariant coloring $\chi_{\mathsf{CR}}[G]$ as follows. The initial coloring for the algorithm is defined as $\chi_{(0)}[G] \coloneqq \chi_V$, the vertex-coloring of the input graph. The initial coloring is refined by iteratively computing colorings $\chi_{(i)}[G]$ for $i > 0$. For $i > 0$ define $\chi_{(i)}[G](v) \coloneqq (\chi_{(i-1)}[G](v), \mathcal{M}_i(v))$ where

$$\mathcal{M}_i(v) \coloneqq \{\{\big(\chi_{(i-1)}[G](w), \chi_E(v,w), \chi_E(w,v)\big) \mid w \in N_G(v)\}\}.$$

From the definition of the colorings it is immediately clear that $\chi_{(i+1)}[G] \preceq \chi_{(i)}[G]$. Now let $i \in \mathbb{N}$ be the minimal number such that $\chi_{(i)}[G] \equiv \chi_{(i+1)}[G]$. For this i, the coloring $\chi_{(i)}[G]$ is called the *stable* coloring of G and is denoted by $\chi_{\mathsf{CR}}[G]$.

The *Color Refinement algorithm* takes as input a (vertex- and arc-colored) graph G and computes (a coloring that is equivalent to) $\chi_{\mathsf{CR}}[G]$. I remark that this can be implemented in time almost linear in the number of vertices and edges (see, e.g., [9]).

4.2 Splitting Small Color Classes

Having defined the Color Refinement algorithm, we can now define the notion of t-CR-bounded graphs. The basic idea behind t-CR-bounded graphs is the following. Suppose (G,χ) is a vertex-colored graph. Then (G,χ) is t-CR-bounded if it is possible to transform χ into a discrete coloring (i.e., a coloring where each vertex has its own color) by repeatedly performing the following two operations: applying the Color Refinement algorithm and completely splitting color classes of size at most t (i.e., assigning every vertex in such a color class its own color).

For formal reasons, we actually define t-CR-bounded pairs where an additional set $S \subseteq V(G)$ is provided each vertex of which may also be individualized. The intuition behind this is that we may already have good knowledge about the structure of the automorphism group of (G,χ) on the set S which can be exploited for isomorphism testing.

▶ **Definition 17.** *A pair (G, S), where $G = (V, E, \chi_V, \chi_E)$ is a vertex- and arc-colored graph and $S = \chi_V^{-1}(c)$ for some color c (not necessarily in the image of χ_V, i.e., S may be the empty set), is* t-CR-bounded *if the sequence $(\chi_i)_{i \geq 0}$ reaches a discrete coloring where*

$$\chi_0(v) := \begin{cases} (v, 1) & \text{if } v \in S \\ (\chi_V(v), 0) & \text{if } v \notin S \end{cases},$$

$\chi_{2i+1} := \chi_{CR}[V, E, \chi_{2i}, \chi_E]$ and

$$\chi_{2i+2}(v) := \begin{cases} (v, 1) & \text{if } |\chi_{2i+1}^{-1}(\chi_{2i+1}(v))| \leq t \\ (\chi_{2i+1}(v), 0) & \text{otherwise} \end{cases}$$

for all $i \geq 0$. A graph G is t-CR-bounded *if the pair $(G, \emptyset)$ is t-CR-bounded.*

I remark that the name "t-CR-bounded" is inspired by color-t-bounded graphs [4] defined in a similar manner where the letters *CR* refer to the Color Refinement algorithm. Also, I note that a similar notion of graphs has been exploited in [28] for designing a polynomial-time isomorphism test for unit square graphs.

▶ **Theorem 18.** *Let (G_1, S_1) and (G_2, S_2) be two t-CR-bounded pairs and also let $\Gamma \leq \mathrm{Sym}(S_1)$ be a $\widehat{\Gamma}_t$-group and $\theta \colon S_1 \to S_2$ a bijection. Then a representation for the set*

$$\mathrm{Iso}_{\Gamma\theta}((G_1, S_1), (G_2, S_2)) := \{\sigma \colon G_1 \cong G_2 \mid \sigma|_{S_1} \in \Gamma\theta\}$$

can be computed in time $n^{\mathcal{O}((\log t)^c)}$ for some absolute constant c.

▶ **Corollary 19.** *The Graph Isomorphism Problem for t-CR-bounded graphs can be solved in time $n^{\mathcal{O}((\log t)^c)}$ for some absolute constant c.*

In particular, this includes color-t-bounded graphs considered in [4] in the context of the isomorphism problem for strongly regular graphs.

The algorithm underlying Theorem 18 actually describes a polynomial-time Turing reduction from the isomorphism problem for t-CR-bounded graphs to the Hypergraph Isomorphism Problem for $\widehat{\Gamma}_t$-groups. This result may be of independent interest.

▶ **Lemma 20.** *There is a polynomial-time Turing reduction from the Graph Isomorphism Problem for t-CR-bounded graphs to the Hypergraph Isomorphism Problem for $\widehat{\Gamma}_t$-groups.*

4.3 Hereditarily Finite Sets

As the first application to the isomorphism problem for t-CR-bounded pairs we consider hereditarily finite sets. Intuitively speaking, a hereditarily finite set over ground set V is any type of combinatorial object that can be build by iteratively taking sets and tuples over previously created objects.

In this section we extend the algorithm for solving the Hypergraph Isomorphism Problem for $\widehat{\Gamma}_d$-groups to work on any type of hereditarily finite set.

Let V be finite set of elements (e.g., the vertex of a graph). The class of *hereditarily finite sets* over universe (ground set) V is inductively defined as follows. Each $v \in V$ is an *atom* which in particular is a hereditarily finite set. Also, for hereditarily finite sets $\mathfrak{A}_1, \dots, \mathfrak{A}_k$ (over universe V) the set $\{\mathfrak{A}_1, \dots, \mathfrak{A}_k\}$ as well as the tuple $(\mathfrak{A}_1, \dots, \mathfrak{A}_k)$ is a hereditarily finite set (over universe V). Examples of hereditarily finite sets include graphs, hypergraphs and relational structures.

In order to apply the isomorphism test for t-CR-bounded graphs, we can transform a hereditarily finite set $\mathfrak{A}$ over ground set V into a graph $G(\mathfrak{A})$ in the natural way so that the pair $(G(\mathfrak{A}), V)$ is 0-CR-bounded.

▶ **Corollary 21.** *Let $\mathfrak{A}_1$ and $\mathfrak{A}_2$ be two hereditarily finite sets over the universe V and let $\Gamma \leq \mathrm{Sym}(V)$ be a $\widehat{\Gamma}_d$-group. Then a representation for the set $\mathrm{Iso}_\Gamma(\mathfrak{A}_1, \mathfrak{A}_2) := \{\gamma \in \Gamma \mid \gamma\colon \mathfrak{A}_1 \cong \mathfrak{A}_2\}$ can be computed in time $(n+m)^{\mathcal{O}((\log d)^c)}$ for some absolute constant c where $n := |V|$ and $m := |V(G(\mathfrak{A}_1))|$.*

▶ **Corollary 22.** *The isomorphism problem for hereditarily finite sets can be solved in time $(n+m)^{\mathcal{O}((\log n)^c)}$.*

I remark that a variant of the previous corollary was independently obtained by Daniel Wiebking [37] (see *Related Work*).

5 Isomorphism for Graphs of Bounded Genus

Next, we present a second, slightly more involved application of our results and give an algorithm solving the isomorphism problem for graphs of Euler genus at most g in time $n^{\mathcal{O}((\log g)^c)}$ for some absolute constant c. Actually, we prove a more general result.

Recall that a graph H is a minor of another graph G if H can be obtained from G by removing vertices as well as removing and contracting edges. Also define $K_{m,n}$ to be the complete bipartite graph with m vertices on the left side and n vertices on the right side. Let $h \geq 3$ and define $\mathcal{C}_h$ to be the class of graphs that exclude $K_{3,h}$ as a minor.

We present a polynomial-time reduction from the isomorphism problem for the class $\mathcal{C}_h$ to the isomorphism problem for t-CR-bounded graphs where $t := h-1$. The following lemma is the key tool for the reduction. Intuitively, it investigates the structure of certain colorings that are stable with respect to the Color Refinement algorithm for graphs in the class $\mathcal{C}_h$. Recall that a graph G is 3-*connected* if there are no two vertices $v, w \in V(G)$ such that $G - \{v, w\}$ is disconnected.

▶ **Lemma 23.** *Let (G, χ) be a 3-connected, colored graph that excludes $K_{3,h}$ as a minor and suppose $V_1 \uplus V_2 = V(G)$ such that*
1. *each $v \in V_1$ forms a singleton color class with respect to χ,*
2. *χ is stable with respect to the Color Refinement algorithm,*
3. *$|V_1| \geq 3$, and*
4. *$N(V_2) = V_1$.*

Then there is a color class $U \subseteq V_2$ with respect to χ of size $|U| \leq h-1$.

Proof. Let $C := \mathrm{im}(\chi)$, $C_1 := \chi(V_1)$ and $C_2 := \chi(V_2)$. Also define H to be the graph with vertex set $V(H) := C$ and edge set

$$E(H) = \{c_1c_2 \mid \exists v_1 \in \chi^{-1}(c_1), v_2 \in \chi^{-1}(c_2)\colon v_1v_2 \in E(G)\}.$$

Let $C' \subseteq C_2$ be the vertex set of a connected component of $H[C_2]$. Then $|N_H(C')| \geq 3$ since each $v \in V_1$ forms a singleton color class with respect to χ and G is 3-connected.

Now let $c_1, c_2, c_3 \in N_H(C')$ be distinct and also let $v_i \in \chi^{-1}(c_i)$ for $i \in [3]$. Also let T be a spanning tree of $H[C' \cup \{c_1, c_2, c_3\}]$ such that $c_1, c_2, c_3 \in L(T)$ where $L(T)$ denotes the set of leaves of T. Moreover, let T' be the subtree of T obtained from repeatedly removing all leaves $c \in C'$. Hence, $L(T') = \{c_1, c_2, c_3\}$. Then there is a unique color c such that $\deg_{T'}(c) = 3$. Also, for $i \in [3]$, define C_i' to be the set of internal vertices on the unique path from c_i to c in the tree T'.

Since $|\chi^{-1}(c_i)| = 1$ and χ is stable with respect to the Color Refinement algorithm it holds that

$$G\left[\chi^{-1}(C_i' \cup \{c_i\})\right]$$

is connected. Let $U_i \coloneqq \chi^{-1}(C_i' \cup \{c_i\})$, $i \in [3]$. Also let $U = \chi^{-1}(c)$ and suppose that $|U| \geq h$. Then $N(U_i) \cap U \neq \emptyset$ by the definition of the tree T. Moreover, this implies $U \subseteq N(U_i)$ since χ is stable with respect to the Color Refinement algorithm. Hence, G contains a minor isomorphic to $K_{3,h}$. ◀

▶ **Corollary 24.** *Let $(G, \chi_V, \chi_E) \in \mathcal{C}_h$ be a 3-connected, vertex- and arc-colored graph and let $v_1, v_2, v_3 \in V(G)$. Also define $\chi_V^*(v_i) \coloneqq (i, 1)$ for $i \in [3]$ and $\chi_V^*(v) \coloneqq (\chi_V(v), 0)$ for all $v \in V(G) \setminus \{v_1, v_2, v_3\}$. Then (G, χ_V^*, χ_E) is $(h-1)$-CR-bounded.*

Proof. Let $(\chi_i)_{i\geq 0}$ be the sequence of colorings obtained from the definition of $(h-1)$-CR-bounded graphs (Definition 17) for the graph (G, χ_V^*, χ_E). Let $\chi^* \coloneqq \chi_i$ for the minimal $i \geq 0$ such that $\chi_i \equiv \chi_{i+1}$.

Suppose towards a contradiction that χ^* is not discrete (i.e., not every color class is a singleton). Let $V_2 \coloneqq \{v \in V(G) \mid |(\chi^*)^{-1}(\chi^*(v))| > 1\}$ and let $V_1 \coloneqq N_G(V_2)$. Then $|V_1| \geq 3$ since $|V(G) \setminus V_2| \geq 3$ and G is 3-connected. Also note that $\chi^*|_{V_1 \cup V_2}$ is a stable coloring for the graph $G[V_1 \cup V_2]$. Hence, by Lemma 23, there is some color c such that $1 < |(\chi^*)^{-1}(c)| \leq h - 1$. But this contradicts the definition of the coloring χ^* (cf. Definition 17). ◀

The last corollary gives some insights into the structure of the automorphism group of graphs $G \in \mathcal{C}_h$.

▶ **Theorem 25.** *Let $G \in \mathcal{C}_h$ be a 3-connected graph and let $v_1, v_2, v_3 \in V(G)$ be distinct vertices. Then $(\mathrm{Aut}(G))_{(v_1,v_2,v_3)}$ is a $\widehat{\Gamma}_{h-1}$-group.*

Also, the corollary can be used to design an isomorphism test for the class $\mathcal{C}_h$.

▶ **Corollary 26** (Theorem 3 restated)**.** *The Graph Isomorphism Problem for the class $\mathcal{C}_h$ can be solved in time $n^{\mathcal{O}((\log h)^c)}$ for some absolute constant c.*

Proof. Let $G_1, G_2 \in \mathcal{C}_h$. By standard decomposition techniques it suffices to consider the case where G_1 and G_2 are (vertex- and arc-colored) 3-connected graphs.

Suppose $G_1 = (V_1, E_1, \chi_V^1, \chi_E^1)$ and $G_2 = (V_2, E_2, \chi_V^2, \chi_E^2)$. Let $v_1, v_2, v_3 \in V(G_1)$ be three arbitrary vertices. For every $w_1, w_2, w_3 \in V(G_2)$ it is checked whether there is some isomorphism $\varphi\colon G_1 \cong G_2$ such that $\varphi(v_i) = w_i$ for all $i \in [3]$. Towards this end, define $\widehat{\chi}_V^1(v_i) = (i, 1)$ for $i \in [3]$ and $\widehat{\chi}_V^1(v) = (\chi_V^1(v), 0)$ for all $v \in V(G_1) \setminus \{v_1, v_2, v_3\}$. Similarly, define $\widehat{\chi}_V^2(w_i) = (i, 1)$ for $i \in [3]$ and $\widehat{\chi}_V^2(w) = (\chi_V^2(w), 0)$ for all $w \in V(G_2) \setminus \{w_1, w_2, w_3\}$. Hence, it needs to be checked whether $\widehat{G}_1 \cong \widehat{G}_2$ where $\widehat{G}_j \coloneqq (V_j, E_j, \widehat{\chi}_V^j, \chi_E^j)$, $j \in [2]$. By Corollary 24 the graphs $\widehat{G}_1$ and $\widehat{G}_2$ are $(h-1)$-CR-bounded. Hence, isomorphism of the two graphs can be tested within the desired time by Corollary 19. ◀

Note that the algorithm from the corollary describes a polynomial-time Turing reduction from the Graph Isomorphism Problem for $\mathcal{C}_h$ to the Graph Isomorphism Problem for $(h-1)$-CR-bounded graphs. In combination with Lemma 20 this means the the Graph Isomorphism Problem for $\mathcal{C}_h$ is polynomial-time Turing reducible to the Hypergraph Isomorphism Problem for $\widehat{\Gamma}_{h-1}$-groups.

Since the class of graphs of Euler genus at most g excludes $K_{3,4g+3}$ as a minor [32, 18], we obtain the following result.

▶ **Corollary 27** (Corollary 4 restated)**.** *The Graph Isomorphism Problem for graphs of genus at most g can be solved in time $n^{\mathcal{O}((\log g)^c)}$ for some absolute constant c.*

6 Conclusion

We provided a faster algorithm for the Hypergraph Isomorphism Problem for $\widehat{\Gamma}_d$-groups running in time $(n+m)^{\mathcal{O}((\log d)^c)}$ for some absolute constant c. As an application, we obtained, for example, an algorithm testing isomorphism of graphs excluding $K_{3,h}$ as a minor in time $n^{\mathcal{O}((\log h)^c)}$. In particular, this gives an isomorphism test for graphs of Euler genus at most g running in time $n^{\mathcal{O}((\log g)^c)}$.

With the Hypergraph Isomorphism Problem for $\widehat{\Gamma}_d$-groups being exploited as a subroutine in a number of algorithms testing isomorphism, it seems plausible to hope for further applications beyond the ones presented in this paper. Indeed, a very recent work by Grohe, Wiebking and the present author [16] gives an isomorphism test running in time $n^{\mathcal{O}((\log h)^c)}$ for n-vertex graphs excluding an arbitrary h-vertex graph as a minor. This algorithm crucially builds on the isomorphism test for hypergraphs as well as the notion of t-CR-bounded graphs. Actually, extending the applicability of our techniques further, it might be possible to provide an algorithm with a similar running time for all classes excluding only a topological minor building on a decomposition theorem for such graph classes due to Grohe and Marx [13].

Another open question concerns the complexity of testing isomorphism of hypergraphs for a given $\widehat{\Gamma}_d$-group. The algorithm presented in this work is significantly faster than the previous best algorithm [34] running in time $n^{\mathcal{O}(d)}m^{\mathcal{O}(1)}$ only for small numbers of hyperedges. Indeed, for large numbers of hyperedges $m = n^{\Omega(d)}$, our algorithm becomes slower than the algorithm due to Schweitzer and Wiebking [34]. Can the Hypergraph Isomorphism Problem for $\widehat{\Gamma}_d$-groups be solved in time $n^{\mathcal{O}((\log d)^c)}m^{\mathcal{O}(1)}$ for some absolute constant c? Observe that it is already completely unclear whether isomorphism of hypergraphs can be tested in time $n^{\mathcal{O}((\log n)^c)}m^{\mathcal{O}(1)}$ for some absolute constant c.

References

1 László Babai. Graph isomorphism in quasipolynomial time [extended abstract]. In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 684–697. ACM, 2016. doi:10.1145/2897518.2897542.

2 László Babai. Canonical form for graphs in quasipolynomial time: preliminary report. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 1237–1246. ACM, 2019. doi:10.1145/3313276.3316356.

3 László Babai, Peter J. Cameron, and Péter P. Pálfy. On the orders of primitive groups with restricted nonabelian composition factors. *J. Algebra*, 79(1):161–168, 1982. doi:10.1016/0021-8693(82)90323-4.

4 László Babai, Xi Chen, Xiaorui Sun, Shang-Hua Teng, and John Wilmes. Faster canonical forms for strongly regular graphs. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 157–166. IEEE Computer Society, 2013. doi:10.1109/FOCS.2013.25.

5 László Babai and Paolo Codenotti. Isomorhism of hypergraphs of low rank in moderately exponential time. In *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*, pages 667–676. IEEE Computer Society, 2008. doi:10.1109/FOCS.2008.80.

6 László Babai, William M. Kantor, and Eugene M. Luks. Computational complexity and the classification of finite simple groups. In *24th Annual Symposium on Foundations of Computer Science, Tucson, Arizona, USA, 7-9 November 1983*, pages 162–171. IEEE Computer Society, 1983. doi:10.1109/SFCS.1983.10.

7 László Babai and Eugene M. Luks. Canonical labeling of graphs. In David S. Johnson, Ronald Fagin, Michael L. Fredman, David Harel, Richard M. Karp, Nancy A. Lynch, Christos H. Papadimitriou, Ronald L. Rivest, Walter L. Ruzzo, and Joel I. Seiferas, editors, *Proceedings of the 15th Annual ACM Symposium on Theory of Computing, 25-27 April, 1983, Boston, Massachusetts, USA*, pages 171–183. ACM, 1983. doi:10.1145/800061.808746.

8 László Babai and John Wilmes. Quasipolynomial-time canonical form for steiner designs. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 261–270. ACM, 2013. doi:10.1145/2488608.2488642.

9 Christoph Berkholz, Paul S. Bonsma, and Martin Grohe. Tight lower and upper bounds for the complexity of canonical colour refinement. *Theory Comput. Syst.*, 60(4):581–614, 2017. doi:10.1007/s00224-016-9686-0.

10 Jin-yi Cai, Martin Fürer, and Neil Immerman. An optimal lower bound on the number of variables for graph identifications. *Combinatorica*, 12(4):389–410, 1992. doi:10.1007/BF01305232.

11 John D. Dixon and Brian Mortimer. *Permutation groups*, volume 163 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 1996. doi:10.1007/978-1-4612-0731-3.

12 Michael Elberfeld and Ken-ichi Kawarabayashi. Embedding and canonizing graphs of bounded genus in logspace. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 383–392. ACM, 2014. doi:10.1145/2591796.2591865.

13 Martin Grohe and Dániel Marx. Structure theorem and isomorphism test for graphs with excluded topological subgraphs. *SIAM J. Comput.*, 44(1):114–159, 2015. doi:10.1137/120892234.

14 Martin Grohe, Daniel Neuen, and Pascal Schweitzer. A faster isomorphism test for graphs of small degree. In Mikkel Thorup, editor, *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 89–100. IEEE Computer Society, 2018. doi:10.1109/FOCS.2018.00018.

15 Martin Grohe, Daniel Neuen, Pascal Schweitzer, and Daniel Wiebking. An improved isomorphism test for bounded-tree-width graphs. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 67:1–67:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.67.

16 Martin Grohe, Daniel Neuen, and Daniel Wiebking. Isomorphism testing for graphs excluding small minors. *CoRR*, abs/2004.07671, 2020. arXiv:2004.07671.

17 Martin Grohe and Pascal Schweitzer. Isomorphism testing for graphs of bounded rank width. In Venkatesan Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 1010–1029. IEEE Computer Society, 2015. doi:10.1109/FOCS.2015.66.

18 Frank Harary. *Graph theory.* Addison-Wesley, 1991.

19 Neil Immerman and Eric Lander. Describing graphs: A first-order approach to graph canonization. In Alan L. Selman, editor, *Complexity Theory Retrospective: In Honor of Juris Hartmanis on the Occasion of His Sixtieth Birthday, July 5, 1988*, pages 59–81. Springer New York, New York, NY, 1990. doi:10.1007/978-1-4612-4478-3_5.

20 Ken-ichi Kawarabayashi. Graph isomorphism for bounded genus graphs in linear time. *CoRR*, abs/1511.02460, 2015. arXiv:1511.02460.

21 Eugene M. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. *J. Comput. Syst. Sci.*, 25(1):42–65, 1982. doi:10.1016/0022-0000(82)90009-5.

22 Eugene M. Luks. Hypergraph isomorphism and structural equivalence of boolean functions. In Jeffrey Scott Vitter, Lawrence L. Larmore, and Frank Thomson Leighton, editors, *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing, May 1-4, 1999, Atlanta, Georgia, USA*, pages 652–658. ACM, 1999. doi:10.1145/301250.301427.

23 Eugene M. Luks and Takunari Miyazaki. Polynomial-time normalizers. *Discret. Math. Theor. Comput. Sci.*, 13(4):61–96, 2011. URL: http://dmtcs.episciences.org/531.

24 Brendan D. McKay. Practical graph isomorphism. *Congr. Numer.*, 30:45–87, 1981.

25 Brendan D. McKay and Adolfo Piperno. Practical graph isomorphism, II. *J. Symb. Comput.*, 60:94–112, 2014. doi:10.1016/j.jsc.2013.09.003.

26 Gary L. Miller. Isomorphism of graphs which are pairwise k-separable. *Information and Control*, 56(1/2):21–33, 1983. doi:10.1016/S0019-9958(83)80048-5.

27 Gary L. Miller. Isomorphism of k-contractible graphs. A generalization of bounded valence and bounded genus. *Information and Control*, 56(1/2):1–20, 1983. doi:10.1016/S0019-9958(83)80047-3.

28 Daniel Neuen. Graph isomorphism for unit square graphs. In Piotr Sankowski and Christos D. Zaroliagis, editors, *24th Annual European Symposium on Algorithms, ESA 2016, August 22-24, 2016, Aarhus, Denmark*, volume 57 of *LIPIcs*, pages 70:1–70:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.ESA.2016.70.

29 Daniel Neuen. *The Power of Algorithmic Approaches to the Graph Isomorphism Problem*. PhD thesis, RWTH Aachen University, Aachen, Germany, 2019. doi:10.18154/RWTH-2020-00160.

30 Daniel Neuen. Hypergraph isomorphism for groups with restricted composition factors. *CoRR*, abs/2002.06997, 2020. arXiv:2002.06997.

31 Ilia N. Ponomarenko. The isomorphism problem for classes of graphs closed under contraction. *Journal of Soviet Mathematics*, 55(2):1621–1643, 1991. doi:10.1007/BF01098279.

32 Gerhard Ringel. Das geschlecht des vollständigen paaren graphen. *Abh. Math. Semin. Univ. Hambg.*, 28(3):139–150, 1965. doi:10.1007/BF02993245.

33 Joseph J. Rotman. *An introduction to the theory of groups*, volume 148 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, fourth edition, 1995. doi:10.1007/978-1-4612-4176-8.

34 Pascal Schweitzer and Daniel Wiebking. A unifying method for the design of algorithms canonizing combinatorial objects. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 1247–1258. ACM, 2019. doi:10.1145/3313276.3316338.

35 Ákos Seress. *Permutation group algorithms*, volume 152 of *Cambridge Tracts in Mathematics*. Cambridge University Press, Cambridge, 2003. doi:10.1017/CBO9780511546549.

36 Xiaorui Sun and John Wilmes. Faster canonical forms for primitive coherent configurations: Extended abstract. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 693–702. ACM, 2015. doi:10.1145/2746539.2746617.

37 Daniel Wiebking. Graph isomorphism in quasipolynomial time parameterized by treewidth. *CoRR*, abs/1911.11257, 2019. arXiv:1911.11257.

On Solving (Non)commutative Weighted Edmonds' Problem

Taihei Oki
Department of Mathematical Informatics, Graduate School of Information Science and Technology, University of Tokyo, Japan
taihei_oki@mist.i.u-tokyo.ac.jp

Abstract

In this paper, we consider computing the degree of the Dieudonné determinant of a polynomial matrix $A = A_\ell + A_{\ell-1}s + \cdots + A_0 s^\ell$, where each A_d is a linear symbolic matrix, i.e., entries of A_d are affine functions in symbols $x_1, \dots, x_m$ over a field K. This problem is a natural "weighted analog" of Edmonds' problem, which is to compute the rank of a linear symbolic matrix. Regarding $x_1, \dots, x_m$ as commutative or noncommutative, two different versions of weighted and unweighted Edmonds' problems can be considered. Deterministic polynomial-time algorithms are unknown for commutative Edmonds' problem and have been proposed recently for noncommutative Edmonds' problem.

The main contribution of this paper is to establish a deterministic polynomial-time reduction from (non)commutative weighted Edmonds' problem to unweighed Edmonds' problem. Our reduction makes use of the discrete Legendre conjugacy between the integer sequences of the maximum degree of minors of A and the rank of linear symbolic matrices obtained from the coefficient matrices of A. Combined with algorithms for noncommutative Edmonds' problem, our reduction yields the first deterministic polynomial-time algorithm for noncommutative weighted Edmonds' problem with polynomial bit-length bounds. We also give a reduction of the degree computation of quasideterminants and its application to the degree computation of noncommutative rational functions.

2012 ACM Subject Classification Theory of computation → Algebraic complexity theory; Computing methodologies → Linear algebra algorithms; Computing methodologies → Combinatorial algorithms

Keywords and phrases skew fields, Edmonds' problem, Dieudonné determinant, degree computation, Smith–McMillan form, matrix expansion, discrete Legendre conjugacy

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.89

Category Track A: Algorithms, Complexity and Games

Funding This work was supported by JST CREST Grant Number JPMJCR14D2, Japan, Grant-in-Aid for JSPS Research Fellow Grant Number JP18J22141, Japan and JST ACT-I Grant Number JPMJPR18U9, Japan.

Acknowledgements The author thanks Satoru Iwata and Hiroshi Hirai for careful reading and helpful suggestions, and Christian Engels, Tasuku Soma, Shuichi Hirahara and Yuni Iwamasa for discussions.

1 Introduction

The background of this paper goes back to Edmonds [10]. In 1967, Edmonds posed a question whether there exists a polynomial-time algorithm to compute the rank of a *linear* (*symbolic*) *matrix* B over a field K, which is in the form

$$B = B_0 + B_1 x_1 + \cdots + B_m x_m,$$

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 89; pp. 89:1–89:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

where $B_0, B_1 \ldots, B_m \in K^{n \times n}$ and $x_1, \ldots, x_m$ are commutative symbols. Here, B is regarded as a matrix over the polynomial ring $K[x_1, \ldots, x_m]$ or the rational function field $K(x_1, \ldots, x_m)$. In case where B is the Edmonds or Tutte matrix of a bipartite or nonbipartite graph G, the rank computation for B corresponds to solving the maximum matching problem on G. More generally, Lovász [27] showed that Edmonds' problem is equivalent to a linear matroid intersection problem if all B_i are of rank 1, and to a linear matroid parity problem if all B_i are skew-symmetric matrices of rank 2. For general linear matrices, the celebrated Schwartz–Zippel lemma [34] provides a simple randomized algorithm if $|K|$ is large enough [27]. However, no deterministic polynomial-time algorithm still has been known; the existence of such an algorithm would imply nontrivial circuit complexity lower bounds [22, 36].

Recent studies [11, 17, 20] address the noncommutative version of Edmonds' problem (nc-Edmonds' problem). This is a problem of computing the *noncommutative rank* (nc-rank) of B, which is the rank defined by regarding $x_1, \ldots, x_m$ as pairwise noncommutative, i.e., $x_i x_j \neq x_j x_i$ if $i \neq j$. In this way, B is viewed as a matrix over the free ring $K\langle x_1, \ldots, x_m \rangle$ generated by noncommutative symbols $x_1, \ldots, x_m$. The nc-rank of B is precisely the rank of B over a skew (noncommutative) field $K \langle\!(x_1, \ldots, x_m)\!\rangle$, called a *free skew field*, which is the quotient of $K\langle x_1, \ldots, x_m \rangle$ defined by Amitsur [2]. We call a linear matrix over K having noncommutative symbols an *nc-linear matrix* over K. The recent studies [11, 17, 20] revealed that nc-Edmonds' problem is deterministically tractable. For the case where K is the set $\mathbb{Q}$ of rational numbers, Garg et al. [11] proved that Gurvits' *operator scaling algorithm* [16] deterministically computes the nc-rank of B in poly(n, m) arithmetic operations on $\mathbb{Q}$. Algorithms over general field K were later given by Ivanyos et al. [20] and Hamada–Hirai [17] exploiting the min-max theorem established for nc-rank. In [16] and [20] applied to the case of $K = \mathbb{Q}$, bit-lengths of intermediate numbers are proved to be bounded by a polynomial of the input bit-length.

In this paper, we shall consider "weighted" versions of commutative and noncommutative Edmonds' problem introduced by Hirai [18]. First, consider commutative symbols $x_1, \ldots, x_m$ and an extra commutative symbol s. Define a matrix

$$A = A_\ell + A_{\ell-1} s + \cdots + A_0 s^\ell, \tag{1}$$

where $A_d = A_{d,0} + A_{d,1} x_1 + \cdots + A_{d,m} x_m \in K[x_1, \ldots, x_m]^{n \times n}$ is a linear matrix over K for $d = 0, \ldots, \ell$. We call (1) a *linear polynomial matrix* over K. The *weighted Edmonds' problem* (WEP) is the problem to compute the degree (in s) of the determinant of A. Analogously to Edmonds' problem, WEP includes a bunch of weighted combinatorial optimization problems as special cases, such as a maximum weighted perfect matching problem, a weighted linear matroid intersection problem and a weighted linear matroid parity problem; see [18, Section 5].

Defining *noncommutative weighted Edmonds' problem* (nc-WEP) requires some more involved algebraic notions due to noncommutativity. Let $x_1, \ldots, x_m$ be noncommutative symbols and s an extra symbol that commutes with any element in $K\langle x_1, \ldots, x_m \rangle$. An *nc-linear polynomial matrix* A over K is a matrix in the form of (1) with each A_d regarded as an nc-linear matrix. Then A can be viewed as a matrix over the rational function (skew) field $F(s)$ over $F := K \langle\!(x_1, \ldots, x_m)\!\rangle$. Since entries of A are noncommutative, the determinant of A is nontrivial. Here, we employ the *Dieudonné determinant* [9], which is a noncommutative generalization of the usual determinant defined for matrices over skew fields. We denote the Dieudonné determinant of A by $\operatorname{Det} A$. The Dieudonné determinant retains useful properties of the usual determinant such as $\operatorname{Det} AB = \operatorname{Det} A \operatorname{Det} B$. While the value of $\operatorname{Det} A$ is no longer in $F(s)$, its degree (in s) is well-defined [8, 35]. See Section 2.1 for the definition of Dieudonné determinant.

The nc-WEP is the problem to compute $\deg \mathrm{Det}$ of a given nc-linear polynomial matrix. Hirai [18] formulated the dual problem of nc-WEP as the minimization of an *L-convex function* on a *uniform modular lattice*, and gave an algorithm based on the steepest gradient descent. Hirai's algorithm uses $\mathrm{poly}(n, m, \ell)$ arithmetic operations on K while no bit-length bound has been given for $K = \mathbb{Q}$.

A weighted combinatorial optimization problem often reduces to an unweighted problem. This paper explores a reduction from (nc-)WEP (a weighted problem) to (nc-)Edmonds' problem (an unweighted problem). The main result of this paper is the following.

▶ **Theorem 1.** *The (nc-)WEP deterministically reduces to (nc-)Edmonds' problem for an (nc-)linear matrix of size ℓn^2 with m symbols.*

Theorem 1 provides an efficient randomized algorithm for WEP through the Schwartz–Zippel lemma and a deterministic polynomial-time algorithm for nc-WEP via the rank computation algorithms [11, 18, 20] for nc-linear matrices. This algorithm for nc-WEP is much different from Hirai's algorithm [18]; in particular, while Hirai's algorithm calls an oracle of nc-Edmonds' problem polynomially many times, our algorithm calls it only once (the matrix size would be augmented instead). Furthermore, in case of $K = \mathbb{Q}$, our reduction does not exponentially swell the input bit-length because every entry of the nc-linear matrix constructed in this reduction is some coefficient of an entry in the input nc-linear polynomial matrix. Thus, by employing an algorithm [11, 18] for nc-Edmonds' problem with bit-length bounds, we obtain the following.

▶ **Theorem 2.** *We can deterministically solve nc-WEP in* $\mathrm{poly}(n, m, \ell)$ *arithmetic operations on K. In addition, if $K = \mathbb{Q}$, the bit-lengths of intermediate numbers are bounded by a polynomial of the input bit-length.*

We also give a reduction from computing the degree of a *quasideterminant* [12, 13], which is another noncommutative analogy of the determinant, to computing the degree of Dieudonné determinant. This can be applied to the degree computation of noncommutative rational functions represented as a noncommutative formula with division. See Section 4 for details.

Techniques

Let A be an $n \times n$ (nc-)linear polynomial matrix over a field K and put $F := K(x_1, \dots, x_m)$ or $K\langle\!\langle x_1, \dots, x_m \rangle\!\rangle$. Slightly generalizing (nc-)WEP, we consider the problem to compute

$$d_k(A) := \max\{\deg \mathrm{Det}\, A[I, J] \mid |I| = |J| = k\} \tag{2}$$

for given k, where $A[I, J]$ denotes the submatrix of A indexed by a row set I and a column set J. Clearly $\deg \mathrm{Det}\, A = d_n(A)$. In view of combinatorial optimization, computation of $d_k(A)$ corresponds to solving weighted problems under cardinality constraints.

Our reduction scheme is based on a method, which we call *matrix expansion*, that constructs an (nc-)linear matrix $\Omega_\mu(A) \in F^{\mu n \times \mu n}$ obtained by arranging the coefficient matrices of A. Through a canonical form of A called the *Smith–McMillan form*, it is shown that the integer sequences of $(d_0(A), d_1(A), \dots, d_r(A))$ with $r := \mathrm{rank}\, A$ and $(\omega_0(A), \omega_1(A), \dots)$ with $\omega_\mu(A) := \mathrm{rank}\, \Omega_\mu(A)$ are concave and convex, respectively. In addition, they are in the relation of the *discrete Legendre conjugate*, that is, they satisfy

$$d_k(A) = \min_{\mu \geq 0}(\omega_\mu(A) - k\mu) \qquad (0 \leq k \leq r), \tag{3}$$

$$\omega_\mu(A) = \max_{0 \leq k \leq r}(d_k(A) + k\mu) \quad (\mu \geq 0). \tag{4}$$

The Legendre conjugacy is an important duality relation on discrete convex and concave functions treated in *discrete convex analysis* [32]. The formulas (3) and (4) are a generalization of results on matrix pencils over fields given by Murota [33] and on polynomial matrices over algebraically closed fields by Moriyama–Murota [28]. For proving the conjugacy, equalities connecting $d_k(A)$ and $\omega_\mu(A)$ are necessary. We present a new short and simple connection which works even on skew fields through the multiplicativity of Ω_μ, i.e.,

$$\Omega_\mu(A)\Omega_\mu(B) = \Omega_\mu(AB). \tag{5}$$

The conjugacy formula (3) reduces the computation of $d_k(A)$ to a one-dimensional discrete convex optimization problem, which can be efficiently done by binary search. In each iteration, the objective function can be evaluated by solving (nc-)Edmonds' problem. Moreover, we derive direct formulas with respect to r and $d_r(A)$ from (3), which proves Theorem 1.

Related Work

In the field of computer algebra, algorithms were proposed for computing various kinds of canonical forms of a polynomial matrix $A \in F[s]^{n \times n}$ (or of its generalization) such as the *Jacobson normal form* [26], the *Hermite normal form* [14], the *Popov normal form* [23] and their weaker form called a *row-reduced form* [1, 4]. These algorithms iteratively solve systems of linear equations over F whose coefficient matrices are variants of expanded matrices $\Omega_\mu(A)$ under the name of "linearized matrices" [23] or "striped Krylov matrices" [4]. While these algorithms can compute $\deg \operatorname{Det} A$, their running time is bounded in terms of the number of operations on F. Hence if $F = K(x_1, \ldots, x_m)$ or $F = K\langle\!\langle x_1, \ldots, x_m \rangle\!\rangle$, the expression size of intermediate numbers might be exponentially large.

Combinatorial relaxation [29, 30] is another framework for deg-det computation based on combinatorial optimization. Hirai's algorithm [18] for nc-WEP can also be viewed as a variant of combinatorial relaxation. Unlike the matrix expansion, it is difficult to give bit complexity bounds for combinatorial relaxation algorithms because they iteratively perform the Gaussian elimination on the same matrix and thus the magnitude of its entries might swell.

Organization

The rest of this paper is organized as follows. Section 2 provides preliminaries on matrices and polynomials over skew fields. Section 3 describes our proposed reductions after introducing the matrix expansion and the Legendre conjugacy. Section 4 describes the computation of the degree of quasideterminants and its application to the degree computation of noncommutative rational functions.

2 Preliminaries

Let $\mathbb{Z}$ denote the set of integers and $\mathbb{N}$ the set of nonnegative integers. For $n \in \mathbb{N}$, we denote the set $\{1, 2, \ldots, n\}$ by $[n]$ and $\{0, 1, 2, \ldots, n\}$ by $[0, n]$.

2.1 Matrices over Skew Fields

A *skew field*, or a *division ring* is a ring F such that every nonzero element has a multiplicative inverse in F. A right (left) F-module is especially called a *right (left) F-vector space*. The *dimension* of a right (left) F-vector space V is defined as the rank of V as a module, that is, the cardinality of any basis of V. The usual facts from linear algebra on independent sets and generating sets in vector spaces are valid even on skew fields [25].

A square matrix $A \in F^{n\times n}$ is said to be *nonsingular* if there exists a unique $n \times n$ matrix over F, denoted by A^{-1}, such that AA^{-1} and $A^{-1}A$ are the identity matrix I_n of size n. A square matrix is *singular* if it is not nonsingular. The *rank* $\operatorname{rank} A$ of a matrix $A \in F^{n\times n'}$ is the dimension of the right F-vector space spanned by the column vectors of A, and is equal to the dimension of the left F-vector space spanned by the row vectors of A. The rank is invariant under (right and left) multiplications of nonsingular matrices. It is observed that a square matrix $A \in F^{n\times n}$ is nonsingular if and only if $\operatorname{rank} A = n$.

The *Bruhat decomposition* uniquely factors a nonsingular matrix $A \in F^{n\times n}$ into the product of four $n \times n$ matrices over F as $A = LDPU$, where L is lower unitriangular, D is diagonal, P is a permutation matrix and U is upper unitriangular [7, Theorem 2.2 in Section 11.2]. Here, a lower (upper) unitriangular matrix is a lower (upper) triangular matrix whose diagonal entries are 1. Let $F^\times_{\mathrm{ab}} := F^\times / [F^\times, F^\times]$ denote the abelianization of the multiplicative subgroup $F^\times = F \setminus \{0\}$ of F, where $[F^\times, F^\times] := \langle \{aba^{-1}b^{-1} \mid a, b \in F^\times\} \rangle$ is the commutator subgroup of $F^\times$. The *Dieudonné determinant* $\operatorname{Det} A$ of a nonsingular matrix $A \in F^{n\times n}$, which is decomposed as $A = LDPU$, is an element of $F^\times_{\mathrm{ab}}$ defined by

$$\operatorname{Det} A := \operatorname{sgn}(P) e_1 e_2 \cdots e_n \bmod [F^\times, F^\times],$$

where $\operatorname{sgn}(P) \in \{-1, +1\}$ is the sign of the permutation P and $e_1, \ldots, e_n \in F^\times$ are the diagonal entries of D [9]. For a singular matrix $A \in F^{n\times n}$, define $\operatorname{Det} A$ as 0 for convenience. In case where F is commutative, the Dieudonné determinant coincides with the usual determinant. As the usual determinant, the Dieudonné determinant satisfies the following properties [3, Chapter 4.1]:

(D1) $\operatorname{Det} AB = \operatorname{Det} A \operatorname{Det} B$ for $A, B \in F^{n\times n}$.

(D2) $\operatorname{Det} \begin{pmatrix} A & * \\ O & B \end{pmatrix} = \operatorname{Det} \begin{pmatrix} A & O \\ * & B \end{pmatrix} = \operatorname{Det} A \operatorname{Det} B$ for $A \in F^{n\times n}$ and $B \in F^{n'\times n'}$, where blocks in O and $*$ represent zero and any matrices of appropriate size, respectively.

2.2 Polynomials over Skew Fields

Let us consider the polynomial ring $F[s]$ over a skew field F, where s is an indeterminate that commutes with any element of F. A nonzero polynomial $p \in F[s]$ is uniquely written as $p = \sum_{d=0}^{\ell} a_{\ell-d} s^d$, where $a_0, \ldots, a_\ell \in F$ with $a_0 \neq 0$. The addition and the multiplication in $F[s]$ are naturally defined. The *degree* $\deg p$ of p is defined by $\deg p := \ell$ and we set $\deg 0 := -\infty$. Then the minus of the degree enjoys the *discrete valuation* property, that is, deg satisfies $\deg(p + q) \leq \max\{\deg p, \deg q\}$ and $\deg pq = \deg p + \deg q$ for $p, q \in F[s]$.

The polynomial ring $F[s]$ is a (right and left) *Ore domain*, i.e., for each $p, q \in F[s] \setminus \{0\}$, there exists $u, u', v, v' \in F[s] \setminus \{0\}$ such that $pu = pv$ and $u'p = v'q$. This property enables $F[s]$ to have the (right and left) *Ore quotient ring*, which is a skew field of fractions each of whose elements is expressed as $f = pq^{-1} = q'^{-1}p'$ for some $p, p', q, q' \in F[s]$ with $q, q' \neq 0$. Elements of $F(s)$ are called *rational functions* over F and $F(s)$ is called the *rational function field* over F. See [7, Section 9.1] and [15, Chapter 6] for the construction of $F(s)$. The degree on $F[s]$ is uniquely extended to a valuation on $F(s)$ by $\deg f := \deg p - \deg q$ for $f = pq^{-1} \in F(s)$; see [8, Proposition 9.1.1]. A rational function $f \in F(s)$ is said to be *proper* if $\deg f \leq 0$.

The *Laurent series field* $F((s^{-1}))$ over F in s^{-1} is the set of formal power series over F in the form of

$$f = \sum_{d=-\ell}^{\infty} a_d s^{-d} \tag{6}$$

for some $\ell \in \mathbb{Z}$ and $a_{-\ell}, a_{-\ell+1}, \ldots \in F$. This skew field has the natural addition and multiplication. The rational function field $F(s)$ can be embedded in $F((s^{-1}))$ [6, Proposition 7.1]. Namely, any rational function $f \in F(s)$ can be uniquely expanded in form of (6). In particular, ℓ coincides with $\deg f$.

Let $A \in F(s)^{n \times n}$ be a square matrix over $F(s)$, called a *rational function matrix* over F. The degree of the Dieudonné determinant of A is well-defined since all commutators of $F(s)^\times$ have degree zero. Note that deg Det of singular matrices are $-\infty$. The following properties on deg Det are easily seen from (D1) and (D2).

(MV1) $\deg \operatorname{Det} AB = \deg \operatorname{Det} A + \deg \operatorname{Det} B$ for $A, B \in F(s)^{n \times n}$.

(MV2) $\deg \operatorname{Det} \begin{pmatrix} A & * \\ O & B \end{pmatrix} = \deg \operatorname{Det} \begin{pmatrix} A & O \\ * & B \end{pmatrix} = \deg \operatorname{Det}(A) + \deg \operatorname{Det}(B)$ for $A \in F(s)^{n \times n}$ and $B \in F(s)^{n' \times n'}$.

Recall the notation $d_k(A)$ in (2) for $A \in F(s)^{n \times n'}$. Note that $d_1(A)$ is the maximum degree of an entry in A, and we call $d_1(A)$ the *degree* of A. Similarly to (6), A can be uniquely expanded as

$$A = \sum_{d=-\ell}^{\infty} A_d s^{-d} \tag{7}$$

with $\ell = d_1(A)$ and some $A_{-\ell}, A_{-\ell+1}, \ldots \in F(s)^{n \times n'}$. The following proposition gives lower and upper bounds on $d_k(A)$.

► **Proposition 3.** *Let $A \in F(s)^{n \times n'}$ be a rational function matrix over a skew field F. For $k \in [0, n^*]$ with $n^* := \min\{n, n'\}$, the following hold:*

(1) *$d_k(As^\ell) = d_k(A) + \ell k$ for $\ell \in \mathbb{Z}$.*

(2) *$d_k(A) \leq \ell k$, where ℓ is the degree of A.*

(3) *$d_k(A) > -\infty$ if and only if $k \leq \operatorname{rank} A$. In addition, if A is a polynomial matrix, then $d_k(A) \geq 0$ for $k \leq \operatorname{rank} A$.*

Proof. (1) follows from the fact that for any $k \times k$ submatrix $A[I, J]$ of A, it holds

$$\begin{aligned} \deg \operatorname{Det} A[I, J] s^\ell &= \deg \operatorname{Det}(A[I, J] \cdot s^\ell I_k) \\ &= \deg \operatorname{Det} A[I, J] + \deg \det s^\ell I_k \\ &= \deg \operatorname{Det} A[I, J] + \ell k. \end{aligned}$$

(2) Let $\alpha_1, \ldots, \alpha_k$ be the exponents of the Smith–McMillan form of a nonsingular $k \times k$ submatrix $A[I, J]$ of A. Then the claim follows from $\deg \operatorname{Det} A[I, J] = \alpha_1 + \cdots + \alpha_k$ and $\ell \geq \alpha_1 \geq \cdots \geq \alpha_k$.

(3) The former part is obtained from the fact that rank A is equal to the maximum size of a nonsingular submatrix of A. The latter part can be proved using the *Smith normal form*, see e.g. [18, Lemma 2.11]. ◄

A rational function matrix is said to be *proper* if its degree is nonpositive. A square rational function matrix is said to be *biproper* if it is proper and nonsingular, and its inverse is also proper. We abbreviate proper and biproper rational function matrices as proper and biproper matrices, respectively. It is easy to see that the product of proper matrices are proper, which implies that the product of biproper matrices are biproper again. Equivalent conditions for proper matrices to be biproper are established as follows.

▶ **Lemma 4** ([18, Lemma 2.10])**.** *Let $A \in F(s)^{n\times n}$ be a square proper matrix over a skew field F. Then the following are equivalent:*

(1) *A is biproper.*

(2) $\deg \operatorname{Det} A = 0$.

(3) *The coefficient matrix A_0 of s^0 in the expansion* (7) *of A is nonsingular.*

A *biproper transformation* is a transformation of a rational function matrix $A \in F(s)^{n\times n'}$ in the form $A \mapsto SAT$, where $S \in F(s)^{n\times n}$ and $T \in F(s)^{n'\times n'}$ are biproper matrices. Under biproper transformations, we can establish a canonical form of rational function matrices, called the *Smith–McMillan form*. This is well-known for complex rational function matrices as the *Smith–McMillan form at infinity* [31, 37] in the context of control theory.

▶ **Proposition 5** (Smith–McMillan form)**.** *Let $A \in F(s)^{n\times n'}$ be a rational function matrix of rank r over a skew field F. There exist biproper matrices $S \in F(s)^{n\times n}$, $T \in F(s)^{n'\times n'}$ and integers $\alpha_1 \ge \alpha_2 \ge \cdots \ge \alpha_r$ such that*

$$SAT = \begin{pmatrix} \operatorname{diag}(s^{\alpha_1}, \ldots, s^{\alpha_r}) & O \\ O & O \end{pmatrix}.$$

The integer α_i is uniquely determined by

$$\alpha_i = d_i(A) - d_{i-1}(A) \tag{8}$$

for $i \in [r]$. In particular, $d_k(A)$ is invariant under biproper transformations for $k \in [0, r]$.

Proof. The proof is the same as that for nonsingular $A \in F(s)^{n\times n}$ in [18, Proposition 2.9], which iteratively determines α_i from $i = 1$ to n, except that the iterations stops when $i = r$. ◀

Solving (8) for $d_k(A)$, we obtain

$$d_k(A) = \sum_{i=1}^{k} \alpha_i \tag{9}$$

for $k \in [0, r]$. This is a key identity that connects $d_k(A)$ and the Smith–McMillan form of A. It is worth mentioning that all α_i are nonpositive for a proper matrix A since α_1 is equal to the degree $d_1(A)$ of A by (8).

3 Computing the Degree of Dieudonné Determinant

Let $A = \sum_{d=0}^{\ell} A_{\ell-d} s^d \in F[s]^{n\times n'}$ be a polynomial matrix over a skew field F; we typically consider an (nc-)linear polynomial matrix with $F \coloneqq K(x_1, \ldots, x_m)$ or $K\lbrace\!\!\lbrace x_1, \ldots, x_m \rbrace\!\!\rbrace$. In this section, we give reductions of computing $d_k(A)$ and $\operatorname{rank} A$ to rank computations over F. Instead of A, we deal with a proper matrix obtained from A by

$$As^{-\ell} = \sum_{d=0}^{\ell} A_d s^{-d} \in F(s)^{n\times n'}. \tag{10}$$

The value of $d_k(A)$ can be recovered from that of (10) through Proposition 3 (1).

Section 3.1 introduces *matrix expansion* which is our key tool. Section 3.2 connects the sequence of d_k to the rank of expanded matrices via the *Legendre conjugacy*. Making use of them, we give reductions and algorithms in Section 3.3, which proves Theorem 1.

3.1 Matrix Expansion

For a proper matrix $A \in F(s)^{n \times n'}$ and $\mu \in \mathbb{N}$, we define the *μth-order expanded matrix* $\Omega_\mu(A)$ of A as the following $\mu n \times \mu n'$ block matrix

$$\Omega_\mu(A) \coloneqq \begin{pmatrix} A_0 & A_1 & A_2 & \cdots & \cdots & A_{\mu-1} \\ O & A_0 & A_1 & A_2 & & \vdots \\ \vdots & O & A_0 & A_1 & \ddots & \vdots \\ \vdots & & \ddots & \ddots & \ddots & A_2 \\ \vdots & & & \ddots & A_0 & A_1 \\ O & \cdots & \cdots & \cdots & O & A_0 \end{pmatrix} \in F^{\mu n \times \mu n'},$$

where $A_0, \ldots, A_{\mu-1} \in F^{n \times n'}$ are matrices in the expansion (7) of A. Note that $\Omega_\mu(A)$ is an (nc-)linear matrix over K. Expanded matrices satisfy the multiplicativity (5).

▶ **Lemma 6.** *Let $A \in F(s)^{n \times n'}$ and $B \in F(s)^{n' \times n''}$ be proper matrices over a skew field F. Then it holds* (5) *for any $\mu \in \mathbb{N}$.*

Proof. Expand A and B by (7) as $A = \sum_{d=0}^{\infty} A_d s^{-d}$ and $B = \sum_{d=0}^{\infty} B_d s^{-d}$. By

$$AB = \left(\sum_{d=0}^{\infty} A_d s^{-d}\right)\left(\sum_{d=0}^{\infty} B_d s^{-d}\right) = \sum_{d=0}^{\infty} A_d \left(\sum_{j=0}^{d} B_{d-j} s^{-j}\right) = \sum_{j=0}^{\infty}\left(\sum_{d=0}^{j} A_d B_{d-j}\right) s^{-j},$$

the (i, j)th block in $\Omega_\mu(AB)$ is $\sum_{d=0}^{j-i} A_d B_{d-j}$ if $i \leq j$ and O otherwise. This coincides with the (i, j)th block in $\Omega_\mu(A)\Omega_\mu(B)$. ◀

Let $\omega_\mu(A)$ denote the rank of $\Omega_\mu(A)$. The following lemma claims that $\omega_\mu(A)$ coincides with that of the Smith–McMillan form of A.

▶ **Lemma 7.** *Let $A \in F(s)^{n \times n'}$ be a proper matrix over a skew field F. Then it holds $\omega_\mu(A) = \omega_\mu(D)$ for $\mu \in \mathbb{N}$, where D is the Smith–McMillan form of A.*

Proof. Let $S \in F(s)^{n \times n}$ and $T \in F(s)^{n' \times n'}$ be biproper matrices such that $SAT = D$. From Lemma 6, we have $\omega_\mu(D) = \operatorname{rank} \Omega_\mu(SAT) = \operatorname{rank} \Omega_\mu(S)\Omega_\mu(A)\Omega_\mu(T)$. Let S_0 and T_0 be the coefficient matrices of s^0 in the expansion (7) of S and T, respectively. Since S_0 and T_0 are nonsingular by Lemma 4,the block matrices $\Omega_\mu(S)$ and $\Omega_\mu(T)$ are also nonsingular. Therefore we have $\omega_\mu(D) = \omega_\mu(A)$. ◀

Let $0 \geq \alpha_1 \geq \cdots \geq \alpha_r$ be the exponents of the Smith–McMillan form of A with $r \coloneqq \operatorname{rank} A$. Put

$$N_d \coloneqq |\{i \in [r] \mid -\alpha_i \leq d\}| \tag{11}$$

for $d \in \mathbb{N}$. Lemma 7 leads us to the following lemma; a similar result based on the Kronecker canonical form is also known for matrix pencils over a field [21, Theorem 2.3].

▶ **Lemma 8.** *Let $A \in F(s)^{n \times n'}$ be a proper matrix over a skew field F. For $\mu \in \mathbb{N}$, it holds*

$$\omega_\mu(A) = \sum_{d=0}^{\mu-1} N_d, \tag{12}$$

where N_d is defined in (11)*.*

Proof. Let $D = \sum_{d=0}^{\infty} D_d s^{-d}$ be the Smith–McMillan form of A and $\alpha_1, \dots, \alpha_r$ the exponents of diagonal entries of D, where $r := \operatorname{rank} A$. The ith diagonal entry of D_d is 1 if $i \leq r$ and $\alpha_i = -d$, and 0 otherwise. Thus each row and column in $\Omega_\mu(D)$ has at most one nonzero entry. Hence $\omega_\mu(D)$, which is equal to $\omega_\mu(A)$ by Lemma 7, is equal to the number of nonzero entries in $\Omega_\mu(D)$. It is easily checked that the $(\mu - d)$th block row of $\Omega_\mu(D)$ contains N_d nonzero entries for $d = 0, \dots, \mu - 1$. ◀

The equality (12) is a key identity that connects $\omega_\mu(A)$ and the Smith–McMillan form of A. We remark that, for $d \in \mathbb{N}$, the equality (12) can be rewritten as

$$N_d = \omega_{d+1}(A) - \omega_d(A). \tag{13}$$

3.2 Legendre Conjugacy of $d_k(A)$ and $\omega_\mu(A)$

Let $A \in F(s)^{n \times n'}$ be a proper matrix of rank r and $\alpha_1 \geq \dots \geq \alpha_r$ the exponents of the Smith–McMillan form of A. Put $d_k := d_k(A)$ for $k = 0, \dots, r$. From $\alpha_k \geq \alpha_{k+1}$ and (8), the inequality $d_{k-1} + d_{k+1} \leq 2d_k$ holds for all $k \in [r-1]$. In addition, for $\mu \in \mathbb{N}$, put $\omega_\mu := \omega_\mu(A)$ and define N_μ by (11). From $N_{\mu-1} \leq N_\mu$ and (13), we have $\omega_{\mu-1} + \omega_{\mu+1} \geq 2\omega_\mu$ for all $\mu \geq 1$. These two inequalities for d_k and ω_μ indicate the *concavity* of d_k and the *convexity* of ω_μ in the following sense.

A (discrete) function $f \colon \mathbb{Z} \to \mathbb{Z} \cup \{+\infty\}$ is said to be *convex* if

$$f(x-1) + f(x+1) \geq 2f(x)$$

for all $x \in \mathbb{Z}$. We call a function $g \colon \mathbb{Z} \to \mathbb{Z} \cup \{-\infty\}$ *concave* if $-g$ is convex. An integer sequence $(a_k)_{k \in K}$ indexed by $K \subseteq \mathbb{Z}$ can be identified with a function $\check{a} \colon \mathbb{Z} \to \mathbb{Z} \cup \{+\infty\}$ by letting $\check{a}(k)$ be a_k if $k \in K$ and $+\infty$ otherwise. We can also identify a with $\hat{a} \colon \mathbb{Z} \to \mathbb{Z} \cup \{-\infty\}$ defined by $\hat{a}(k) := a_k$ if $k \in K$ and $\hat{a}(k) := -\infty$ otherwise. In this way, we identify integer sequences $(d_0, d_1, \dots, d_r)$ and $(\omega_0, \omega_1, \omega_2, \dots)$ with discrete functions $\check{d} \colon \mathbb{Z} \to \mathbb{Z} \cup \{-\infty\}$ and $\hat{\omega} \colon \mathbb{Z} \to \mathbb{Z} \cup \{+\infty\}$, respectively. From the argument in the previous paragraph, $(d_0, d_1, \dots, d_r)$ is concave and $(\omega_0, \omega_1, \omega_2, \dots)$ is convex.

Let $f \colon \mathbb{Z} \to \mathbb{Z} \cup \{+\infty\}$ be a function such that $f(x) \in \mathbb{Z}$ for some $x \in \mathbb{Z}$. The *concave conjugate* of f is a function $f^\circ \colon \mathbb{Z} \to \mathbb{Z} \cup \{-\infty\}$ defined by

$$f^\circ(y) := \inf_{x \in \mathbb{Z}} (f(x) - xy)$$

for $y \in \mathbb{Z}$. Similarly, for a function $g \colon \mathbb{Z} \to \mathbb{Z} \cup \{-\infty\}$ with $g(y) \in \mathbb{Z}$ for some $y \in \mathbb{Z}$, the *convex conjugate* of g is a function $g^\bullet \colon \mathbb{Z} \to \mathbb{Z} \cup \{+\infty\}$ given by

$$g^\bullet(x) := \sup_{y \in \mathbb{Z}} (g(y) + xy)$$

for $x \in \mathbb{Z}$. The maps $f \mapsto f^\circ$ and $g \mapsto g^\bullet$ are referred to as the *concave* and *convex discrete Legendre transform*, respectively. In general, f° is concave and $g^\bullet$ is convex. In addition, if f is convex and g is concave,

$$(f^\circ)^\bullet = f, \quad (g^\bullet)^\circ = g \tag{14}$$

hold. Hence the Legendre transformation establishes a one-to-one correspondence between discrete convex and concave functions. See [32] for details of discrete convex/concave functions and their Legendre transform.

Indeed, as explained in Section 1, the sequences of d_k and ω_μ are in the relation of Legendre conjugate. This can be shown from the key identities (9) and (12) that connect $d_k(A)$ and $\omega_\mu(A)$ through the Smith–McMillan form of A.

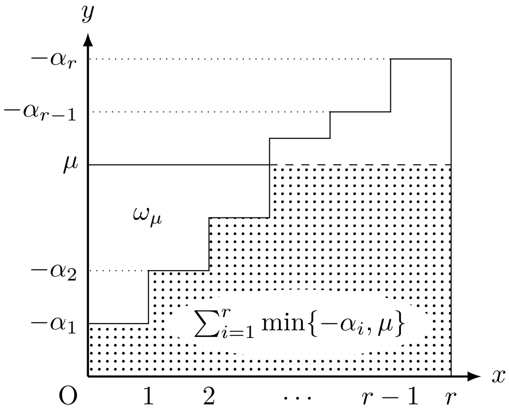


Figure 1 Graphic explanation of (15).

▶ **Theorem 9.** *Let $A \in F(s)^{n\times n'}$ be a proper matrix of rank r over a skew field F. Then* (3) *and* (4) *hold.*

Proof. Put $d_k := d_k(A)$ for $k = 0,\dots,r$ and $\omega_\mu := \omega_\mu(A)$ for $\mu \in \mathbb{N}$. Since $(d_0, d_1, \dots, d_r)$ is concave and $(\omega_0, \omega_1, \omega_2, \dots)$ is convex, (3) and (4) are equivalent by (14). We show (4).

First we give an equality

$$\omega_\mu = r\mu - \sum_{i=1}^{r}\min\{-\alpha_i,\mu\} \tag{15}$$

for $\mu \in \mathbb{N}$, where $\alpha_1 \geq \dots \geq \alpha_r$ are the exponents of the Smith–McMillan form of A. Figure 1 graphically shows this equality. Let x and y be the coordinates along the horizontal and vertical axes in Figure 1, respectively. For $i = 1,\dots,r$, the height of the dotted rectangle with $i-1 \leq x < i$ is $\min\{-\alpha_i,\mu\}$. Hence the area of the dotted region is equal to $\sum_{i=1}^{r}\min\{-\alpha_i,\mu\}$. In addition, the width of the white rectangle with $d \leq y < d+1$ is equal to N_d for $d = 0,\dots,\mu-1$, where N_d is defined by (11). Hence the area of the white stepped region is equal to $N_0 + \cdots + N_{\mu-1} = \omega_\mu$ by (12). Now we have (15) since the sum of the areas of these two regions is $r\mu$.

Substituting (9) into the right hand side of (4), we have

$$\max_{0\leq k\leq r}(d_k + k\mu) = \max_{0\leq k\leq r}\sum_{i=1}^{k}(\alpha_i+\mu) = \sum_{i=1}^{k^*}\alpha_i + k^*\mu, \tag{16}$$

where k^* is the maximum $0 \leq k \leq r$ such that $\alpha_k + \mu \geq 0$. Since $\min\{-\alpha_i,\mu\}$ is $-\alpha_i$ if $i \leq k^*$ and is μ if $i > k^*$, it holds

$$\sum_{i=1}^{r}\min\{-\alpha_i,\mu\} = -\sum_{i=1}^{k^*}\alpha_i + (r-k^*)\mu. \tag{17}$$

From (16) and (17), we have

$$\max_{0\leq k\leq r}(d_k + k\mu) = r\mu - \sum_{i=1}^{r}\min\{-\alpha_i,\mu\},$$

in which the right hand side is equal to ω_μ by (15). ◀

3.3 Reductions and Algorithms

Let $A = A_0 + A_1 s^{-1} + \cdots + A_\ell s^{-\ell} \in F(s)^{n \times n'}$ be the proper matrix (10) of rank r. The expression (15) of $d_k(A)$ indicates that $d_k(A)$ is equal to the optimal value of an minimization problem with objective function

$$f_k(\mu) \coloneqq \omega_\mu(A) - k\mu. \tag{18}$$

Since f_k is convex, it is minimized by the minimum μ such that $f_k(\mu+1) - f_k(\mu) \geq 0$. This can be found by the binary search in $\mathrm{O}(\log M)$ evaluations of f_k, where M is an upper bound on a minimizer of f_k. The following lemma claims that we can adopt ℓr as the upper bound.

▶ **Lemma 10.** *Let $A = \sum_{d=0}^{\ell} A_d s^{-d} \in F(s)^{n \times n'}$ be the proper matrix* (10) *of rank r. Then the following hold:*

(1) *The exponents $\alpha_1, \ldots, \alpha_r$ of the Smith–McMillan form of A are at least $-\ell r$.*

(2) *For $k \in [0, r]$, the function f_k in* (18) *has a minimizer μ^* satisfying $0 \leq \mu^* \leq \ell r$.*

Proof. The claims are trivial if $r = 0$. Suppose $r \geq 1$.

(1) It suffices to show $\alpha_r \geq -\ell r$. Since A is proper, $d_{r-1}(A)$ is nonpositive. In addition, since As^ℓ is a polynomial matrix of rank r, we have $0 \leq d_r(As^\ell) = d_r(A) + \ell r$ by Proposition 3 (1) and (3). Thus $\alpha_r = d_r(A) - d_{r-1}(A) \geq -\ell r$ holds.

(2) From Lemma 8, the objective function f_k can be written as

$$f_k(\mu) = \sum_{d=0}^{\mu-1} (N_d - k)$$

for $\mu \in \mathbb{N}$. Hence f_k is minimized by the maximum $\mu \in \mathbb{N}$ such that $N_\mu + k < 0$. Note that such μ exists since f_k has the minimum value. From the definition (11) of N_d, it holds $N_d = N_{-\alpha_r}$ for all $d \geq -\alpha_r$. Hence f_k has a minimizer less than or equal to $-\alpha_r$, which is at most ℓr by (1). ◀

Finally, we show direct formulas of rank A and $d_r(A)$ for a proper matrix A in (10). These formulas naturally yield efficient algorithms to compute them, which proves Theorem 1.

▶ **Lemma 11.** *Let $A = \sum_{d=0}^{\ell} A_d s^{-d} \in F(s)^{n \times n'}$ be the proper matrix* (10) *of rank r. Then it holds $r = \omega_{ln^*+1}(A) - \omega_{ln^*}(A)$ and $d_r(A) = \omega_{lr}(A) - lr^2$, where $n^* \coloneqq \min\{n, n'\}$.*

Proof. We first show the formula on r. We have $\omega_{\ell n^*+1}(A) - \omega_{\ell n^*}(A) = N_{\ell n^*}$ by (13). Since $-\alpha_i$ is at most $\ell r \leq \ell n^*$ for all $i \in [r]$ by Lemma 10 (1), we have $r = N_{\ell n^*}$.

Next we show the formula on $d_r(A)$. From (3) and (12), it holds

$$d_r(A) = \min_{\mu \geq 0} \sum_{d=0}^{\mu-1} (N_d - r). \tag{19}$$

Since $N_0 \leq N_1 \leq \cdots \leq N_{\ell r} = N_{\ell r+1} = \cdots = r$ by Lemma 10 (1), the minimum value of the right hand side of (19) is attained by $\mu = \ell r$. Thus we are done. ◀

▶ Remark 12. In view of combinatorial optimization, our algorithms are regarded as pseudo-polynomial time algorithms since the running time depends on a polynomial of the maximum exponent ℓ of s instead of $\mathrm{poly}(\log \ell)$. Thus it is natural to try to solve the following problem:

Sparse Degree of Determinant (SDD)

Input : $A = A_1 s^{w_1} + \cdots + A_m s^{w_m} \in K[s]^{n \times n}$, where $0 \leq w_1 \leq \ldots \leq w_m$ are integers.

Output: $\deg \det A$.

However, setting $w_k := (n+1)^k$ for $k \in [m]$ would make the rank of A the same as that of a linear matrix $A_1x_1 + \cdots + A_mx_m \in K[x_1, \ldots, x_m]^{n \times n}$ (known as the *Kronecker substitution* [24]). Since giving a deterministic polynomial-time algorithm for Edmonds' problem has still been open for more than half a century, SDD is also a quite challenging problem.

4 Computing the Degree of Quasideterminants

The *quasideterminant* [12, 13] is another noncommutative analogy of the determinant than the Dieudonné determinant. Let $A \in F^{n \times n}$ be a square matrix over a skew field F. Fix $i, j \in [n]$ and put $I := [n] \setminus \{i\}$, $J := [n] \setminus \{j\}$. The (i,j)th *quasideterminant* $|A|_{i,j}$ of A is defined if $A[I, J]$ nonsingular as

$$|A|_{i,j} := A[\{i\}, \{j\}] - A[I, \{j\}]A[I, J]^{-1}A[\{i\}, J] \in F.$$

Analogous to the usual determinant, A is nonsingular if and only if at least one quasideterminant of A is defined and nonzero [12, Proposition 1.4.6]. When A is nonsingular, $|A|_{i,j}$ is defined if and only if the (i,j)th entry a of A^{-1} is nonzero, and if this is the case, $|A|_{i,j} = a^{-1}$ holds.

Through the Dieudonné determinant, we can compute the degree of a quasideterminant of rational function matrices without computing the quasideterminant.

▶ **Proposition 13.** *Let $A \in F(s)^{n \times n}$ be a square rational function matrix over a skew field F. Then for $i, j \in [n]$ with $S := A[[n] \setminus \{i\}, [n] \setminus \{j\}]$ being nonsingular, it holds*

$$\deg |A|_{i,j} = \deg \operatorname{Det} A - \deg \operatorname{Det} S.$$

Proof. We assume $i = j = 1$ without loss of generality. Express A as

$$A = \begin{pmatrix} a & r \\ c & S \end{pmatrix},$$

where $a \in F(s), r \in F(s)^{1 \times (n-1)}$ and $c \in F(s)^{(n-1) \times 1}$. By elementary row and column operations, it holds

$$A = \begin{pmatrix} a & r \\ c & S \end{pmatrix} = \begin{pmatrix} 1 & rS^{-1} \\ 0 & I_{n-1} \end{pmatrix} \begin{pmatrix} |A|_{i,j} & 0 \\ 0 & S \end{pmatrix} \begin{pmatrix} 1 & 0 \\ S^{-1}c & I_{n-1} \end{pmatrix},$$

where we used $|A|_{i,j} = a - rS^{-1}c$. Hence $\deg \operatorname{Det} A = \deg |A|_{i,j} + \deg \operatorname{Det} S$, as required. ◀

Proposition 13 can be applied to the problem of computing the degree of a *noncommutative rational function* (nc-rational function) expressed by a *noncommutative formula* (nc-formula). Let K be a field and consider pairwise noncommutative symbols $x_1, \ldots, x_m$. An *nc-rational function* is an element of the free skew field $K\langle\!(x_1, \ldots, x_m)\!\rangle$. An *nc-formula* (with division) Φ is a binary tree, whose every leaf is labeled with an element of $\{x_1, \ldots, x_m\} \cup K$ and every non-leaf node is labeled with "+", "×" or "÷". Each node computes an nc-rational function in the obvious way, and the output of Φ is the rational function computed by the root. The *size* of Φ is the number of nodes.

Cohn [5] showed that for an nc-formula of size r computing f, we can construct a nonsingular $n \times n$ nc-linear matrix B with $n = \operatorname{poly}(r)$ such that the top-left entry of B^{-1} is f. In addition, the top-left entry of B^{-1} is nonzero if and only if the submatrix of B

without the first row and column is nonsingular as mentioned above. Therefore, as indicated by Hrubeš–Wigderson [19], the problem of checking if an nc-formula represents zero can be reduced to nc-Edmonds' problem.

We can consider a weighted analog of this reduction. Unlike the commutative case, an nc-rational function $f \in K\{\!\!\{x_1, \ldots, x_m\}\!\!\}$ cannot always be expressed as the ratio of two noncommutative polynomials. Nevertheless, we can define the (total) degree of f as the degree (in s) of the rational function $g \in K\{\!\!\{x_1, \ldots, x_m\}\!\!\}(s)$ obtained by replacing each x_i with $x_i s$. Then given an nc-formula computing f, we construct an nc-linear polynomial matrix A such that $|A|_{1,1} = f^{-1}$ and reduce the degree computation of f to nc-WEP using Proposition 13. By Theorem 2, we have:

▶ **Theorem 14.** *We can deterministically compute the degree of the nc-rational function represented by an nc-formula of size r over a field K in* poly(r) *arithmetic operations on K. If $K = \mathbb{Q}$, the bit-lengths of intermediate numbers are polynomially bounded.*

References

1 S. A. Abramov and M. A. Barkatou. On solution spaces of products of linear differential or difference operators. *ACM Communications in Computer Algebra*, 48(4):155–165, 2014. doi:10.1145/2733693.2733719.

2 S. A. Amitsur. Rational identities and applications to algebra and geometry. *Journal of Algebra*, 3(3):304–359, 1966.

3 E. Artin. *Geometric Algebra.* Interscience Publishers, Inc., New York, NY, 1957. doi:10.1002/9781118164518.

4 B. Beckermann, H. Cheng, and G. Labahn. Fraction-free row reduction of matrices of Ore polynomials. *Journal of Symbolic Computation*, 41(5):513–543, 2006. doi:10.1016/j.jsc.2005.10.002.

5 P. M. Cohn. The embedding of firs in skew fields. *Proceedings of the London Mathematical Society*, s3-23(2):193–213, 1971. doi:10.1112/plms/s3-23.2.193.

6 P. M. Cohn. *Free Rings and Their Relations*, volume 19 of *London Mathematical Society Monograph.* Academic Press, London, 2nd edition, 1985.

7 P. M. Cohn. *Algebra*, volume 3. John Wiley & Sons, Chichester, 2nd edition, 1991.

8 P. M. Cohn. *Skew Fields. Theory of General Division Rings.* Cambridge University Press, Cambridge, 1995.

9 J. Dieudonné. Les déterminants sur un corps non commutatif. *Bulletin de la Société Mathématique de France*, 71:27–45, 1943.

10 J. Edmonds. Systems of distinct representatives and linear algebra. *Journal of Research of the National Bureau of Standards*, 71B(4):241–245, 1967. doi:10.6028/jres.071B.033.

11 A. Garg, L. Gurvits, R. Oliveira, and A. Wigderson. Operator scaling: theory and applications. *Foundations of Computational Mathematics*, 20(2):223–290, 2020. doi:10.1007/s10208-019-09417-z.

12 I. Gelfand, S. Gelfand, V. Retakh, and R. Lee Wilson. Quasideterminants. *Advances in Mathematics*, 193(1):56–141, 2005. doi:10.1016/j.aim.2004.03.018.

13 I. M. Gel'fand and V. S. Retakh. Determinants of matrices over noncommutative rings. *Functional Analysis and Its Applications*, 25(2):91–102, 1991. doi:10.1007/BF01079588.

14 M. Giesbrecht and M. S. Kim. Computing the Hermite form of a matrix of Ore polynomials. *Journal of Algebra*, 376:341–362, 2013. doi:10.1016/j.jalgebra.2012.11.033.

15 K. R. Goodearl and R. B. Warfield, Jr. *An Introduction to Noncommutative Noetherian.* Cambridge University Press, Cambridge, 2nd edition, 2004.

16 L. Gurvits. Classical complexity and quantum entanglement. *Journal of Computer and System Sciences*, 69(3):448–484, 2004.

17 M. Hamada and H. Hirai. Maximum vanishing subspace problem, CAT(0)-space relaxation, and block-triangularization of partitioned matrix, 2017. arXiv:1705.02060.

18 H. Hirai. Computing the degree of determinants via discrete convex optimization on Euclidean buildings. *SIAM Journal on Applied Geometry and Algebra*, 3(3):523–557, 2019.

19 P. Hrubeš and A. Wigderson. Non-commutative arithmetic circuits with division. *Theory of Computing*, 11(14):357–393, 2015. doi:10.4086/toc.2015.v011a014.

20 G. Ivanyos, Y. Qiao, and K. V. Subrahmanyam. Constructive non-commutative rank computation is in deterministic polynomial time. *Computational Complexity*, 27(4):561–593, 2018.

21 S. Iwata and R. Shimizu. Combinatorial analysis of generic matrix pencils. *SIAM Journal on Matrix Analysis and Applications*, 29(1):245–259, 2007.

22 V. Kabanets and R. Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. *Computational Complexity*, 13(1–2):1–46, 2004.

23 M. Khochtali, J. Rosenkilde né Nielsen, and A. Storjohann. Popov form computation for matrices of Ore polynomials. In *Proceedings of the 42nd International Symposium on Symbolic and Algebraic Computation (ISSAC '17)*, pages 253–260, New York, NY, 2017. ACM Press. doi:10.1145/3087604.3087650.

24 L. Kronecker. Grundzüge einer arithmetischen Theorie der algebraischen Grössen. *Journal für die reine und angewandte Mathematik*, 92:1–122, 1882.

25 T. Y. Lam. *Lectures on Modules and Rings*, volume 189 of *Graduate Texts in Mathematics*. Springer, New York, NY, 1999.

26 V. Levandovskyy and K. Schindelar. Computing diagonal form and Jacobson normal form of a matrix using Gröbner bases. *Journal of Symbolic Computation*, 46(5):595–608, 2011. doi:10.1016/j.jsc.2010.10.009.

27 L. Lovász. Singular spaces of matrices and their application in combinatorics. *Boletim da Sociedade Brasileira de Matemática*, 20(1):87–99, 1989.

28 S. Moriyama and K. Murota. Discrete Legendre duality in polynomial matrices (in Japanese). *The Japan Society for Industrial and Applied Mathematics*, 23(2):183–202, 2013.

29 K. Murota. Combinatorial relaxation algorithm for the maximum degree of subdeterminants: Computing Smith-McMillan form at infinity and structural indices in Kronecker form. *Applicable Algebra in Engineering, Communication and Computing*, 6(4-5):251–273, 1995.

30 K. Murota. Computing the degree of determinants via combinatorial relaxation. *SIAM Journal on Computing*, 24(4):765–796, 1995. doi:10.1137/S0097539791201897.

31 K. Murota. *Matrices and Matroids for Systems Analysis.* Springer, Berlin Heidelberg, 2000.

32 K. Murota. *Discrete Convex Analysis.* SIAM, Philadelphia, 2003.

33 K. Murota. Legendre duality in combinatorial study of matrix pencils. *Japan Journal of Industrial and Applied Mathematics*, 29(2):205–236, 2012.

34 J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM*, 27(4):701–717, 1980. doi:10.1145/322217.322225.

35 L. Taelman. Dieudonné determinants for skew polynomial rings. *Journal of Algebra and Its Applications*, 5(1):89–93, 2006.

36 L. G. Valiant. Completeness classes in algebra. In *Proceedings of the 11th Annual ACM Symposium on Theory of Computing (STOC '79)*, pages 249–261, New York, NY, 1979. ACM Press. doi:10.1145/800135.804419.

37 G. C. Verghese and T. Kailath. Rational matrix structure. *IEEE Transactions on Automatic Control*, 26(2):434–439, 1981.

A General Stabilization Bound for Influence Propagation in Graphs

Pál András Papp
ETH Zürich, Switzerland
apapp@ethz.ch

Roger Wattenhofer
ETH Zürich, Switzerland
wattenhofer@ethz.ch

Abstract

We study the stabilization time of a wide class of processes on graphs, in which each node can only switch its state if it is motivated to do so by at least a $\frac{1+\lambda}{2}$ fraction of its neighbors, for some $0 < \lambda < 1$. Two examples of such processes are well-studied dynamically changing colorings in graphs: in majority processes, nodes switch to the most frequent color in their neighborhood, while in minority processes, nodes switch to the least frequent color in their neighborhood. We describe a non-elementary function $f(\lambda)$, and we show that in the sequential model, the worst-case stabilization time of these processes can completely be characterized by $f(\lambda)$. More precisely, we prove that for any $\epsilon > 0$, $O(n^{1+f(\lambda)+\epsilon})$ is an upper bound on the stabilization time of any proportional majority/minority process, and we also show that there are graph constructions where stabilization indeed takes $\Omega(n^{1+f(\lambda)-\epsilon})$ steps.

2012 ACM Subject Classification Mathematics of computing → Graph coloring; Theory of computation → Distributed computing models; Theory of computation → Self-organization

Keywords and phrases Minority process, Majority process

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.90

Category Track A: Algorithms, Complexity and Games

Related Version The full version of the paper is available at `https://arxiv.org/abs/2004.09185`.

1 Introduction

Many natural phenomena can be modeled by graph processes, where each node of the graph is in a state (represented by a color), and each node can change its state based on the states of its neighbors. Such processes have been studied since the dawn of computer science, by, e.g., von Neumann, Ulam, and Conway. Among the numerous applications of these graph processes, the most eminent ones today are possibly neural networks, both biological and artificial.

Two fundamental graph processes are majority and minority processes. In a *majority process*, each node wants to switch to the most frequent color in its neighborhood. Such a process is a straightforward model of influence spreading in networks, and as such, it has various applications in social science, political science, economics, and many more [29, 9, 12, 18, 23].

In contrast, in a *minority process*, each node wants to switch to the least frequent color in its neighborhood. Minority processes are used to model scenarios where the nodes are motivated to anti-coordinate with each other, like frequency selection in wireless communication, or differentiating from rival companies in economics [24, 6, 7, 11, 8].

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 90; pp. 90:1–90:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Majority and minority processes have been studied in several different models, the most popular being the synchronous model (where in each step, all nodes can switch simultaneously) and the sequential model (where in each step, exactly one node switches). Since in many application areas, it is unrealistic to assume that nodes switch at the exact same time, we focus on the sequential model in this paper. We are interested in the worst-case stabilization time of such processes, i.e. the maximal number of steps until no node wants to change its color anymore.

Our main parameter describes how easily nodes will switch their color. Previously, the processes have mostly been studied under the basic switching rule, when nodes are willing switch their color for any small improvement. However, it is often more reasonable to assume a *proportional switching rule*, i.e. that nodes only switch their color if they are motivated by at least, say, 70% of their neighbors to do so. In general, we describe such proportional processes by a parameter $\lambda \in (0, 1)$, and say that a node is switchable if it is in conflict with a $\frac{1+\lambda}{2}$ portion of its neighborhood. The stabilization time in such proportional processes (possibly as a function of λ) has so far remained unresolved.

The reason we can analyze proportional majority and minority processes together is that both can be viewed as a special case of a more general process of propagating conflicts through a network, where the cost of relaying conflicts through a node is proportional to the degree of the node. This more general process could also be used to model the propagation of information, energy, or some other entity through a network. This suggests that our results might also be useful for gaining insights into different processes in a wide range of other application areas, e.g. the behavior of neural networks.

In the paper, we provide a tight characterization of the maximal possible stabilization time of proportional majority and minority processes. We show that for maximal stabilization, a critical parameter is the portion φ of the neighborhood that nodes use as "outputs", i.e. neighbors they propagate conflicts to. Based on this, we prove that the stabilization time of proportional processes follows a transition between quadratic and linear time, described by the non-elementary function

$$f(\lambda) := \max_{\varphi \in (0, \frac{1-\lambda}{2}]} \frac{\log\left(\frac{1-\varphi}{\lambda+\varphi}\right)}{\log\left(\frac{1-\varphi}{\varphi}\right)}. \tag{1}$$

More specifically, for any $\epsilon > 0$, we show that on the one hand, $O(n^{1+f(\lambda)+\epsilon})$ is an upper bound on the number of steps of any majority/minority process, and on the other hand, there indeed exists a graph construction where the processes last for $\Omega(n^{1+f(\lambda)-\epsilon})$ steps.

2 Related Work

Various aspects of both majority and minority processes on two colors have been studied extensively. This includes basic properties of the processes [17, 36], sets of critical nodes that dominate the process [12, 15, 20], complexity and approximability results [21, 3, 10], threshold behavior in random graphs [14, 26], and the analysis of stable states in the process [16, 33, 4, 5, 34, 24]. Modified process variants have also been studied [35, 25], with numerous generalizations aiming to provide a more realistic model for social networks [2, 1].

However, the question of stabilization time in the processes has almost exclusively been studied for the basic switching rule (defined in Section 3.2). Even for the basic rule, apart from a straightforward $O(n^2)$ upper bound, the question has remained open for a long time in case of both processes. It has recently been shown in [13] and [27] that both processes can exhibit almost-quadratic stabilization time in case of basic switching, both in the sequential adversarial and in the synchronous model. On the other hand, the maximal stabilization time under proportional switching has remained open so far.

It has also been shown that if the order of nodes is chosen by a benevolent player, then the behavior of the two processes differs significantly, with the worst-case stabilization time being $O(n)$ for majority processes [13] and almost-quadratic for minority processes [27]. In weighted graphs, where the only available upper bound on stabilization time is exponential, it has been shown that both majority and minority can indeed last for an exponential number of steps in various models [22, 28]. The result of [28] is the only one to also study the proportional switching rule, showing that the exponential lower bound also holds in this case; however, since the paper studies weighted graphs with arbitrarily high weights, this model differs significantly from our unweighted setting.

Stabilization time has also been examined in several special cases, mostly assuming the synchronous model. The stabilization of a slightly different minority process variant (based on closed neighborhoods) has been studied in special classes of graphs including grids, trees and cycles [30, 31, 32]. The work of [19] describes slightly modified versions of minority processes which may take $O(n^5)$ or $O(n^6)$ steps to stabilize, but provide better local minima (stable states) upon termination. For majority processes, stabilization has mostly been studied from a random initial coloring, on special classes of graphs such as grids, tori and expanders [14, 26].

Various aspects of majority processes have also been studied under the proportional switching rule, including sets of critical nodes that dominate the process, and sets of nodes that always preserve a specific color [38, 37]. However, to our knowledge, the stabilization time of the processes with proportional switching has not been studied before.

3 Model and Notation

3.1 Preliminaries

We define our processes on simple, unweighted, undirected graphs $G(V, E)$, with V denoting the set of nodes and E the set of edges. We denote the number of nodes by $n = |V|$. The neighborhood of v is denoted by $N(v)$, the degree of v by $\deg(v) = |N(v)|$.

We also use simple directed graphs in our proofs. A directed graph is called a DAG if it contains no directed cycles. A *dipartitioning* of a DAG is a disjoint partitioning (V_1, V_2) of V such that each source node is in V_1, and all edges between V_1 and V_2 all go from V_1 to V_2. We refer to the set of edges from V_1 to V_2 as a *dicut*.

Given an undirected graph G with edge set E, we also define the *directed edge set* of G as $\widehat{E} = \{(u, v), (v, u) \,|\, (u, v) \in E\}$, i.e. the set of directed edges obtained by taking each edge with both possible orientations.

A *coloring* is a function $\gamma : V \to \{\text{black}, \text{white}\}$. A *state* is a current coloring of G. Under a given coloring, we define $N_s(v) = \{u \in N(v) | \gamma(v) = \gamma(u)\}$ and $N_o(v) = \{u \in N(v) | \gamma(v) \neq \gamma(u)\}$ as the same-color and opposite-color neighborhood of v, respectively.

We say that there is a *conflict* on edge (u, v), or that (u, v) is a *conflicting edge*, if $u \in N_o(v)$ in case of a majority process, and if $u \in N_s(v)$ in case of a minority process. In general, we denote the conflict neighborhood by $N_c(v)$, meaning $N_c(v) = N_o(v)$ and $N_c(v) = N_s(v)$ in case of majority and minority processes, respectively. We occasionally also use $N_{\neg c}(v) = N(v) \setminus N_c(v)$.

If a node v has more conflicts than a predefined threshold (depending on the so-called *switching rule* in the model, discussed later) in the current state, then v is *switchable*. Switching v changes its color to the opposite color. If edge (u, v) becomes (ceases to be) a conflicting edge when node v switches, then we say that v has *created* this conflict (*removed* this conflict, respectively).

A *majority/minority process* is a sequence of steps (states), where each state is obtained from the previous state by a set of switchable nodes switching. In this paper, we examine sequential processes, when in each step, exactly one node switches. Such a process is *stable* when there are no more switchable nodes in the graph. By *stabilization time*, we mean the number of steps until a stable state is reached.

3.2 Model and switching rule

We study the worst-case stabilization time of majority/minority processes, that is, the maximal number of steps achievable on any graph, from any initial coloring. In other words, we assume the *sequential adversarial model*, when the order of nodes (i.e., the next switchable node to switch in each time step) is chosen by an adversary who maximizes stabilization time.

It only remains to specify the condition that allows a node to switch its color. The most straightforward switching rule is the following:

▶ **Rule I** (Basic Switching). *Node v is switchable if* $|N_c(v)| - |N_{\neg c}(v)| > 0$.

An equivalent form of this rule is $|N_c(v)| > \frac{1}{2} \cdot \deg(v)$. This rule is shown to allow up to $\widetilde{\Theta}(n^2)$ stabilization time for both majority [13] and minority [27] processes. However, it is often more realistic to assume a proportional switching rule, based on a real parameter $\lambda \in (0, 1)$:

▶ **Rule II** (Proportional Switching). *Node v is switchable if* $|N_c(v)| - |N_{\neg c}(v)| \geq \lambda \cdot deg(v)$.

Since we have $|N_c(v)| + |N_{\neg c}(v)| = \deg(v)$, this is equivalent to saying that v is switchable exactly if $|N_c(v)| \geq \frac{1+\lambda}{2} \cdot \deg(v)$. In the limit when λ is infinitely small (or, equivalently, as $\frac{1+\lambda}{2}$ approaches $\frac{1}{2}$ from above), we obtain Rule I as a special case of Rule II.

In case of Rule I, whenever a node v switches, it is possible that the total number of conflicts in the graph decreases by 1 only. On the other hand, Rule II implies that the switching of v decreases the total number of conflicts at least by $\lambda \cdot \deg(v)$ (we say that v *wastes* these conflicts), so in case of Rule II, the total number of conflicts can decrease more rapidly, allowing only a smaller stabilization time. Our findings show that the maximal number of steps is different for every distinct λ.

3.3 On the $f(\lambda)$ function

While the processes have a symmetric definition on each edge by default, it turns out that in order to maximize stabilization time, each edge has to be used in an asymmetric way. The most important parameter at each node v is the ratio of neighbors v uses as "inputs" and as "outputs". That is, the optimal behavior for each node v is to select $\varphi \cdot \deg(v)$ of its neighbors as outputs (for some $\varphi \in (0, 1)$), and create all new conflicts on the edges leading to these output nodes, and similarly, mark the remaining $(1 - \varphi) \cdot \deg(v)$ neighbors as inputs, and only remove conflicts from the edges coming from these input nodes. Note that with Rule II, whenever a node switches, it can create at most $\left(1 - \frac{1+\lambda}{2}\right) \cdot \deg(v) = \frac{1-\lambda}{2} \cdot \deg(v)$ new conflicts, so it is reasonable to assume $\varphi \in \left(0, \frac{1-\lambda}{2}\right]$.

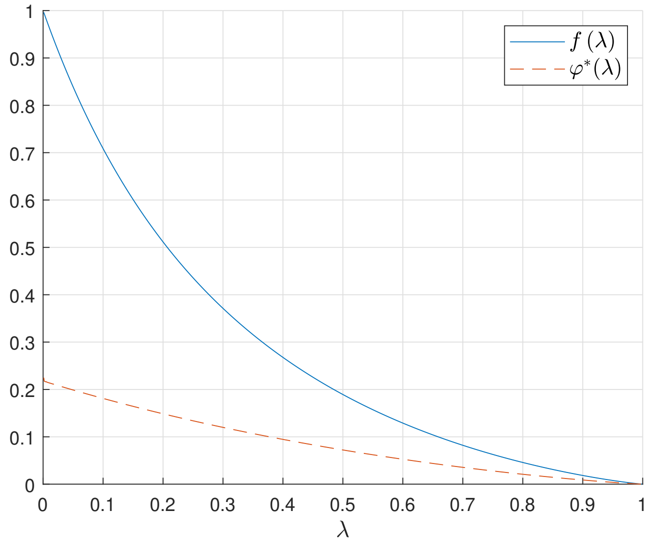


Figure 1 Plot of $f(\lambda)$ and $\varphi^*(\lambda)$ for $\lambda \in (0,1)$.

Our results show that if all nodes select φ as their output rate, then the maximal achievable stabilization time is a function of

$$\frac{\log\left(\frac{1-\varphi}{\lambda+\varphi}\right)}{\log\left(\frac{1-\varphi}{\varphi}\right)}. \tag{2}$$

As such, the largest stabilization time can be achieved by maximizing this expression by selecting the optimal φ value, as shown in the definition of f in Equation 1. We denote the optimal value of φ (i.e., the argmax of Equation 2) by φ^*. The function f has no straightforward closed form, as such a form would require solving

$$(\lambda+1)\cdot\varphi\cdot\log\left(\frac{1-\varphi}{\varphi}\right) = (\lambda+\varphi)\log\left(\frac{1-\varphi}{\lambda+\varphi}\right),$$

for φ, with λ as a parameter. We discuss f in more detail in the full version of the paper.

Figure 1 shows the values of f and φ^* as a function of λ. The figure shows that both $f(\lambda)$ and $\varphi^*(\lambda)$ are continuous, monotonically decreasing and convex.

It is visible that $\lim_{\lambda\to 0} f(\lambda) = 1$ and $\lim_{\lambda\to 1} f(\lambda) = 0$. This is in line with what we would expect: the simple switching rule allows a stabilization time up to $\widetilde{\Theta}(n^2)$ [13, 27], while even for any large $\lambda < 1$, it is still straightforward to present a graph with $\Omega(n)$ stabilization time. Our main result is showing that $f(\lambda)$ describes the continuous transition between these two extremes.

4 General intuition behind the proofs

Note that initially, each node v can have at most $\deg(v)$ conflicts on its incident edges, and each time when v switches, it wastes $\lambda \cdot \deg(v)$ conflicts. Therefore, if each node were to "use" its own initial conflicts only, then each node could switch at most $\frac{1}{\lambda}$ times, and stabilization time could never go above $O(n)$.

Instead, the idea is to take the high number of conflicts initially available at high-degree nodes, and use these conflicts to switch the less wasteful low-degree nodes many times. Specifically, we could have a set of $\Theta(n)$-degree nodes that initially have $\Omega(n^2)$ conflicts altogether on their incident edges, and somehow relay these conflicts to another set of $O(1)$-degree nodes, which only waste $O(1)$ conflicts at each switching. However, due to the large difference both in degree and in the number of switches, it is not possible to connect these two sets directly; instead, we need to do this through a range of intermediate levels, which exhibit decreasing degree and increasingly more switches. In order to maximize stabilization time, our main task is to move conflicts through these levels as efficiently (i.e., wasting as few conflicts in the process) as possible.

The formula of $f(\lambda)$ describes the efficiency of this process. The rate of inputs to outputs $\frac{1-\varphi}{\varphi}$ determines the factor by which the degree decreases at every new level. If φ is chosen small, then $\frac{1-\varphi}{\varphi}$ is high, so we only have a few levels until we reach constant degree, and hence the number of switches is increased only a few times. On the other hand, the increase in the number of switches per level is expressed by $\frac{1-\varphi}{\lambda+\varphi}$, which is a decreasing function of φ. If φ is too large, then although we execute this increase more times, each of these increases is significantly smaller.

With a degree decrease rate of $\frac{1-\varphi}{\varphi}$, we can altogether have about $\log_{\frac{1-\varphi}{\varphi}}(n)$ levels until the degree decreases from $\Theta(n)$ to $\Theta(1)$. If we increase the number of switches by a factor of $\frac{1-\varphi}{\lambda+\varphi}$ each time, then the $O(1)$-degree nodes will exhibit

$$\left(\frac{1-\varphi}{\lambda+\varphi}\right)^{\log_{\frac{1-\varphi}{\varphi}}(n)} = n^{\frac{\log\left(\frac{1-\varphi}{\lambda+\varphi}\right)}{\log\left(\frac{1-\varphi}{\varphi}\right)}} \leq n^{f(\lambda)} \tag{3}$$

switches, with an equation only if $\varphi = \varphi^*(\lambda)$. Having $\widetilde{\Theta}(n)$ nodes in the last level, this sums up to about $n^{1+f(\lambda)}$ switches altogether.

4.1 Conflict propagation systems

The upper bound on stabilization time is easiest to present in a general form that only focuses on this flow of conflicts in the graph. We define a simpler representation of the processes which only keeps a few necessary concepts to describe the flow of conflicts, and ignores e.g. the color of nodes or the timing of the switches at each node. In fact, we only require the number of times $s(v)$ each $v \in V$ switches, and the number $c(u, v)$ of conflicts that were created by node u and then removed by node v, for each $(u, v) \in \widehat{E}$.

For simplicity, given a function $c : \widehat{E} \to \mathbb{N}$, let us introduce the notation $c_{in}(v) := \sum_{u \in N(v)} c(u, v)$ and $c_{out}(v) := \sum_{u \in N(v)} c(v, u)$.

▶ **Definition 1** (Conflict Propagation System, CPS). *Given an undirected graph G, a conflict propagation system is an assignment $s : V \to \mathbb{N}$ and $c : \widehat{E} \to \mathbb{N}$ such that*

1. *for each $v \in V$, we have $c_{in}(v) + deg(v) \geq \lambda \cdot deg(v) \cdot s(v) + c_{out}(v)$,*
2. *for each $v \in V$, we have $c_{out}(v) \leq \frac{1-\lambda}{2} \cdot deg(v) \cdot s(v)$, and*
3. *for each $(u, v) \in \widehat{E}$, we have $c(u, v) \leq s(u)$.*

With the choice of $s(v)$ and $c(u, v)$ described above, any proportional majority or minority process indeed satisfies these properties, and thus provides a CPS. Hence if we upper bound the stabilization time (i.e. the total number of switches $\sum_{v \in V} s(v)$) of any CPS, this establishes the same bound on the stabilization time of any majority/minority process.

Condition 1 is the most complex of the three; it expresses the amount of "input conflicts" $c_{in}(v)$ required to switch v an $s(v)$ times altogether. Every time after v switches, it has at most $\frac{1-\lambda}{2} \cdot \deg(v)$ conflicts on the incident edges, so it needs to acquire $\lambda \cdot \deg(v)$ new conflicts to reach the threshold of $\frac{1+\lambda}{2} \cdot \deg(v)$ and be switchable again; this results in the

$\lambda \cdot \deg(v) \cdot s(v)$ term. Moreover, if in the meantime, the neighboring nodes remove some of the conflicts from the incident edges (expressed by $c_{out}(v)$), then this also has to be compensated for by extra input conflicts. Finally, the extra $\deg(v)$ term comes from the (at most) $\deg(v)$ conflicts that are already on the incident edges in the initial coloring. For a detailed discussion of this condition, see the full version of the paper.

Condition 2 also holds, since each time when v switches, it creates at most $\frac{1-\lambda}{2} \cdot \deg(v)$ conflicts on the incident edges. Each time u switches, it can only create one conflict on a specific edge, so condition 3 also follows. Hence any majority/minority process indeed provides a CPS.

Finally, we need a technical step to get rid of the extra $\deg(v)$ term in condition 1. Note that this term becomes asymptotically irrelevant as $s(v)$ grows; hence, our approach is to handle fewer-switching nodes separately, and require condition 1 only for nodes with large $s(v)$. More formally, we select a constant s_0, and we refer to nodes v with $s(v) < s_0$ as *base nodes*. We then consider *Relaxed CPSs*, where, given this extra parameter s_0, condition 1 is replaced by:

1R. *for each $v \in V$ with $s(v) \geq s_0$, we have $c_{in}(v) \geq \lambda \cdot deg(v) \cdot s(v) + c_{out}(v)$,*

This relaxation comes at the cost of an extra ϵ additive term in the exponent of our upper bound.

5 Upper bound proof

We now outline the proof of the upper bound on the number of switches. A more detailed discussion of this proof is available in the full version of the paper.

5.1 Properties of an optimal construction

We start by noting that since moving a conflict through a node is wasteful, it is suboptimal to have two neighboring nodes that both transfer a conflict to each other, or more generally, to move a conflict along any directed cycle. Therefore, in a CPS with maximal stabilization time, the conflicts are essentially moved along the edges of a DAG. To formalize this, given a CPS, let us say that a directed edge $(u, v) \in \widehat{E}$ is a *real edge* if $c(u, v) > 0$.

▶ **Lemma 2.** *There exists a CPS with maximal stabilization time where the real edges form a DAG.*

Proof. Among the CPSs on n nodes with maximal stabilization time, let us take the CPS P where the sum $\sum_{e \in \widehat{E}} c(e)$ is minimal. Assume that there is a directed cycle along the real edges of this CPS, and let $c(e_0)$ denote the minimal value of function c along this cycle.

Now consider the CPS P' where the value of c on each edge of this directed cycle is decreased by $c(e_0)$. Since in each affected node, the inputs and outputs have been decreased by the same value, P' still satisfies all three conditions, and thus it is also a valid CPS. Moreover, P' has the same amount of total switches as P. However, since $c(e_0) > 0$, the sum of $c(e)$ values in P' is less than in P, which contradicts the minimality of P. ◀

Hence for the upper bound proof, we can assume that the real edges of the CPS form a DAG. In the rest of the section, we focus on this DAG composed of the real edges of the CPS. We first show that for convenience, we can also assume that each base node is a source in this DAG.

▶ **Lemma 3.** *There exists a CPS with maximal stabilization time where each base node is a source node of the DAG.*

Proof. Note that by removing an input edge (u, v) of a base node v (that is, setting $c(u, v)$ to 0), the remaining CPS is still valid, since node v does not have to satisfy condition 1R, and in node u, only the sum of outputs was decreased. Therefore, we can remove all the input edges of each base node, and hence base nodes will all become source nodes of the DAG. ◀

▶ **Lemma 4.** *For each directed edge (u, v) in the DAG where u is a source node, $c(u, v) = O(1)$. More specifically, $c(u, v) \leq s_0$.*

Proof. If u is a base node, then $s(u) \leq s_0$, so $c(u, v) \leq s_0$ due to condition 3. Otherwise, condition 1R must hold, and since u has no input nodes, we get $0 \geq c_{out}(u) + \lambda \cdot \deg(u) \cdot s(u)$, hence $c_{out}(u) = 0$, so $c(u, v) = 0$ for every v. Thus $c(u, v) \leq s_0$. ◀

5.2 Edge potential

As a main ingredient of the proof, we define a way to measure how close we are to propagating conflicts optimally.

▶ **Definition 5** (Potential). *Given a real edge $e \in \widehat{E}$, the potential of e is defined as $P(e) = c(e)^{1/f(\lambda)}$.*

For simplicity of notation, we also use P to denote the function $x \to x^{1/f(\lambda)}$ on real numbers instead of edges.

Intuitively speaking, the potential function describes the cost of sending a specific number of conflicts through a single edge, in terms of the number of initial conflicts used up for this. Note that since $f(\lambda) < 1$, the function P is always convex. This shows that sending a high number of conflicts through a single edge is more costly than sending the same amount of conflicts through multiple edges.

As the following lemma shows, the potential is defined in such a way that the total potential can never increase when passing through a node in the DAG; the best that a node can do is to preserve the input potential if it relays conflicts optimally.

▶ **Lemma 6.** *For any non-source node v of the DAG, with input edges from $N_{in}(v)$ and output edges to $N_{out}(v)$, we have*

$$\sum_{u \in N_{in}(v)} P(u, v) \geq \sum_{u \in N_{out}(v)} P(v, u).$$

Proof. If v is not a source, then by Lemma 3 it is not a base node, and thus has to satisfy condition 1R. In our DAG, c_{in} and c_{out} correspond to $\sum_{u \in N_{in}(v)} c(u, v)$ and $\sum_{u \in N_{out}(v)} c(v, u)$, respectively. Assume that we fix the value of c_{in} and c_{out}. Since the potential function P is convex, the incoming potential (left side) is minimized if c_{in} is split as equally among the input neighbors as possible. On the other hand, the outgoing potential (right side) is maximized if c_{out} is split as unequally among outputs as possible, so all output edges present in the DAG have the maximal possible number of switches, meaning $c(v, u) = s(v)$ for every $u \in N_{out}(v)$.

Assume that a fraction φ of v's incident edges are outgoing, i.e. $|N_{out}(v)| = \varphi \cdot \deg(v)$ and $|N_{in}(v)| = (1 - \varphi) \cdot \deg(v)$. By condition 1R, we have $c_{in} \geq \lambda \cdot \deg(v) \cdot s(v) + c_{out}$; with $c_{out} = \varphi \cdot \deg(v) \cdot s(v)$, this gives $c_{in} \geq (\lambda + \varphi) \cdot \deg(v) \cdot s(v)$. If split evenly among the $(1 - \varphi) \cdot \deg(v)$ inputs, this means

$$\frac{c_{in}}{|N_{in}(v)|} \geq \frac{(\lambda + \varphi) \cdot \deg(v) \cdot s(v)}{(1 - \varphi) \cdot \deg(v)} = \left(\frac{\lambda + \varphi}{1 - \varphi}\right) \cdot s(v)$$

switches for each input node. The inequality on the potential then comes down to

$$\sum_{u \in N_{in}(v)} P(u,v) \geq (1-\varphi) \cdot \deg(v) \cdot \left(\frac{\lambda+\varphi}{1-\varphi} \cdot s(v) \right)^{1/f(\lambda)} \geq$$
$$\geq \varphi \cdot \deg(v) \cdot s(v)^{1/f(\lambda)} \geq \sum_{u \in N_{out}(v)} P(v,u).$$

To show that the inequality in the middle holds, we only require

$$\left(\frac{\lambda+\varphi}{1-\varphi} \right)^{1/f(\lambda)} \geq \frac{\varphi}{1-\varphi},$$

or, put otherwise,

$$\frac{1}{f(\lambda)} \log \left(\frac{\lambda+\varphi}{1-\varphi} \right) \geq \log \left(\frac{\varphi}{1-\varphi} \right).$$

Since $\frac{\varphi}{1-\varphi} < 1$ (thus its logarithm is negative), we get

$$\frac{\log\left(\frac{\lambda+\varphi}{1-\varphi}\right)}{\log\left(\frac{\varphi}{1-\varphi}\right)} = \frac{\log\left(\frac{1-\varphi}{\lambda+\varphi}\right)}{\log\left(\frac{1-\varphi}{\varphi}\right)} \leq f(\lambda).$$

This holds by the definition of $f(\lambda)$. Note that this also shows that equality can only be achieved if the output rate φ is indeed chosen as the argmax value $\varphi^*(\lambda)$. ◀

Lemma 6 provides the key insight to the main idea of our proof: if we process the nodes of a DAG according to a topological ordering, always maintaining a dicut of outgoing edges from the already processed part of the DAG, then this potential cannot ever increase when adding a new node.

▶ **Lemma 7.** *Given a dicut S of a dipartitioning in the DAG, we have*

$$\sum_{e \in S} P(e) = O(n^2).$$

Proof (Sketch). Each dipartitioning can be obtained by starting from the trivial dipartitioning where V_1 only contains the source nodes of the DAG, and then iteratively adding nodes one by one to this initial V_1. The number of outgoing edges from this initial V_1 (the set of source nodes) is upper bounded by $|E| = O(n^2)$. According to Lemma 4, the number of switches (and hence the potential) on each edge of the dicut is at most constant, so the sum of potential in this initial dicut is also $O(n^2)$.

Now consider the process of iteratively adding nodes to this initial V_1 to obtain a specific dipartitioning. Whenever we add a new node v to V_1, the incoming edges of v are removed from the dicut, and the outgoing edges of v are added to the dicut. According to Lemma 6, the potential on the outgoing edges of v is at most as much as the potential on the incoming edges, so the sum of potential can not increase in any of these steps. Therefore, when arriving at the final V_1, the sum of potential on the cut edges is still at most $O(n^2)$. ◀

5.3 Upper bounding switches

Finally, we present our main lemma that uses the previous upper bound on potential in order to upper bound the number of switches in the CPS.

▶ **Lemma 8.** *Given a CPS and an integer $a \in \{1, ..., n\}$, let $A = \{v \in V \mid a \leq deg(v) < 2a\}$. For the total number of switches $s(A) = \sum_{v \in A} s(v)$, we have*

$$s(A) = O\left(n^{1+f(\lambda)} \cdot a^{-f(\lambda)}\right).$$

Proof (Sketch). If the input edges of the nodes in A would form the dicut of a dipartitioning, then we could directly use Lemma 7 to upper bound the number of switches in A through the potential of the input edges. However, the nodes of A might be scattered arbitrarily in the DAG, and if there is a directed path from one node in A to another, then the "same" potential might be used to switch more than one node in A. Thus we cannot apply Lemma 7 directly. Instead, our proof consists of two parts.

1. First, we define so-called responsibilities for the nodes in A. Given a node $v_0 \in A$, the idea is to devise two different functions: (i) a function $\Delta c(e)$, defined on each edge e which is contained in any directed path starting from v_0, and (ii) a function $\Delta s(v)$, which is defined on any node v that is reachable from v_0 on a directed path. Intuitively, we will consider the conflicts $\Delta c(e)$ and the switches $\Delta s(v)$ to be those that are indirectly "the effects of the switches of v_0". More specifically, Δc and Δs are chosen such that if they are removed (subtracted from the CPS), then v_0 has no output edges in the DAG anymore, and the resulting assignment $s'(v) = s(v) - \Delta s(v)$ and $c'(e) = c(e) - \Delta c(e)$ still remains a valid CPS. Hence the subtraction results in a CPS where v_0 has no directed path to other nodes in A anymore. This shows that we can keep on executing this step for each $v_0 \in A$ until no two nodes in A are connected by a directed path, at which point we can apply Lemma 7 to the resulting graph.

Whenever we process such a node $v_0 \in A$, we define the *responsibility* of v_0 as $R(v_0) := s(v_0) + \sum \Delta s(v)$, where the sum is understood over all the nodes $v \in A$ that are reachable from v_0. The main idea is that we "reassign" these switches to v_0 from other nodes in A. This method is essentially a redistribution of switches in the CPS, so we have $\sum_{v \in A} s(v) = \sum_{v \in A} R(v)$ altogether.

Furthermore, our definition of $\Delta s(v)$ will ensure that $R(v_0) = O(1) \cdot s(v_0)$. Intuitively, this can be explained as follows. Recall that with Rule II, the ratio of output to input conflicts is always upper bounded by a constant factor (below 1) at every node, since switching always wastes a specific proportion of conflicts. Hence, over any path starting from v_0, the number of outputs that can be attributed to v_0 forms a geometric series. As the ratio of the geometric series is below 1, the total amount of conflicts caused by v_0 this way is still within the magnitude of the input conflicts of v_0. Since each node in A has similar degree (and thus requires similar number of input conflicts for one switching), these conflicts can only switch nodes in A approximately the same number of times as v_0 can be switched by its own inputs. A detailed discussion of this responsibility technique is available in the full version of the paper.

2. For the second part of the proof, we show the claim in this modified CPS with no directed path between nodes in A. This implies that there exists a dipartitioning where the nodes of A are in V_2, but all their input nodes are in V_1. This means that all the input edges of each node in A are included in the dicut S of the partitioning.

Consider a node $v \in A$. Due to condition 1R, v has at least $\lambda \cdot \deg(v) \cdot s(v)$ input conflicts. Even if these are distributed equally on all incident edges of v (this is the case that amounts to the lowest total potential, since P is convex), this requires a total input potential of

$$\deg(v) \cdot P(\lambda \cdot s(v)) = \deg(v) \cdot s(v)^{1/f(\lambda)} \cdot \lambda^{1/f(\lambda)}$$

at least. Recall that Lemma 7 shows that the total potential on all edges in S is $O(n^2)$. Our task is hence to find an upper bound on $\sum_{v \in A} s(v)$, subject to

$$\sum_{v \in A} \deg(v) \cdot s(v)^{1/f(\lambda)} \cdot \lambda^{1/f(\lambda)} = O(n^2).$$

Since the last factor on the left side is a constant, we can simply remove it and include it in the $O(n^2)$ term. Furthermore, the degree of each node in A is at least a, so by lower bounding each degree by a, we get

$$\sum_{v \in A} s(v)^{1/f(\lambda)} = O(n^2) \cdot \frac{1}{a}.$$

Given this upper bound on $\sum_{v \in A} P(s(v))$, since the function P is convex, the sum of switches $\sum_{v \in A} s(v)$ is maximal when each node in A switches the same amount of times (i.e. there is an s such that $s(v) = s$ for every $v \in A$), giving

$$|A| \cdot s^{1/f(\lambda)} = O(n^2) \cdot \frac{1}{a}.$$

With this upper bound, $|A| \cdot s$ is maximal if $|A|$ is as large as possible and s as small as possible (again because P grows faster than linearly). Clearly $|A| \leq n$, so assuming $|A| = n$, we get

$$s^{1/f(\lambda)} = O(n) \cdot \frac{1}{a},$$

which means that

$$s = O(n^{f(\lambda)}) \cdot a^{-f(\lambda)},$$

and thus for the total number of switches in A, we get

$$|A| \cdot s = O(n^{1+f(\lambda)}) \cdot a^{-f(\lambda)}. \quad ◀$$

It only remains to sum up this bound for the appropriate intervals to obtain our final bound. Let us consider the intervals $[1, 2)$, $[2, 4)$, $[4, 8)$, ..., i.e. $a = 2^k$ for each factor of 2 up to n, which is a disjoint partitioning of the possible degrees. Note that for these specific values of a, the sum $\sum_{k=0}^{\infty} (2^k)^{-f(\lambda)}$ converges to a constant according to the ratio test. In other words, the sum is dominated by the number of switches of the lowest (constant) degree nodes, and hence, the total number of switches in the graph can be upper bounded by $O(1) \cdot n^{1+f(\lambda)}$.

Recall that since we work with Relaxed CPSs, we lose an ϵ in the exponent of this upper bound when we carry the result over to an original CPS.

▶ **Theorem 9.** *In any CPS with parameter λ, we have $\sum_{v \in V} s(v) = O(n^{1+f(\lambda)+\epsilon})$ for any $\epsilon > 0$.*

Since we have established that every majority/minority process provides a CPS, the upper bound on their stabilization time also follows.

▶ **Corollary 10.** *Under Rule II with any $\lambda \in (0, 1)$, every majority/minority process stabilizes in time $O(n^{1+f(\lambda)+\epsilon})$ for any $\epsilon > 0$.*

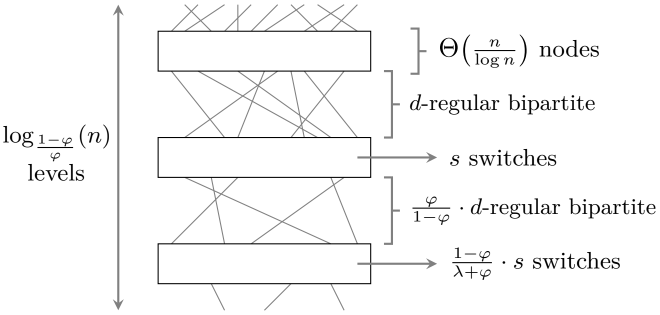

$\log_{\frac{1-\varphi}{\varphi}}(n)$ levels
$\Theta\left(\frac{n}{\log n}\right)$ nodes
d-regular bipartite
s switches
$\frac{\varphi}{1-\varphi} \cdot d$-regular bipartite
$\frac{1-\varphi}{\lambda+\varphi} \cdot s$ switches

Figure 2 Consecutive levels of the lower bound construction.

6 Lower bound construction

Having established the most efficient way to relay conflicts, the high-level design of the matching lower bound construction is rather straightforward, following the level-based idea described in Section 4.

Given λ, we first determine the optimal output rate $\varphi = \varphi^*(\lambda)$. We then create a construction consisting of distinct levels, where each level has the same size, and each consists of a set of nodes that have the same degree. Since the degree should decrease by a factor of $\frac{\varphi}{1-\varphi}$ in each new level from top to bottom, we can add $L = \log_{\frac{1-\varphi}{\varphi}}(n)$ such levels to the graph. If each of these level has $\Theta(\frac{n}{\log n})$ nodes, then with the appropriate choice of constants, the total number of nodes is below n.

Each node in the construction is only connected to other nodes on the levels immediately above or below its own. All conflicts are propagated down in the graph, from upper to lower levels, so the upper neighbors of a node are always used as inputs, while the lower neighbors are always used as outputs. For the optimal propagation of conflicts, each node v must have the optimal input-output rate, i.e. an up-degree of $(1-\varphi) \cdot \deg(v)$ and a down-degree of $\varphi \cdot \deg(v)$. Thus each consecutive level pair forms a regular bipartite graph, with $\frac{\varphi}{1-\varphi}$ of the degree of the level pair above. The construction is illustrated in Figure 2.

Our parameters λ and φ also determine that the number of switches should increase by a factor $\frac{1-\varphi}{\lambda+\varphi}$ on each new level. If we can always increase the switches at this rate, then each node on the lowermost level will switch

$$\left(\frac{1-\varphi}{\lambda+\varphi}\right)^{\log_{\frac{1-\varphi}{\varphi}}(n)} = n^{\frac{\log\left(\frac{1-\varphi}{\lambda+\varphi}\right)}{\log\left(\frac{1-\varphi}{\varphi}\right)}} = n^{f(\lambda)},$$

times, where the last equation holds because we are using $\varphi = \varphi^*(\lambda)$. Since there are $\widetilde{\Theta}(n)$ nodes on the lowermost level, the switches in this level already amount to a total of $\widetilde{\Theta}(n^{1+f(\lambda)})$, matching the upper bound.

However, note that when $\varphi^*(\lambda)$ or $\frac{1-\varphi}{\lambda+\varphi}$ is irrational, we can only use close enough rational approximations of these values. This comes at the cost of losing a small ϵ in the exponent.

▶ **Theorem 11.** *Under Rule II with a wide range of λ values, there is a graph construction and initial coloring where majority/minority processes stabilize in time $\Omega(n^{1+f(\lambda)-\epsilon})$ for any $\epsilon > 0$.*

This level-based structure describes the general idea behind our lower bound construction. However, the main challenge of the construction is in fact designing the connection between subsequent levels. In particular, this connection has to make sure that conflicts are indeed always relayed optimally, i.e. no potential is wasted between any two levels.

Recall from the proof of Lemma 6 that this is only possible if between any two consecutive switches of a node v, it is exactly a $\frac{\lambda+\varphi}{1-\varphi}$ fraction of v's upper neighbors that switch. Moreover, these switching $\frac{\lambda+\varphi}{1-\varphi} \cdot \deg(v)$ upper neighbors always have to be of the right color, i.e. they need to switch to the opposite of v's current color in case of majority processes, and to the same color in case of minority processes. Since the upper neighbors of v are in the same level, we also have to ensure that throughout the entire process, each upper neighbor switches the same number of times altogether.

These conditions impose heavy restrictions on the possible ways to connect two subsequent levels. If the conditions hold for a node v (i.e. the sequence of switches of v's upper neighbors can be split into $\frac{\lambda+\varphi}{1-\varphi} \cdot \deg(v)$-size consecutive appropriate-colored subsets, in an altogether balanced way), then we say that v's upper neighbors follow a valid *control sequence*.

On the other hand, in order to argue about levels in general, we want each level to behave in a similar way. The easiest way to achieve this is to have a one-to-one correspondence between the nodes of different levels, and ensure that each level repeats the same sequence of steps periodically, but in a different pace. That is, we want to connect the levels in such a way that when a level exhibits a specific pattern of switches, then this allows the nodes of the next level to replicate the exact same pattern of switches, but more times.

Thus the key task in our lower bound constructions is to develop a so-called *control gadget*, which is essentially a bipartite graph that fulfills these two requirements: it admits a scheduling of switches such that (i) the upper neighborhood of each lower node follows a valid control sequence, and (ii) while the upper level executes a sequence s times, the lower level executes the same sequence $\frac{1-\varphi}{\lambda+\varphi} \cdot s$ times. Given such a control gadget, we can connect the subsequent level pairs of our construction using this gadgets. This allows us to indeed increase the number of switches by a $\frac{1-\varphi}{\lambda+\varphi}$ factor in each new level, resulting in a total of $\widetilde{\Theta}(n^{1+f(\lambda)})$ switches as described above.

However, developing a control gadget is a difficult combinatorial task in general: it depends on many factors including divisibility questions, and whether our parameters can be expressed as a fraction of small integers. A detailed discussion of control gadget design and the λ values covered by Theorem 11 is available in the full version of the paper. In particular, we present a method which allows us to develop a control gadget for every small λ value below a threshold of approximately 0.476 (more specifically, as long as $\frac{\lambda+\varphi}{1-\varphi} \leq \frac{3}{5}$). The same technique also provides a control gadget for some larger λ values above the threshold, but only when the corresponding switch increase ratio $\frac{1-\varphi}{\lambda+\varphi}$ can be expressed as a fraction of relatively small integers. Furthermore, the full version also describes a simpler solution technique to the control gadget problem; this leaves a slightly larger gap to the upper bound, but it works for any λ without much difficulty.

References

1 Victor Amelkin, Francesco Bullo, and Ambuj K Singh. Polar opinion dynamics in social networks. *IEEE Transactions on Automatic Control*, 62(11):5650–5665, 2017.

2 Vincenzo Auletta, Ioannis Caragiannis, Diodato Ferraioli, Clemente Galdi, and Giuseppe Persiano. Generalized discrete preference games. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI'16, page 53–59. AAAI Press, 2016.

3 Cristina Bazgan, Zsolt Tuza, and Daniel Vanderpooten. Complexity and approximation of satisfactory partition problems. In *International Computing and Combinatorics Conference*, pages 829–838. Springer, 2005.

4 Cristina Bazgan, Zsolt Tuza, and Daniel Vanderpooten. The satisfactory partition problem. *Discrete applied mathematics*, 154(8):1236–1245, 2006.

5 Cristina Bazgan, Zsolt Tuza, and Daniel Vanderpooten. Satisfactory graph partition, variants, and generalizations. *European Journal of Operational Research*, 206(2):271–280, 2010.

6 Olivier Bodini, Thomas Fernique, and Damien Regnault. Crystallization by stochastic flips. In *Journal of Physics: Conference Series*, volume 226, page 012022. IOP Publishing, 2010.

7 Olivier Bodini, Thomas Fernique, and Damien Regnault. Stochastic flips on two-letter words. In *2010 Proceedings of the Seventh Workshop on Analytic Algorithmics and Combinatorics (ANALCO)*, pages 48–55. SIAM, 2010.

8 Zhigang Cao and Xiaoguang Yang. The fashion game: Network extension of matching pennies. *Theoretical Computer Science*, 540:169–181, 2014.

9 Luca Cardelli and Attila Csikász-Nagy. The cell cycle switch computes approximate majority. *Scientific reports*, 2:656, 2012.

10 Ning Chen. On the approximability of influence in social networks. *SIAM Journal on Discrete Mathematics*, 23(3):1400–1415, 2009.

11 Jacques Demongeot, Julio Aracena, Florence Thuderoz, Thierry-Pascal Baum, and Olivier Cohen. Genetic regulation networks: circuits, regulons and attractors. *Comptes Rendus Biologies*, 326(2):171–188, 2003.

12 MohammadAmin Fazli, Mohammad Ghodsi, Jafar Habibi, Pooya Jalaly, Vahab Mirrokni, and Sina Sadeghian. On non-progressive spread of influence through social networks. *Theoretical Computer Science*, 550:36–50, 2014.

13 Silvio Frischknecht, Barbara Keller, and Roger Wattenhofer. Convergence in (social) influence networks. In *International Symposium on Distributed Computing*, pages 433–446. Springer, 2013.

14 Bernd Gärtner and Ahad N Zehmakan. Color war: Cellular automata with majority-rule. In *International Conference on Language and Automata Theory and Applications*, pages 393–404. Springer, 2017.

15 Bernd Gärtner and Ahad N Zehmakan. Majority model on random regular graphs. In *Latin American Symposium on Theoretical Informatics*, pages 572–583. Springer, 2018.

16 Michael U Gerber and Daniel Kobler. Algorithmic approach to the satisfactory graph partitioning problem. *European Journal of Operational Research*, 125(2):283–291, 2000.

17 Eric Goles and Jorge Olivos. Periodic behaviour of generalized threshold functions. *Discrete Mathematics*, 30(2):187–189, 1980.

18 Mark Granovetter. Threshold models of collective behavior. *American Journal of Sociology*, 83(6):1420–1443, 1978.

19 Sandra M Hedetniemi, Stephen T Hedetniemi, KE Kennedy, and Alice A Mcrae. Self-stabilizing algorithms for unfriendly partitions into two disjoint dominating sets. *Parallel Processing Letters*, 23(01):1350001, 2013.

20 Clemens Jeger and Ahad N Zehmakan. Dynamic monopolies in reversible bootstrap percolation. *arXiv preprint arXiv:1805.07392*, 2018.

21 Dominik Kaaser, Frederik Mallmann-Trenn, and Emanuele Natale. On the voting time of the deterministic majority process. In *41st International Symposium on Mathematical Foundations of Computer Science (MFCS 2016)*, 2016.

22 Barbara Keller, David Peleg, and Roger Wattenhofer. How even tiny influence can have a big impact! In *International Conference on Fun with Algorithms*, pages 252–263. Springer, 2014.

23 David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 137–146. ACM, 2003.

24 Jeremy Kun, Brian Powers, and Lev Reyzin. Anti-coordination games and stable graph colorings. In *International Symposium on Algorithmic Game Theory*, pages 122–133. Springer, 2013.

25 Yuezhou Lv and Thomas Moscibroda. Local information in influence networks. In *International Symposium on Distributed Computing*, pages 292–308. Springer, 2015.

26 Ahad N Zehmakan. Opinion forming in erdös-rényi random graph and expanders. In *29th International Symposium on Algorithms and Computations*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik GmbH, Wadern/Saarbruecken . . . , 2018.

27 Pál András Papp and Roger Wattenhofer. Stabilization Time in Minority Processes. In *30th International Symposium on Algorithms and Computation (ISAAC 2019)*, volume 149 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 43:1–43:19, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

28 Pál András Papp and Roger Wattenhofer. Stabilization Time in Weighted Minority Processes. In *36th International Symposium on Theoretical Aspects of Computer Science (STACS 2019)*, volume 126 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 54:1–54:15, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

29 David Peleg. Local majorities, coalitions and monopolies in graphs: a review. *Theoretical Computer Science*, 282(2):231–257, 2002.

30 Damien Regnault, Nicolas Schabanel, and Éric Thierry. Progresses in the analysis of stochastic 2d cellular automata: A study of asynchronous 2d minority. In Luděk Kučera and Antonín Kučera, editors, *Mathematical Foundations of Computer Science 2007*, pages 320–332. Springer Berlin Heidelberg, 2007.

31 Damien Regnault, Nicolas Schabanel, and Éric Thierry. On the analysis of "simple" 2d stochastic cellular automata. In *International Conference on Language and Automata Theory and Applications*, pages 452–463. Springer, 2008.

32 Jean-Baptiste Rouquier, Damien Regnault, and Éric Thierry. Stochastic minority on graphs. *Theoretical Computer Science*, 412(30):3947–3963, 2011.

33 Khurram H Shafique and Ronald D Dutton. On satisfactory partitioning of graphs. *Congressus Numerantium*, pages 183–194, 2002.

34 Saharon Shelah and Eric C Milner. Graphs with no unfriendly partitions. *A tribute to Paul Erdös*, pages 373–384, 1990.

35 Ariel Webster, Bruce Kapron, and Valerie King. Stability of certainty and opinion in influence networks. In *Advances in Social Networks Analysis and Mining (ASONAM), 2016 IEEE/ACM International Conference on*, pages 1309–1320. IEEE, 2016.

36 Peter Winkler. Puzzled: Delightful graph theory. *Commun. ACM*, 51(8):104–104, August 2008.

37 Ahad N Zehmakan. Target set in threshold models. *Acta Mathematica Universitatis Comenianae*, 88(3), 2019.

38 Ahad N Zehmakan. Tight bounds on the minimum size of a dynamic monopoly. In *International Conference on Language and Automata Theory and Applications*, pages 381–393. Springer, 2019.

Network-Aware Strategies in Financial Systems

Pál András Papp
ETH Zürich, Switzerland
apapp@ethz.ch

Roger Wattenhofer
ETH Zürich, Switzerland
wattenhofer@ethz.ch

Abstract

We study the incentives of banks in a financial network, where the network consists of debt contracts and credit default swaps (CDSs) between banks. One of the most important questions in such a system is the problem of deciding which of the banks are in default, and how much of their liabilities these banks can pay. We study the payoff and preferences of the banks in the different solutions to this problem. We also introduce a more refined model which allows assigning priorities to payment obligations; this provides a more expressive and realistic model of real-life financial systems, while it always ensures the existence of a solution.

The main focus of the paper is an analysis of the actions that a single bank can execute in a financial system in order to influence the outcome to its advantage. We show that removing an incoming debt, or donating funds to another bank can result in a single new solution that is strictly more favorable to the acting bank. We also show that increasing the bank's external funds or modifying the priorities of outgoing payments cannot introduce a more favorable new solution into the system, but may allow the bank to remove some unfavorable solutions, or to increase its recovery rate. Finally, we show how the actions of two banks in a simple financial system can result in classical game theoretic situations like the prisoner's dilemma or the dollar auction, demonstrating the wide expressive capability of the financial system model.

2012 ACM Subject Classification Theory of computation → Network games; Applied computing → Economics; Theory of computation → Algorithmic mechanism design

Keywords and phrases Financial network, credit default swap, creditor priority, clearing problem, prisoner's dilemma, dollar auction

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.91

Category Track A: Algorithms, Complexity and Games

1 Introduction

The world's financial system is a complex network where financial institutions such as banks are connected via various kinds of financial contracts. If some financial institutions go bankrupt, then others might suffer as well; the financial network might experience a ripple effect. Two of the most common financial contracts are (i) debt contracts (some bank owes a specific amount of money to another bank) and (ii) Credit Default Swaps (CDSs). A CDS is a simple financial derivative where the payment obligation depends on the defaulting of another bank in the system. The combination of debt contracts and CDSs turns out to provide a simple and yet expressive model, which is able to capture a wide range of interesting phenomena in real-life financial markets [10, 22, 18, 17].

Given a set of banks and a set of payment obligations between these banks, one of the most natural questions is to decide which of the banks can fulfill these obligations, and which of them cannot, and hence are in default. The problem of deciding what portion of obligations banks can fulfill is known as the *clearing problem.* One can easily encounter

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 91; pp. 91:1–91:17
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

a situation when this problem has multiple different solutions in a financial system. It is natural to study how much the individual banks prefer these solutions, i.e. what is their payoff in specific solutions of the system.

In this paper we study the problem from the point of view of a single bank v. We analyze whether some simple actions of v can improve its situation in the network. In a financial system, the complex interconnection between the banks can easily result in situations where banks can achieve a better outcome in surprising and somewhat counterintuitive ways. For example, being on the receiving end of a debt contract is generally considered beneficial, because the bank obtains payment from this contract. However, in a system with debts and CDSs, it is also possible that if a bank v nullifies a debt contract as a creditor, then (through a number of intermediate steps in the network) this results in an even higher total payoff for v. Such phenomena are crucial to understand, since if banks indeed execute these actions to obtain a better outcome, then these opportunities will determine how the financial system changes and evolves in the future.

We begin with a description of the financial system model recently developed by Schuldenzucker *et. al.* [22], which serves as the base model for our findings. We then introduce a more refined version of this model which also assigns *priorities* to each contract, and assumes that banks have to fulfill their payment obligations in the order defined by these priorities. We show that besides being more expressive and realistic, this augmented model still ensures the existence of a solution.

Our main contribution is an analysis of various different actions that banks in the system can execute in order to increase their final payoff when the system is cleared. We first show that by removing an incoming debt (partially or entirely) or by donating extra funds to another bank, a bank might be able to increase its payoff. We then show that investing more external assets or reprioritizing its outgoing payments can also allow a bank to influence the system. However, these actions do not allow a bank to introduce more favorable new solutions, but they can allow the bank to remove unfavorable solutions from the system, or increase its own recovery rate.

Finally, we present some simple examples where two banks try to influence the financial system simultaneously, resulting in situations that are identical to the classical prisoner's dilemma or dollar auction game. This suggests that financial systems in this model can exhibit very rich behavior, and if two or more banks execute these actions simultaneously, this can easily lead to complex game-theoretic settings.

2 Related Work

Numerous studies on the properties of financial systems are directly or indirectly based on the financial system model introduced by Eisenberg and Noe in [11]. This model only assumes simple debt contracts between banks. Different studies have later also extended this model with default costs [21], cross-ownership relations [25, 12] or so-called covered CDSs [17]. The related literature has studied the propagation of shocks in many different variants of these models [2, 8, 5, 4, 1, 13].

One disadvantage of these models is that they can only describe *long positions* of banks on each other, meaning that a worse situation for one bank is always worse (or the same) for any other bank. For example, if a bank is unable to pay its debt, then its creditor receives less money, and it might not be able to pay its debts either. This already enables the model to capture many interesting phenomena, e.g. how a small shock causes a ripple effect in the network. However, long positions imply that there is a solution in these systems which is simultaneously the best for all banks. As such, the models cannot represent the opposing interests of banks in many real-world situations, and thus these models are not so interesting from a game-theoretic point of view.

On the other hand, a more realistic model was recently introduced by Schuldenzucker, Seuken and Battiston [22]; we assume this model of financial systems in our paper. Besides debt contracts, this new model also allows credit default swaps between banks, which are essentially financial derivatives where banks are betting on the default of another bank. CDSs are a prominent kind of derivative that played a significant role in the 2008 financial crisis [14]; as such, they have been studied in various works in the financial literature [9, 18, 10]. While the model still remains relatively simple with these two kind of contracts, it now also allows us to model *short positions*, when it is more favorable for a bank if another bank is worse off. This increases the expressive power of the model dramatically, allowing us to capture a wide range of properties of practical financial systems.

The work of Schuldenzucker *et. al.* analyzes their model from a complexity-theoretic perspective. The authors show that in the base variant of this model, each system has at least one solution; however, if we also assume so-called default costs, then some systems might not have a solution at all. In case of default costs, they also describe sufficient conditions for the existence of a solution. Their follow-up work shows that it is computationally hard to decide if a solution exists, and also to find or approximate a solution of the system [23].

However, to our knowledge, the model has not been analyzed from a game-theoretic perspective before. Our paper aims to lay the foundations of such an analysis, by evaluating a variety of simple (and yet realistic) actions that allow nodes to influence the network due to the presence of short positions. Since banks often have conflicting interests in these systems, these actions can easily lead to interesting game-theoretical dilemmas.

The only similar game-theoretic analysis we are aware of is the recent work of Bertschinger *et. al.* [6], set in the original model of Eisenberg and Noe. Instead of having institutional rules for payment obligations in case of default, [6] assumes that banks can freely select the order of paying their outgoing debts, or even decide to make partial payments in some contracts. The paper discusses the properties of Nash-Equilibria and Social Optima in this setting. While this has a connection to our observations in Section 5.3, we analyze the results of such actions in a significantly more complex model with CDSs.

In general, measuring the sensitivity or complexity of a financial network has also been exhaustively studied [15, 3, 5, 4]. The topic also has a major importance for financial authorities in practice, who regularly conduct stress tests to analyze real-world financial systems. The clearing problem, in particular, also plays an important role in the European Central Bank's stress test framework [7], for example.

3 Financial system model

The model introduced by [22] describes a financial network as a set of *banks* (i.e. nodes), denoted by V, with different kinds of financial contracts (i.e. directed edges) between specific pairs of banks. Banks in our examples are usually denoted by u, v or w. Every bank in the system has a predefined amount of *external assets*, denoted by e_v for bank v.

3.1 Debt and CDS contracts

We assume that each contract in the system is between two specific banks u and v. A contract obliges u (the debtor) to pay a specific amount of money to bank v (the creditor), either unconditionally or based on a specific event. The amount of payment obligation in the contract is the *weight* (in financial terms: the notional) of the contract.

While these contracts might be connected to earlier transactions between the banks (e.g. a loan offered by v to u in the past which results in a debt contract from u to v in the present), we assume that these initial payments are implicitly represented in the external assets of banks, and thus the external assets and the contracts together provide all the necessary information to describe the current state of the system.

The outgoing contracts of bank v altogether specify a given amount of total payment obligations for v. If v is unable to fulfill all these obligations, then we say that v is *in default*. In this case, we are interested in the portion of liabilities that v is still able to pay, known as the *recovery rate* of v and denoted by r_v. The definition implies that we always have $r_v \in [0, 1]$, and v is in default exactly if $r_v < 1$. The recovery rates of all banks is represented together in a recovery rate vector $r \in [0, 1]^V$.

The model allows two kinds of contracts between banks in the system. In case of a simple *debt* contract, u has to pay a specific amount to v unconditionally, i.e. in any case. On the other hand, *credit default swaps* (*CDSs*) are ternary financial contracts, made in reference to a third bank w known as the *reference entity*. A CDS describes a conditional debt which only requires u to pay a specific amount to v if w is in default. In particular, if the weight of the CDS is δ and the recovery rate of w is r_w, then the CDS incurs a payment obligation of $\delta \cdot (1 - r_w)$ from u to v.

In practice, CDSs often describe an insurance policy on debt contracts for the creditor bank. If v is the creditor of a debt coming from w, and v suspects that w might go into default and thus will be unable to pay some of its debt, then v can enter into a CDS as a creditor with some other bank u in the system, in reference to w. If w indeed defaults and cannot pay its liabilities to v, then v instead receives some payment from u. Nonetheless, there could be other reasons for banks to enter CDS contracts, e.g. speculative bets about future developments in the market.

3.2 Assets and liabilities

Since payment obligations in CDSs depend on the recovery rate of other banks, the assets and liabilities of a bank are defined as a function of the vector r. The *liability* of u towards v is the sum of payment obligations from all simple debt contracts and CDSs, i.e.

$$l_{u,v}(r) = c_{u,v} + \sum_{w \in V} c_{u,v}^{w} \cdot (1 - r_w),$$

where $c_{u,v}$ denotes the weight of the simple debt from u to v, and $c_{u,v}^{w}$ denotes the weight of the CDS from u to v with reference to w (understood as 0 if the contracts do not exist). The total liabilities of u is then the sum of liabilities to all other banks, i.e.

$$l_u(r) = \sum_{v \in V} l_{u,v}(r).$$

In contrast to this, the actual *payment* from u to v can be lower than $l_{u,v}(r)$ if u is in default. In this case, the model assumes that u makes payments based on the *principle of proportionality*, i.e. it uses all of its assets to make payments to creditors, in proportion to the respective liabilities. In practice, this means that u can pay an r_u portion of each liability, and thus its payment to v is defined as $p_{u,v}(r) = r_u \cdot l_{u,v}(r)$.

On the other hand, the *assets* of v is the sum of its external assets and its incoming payments, i.e.

$$a_v(r) = e_v + \sum_{u \in V} p_{u,v}(r).$$

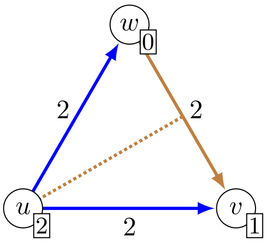


Figure 1 Example financial system with three banks. External assets are shown in rectangles besides the nodes, simple debt contracts are shown as blue arrows from debtor to creditor, and CDSs are shown as brown arrows from debtor to creditor, with a dotted line specifying the reference entity.

Recall that a recovery rate describes the portion of liabilities that a bank can pay. Hence given the assets and liabilities of each bank v, the recovery rate r_v must satisfy $r_v = 1$ if $a_v(r) \geq l_v(r)$, and $r_v = \frac{a_v(r)}{l_v(r)}$ otherwise. A vector r is called a *solution* (in financial terms: a clearing vector) if it describes an equilibrium point for these equalities, i.e. if for each bank v, r_v satisfies this constraint for the assets and liabilities defined by r. Previous work has expressed this by defining the *update function* $f : [0,1]^V \to [0,1]^V$ as

$$f_v(r) = \begin{cases} 1, & \text{if } a_v(r) \geq l_v(r) \\ \frac{a_v(r)}{l_v(r)}, & \text{if } a_v(r) < l_v(r) \end{cases},$$

and defining a solution as a fixed point of the update function.

In order to model the utility function of nodes in the system, we define the *payoff* (in financial terms: equity) of a bank v as the amount of remaining assets after payments if a node is not in default, and 0 otherwise, i.e. $q_v(r) = \max(a_v(r) - l_v(r), 0)$. We assume that the aim of each bank is to maximize its own payoff.

Note that assets, liabilities and payoffs are always defined with regard to a certain recovery rate vector r. However, in order to simplify notation, we do not show r explicitly when it is clear from the context, and instead we simply write e.g. a_v or q_v.

Figure 1 shows an example financial system with three banks u, v and w, with a consistent notation to that of [22, 23]. The system has $e_u = 2$, $e_v = 1$ and $e_w = 0$. There are two debts of weight 2 in the system: one from u to v, the other from u to w. Finally, the system contains a CDS from w to v (also of weight 2), which is in reference to bank u.

Regardless of recovery rates, bank u has liabilities $l_u = 4$ and assets $a_u = 2$, so $r_u = \frac{1}{2}$ in any case. This implies that u can only make payments of $r_u \cdot 2 = 1$ to both v and w. Given $r_u = \frac{1}{2}$, the CDS induces a liability of $2 \cdot (1 - r_u) = 1$ from w to v. Since w receives an incoming payment of $p_{u,w} = 1$ from u, we have $a_w = l_w = 1$, so w can still pay its liability and has a recovery rate of $r_w = 1$. Finally, v has incoming payments $p_{u,v} = 1$ and $p_{w,v} = 1$, external assets $e_w = 1$, and no liabilities. This implies $a_v = 3$ and $l_v = 0$, and thus $r_v = 1$. Hence $(r_u, r_v, r_w) = (\frac{1}{2}, 1, 1)$ is the only solution of the system, providing a payoff of $q_u = 0$, $q_w = 0$ and $q_v = 3$ to the banks.

We also use two sanity assumptions introduced by previous work to exclude degenerate cases [22]. First, we assume that no bank enters into a contract with itself or in reference to itself. Furthermore, since CDSs are regarded as an insurance on debt, we require that if a bank w is a reference entity of some CDS, then w is the debtor of at least one debt contract of positive weight.

4 Payments with priorities

While the principle of proportionality is a simple and natural assumption, financial systems often have more complex payment rules in practice. Thus we also introduce a more general model of *payments with priorities.*

That is, we assume that there is a constant number of priority classes P, and each contract belongs to one of these priority classes. If a node v is in default, then it first spends all its assets to fulfill its liabilities in the highest priority class. If v does not have enough assets to fulfill all such obligations, it spends all its assets on the payments for these edges, proportionally to the amount of liabilities. On the other hand, if v has more assets than highest-priority liabilities, then v pays for all the liabilities in this highest priority level, and continues using the rest of its assets for the lower-priority liabilities in a similar fashion.

More formally, in our modified model, each contract in the network receives another *priority* parameter (besides its weight), which is an integer in $\{1, ..., P\}$. The value 1 denotes the highest priority (i.e. liabilities that have to be paid first), while class P denotes the lowermost priority level.

Given a clearing vector r, for each node v, let $l_v^{(\rho)}$ denote the total amount of liabilities of v due to edges on priority level ρ. Let us also introduce the notation $l_v^{(\leq\rho)} = \sum_{i=1}^{\rho} l_v^{(i)}$. Assume that v has total assets of a_v, and a liability of $l_{v,u}$ on priority level ρ towards another node u. Then the payment of v to u is defined as

$$p_{v,u} = \begin{cases} 0, & \text{if } a_v \leq l_v^{(\leq\rho-1)} \\ \frac{a_v - l_v^{(\leq\rho-1)}}{l_v^{(\rho)}} \cdot l_{v,u}, & \text{if } a_v \in \left(l_v^{(\leq\rho-1)}, l_v^{(\leq\rho)}\right) \\ l_{v,u}, & \text{if } a_v \geq l_v^{(\leq\rho)}. \end{cases}$$

For an example, consider a modified version of the network in Figure 1. Assume we now have 2 priority levels: the debt from u to w is on the higher level, while the other two contracts are on the lower level. For the case of u, this still means $l_u = 4$, $a_u = 2$ and $r_u = \frac{1}{2}$ as before. However, now u uses its 2 units of assets to pay its full liability to w, since this contract has higher priority than the debt to v. Hence $p_{u,v} = 0$ and $p_{u,w} = 2$, resulting in $a_w = 2$. Since $r_u = \frac{1}{2}$ still implies $l_w = 1$ for the CDS, the rest of the payments and recovery rates remain unchanged: we still have $p_{w,v} = 1$ and $r_w = r_v = 1$. However, the payoffs of the banks in the system are now $q_u = 0$, $q_w = 1$ and $q_v = 2$.

The main motivation for introducing payment priorities is that in many cases, it is very close to what happens in real-world financial systems. In many countries, economic laws provide a specific priority list for companies to follow when paying their debts in case of a default. This might start with salaries and other payments to the employees of the company first, then specific kind of debt contracts, and so on.

Another advantage of priorities is that we can often use them to replace so-called *default costs*. Default costs (also studied in [22, 23]) are an extension of the original model, assuming that when banks go into default, they immediately lose a specific portion of their assets. This represents the fact that in practice, once a company goes into default, it has a range of immediate payment obligations (e.g. employees' wages) before it can make payments to other banks. If we instead represent the bank's employees as a separate node in the network, and model this payment obligation with a high-priority edge, then this might allow us to model some of these obligations *without* the use of default costs.

This observation is crucial because the introduction of default costs comes at a significant price: intuitively speaking, default costs introduce a point of discontinuity into the update function, and as a result, some financial systems do not have a solution at all [22]. In contrast to this, without default costs, systems always have at least one solution, as shown by a fixed-point argument in [22]. We point out that the same fixed-point theorem proof also applies in our model with payment priorities: even though the functions $p_{u,v}(r)$ and $a_v(r)$ become significantly more complicated, they are still continuous.

This shows that by introducing priorities, we obtain a model that is significantly more realistic on one hand, but also ensures the existence of a solution at the same time.

▶ **Theorem 1.** *Every financial system with payment priorities has at least one solution.*

Proof (sketch). The proof of this claim is identical to the same proof in the original financial system model, described in the results of [22]. The main idea of the proof is to apply the fixed-point theorem of Kakutani [16], which ensures the existence of a fixed point of the update function f, and thus a solution. This proof can still be applied after the introduction of priorities, since both $a_v(r)$ and $l_v(r)$ still remain a continuous function of r, and so does the update function $f_v(r) = \min(\frac{a_v(r)}{l_v(r)}, 1)$, at least in the domain where $l_v(r) > 0$. The technical part of the proof is slightly more complicated, since one has to consider the $l_v(r) = 0$ case separately. For more details on this proof, we refer the reader to the work of [22]. ◀

5 Influencing the financial system

We now discuss a wide range of actions that a bank can execute in order to obtain a more favorable outcome in the system. Note that except for Section 5.3 which explicitly studies readjusting priorities, all the results also hold in the base model without priorities.

5.1 Removing an incoming debt

One of the most natural actions for a bank v would be to simply cancel a debt contract in which v is a creditor. Since the creditor is considered the beneficiary of a debt, in some financial/legal frameworks, the regulations may indeed allow a bank to nullify an incoming debt contract. However, in case of a financial system with short positions, it is actually possible that in the end, this indirectly increases the payoff of v.

▶ **Theorem 2.** *Removing an incoming debt of v can increase the payoff of v.*

More precisely, our claim is as follows: there exists a financial system S such that (i) S has only one solution r, in which v has payoff q_v and an incoming debt contract, and (ii) in the modified financial system S' obtained by removing this debt, there is again only one solution r', in which the payoff q'_v satisfies $q'_v > q_v$.

Proof. Consider the network in Figure 2. Note that the unlabeled nodes in this system can always pay all their liabilities, so their recovery rate is always 1. Originally, the system has $a_u = 1$ and $l_u = 2$, thus $r_u = \frac{1}{2}$ in any case. This implies $a_w = 2 \cdot (1 - \frac{1}{2}) = 1$, and thus $r_w = 1$. With $r_w = 1$, v obtains no payment from its incoming CDS at all, so the payoff of v in this only solution is $q_v = p_{u,v} = r_u \cdot 1 = \frac{1}{2}$.

One the other hand, consider the system obtained by removing the debt contract from u to v. In this case, $a_u = l_u = 1$, and thus $r_u = 1$. This means that w receives no incoming payments at all, and with $a_w = 0$, we have $r_w = 0$. As a result, v obtains a payment of $2 \cdot (1 - r_w) = 2$ from its incoming CDS, so we have $q_v = 2$. ◀

The proof shows that releasing an outgoing debt increases the recovery rate of u, which indirectly yields an extra payoff for v. Note that v could also achieve this result by donating funds to u, i.e. by increasing e_u by 1. This is even more realistic in a legal framework: the

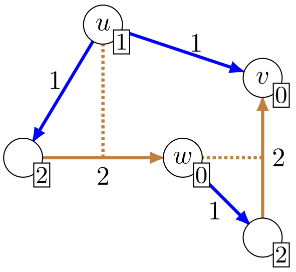


Figure 2 A bank v removing one of its incoming debts.

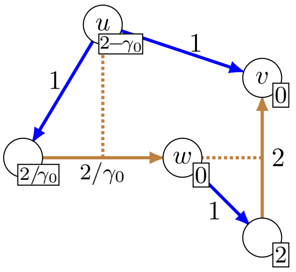


Figure 3 A bank v removing a γ_0 portion of an incoming debt.

owner(s) of bank v can simply donate a specific amount to bank u, who would accept it in hope of avoiding default. Naturally, this is only a favorable step to v if by donating x units of money, it can increase its own payoff by more than x.

▶ **Theorem 3.** *Donating external assets to another node u can be a favorable step.*

More precisely, there is a system S such that (i) S has only one solution r, in which node v has payoff q_v, and (ii) in the system S' obtained by replacing external funds of u by $e'_u := e_u + x$, there is again only one solution r' which satisfies $q'_v > q_v + x$.

The proof of the theorem is identical to that of Theorem 2: if v increases e_u by $x = 1$ in Figure 2, then again $r_u = 1$, which ultimately provides a payoff of $q_v = 3$ (as opposed to the original $\frac{1}{2}$). Note that in general, this action may allow banks to improve their position by affecting a bank that is arbitrarily far in the topology of the network.

While our main focus in the paper is a theoretical analysis of these improvement techniques, we point out that these operations are not only theoretical anomalies in the model; there are known examples when some institutions indeed applied similar techniques in real-world financial networks [19].

We also note that one could prove Theorems 2 and 3 on a smaller example system, where v only has an incoming debt from u and a larger outgoing CDS in reference to u. We have chosen this slightly larger example since it allows us to use a similar proof structure for all our claims in this section.

Finally, if v can increase its payoff by releasing an incoming debt, it is natural to wonder if it is always optimal for v to erase the entire debt, or whether it could be beneficial to only reduce the amount in some cases. We show that reducing a debt to a given portion γ_0 of its original weight can also be an optimal strategy.

▶ **Theorem 4.** *For each constant $\gamma_0 \in [0, 1]$, there is a financial system where bank v achieves its maximal payoff by reducing an incoming debt by a γ_0 portion of its original weight.*

Proof. Consider a modified version of our previous systems, as shown in Figure 3. We show that for any γ_0 parameter, the optimal action of v in this system is to let go of γ_0 portion of the incoming debt from u, i.e. to reduce its weight to $1 - \gamma_0$.

Assume that v reduces the incoming debt by a γ portion for some $\gamma \in [0, 1]$, and let us analyze the final payoff of v as a function of γ. Note that a choice of $\gamma = \gamma_0$ implies that $a_u = l_u$ exactly, and thus $r_u = 1$, $r_w = 0$ and $q_v = (1 - \gamma_0) + 2 = 3 - \gamma_0$ as a result. Hence we have to show that $q_v < 3 - \gamma_0$ in any other case.

First consider the case when $\gamma < \gamma_0$. Since u has $l_u = 1 + (1-\gamma) = 2-\gamma > 2-\gamma_0$, u is in default. Then $r_u = \frac{2-\gamma_0}{2-\gamma}$, and thus w receives an incoming payment of

$$a_w = \frac{2}{\gamma_0} \cdot \left(1 - \frac{2-\gamma_0}{2-\gamma}\right) = \frac{2 \cdot (\gamma_0 - \gamma)}{\gamma_0 \cdot (2-\gamma)}.$$

This is a decreasing function in γ, and it equals 1 exactly for $\gamma = 0$, so $a_w < 1$ for any $\gamma > 0$, and thus w is in default with $r_w = a_w$. Then the amount v receives from the CDS is

$$2 \cdot (1 - r_w) = 2 \cdot \left(1 - \frac{2 \cdot (\gamma_0 - \gamma)}{\gamma_0 \cdot (2-\gamma)}\right) = 2 \cdot \frac{\gamma \cdot (2-\gamma_0)}{\gamma_0 \cdot (2-\gamma)}.$$

Since $q_v = (1-\gamma) \cdot r_u + 2 \cdot (1-r_w)$, we need to show that

$$3 - \gamma_0 > (1-\gamma) \cdot \frac{2-\gamma_0}{2-\gamma} + 2 \cdot \frac{\gamma \cdot (2-\gamma_0)}{\gamma_0 \cdot (2-\gamma)}.$$

After multiplying this by $\gamma_0 \cdot (2-\gamma)$, expanding the brackets and removing terms that cancel out, we are left with $\gamma_0 \cdot (4-\gamma_0) > \gamma \cdot (4-\gamma_0)$, which naturally holds since $\gamma < \gamma_0$.

On the other hand, if $\gamma > \gamma_0$, then $a_u > l_u$, and thus $r_u = 1$. This means $r_w = 0$, so v receives an amount of 2 from the CDS, and has a total payoff of $(1-\gamma) + 2 = 3 - \gamma$, which is again less than $3 - \gamma_0$. Thus selecting $\gamma = \gamma_0$ is indeed the best option for v. ◀

5.2 Investing more external assets

In light of Theorem 3, it is natural to ask if v can also increase its payoff by increasing *its own* external assets. In practice, this could easily happen in multiple ways, e.g. by creating more shares to raise capital for the bank, or by the owners of the bank investing further funds into the bank. If increasing e_v by x would allow v to increase its payoff by more than x in the only solution, then the owners of v would indeed be motivated to invest these extra funds into the bank.

However, somewhat surprisingly, it turns out that this is not possible in the same way as in previous cases: we cannot increase the payoff of v by more than x in the only solution of the system. More specifically, if a vector r' is a solution to the new system and provides a payoff of q'_v, then r' was already a solution of the original system with a payoff of $q'_v - x$.

▶ **Theorem 5.** *Assume that every solution of system S provides a payoff of at most q_v for v. Then setting $e'_v = e_v + x$ cannot introduce a new solution r' with $q'_v > q_v + x$.*

Proof. Assume that such a new solution r' is introduced. Since payoff is always nonnegative, $q_v \geq 0$, and thus $q'_v > x$ in r'. This means that we have $a'_v > x + l'_v$ in r'. Hence, even if e'_v was reduced by x (back to its original value e_v), then v could still pay all of its liabilities; thus the same recovery vector r' and the same payments on each edge also provide a solution in the original system S. The payoff of v in this solution is $q'_v - x$, which is larger than q_v by assumption. This contradicts the fact that q_v was the maximal payoff for v in S. ◀

Naturally, if v is in default, the *recovery rate* of v can indeed be increased in the only solution by injecting extra funds. However, an increase of r_v does not translate to an increase in payoff, so it is a waste for the owners of v to invest resources for this.

On the other hand, while it is not possible to produce a new, more favorable solution for v, it is possible to invalidate solutions that are unfavorable to v. That is, if the original financial system had multiple solutions with different payoffs for v, and v is unsure which

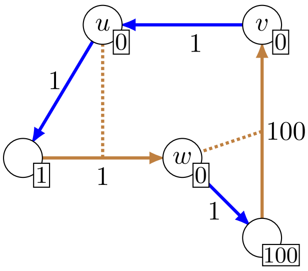


Figure 4 A bank v increasing its own external assets.

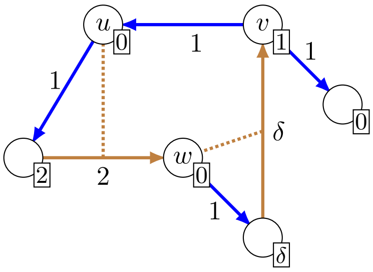


Figure 5 A bank v readjusting the priority of its outgoing contracts.

of these solutions will be implemented by a financial authority, then it is possible that v can inject extra funds to remove a solution where its payoff is much smaller than in other solutions. This may allow v to increase its worst-case payoff, or its payoff in expectation (in case of a randomized choice of solution).

▶ **Theorem 6.** *Given a financial system S with two solutions, it is possible that setting $e'_v = e_v + x$ removes a solution which is unfavorable to v.*

More precisely, there is a system S such that (i) S has two solutions r_1 and r_2, with solution r_2 satisfying $q_v = 0$, and (ii) in the system S' obtained by setting $e'_v := e_v + x$, the only solution is $r' = r_1$, satisfying $q'_v > x$.

Proof. Consider the system in Figure 4, which has two solutions. The design of the system ensures $r_u = r_v$ and $r_w = 1 - r_u$. If $r_v = 1$, then this implies $r_u = 1$ and $r_w = 0$, in which case v has $a_v = 100$, giving a solution with $q_v = 99$. On the other hand, if $r_v < 1$, then it has to satisfy

$$r_v = \frac{100 \cdot (1 - r_w)}{1} = 100 \cdot r_u = 100 \cdot r_v .$$

This is only satisfied if $r_v = 0$, so this is the only other solution, providing $q_v = 0$.

Now assume that v invests $x = 1$ extra funds to have $e_v = 1$. In this case, the system always has $r_v = 1$, hence $r_u = 1$ and $r_w = 0$. This implies that v obtains a payment of 100 in the CDS, resulting in a payoff of $q_v = 100$. Even if we subtract the extra $x = 1$ investment, v has an extra payoff of 99, and thus it has indeed increased its worst-case payoff significantly. ◀

5.3 Readjusting priorities

Assuming payments with priorities as discussed in Section 4, it is also interesting to know if a node can improve its situation by readjusting the priorities of its outgoing edges. That is, in a more flexible regulation framework, banks may be allowed to choose to some extent the order in which they fulfill their payment obligations. However, we show that similarly to the previous case, readjusting the priorities of outgoing edges cannot introduce a better solution.

▶ **Theorem 7.** *Assume that every solution of system S provides a payoff of at most q_v for v. Then redefining v's outgoing priorities cannot introduce a new solution r' with $q'_v > q_v$.*

Proof. Assume that such a new solution r' is introduced. Payoff is nonnegative, so $q_v \geq 0$, and thus $q'_v > 0$. This implies that $a'_v > l'_v$ in r', i.e. v is able to pay all of its liabilities in every outgoing contract. However, in this case, the priorities on the outgoing edges do not matter; hence r' is a solution of S' regardless of how the priorities of outgoing contracts are chosen. In particular, r' is already a solution of the initial system S before the priorities were reorganized, giving the same payoff q'_v in S. This contradicts the fact that q_v was the maximal payoff for v in S. ◀

However, it is again possible that v can increase its recovery rate by readjusting priorities. Recall that in the previous case of increasing the bank's own external assets, we did not explore this possibility, since it required the bank v to invest extra funds while not yielding (the same amount of) extra payoff. However, readjusting priorities is an action that v might be able to execute free of charge. Thus if we define the recovery rate as the secondary objective function of a bank (i.e. even if v is in default and thus has 0 payoff, it is not oblivious to the outcome, and prefers a higher recovery rate), then redefining priorities may allow v to achieve a more preferred outcome without having to invest any extra funds.

▶ **Theorem 8.** *Redefining v's outgoing priorities can increase the recovery rate of v.*

Proof. Consider the system in Figure 5 with a choice of $\delta = \frac{1}{2}$. Originally, each contract is in the same (lower) priority class. Bank v never has enough assets to pay its liabilities, hence u is also in default. In this case, we have $r_u = r_v$ and $r_w = 2 - 2 \cdot r_u$, so v receives $\delta \cdot (1 - r_w) = \delta \cdot (2 \cdot r_v - 1)$ funds from the CDS. This means that

$$r_v = \frac{\delta \cdot (2 \cdot r_v - 1) + 1}{2},$$

which, after reorganization, gives $\delta - 1 = 2 \cdot (\delta - 1) \cdot r_v$, and thus $r_v = \frac{1}{2}$. This is the only solution of the system if $\delta \neq 1$.

Now assume that v is able to raise the debt towards u to the higher priority level. In this new system, v first fulfills its payment obligation to u, which is always possible from its external assets. Hence $r_u = 1$ in this case, implying $r_w = 0$ and thus a payment of $\frac{1}{2}$ to v in the CDS. This implies $r_v = \frac{3}{4}$ in the only solution of the new system. ◀

We again point out that the previous work of Bertschinger *et. al.* [6] discusses a similar phenomenon in debt-only networks, i.e. how redefining the priorities of v's outgoing payments can result in an increased recovery rate for v.

Finally, we show that redefining priorities can allow v to remove an unfavorable solution, and thus increase its worst-case or expected payoff as in the previous subsection.

▶ **Theorem 9.** *Given a financial system S with two solutions, redefining v's outgoing priorities can remove a solution which is unfavorable to v.*

Proof. Consider the system in Figure 5 with a choice of $\delta = 100$. As discussed in the proof of Theorem 8, if $r_v < 1$, then the only solution is $r_v = \frac{1}{2}$. However, the large δ value now allows another solution in the original system: if $r_v = 1$, then $r_u = 1$ and $r_w = 0$, ensuring that v indeed has enough funds to pay its liabilities. The two solutions come with payoffs of $q_v = 0$ and $q_v = 98$, respectively.

Now if v raises its debt towards u to the higher priority level, then $r_u = 1$ is always guaranteed, so $r_w = 0$ and thus v indeed has a payoff of 98 in the only solution. ◀

6 Game-theoretic dilemmas in financial systems

Finally, we briefly show that the attempts of banks to influence the system can also easily lead to situations that can be described by classical game-theoretic settings.

We first show an example where if two nodes simultaneously try to influence the system to their advantage, then the resulting situation is essentially identical to the well-known prisoner's dilemma [20]. We then show that with some modifications to this network, we can also obtain financial systems that represent other well-known two-player-two-strategy games, e.g. the chicken or stag hunt game [20]. Finally, we show a different network design that allows us to model the multiple-round setting of a dollar auction [24].

6.1 Prisoner's dilemma

For an example of the prisoner's dilemma, consider the financial system in Figure 6, where banks v_1 and v_2 want to influence the system to achieve a better outcome. Assume that in the current legal framework, the only step available to these banks is to completely remove their incoming debt contract from u (as in Theorem 2); both banks can decide whether to execute this step or not. Note that canceling a debt increases the recovery rate of u, which indirectly implies a larger payment on the CDS for both v_1 and v_2, and thus can be beneficial for both banks. Applying prisoner's dilemma terminology, we also refer to the step of canceling the debt as *cooperation*, and the step of not canceling the debt as *defection*.

Now let us analyze the payoff of v_1 and v_2 in each strategy profile. Note that $r_w = 1 - r_u$, so the payment on the CDSs for both v_1 and v_2 is $3 \cdot (1 - r_w) = 3 \cdot r_u$ in any case.

If both of the nodes cooperate (i.e. both debts are removed), then u can pay its remaining liabilities, thus $r_u = 1$. This implies a payment of 3 on the CDS, which is the only asset of the acting nodes in this case; hence $q_{v_1} = q_{v_2} = 3$.

If both of the nodes defect (no debt is removed), then we only have $r_u = \frac{1}{3}$, resulting in a payment of 1 from the CDS. However, in this case, both v_1 and v_2 also get a direct payment of $5 \cdot r_u = \frac{5}{3}$ from the defaulting u, which adds up to a total payoff of $\frac{8}{3} = 2.\dot{6}$.

Finally, assume that only one of the nodes cooperate (say, v_1). With only one of the outgoing debts removed, u will have a recovery rate of $r_u = \frac{1}{2}$. This results in a payment of $\frac{3}{2}$ on the CDS for both nodes. However, note that v_2 still has an incoming debt contract from u, and receives a payment of $5 \cdot r_u = \frac{5}{2}$ on this contract. This implies $q_{v_1} = \frac{3}{2} = 1.5$, while $q_{v_2} = 4$ for the strategy profile. The symmetric case yields $q_{v_1} = 4$ and $q_{v_2} = 1.5$.

Since the payoffs are ordered exactly as in a prisoner's dilemma, we obtain an essentially equivalent situation if the two banks are not allowed to coordinate. For both players, defection is always a dominant strategy. E.g. for v_2, defection yields a payoff of 4 (instead of only 3) if v_1 cooperates, and it yields a payoff of $2.\dot{6}$ (instead of only 1.5) if v_1 defects. Thus the Nash-Equilibrium is obtained when both players defect, with $q_{v_1} = q_{v_2} = 2.\dot{6}$. However, both players would be better off with mutual cooperation, which gives $q_{v_1} = q_{v_2} = 3$.

Note that we only assumed for convenience that banks can only remove their entire debt; the behavior is similar if v_1 and v_2 can freely select the portions γ_1 and γ_2 of incoming debt that they keep. In particular, by differentiating the payoff $q_{v_1} = \frac{3+5\cdot\gamma_1}{1+\gamma_1+\gamma_2}$, one can show that defection is indeed the best response of v_1 for any choice of γ_2, and vice versa.

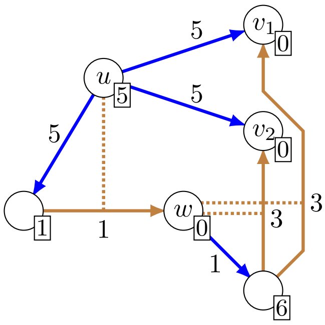


Figure 6 Representation of a prisoner's dilemma in a financial system.

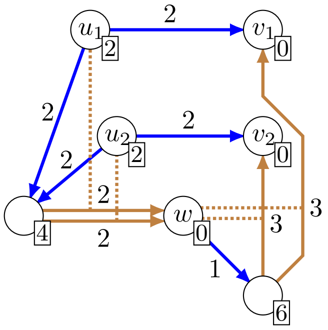


Figure 7 Representation of a stag hunt game in a financial system.

6.2 Stag Hunt

Next, we analyze the financial system in Figure 7, which represents the coordination game known as stag hunt [20]. We again assume that the two acting nodes v_1 and v_2 can only execute the action of completely removing their incoming debt contract from u_1 and u_2, respectively. As before, we refer to the decisions of removing and not removing the debt as cooperation and defection, respectively.

Recall that canceling an incoming debt and donating funds to another bank are very similar operations in some sense. With a slight modification to our system, we could also present the same example game in a setting where the acting banks must decide to donate or not donate a specific amount of funds to a bank. For our example systems, we select the action that allows a simpler presentation.

Let us analyze the payoffs in the different strategy profiles. If both players cooperate, then both u_1 and u_2 will only have a liability of 2, which implies $r_{u_1} = r_{u_2} = 1$. In this case, w receives no payment from either of the CDSs, resulting in $r_w = 0$. This means that both v_1 and v_2 get a payment of 3 from their incoming CDSs. With their debt contracts canceled, we get $q_{v_1} = q_{v_2} = 3$.

If both players defect and keep their debt contract, then both u_1 and u_2 will have a recovery rate of only $\frac{1}{2}$. This implies a payment of 1 to w on both CDSs, so w avoids default with $r_w = 1$. This means that the acting nodes will not receive any payment on the CDS. On their debt contracts, they both receive $\frac{1}{2} \cdot 2$, i.e. $q_{v_1} = q_{v_2} = 1$.

Finally, assume that v_1 cooperates but v_2 defects. In this case, we end up with recovery rates of $r_{u_1} = 1$ and $r_{u_2} = \frac{1}{2}$. Thus w only receives payment on the CDS that is in reference to u_2. However, this payment of $\frac{1}{2} \cdot 2$ is already enough for w to fulfill its liabilities, and hence $r_w = 1$. Again, v_1 and v_2 do not receive any payment on the CDS. However, v_2 still has an incoming debt contract that ensures a payment of $\frac{1}{2} \cdot 2 = 1$, while v_1 has no assets at all. Thus the solution provides $q_{v_1} = 0$ and $q_{v_2} = 1$. In a symmetric manner, the case when v_2 cooperates and v_1 defects incurs $q_{v_1} = 1$, $q_{v_2} = 0$.

Thus the system represents a game where the players are incentivized to coordinate their strategies. Both the case when both banks cooperate and when both banks defect is a pure Nash-Equilibrium, with mutual cooperation being the social optimum. However, if a bank is unsure whether the other bank will cooperate, it might be motivated to defect in order to avoid the risk of getting no payoff at all.

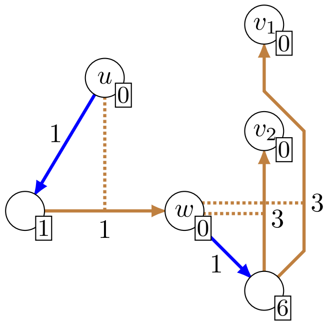


Figure 8 Representation of a chicken game in a financial system.

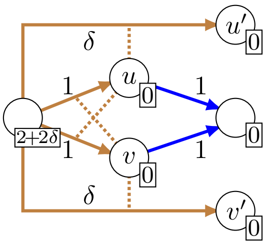


Figure 9 Representation of a dollar auction in a financial system.

6.3 Chicken game

We also provide an example of the chicken game (also known as the hawk-dove game [20]), when the pure Nash-Equilibria are obtained in the asymmetric strategy profiles.

Consider the financial system in Figure 8, and assume the acting banks v_1 and v_2 now have the options to either donate 1 unit or money to another bank, or do not donate money at all. Due to the structure of the network, the nodes are motivated to donate this 1 unit of money to u, since this results in a payment on their incoming CDS contract. We again refer to donating a unit of money to u as cooperation, and not donating as defection.

If both nodes defect, then u still has no assets at all, implying $r_u = 0$. This results in $r_w = 1$, and hence the acting nodes receive no incoming payment, so $q_{v_1} = q_{v_2} = 0$.

If both nodes cooperate, then u has more than enough assets to pay its liabilities, resulting in $r_u = 1$ and $r_w = 0$. This means that both nodes get a payment of 3 in the CDS. After subtracting the amount they have donated, we get $q_{v_1} = q_{v_2} = 2$.

However, to ensure that u does not go into default, it is enough if only one of the two nodes make a donation. I.e. if v_1 cooperates but v_2 defects, then u still has 1 asset, which already implies $r_u = 1$, $r_w = 0$ and a payment of 3 to both v_1 and v_2 on their incoming CDS. After subtracting the donated funds, this gives $q_{v_1} = 2$ and $q_{v_2} = 3$. Similarly, if v_2 cooperates and v_1 defects, we obtain $q_{v_1} = 3$, $q_{v_2} = 2$.

The payoffs show that there is no dominant strategy in the game: if v_1 cooperates, then the best response of v_2 is to defect, while if v_1 defects, then the best response of v_2 is to cooperate. This implies that the two pure Nash-Equilibria are obtained in the strategy profiles when the banks choose the opposite strategies.

Note that we can easily generalize this setting to the case of more than 2 acting nodes, resulting in the so-called volunteer's dilemma. For any k, we can add distinct banks $v_1, v_2, ..., v_k$ that are all connected to the financial network in the same way (through an incoming CDS of weight 3 in reference to w), and all have the same two options of either donating 1 unit of money to u or not acting at all. Note that we also have to ensure that the (currently unlabeled) debtor of the CDSs to these acting nodes has enough resources to make payments on these CDSs in any case, i.e. it must have external assets of at least $3 \cdot k$.

In this case, we obtain a game where again only one volunteer bank v_i is required to make a donation to u, and this already ensures a payoff of 3 for every other bank (and a payoff of 2 for v_i). In this game, the pure Nash-Equilibria are the strategy profiles where exactly one bank cooperates, and the remaining banks all defect.

6.4 Dollar auction

Finally, we show a representation of the dollar auction game [24] in financial systems. We find this example network even more interesting because it builds on a threshold behavior in the financial system model, i.e. that a minor difference in assets can lead to a completely different outcome.

Consider the system in Figure 9, and assume that banks u' and v' want to influence this system by donating extra funds to banks u or v (as in Theorem 3). Note that the payoff of u' and v' depends on the recovery rates of u and v, respectively, which in turn have a recovery rate depending on each other. Due to the design of the system, u' prefers bank u to be in default, and thus it wants to increase e_v; similarly, v' prefers bank v to be in default, so it wants to increase e_u. We assume that 1 unit of money is a very high amount in our context, and thus u' and v' cannot donate enough to ensure that u or v pays its debt entirely from external assets; i.e. we assume that $e_u, e_v < 1$ even after the donation of extra funds.

For a convenient analysis, we assume that there is a small minimum amount ϵ of funds that u' or v' can donate in one step. In our example, we choose a δ value in the magnitude of this ϵ, e.g. $\delta = 6\epsilon$.

Let us now analyze the recovery rates of u and v in the solutions of the system.

- The vector $r_u = r_v = 1$ cannot be a solution, since it would imply no payment on the incoming CDSs, and thus these recovery rates would only be possible if $e_u, e_v \geq 1$.
- If a vector $r_u = 1$, $r_v < 1$ is a solution, then since v receives no incoming payments, we must have $r_v = \frac{e_v}{1} = e_v$. Thus bank u has assets of $e_u + 1 - e_v$, which has to be at least 1 for $r_u = 1$ to hold. Hence this is only a solution if $e_u + 1 - e_v \geq 1$, i.e. $e_u \geq e_v$. In a symmetric manner, $r_v = 1$, $r_u = e_u$ is only a solution if $e_v \geq e_u$.
- If $r_u < 1$, $r_v < 1$ in a solution, then $r_u = e_u + 1 - r_v$ and $r_v = e_v + 1 - r_u$ must hold. This implies $e_u = e_v$, and $r_u + r_v = 1 + e_u$. Hence if $e_u = e_v$, then any r_u, r_v with $r_u + r_v = 1 + e_u$ provides a solution.

Thus as long as $e_u, e_v < 1$, the behavior of the system is as follows:

- If $e_u < e_v$, then the only solution is $r_u = e_u$, $r_v = 1$. This means $q_{u'} = \delta \cdot (1 - e_u)$ and $q_{v'} = 0$.
- If $e_u > e_v$, then the only solution is $r_u = 1$, $r_v = e_v$. This implies $q_{u'} = 0$ and $q_{v'} = \delta \cdot (1 - e_v)$.
- If $e_u = e_v$, then any $r_u, r_v \leq 1$ with $r_u + r_v = 1 + e_u$ is a solution of the system. In the general case, $q_{u'} = \delta \cdot (1 - r_u)$ and $q_{v'} = \delta \cdot (1 - r_v)$.

This describes a setting that is very similar to a dollar auction. In the beginning, with $e_u = e_v = 0$, we have a range of different solutions, and a choice among these depends on a financial authority. One of the banks (say, bank u') decides to donate a small ϵ amount of funds to v; then with $e_v = \epsilon > e_u = 0$, bank u' receives a payment of $\delta \cdot (1 - 0)$ in the only resulting solution. At this point, the payoff of v' is 0; however, at the cost of donating $2 \cdot \epsilon$ funds to u, it could achieve $e_u = 2\epsilon > e_v = \epsilon$, thus resulting in a single solution with a payoff of $q_{v'} = \delta \cdot (1 - \epsilon)$. Since this increases the payoff of v' by $\delta \cdot (1 - \epsilon)$ at the cost of only 2ϵ, this is indeed a rational step for the appropriate δ and ϵ values. However, then u' is again motivated to donate 2ϵ more funds to increase e_v over e_u again, and so on.

Assuming that both u' and v' has at most $\frac{1}{2}$ funds to donate, we always have $e_u, e_v \in [0, \frac{1}{2}]$. This shows that e.g. if we have $e_u > e_v$, then the payoff of bank v' is always within

$$q_{v'} = \delta \cdot (1 - e_v) \; \in \; [\delta/2 \, , \, \delta] = [3\epsilon \, , \, 6\epsilon].$$

Hence in every step, it is indeed rational for v' to donate another 2ϵ funds, since it increases its payoff from 0 to at least 3ϵ. After a couple of rounds, u' and v' will have both donated significantly more money than their payoff of at most 6ϵ. However, the banks are still always tempted to execute the next donation step to mitigate their losses.

References

1 Daron Acemoglu, Vasco M Carvalho, Asuman Ozdaglar, and Alireza Tahbaz-Salehi. The network origins of aggregate fluctuations. *Econometrica*, 80(5):1977–2016, 2012.

2 Daron Acemoglu, Asuman Ozdaglar, and Alireza Tahbaz-Salehi. Systemic risk and stability in financial networks. *American Economic Review*, 105(2):564–608, 2015.

3 Nimalan Arinaminpathy, Sujit Kapadia, and Robert M May. Size and complexity in model financial systems. *Proceedings of the National Academy of Sciences*, 109(45):18338–18343, 2012.

4 Marco Bardoscia, Stefano Battiston, Fabio Caccioli, and Guido Caldarelli. Pathways towards instability in financial networks. *Nature Communications*, 8:14416, 2017.

5 Stefano Battiston, Guido Caldarelli, Robert M May, Tarik Roukny, and Joseph E Stiglitz. The price of complexity in financial networks. *Proceedings of the National Academy of Sciences*, 113(36):10031–10036, 2016.

6 Nils Bertschinger, Martin Hoefer, and Daniel Schmand. Strategic Payments in Financial Networks. In *11th Innovations in Theoretical Computer Science Conference (ITCS 2020)*, volume 151 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 46:1–46:16, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

7 Stéphane Dees, Jérôme Henry, and Reiner Martin. Stamp€: stress-test analytics for macroprudential purposes in the euro area. *Frankfurt am Main: ECB*, 2017.

8 Gabrielle Demange. Contagion in financial networks: a threat index. *Management Science*, 64(2):955–970, 2016.

9 Darrell Duffie and Haoxiang Zhu. Does a central clearing counterparty reduce counterparty risk? *The Review of Asset Pricing Studies*, 1(1):74–95, 2011.

10 Marco D'Errico, Stefano Battiston, Tuomas Peltonen, and Martin Scheicher. How does risk flow in the credit default swap market? *Journal of Financial Stability*, 35:53–74, 2018.

11 Larry Eisenberg and Thomas H Noe. Systemic risk in financial systems. *Management Science*, 47(2):236–249, 2001.

12 Matthew Elliott, Benjamin Golub, and Matthew O Jackson. Financial networks and contagion. *American Economic Review*, 104(10):3115–53, 2014.

13 Helmut Elsinger, Alfred Lehar, and Martin Summer. Risk assessment for banking systems. *Management science*, 52(9):1301–1314, 2006.

14 Ingo Fender and Jacob Gyntelberg. Overview: global financial crisis spurs unprecedented policy actions. *BIS Quarterly Review*, 13(4):1–24, 2008.

15 Prasanna Gai, Andrew Haldane, and Sujit Kapadia. Complexity, concentration and contagion. *Journal of Monetary Economics*, 58(5):453–470, 2011.

16 Shizuo Kakutani et al. A generalization of brouwer's fixed point theorem. *Duke mathematical journal*, 8(3):457–459, 1941.

17 Matt V Leduc, Sebastian Poledna, and Stefan Thurner. Systemic risk management in financial networks with credit default swaps. *Available at SSRN 2713200*, 2017.

18 Yee Cheng Loon and Zhaodong Ken Zhong. The impact of central clearing on counterparty risk, liquidity, and trading: Evidence from the credit default swap market. *Journal of Financial Economics*, 112(1):91–115, 2014.

19 Matthew O'Brien. How to make money for nothing like wall street. *The Atlantic (Business)*, October 2013. Accessed: 22. Apr, 2020. URL: https://www.theatlantic.com/business/archive/2013/10/how-to-make-money-for-nothing-like-wall-street/280825/.

20 Martin J Osborne et al. *An introduction to game theory*, volume 3. Oxford university press New York, 2004.

21 Leonard CG Rogers and Luitgard AM Veraart. Failure and rescue in an interbank network. *Management Science*, 59(4):882–898, 2013.

22 Steffen Schuldenzucker, Sven Seuken, and Stefano Battiston. Clearing payments in financial networks with credit default swaps. In *Proceedings of the 2016 ACM Conference on Economics and Computation*, EC '16, pages 759–759, New York, NY, USA, 2016. ACM.

23 Steffen Schuldenzucker, Sven Seuken, and Stefano Battiston. Finding Clearing Payments in Financial Networks with Credit Default Swaps is PPAD-complete. In *8th Innovations in Theoretical Computer Science Conference (ITCS 2017)*, volume 67 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 32:1–32:20, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

24 Martin Shubik. The dollar auction game: A paradox in noncooperative behavior and escalation. *Journal of conflict Resolution*, 15(1):109–111, 1971.

25 Stefania Vitali, James B Glattfelder, and Stefano Battiston. The network of global corporate control. *PloS one*, 6(10):e25995, 2011.

Nondeterministic and Randomized Boolean Hierarchies in Communication Complexity

Toniann Pitassi
University of Toronto, Canada
Institute for Advanced Study, Princeton, NJ, USA
toni@cs.toronto.edu

Morgan Shirley
University of Toronto, Canada
shirley@cs.toronto.edu

Thomas Watson
University of Memphis, TN, USA
Thomas.Watson@memphis.edu

Abstract

We investigate the power of randomness in two-party communication complexity. In particular, we study the model where the parties can make a constant number of queries to a function with an efficient one-sided-error randomized protocol. The complexity classes defined by this model comprise the Randomized Boolean Hierarchy, which is analogous to the Boolean Hierarchy but defined with one-sided-error randomness instead of nondeterminism. Our techniques connect the Nondeterministic and Randomized Boolean Hierarchies, and we provide a complete picture of the relationships among complexity classes within and across these two hierarchies. In particular, we prove that the Randomized Boolean Hierarchy does not collapse, and we prove a query-to-communication lifting theorem for all levels of the Nondeterministic Boolean Hierarchy and use it to resolve an open problem stated in the paper by Halstenberg and Reischuk (CCC 1988) which initiated the study of this hierarchy.

2012 ACM Subject Classification Theory of computation → Communication complexity

Keywords and phrases Boolean hierarchies, lifting theorems, query complexity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.92

Category Track A: Algorithms, Complexity and Games

Related Version A full version of this paper [27] is available at `https://eccc.weizmann.ac.il/report/2019/043`.

Funding *Toniann Pitassi*: Supported by NSERC.
Morgan Shirley: Supported by NSERC.
Thomas Watson: Supported by NSF grant CCF-1657377.

Acknowledgements We thank Benjamin Rossman for helpful comments and discussions.

1 Introduction

A classic example of the power of randomness in communication is the Equality function: Alice gets an n-bit string x, Bob gets an n-bit string y, and they want to know whether x equals y. Though Equality is maximally hard for deterministic communication [35], it can be solved by a randomized protocol using $O(1)$ bits of communication (in the public-coin model) using the fingerprinting technique. Although this example (known for over 40 years) demonstrates the power of randomized communication in the standard two-party setting, many questions remain about the exact power of randomness in communication.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 92; pp. 92:1–92:19
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Much is still not understood about the power of randomness in other important communication settings beyond the standard two-party model. For example, in the Number-on-Forehead (NOF) model, even for three parties, no explicit function is known to exhibit a superpolylogarithmic separation between randomized and deterministic communication [25] (despite the fact that linear lower bounds for randomized NOF protocols were proven over 30 years ago in the seminal paper [3]). Another example concerns randomness in the context of *nondeterministic* two-party protocols (so-called Arthur–Merlin and Merlin–Arthur models). While strong lower bounds are known for Merlin–Arthur protocols [22] (though even here, explicit linear lower bounds remain elusive), no strong lower bounds are known for Arthur–Merlin protocols computing any explicit function – such bounds are necessary for making progress on rigidity bounds and circuit lower bounds, and also important for delegation [29, 12, 1].

We wish to highlight that even in the standard setting of plain randomized two-party communication protocols, many fundamental questions remain poorly understood. Our goal in this paper is to make progress on some of these questions. Much is known about the *limitations* of randomness – e.g., strong (indeed, linear) lower bounds are known for the classic SET-DISJOINTNESS function [2, 21, 30, 4], which can be viewed as showing that $\mathsf{coNP}^{\mathsf{cc}} \not\subseteq \mathsf{BPP}^{\mathsf{cc}}$ (where we use $^{\mathsf{cc}}$ superscripts to indicate the communication analogues of classical complexity classes). However, surprisingly little is known about the *power* of randomness. Most known efficient randomized protocols for other functions, such as GREATER-THAN, can be viewed as oracle reductions to the aforementioned EQUALITY upper bound: GREATER-THAN $\in \mathsf{P}^{\textsc{Equality}}$. Until recently, it was not even known whether $\mathsf{BPP}^{\mathsf{cc}} = \mathsf{P}^{\textsc{Equality}}$ (assuming the classes are defined to contain only total two-party functions), i.e., whether EQUALITY is the "only" thing randomness is good for. Chattopadhyay, Lovett, and Vinyals [8] answered this question in the negative by exhibiting a total function that is in $\mathsf{BPP}^{\mathsf{cc}}$ (indeed, in $\mathsf{coRP}^{\mathsf{cc}}$) but not in $\mathsf{P}^{\textsc{Equality}}$ (though the upper bound is still a form of fingerprinting). Since EQUALITY $\in \mathsf{coRP}^{\mathsf{cc}}$, we have $\mathsf{P}^{\textsc{Equality}} \subseteq \mathsf{P}^{\mathsf{RP}^{\mathsf{cc}}}$ where the latter class contains functions with efficient deterministic protocols that can make (adaptive) oracle queries to any function in $\mathsf{RP}^{\mathsf{cc}}$. In fact, [8] exhibited a strict infinite hierarchy of classes within $\mathsf{P}^{\mathsf{RP}^{\mathsf{cc}}}$, with $\mathsf{P}^{\textsc{Equality}}$ at the bottom, and with subsequent levels having increasingly powerful specific oracle functions.

However, it remains open whether $\mathsf{BPP}^{\mathsf{cc}} = \mathsf{P}^{\mathsf{RP}^{\mathsf{cc}}}$ for total functions (intuitively, whether two-sided error can be efficiently converted to oracle queries to one-sided error). It is even open whether $\mathsf{BPP}^{\mathsf{cc}} \subseteq \mathsf{P}^{\mathsf{NP}^{\mathsf{cc}}}$ for total functions [15], although this is known to be false if the classes are defined to allow partial functions [26]. We obtain a more detailed understanding of the structure of $\mathsf{P}^{\mathsf{RP}^{\mathsf{cc}}}$ by focusing on *restricting the number of oracle queries* (rather than restricting the $\mathsf{RP}^{\mathsf{cc}}$ function as in [8]). For constants $q = 0, 1, 2, 3, \ldots$, the class $\mathsf{P}_{\parallel}^{\mathsf{RP}[q]^{\mathsf{cc}}}$ consists of all two-party functions with an efficient (polylogarithmic communication cost) deterministic protocol that can make q many nonadaptive queries to an oracle for a function in $\mathsf{RP}^{\mathsf{cc}}$. Even if partial functions are allowed, it was not known whether these classes form a strict infinite hierarchy, i.e., whether $\mathsf{P}_{\parallel}^{\mathsf{RP}[q]^{\mathsf{cc}}} \subsetneq \mathsf{P}_{\parallel}^{\mathsf{RP}[q+1]^{\mathsf{cc}}}$ for all q. One of our main contributions (Theorem 1) implies that this is indeed the case, even for total functions. Our proof introduces new lower bound techniques.

With $\mathsf{NP}^{\mathsf{cc}}$ in place of $\mathsf{RP}^{\mathsf{cc}}$, $\bigcup_q \mathsf{P}_{\parallel}^{\mathsf{NP}[q]^{\mathsf{cc}}}$ forms the communication version of the Boolean Hierarchy from classical complexity, which was previously studied by Halstenberg and Reischuk [16, 17, 18]. We also prove results (Theorem 2 and Theorem 3) that resolve a 31-year-old open question posed in their work. $\bigcup_q \mathsf{P}_{\parallel}^{\mathsf{RP}[q]^{\mathsf{cc}}}$ can be viewed as the communication version of the Randomized Boolean Hierarchy, which has not been studied explicitly in previous works. Overall, we obtain a complete understanding of the relationships among the classes within and across the Nondeterministic and Randomized Boolean Hierarchies in communication complexity. In the following subsection we discuss the relevant communication complexity classes and describe our theorems in detail.

1.1 Background and our contributions

Forgetting about communication complexity for a moment, the Boolean Hierarchy in classical complexity theory consists of problems that have a polynomial-time algorithm making a constant number of queries to an NP oracle. This hierarchy has an intricate relationship with other complexity classes, and its second level (DP) captures the complexity of certain "exact" versions of optimization problems. It consists of an infinite sequence of complexity classes $\mathsf{NP}(q)$ for $q = 1, 2, 3, \ldots$ (where $\mathsf{NP}(1) = \mathsf{NP}$ and $\mathsf{NP}(2) = \mathsf{DP}$). Among the several equivalent ways of defining these classes [34, 7, 23, 32], perhaps the simplest is that $\mathsf{NP}(q)$ consists of all decision problems that can be computed by taking the parity of the answers to a sequence of q NP problems on the given input. As illustrated in Figure 1, it is known that these levels are intertwined with the classes $\mathsf{P}_{\parallel}^{\mathsf{NP}[q]}$ of all decision problems solvable in polynomial time using q nonadaptive NP queries (for constant q) [23, 32, 5]:

$$\mathsf{NP}(q) \subseteq \mathsf{P}_{\parallel}^{\mathsf{NP}[q]} \subseteq \mathsf{NP}(q+1) \qquad \text{and} \qquad \mathsf{coNP}(q) \subseteq \mathsf{P}_{\parallel}^{\mathsf{NP}[q]} \subseteq \mathsf{coNP}(q+1)$$

(by closure of $\mathsf{P}_{\parallel}^{\mathsf{NP}[q]}$ under complementation). Here, $\mathsf{coNP}(q)$ means $\mathsf{co}(\mathsf{NP}(q))$ rather than $(\mathsf{coNP})(q)$.

Analogous to the above *Nondeterministic* Boolean Hierarchy, one can define the *Randomized* Boolean Hierarchy by using RP (one-sided error randomized polynomial time) instead of NP in the definitions [6]. The analogous inclusions like in Figure 1 hold among all the classes $\mathsf{RP}(q)$, $\mathsf{coRP}(q)$, and $\mathsf{P}_{\parallel}^{\mathsf{RP}[q]}$, by similar arguments. Although the (suitably defined) *Polynomial* Hierarchy over RP is known to collapse to its second level, which equals BPP [36], the *Boolean* Hierarchy over RP has not been widely studied.

Recall the basic (deterministic) model of communication [35, 24], where Alice is given an input x and Bob is given an input y, and they wish to collaboratively evaluate some function $F(x, y)$ of their joint input by engaging in a protocol that specifies how they exchange bits of information about their inputs. Many classical complexity classes (P, RP, NP, and so on) have natural two-party communication analogues [2] (including the classes in the Nondeterministic and Randomized Boolean Hierarchies). The area of structural communication complexity, which concerns the properties of and relationships among these classes, is undergoing a renaissance and has turned out to yield new techniques and perspectives for understanding questions in a variety of other areas (circuit complexity, proof complexity, data structures, learning theory, delegation, fine-grained complexity, property testing, cryptography, extended formulations, etc.) [15]. For any classical time-bounded complexity class $\mathcal{C}$, we use $\mathcal{C}^{\mathsf{cc}}$ to denote its communication complexity analogue – the class of all two-party functions on n bits that admit a protocol communicating at most polylog(n) bits, in a model defined by analogy with the classical $\mathcal{C}$.

Halstenberg and Reischuk [16, 17] initiated the study of the Nondeterministic Boolean Hierarchy in two-party communication complexity. They observed that the inclusions shown in Figure 1 hold for the communication versions of the classes, by essentially the same proofs as in the time-bounded setting. They also proved that $\mathsf{NP}(q)^{\mathsf{cc}} \neq \mathsf{coNP}(q)^{\mathsf{cc}}$, which simultaneously implies that each of the inclusions is strict: $\mathsf{NP}(q)^{\mathsf{cc}} \subsetneq \mathsf{P}_{\parallel}^{\mathsf{NP}[q]\mathsf{cc}} \subsetneq \mathsf{NP}(q+1)^{\mathsf{cc}}$.

The communication version of the Randomized Boolean Hierarchy has not been explicitly studied as far as we know, but as mentioned earlier it is interesting since Equality $\in \mathsf{coRP}^{\mathsf{cc}}$ and many randomized protocols have been designed by reduction to this fact (such as

$$\begin{tikzcd}
 & \begin{array}{c}\mathsf{NP} =\\ \mathsf{NP}(1)\end{array} \arrow[dr] & & \begin{array}{c}\mathsf{DP} =\\ \mathsf{NP}(2)\end{array} \arrow[dr] & & \mathsf{NP}(3) \arrow[dr] & & \mathsf{NP}(4) \arrow[dr] & & & & \\
\mathsf{P} \arrow[ur] \arrow[dr] & & \mathsf{P}^{\mathsf{NP}[1]} \arrow[ur] \arrow[dr] & & \mathsf{P}_{\|}^{\mathsf{NP}[2]} \arrow[ur] \arrow[dr] & & \mathsf{P}_{\|}^{\mathsf{NP}[3]} \arrow[ur] \arrow[dr] & & \mathsf{P}_{\|}^{\mathsf{NP}[4]} & \cdots & \mathsf{P}_{\|}^{\mathsf{NP}} \arrow[r] & \mathsf{P}^{\mathsf{NP}} \\
 & \begin{array}{c}\mathsf{coNP}(1)\\ = \mathsf{coNP}\end{array} \arrow[ur] & & \begin{array}{c}\mathsf{coNP}(2)\\ = \mathsf{coDP}\end{array} \arrow[ur] & & \mathsf{coNP}(3) \arrow[ur] & & \mathsf{coNP}(4) \arrow[ur] & & & &
\end{tikzcd}$$

Figure 1 Relations between classes in the Boolean Hierarchy. Here, $\mathcal{C}_1 \to \mathcal{C}_2$ represents $\mathcal{C}_1 \subseteq \mathcal{C}_2$.

Greater-Than $\in \mathsf{P}^{\mathsf{RP}^{\mathsf{cc}}}$). What can we say about the power of a fixed number of queries to an $\mathsf{RP}^{\mathsf{cc}}$ oracle? Our first contribution strengthens the aforementioned separation due to Halstenberg and Reischuk.

▶ **Theorem 1.** *For total functions,* $\mathsf{coRP}(q)^{\mathsf{cc}} \not\subseteq \mathsf{NP}(q)^{\mathsf{cc}}$ *for every constant* q.

Since $\mathsf{RP}^{\mathsf{cc}} \subseteq \mathsf{NP}^{\mathsf{cc}}$, Theorem 1 simultaneously implies that each of the inclusions in the Randomized Boolean Hierarchy is strict: $\mathsf{RP}(q)^{\mathsf{cc}} \subsetneq \mathsf{P}_{\|}^{\mathsf{RP}[q]\mathsf{cc}} \subsetneq \mathsf{RP}(q+1)^{\mathsf{cc}}$, and thus the hierarchy does not collapse. Previously, no separation beyond the first level seemed to be known in the literature. Our proof of Theorem 1 is completely different from (and more involved than) Halstenberg and Reischuk's proof of $\mathsf{coNP}(q)^{\mathsf{cc}} \not\subseteq \mathsf{NP}(q)^{\mathsf{cc}}$, which used the "easy-hard argument" of [20].

In [16, 17], Halstenberg and Reischuk asked whether the inclusion $\mathsf{P}_{\|}^{\mathsf{NP}[q]\mathsf{cc}} \subseteq \mathsf{NP}(q+1)^{\mathsf{cc}} \cap \mathsf{coNP}(q+1)^{\mathsf{cc}}$ is strict. When $q = 0$, this is answered by the familiar results that $\mathsf{P}^{\mathsf{cc}} = \mathsf{NP}^{\mathsf{cc}} \cap \mathsf{coNP}^{\mathsf{cc}}$ when the classes are defined to contain only total functions [18], whereas $\mathsf{P}^{\mathsf{cc}} \subsetneq \mathsf{NP}^{\mathsf{cc}} \cap \mathsf{coNP}^{\mathsf{cc}}$ (indeed, $\mathsf{P}^{\mathsf{cc}} \subsetneq \mathsf{ZPP}^{\mathsf{cc}}$) holds when partial functions (promise problems) are allowed. For $q > 0$, we resolve this 31-year-old open question by proving that the situation is analogous to the $q = 0$ case.

▶ **Theorem 2.** *For total functions,*

$$\mathsf{P}_{\|}^{\mathsf{NP}[q]\mathsf{cc}} = \mathsf{NP}(q+1)^{\mathsf{cc}} \cap \mathsf{coNP}(q+1)^{\mathsf{cc}} \qquad \textit{and} \qquad \mathsf{P}_{\|}^{\mathsf{RP}[q]\mathsf{cc}} = \mathsf{RP}(q+1)^{\mathsf{cc}} \cap \mathsf{coRP}(q+1)^{\mathsf{cc}}$$

for every constant q.

▶ **Theorem 3.** *For partial functions,* $\mathsf{RP}(q+1)^{\mathsf{cc}} \cap \mathsf{coRP}(q+1)^{\mathsf{cc}} \not\subseteq \mathsf{P}_{\|}^{\mathsf{NP}[q]\mathsf{cc}}$ *for every constant* q.

Since $\mathsf{RP}^{\mathsf{cc}} \subseteq \mathsf{NP}^{\mathsf{cc}}$, Theorem 3 implies that

$$\mathsf{P}_{\|}^{\mathsf{NP}[q]\mathsf{cc}} \subsetneq \mathsf{NP}(q+1)^{\mathsf{cc}} \cap \mathsf{coNP}(q+1)^{\mathsf{cc}} \qquad \text{and} \qquad \mathsf{P}_{\|}^{\mathsf{RP}[q]\mathsf{cc}} \subsetneq \mathsf{RP}(q+1)^{\mathsf{cc}} \cap \mathsf{coRP}(q+1)^{\mathsf{cc}}$$

for partial functions. Taken together, Theorem 1, Theorem 2, and Theorem 3 complete the picture of the relationships among the classes within and across both hierarchies, for both total and partial functions.

1.2 Query-to-communication lifting

Our proof of Theorem 3 uses the paradigm of *query-to-communication lifting* [28, 11, 9, 14, 10, 13, 33]. This approach to proving communication lower bounds has led to breakthroughs on fundamental questions in communication complexity and many of its application areas. The idea consists of two steps:

(1) First prove an analogous lower bound in the simpler setting of decision tree depth complexity (a.k.a. query complexity). This step captures the combinatorial core of the lower bound argument without the burden of dealing with the full power of communication protocols.
(2) Then apply a *lifting theorem*, which translates the query lower bound into a communication lower bound for a related two-party function. This step encapsulates the general-purpose machinery for dealing with protocols, and can be reused from one argument to the next.

The availability of a lifting theorem greatly simplifies the task of proving certain communication lower bounds, because it divorces the problem-specific aspects from the generic aspects. The format of a lifting theorem is that if $f\colon \{0,1\}^n \to \{0,1\}$ is any partial function and $g\colon \mathcal{X} \times \mathcal{Y} \to \{0,1\}$ is a certain "small" two-party function called a gadget, then the communication complexity of the two-party composed function $f \circ g^n\colon \mathcal{X}^n \times \mathcal{Y}^n \to \{0,1\}$ – in which Alice gets $x = (x_1, \dots, x_n)$, Bob gets $y = (y_1, \dots, y_n)$, and their goal is to evaluate $(f \circ g^n)(x, y) := f(g(x_1, y_1), \dots, g(x_n, y_n))$ – should be approximately the query complexity of the outer function f. One direction is generally straightforward: given a query upper bound for f, a communication upper bound for $f \circ g^n$ is witnessed by a protocol that simulates the decision tree for f and evaluates $g(x_i, y_i)$ whenever it queries the i^{th} bit of the input to f; the number of bits of communication is at most the number of queries made by the decision tree times the (small) cost of evaluating a copy of g. The other direction is the challenging part: despite Alice and Bob's ability to send messages that depend in arbitrary ways on all n coordinates, they nevertheless cannot do much better than just simulating a decision tree, which involves "talking about" one coordinate at a time.

A lifting theorem must be stated with respect to a particular model of computation, such as deterministic, one-sided error randomized, nondeterministic, etc., which we associate with the corresponding complexity classes. Indeed, lifting theorems are known for P [28, 14], RP [13], NP [11, 9], and many other classes. It is convenient to recycle complexity class names to denote the complexity of a given function in the corresponding model, e.g., $\mathsf{P}^{\mathsf{dt}}(f)$ is the minimum worst-case number of queries made by any decision tree that computes f, and $\mathsf{P}^{\mathsf{cc}}(F)$ is the minimum worst-case communication cost of any protocol that computes F. With this notation, the deterministic lifting theorem from [28, 14] can be stated as: for all f, $\mathsf{P}^{\mathsf{cc}}(f \circ g^n) = \mathsf{P}^{\mathsf{dt}}(f) \cdot \Theta(\log n)$ where $g\colon [m] \times \{0,1\}^m \to \{0,1\}$ is the "index" gadget defined by $g(x, y) = y_x$ with $m := n^{20}$. (Note that $\mathsf{P}^{\mathsf{cc}}(g) = O(\log n)$ since Alice can send her $\log m$-bit "pointer" to Bob, who responds with the pointed-to bit from his string.) The index gadget has also been used in lifting theorems for several other complexity classes.

We prove lifting theorems for all classes in the Nondeterministic Boolean Hierarchy, with the index gadget.

▶ **Theorem 4.** *For every partial function* $f\colon \{0,1\}^n \to \{0,1\}$ *and every constant* q*,*
(i) $\mathsf{NP}(q)^{\mathsf{cc}}(f \circ g^n) \;=\; \mathsf{NP}(q)^{\mathsf{dt}}(f) \cdot \Theta(\log n)$
(ii) $\mathsf{P}_{\parallel}^{\mathsf{NP}[q]\mathsf{cc}}(f \circ g^n) \;=\; \mathsf{P}_{\parallel}^{\mathsf{NP}[q]\mathsf{dt}}(f) \cdot \Theta(\log n)$
where $g\colon [m] \times \{0,1\}^m \to \{0,1\}$ *is the index gadget defined by* $g(x, y) = y_x$ *with* $m := n^{20}$*.*

Only part *(ii)* is needed for proving Theorem 3, but part *(i)* forms an ingredient in the proof of *(ii)* and is of independent interest.

The most closely related lifting theorem to Theorem 4 is the one for P^{NP} [10], corresponding to computations that make an unbounded number of adaptive queries to an NP oracle. In that paper, the overall idea was to approximately characterize P^{NP} complexity in terms of *decision lists* (DL), and then prove a lifting theorem directly for DLs. Briefly, a conjunction

DL (introduced by [31]) is a sequence of small-width conjunctions each with an associated output bit, and the output is determined by finding the first conjunction in the list that accepts the given input. A rectangle DL is similar but with combinatorial rectangles instead of conjunctions. The proof from [10] shows how to convert a rectangle DL for $f \circ g^n$ into a conjunction DL for f.

The gist of our arguments for both parts of Theorem 4 is to approximately characterize (via different techniques than for P^{NP}) these classes in terms of DLs with a bounded number of *alternations* (how many times the associated output bit flips as we walk down the entire DL). The DL lifting argument from [10] does not preserve the number of alternations, but we show how it can be adapted to do this. Our techniques also yield an approximate lifting theorem for $\mathsf{P}_{\|}^{\mathsf{NP}}$ (corresponding to computations that make an unbounded number of nonadaptive NP oracle queries), but we omit the details.

2 Preliminaries

We assume familiarity with deterministic computation in query and communication complexity [19, 24]. Recall the following standard definitions of nondeterministic and one-sided error randomized models:

- An $\mathsf{NP}^{\mathsf{dt}}$ decision tree is a DNF formula Φ. Given an input z, the output of such a decision tree is Φ evaluated on z. A function f is computed by Φ if $f(z) = \Phi(z)$ on all inputs z for which $f(z)$ is defined. The cost of Φ is the maximum width (number of literals) in any conjunction in Φ.
- An $\mathsf{NP}^{\mathsf{cc}}$ protocol is a set $\mathcal{R}$ of combinatorial rectangles. Given an input (x, y), the output of such a protocol is $\mathcal{R}(x, y) \coloneqq 1$ iff there exists an $R \in \mathcal{R}$ containing (x, y). A two-party function F is computed by $\mathcal{R}$ if $F(x, y) = \mathcal{R}(x, y)$ on all inputs (x, y) for which $F(x, y)$ is defined. The cost of $\mathcal{R}$ is $\lceil \log(|\mathcal{R}| + 1) \rceil$, which intuitively represents the number of bits required to specify a rectangle in $\mathcal{R}$ or indicate that the input is in no such rectangle.
- An $\mathsf{RP}^{\mathsf{dt}}$ decision tree is a distribution $\boldsymbol{T}$ over deterministic decision trees. Given an input z, the output of such a decision tree is computed by sampling a deterministic decision tree T from $\boldsymbol{T}$ and evaluating $T(z)$. A function f is computed by $\boldsymbol{T}$ if for all $z \in f^{-1}(0)$, $\Pr_{T \sim \boldsymbol{T}}[T(z) = 1] = 0$ and for all $z \in f^{-1}(1)$, $\Pr_{T \sim \boldsymbol{T}}[T(z) = 1] \geq 1/2$. The cost of $\boldsymbol{T}$ is the maximum number of bits queried in any T in its support.
- An $\mathsf{RP}^{\mathsf{cc}}$ protocol is a distribution $\boldsymbol{\Pi}$ over deterministic communication protocols. Given an input (x, y), the output of such a protocol is computed by sampling a deterministic protocol Π from $\boldsymbol{\Pi}$ and evaluating $\Pi(x, y)$. A function F is computed by $\boldsymbol{\Pi}$ if for all $(x, y) \in F^{-1}(0)$, $\Pr_{\Pi \sim \boldsymbol{\Pi}}[\Pi(x, y) = 1] = 0$ and for all $(x, y) \in F^{-1}(1)$, $\Pr_{\Pi \sim \boldsymbol{\Pi}}[\Pi(x, y) = 1] \geq 1/2$. The cost of $\boldsymbol{\Pi}$ is the maximum number of bits exchanged in any Π in its support.

Let $\mathcal{C}$ be an arbitrary complexity class name representing a model of computation (such as NP or RP). We let $\mathcal{C}^{\mathsf{cc}}(F)$ denote the communication complexity of a two-party function F in the corresponding model: the minimum cost of any $\mathcal{C}^{\mathsf{cc}}$ protocol computing F. We let $\mathcal{C}^{\mathsf{dt}}(f)$ denote the query complexity of a Boolean function f in the corresponding model: the minimum cost of any $\mathcal{C}^{\mathsf{dt}}$ decision tree computing f. Often we will abuse notation by having F or f refer to an infinite *family* of functions, where there is at most one function in the family for each possible input length. In this case, $\mathcal{C}^{\mathsf{cc}}(F)$ or $\mathcal{C}^{\mathsf{dt}}(f)$ will be the complexity parameterized by the input length n; we typically express this with asymptotic notation. When written by itself, $\mathcal{C}^{\mathsf{cc}}$ or $\mathcal{C}^{\mathsf{dt}}$ denotes the class of all families of functions with complexity at most polylogarithmic in n, in the corresponding model. We will always clarify whether a class $\mathcal{C}^{\mathsf{cc}}$ or $\mathcal{C}^{\mathsf{dt}}$ is meant to contain partial functions or just total functions, since this is not explicit in the notation.

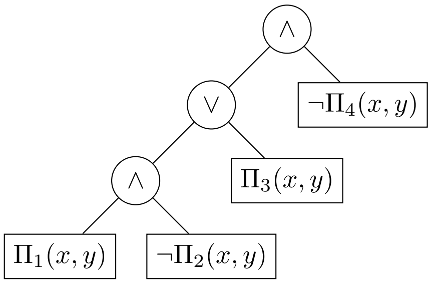


Figure 2 A visualization of a $\mathcal{C}(4)^{\mathsf{cc}}$ protocol, where each Π_i is a $\mathcal{C}^{\mathsf{cc}}$ protocol.

For $\mathsf{RP}^{\mathsf{cc}}$ and $\mathsf{RP}^{\mathsf{dt}}$, the constant $1/2$ in the success probability is arbitrary: by amplification, choosing a different positive constant in the definition would only affect the complexity of any function by a constant factor. Also note that $\mathsf{NP}^{\mathsf{dt}}(f) \leq \mathsf{RP}^{\mathsf{dt}}(f)$ for all f, and since we defined $\mathsf{RP}^{\mathsf{cc}}$ using the public-coin model, we have $\mathsf{NP}^{\mathsf{cc}}(F) \leq \mathsf{RP}^{\mathsf{cc}}(F) + O(\log n)$ for all F (by decreasing the number of random bits for sampling a protocol to $O(\log n)$ and using nondeterminism to guess an outcome that results in output 1).

2.1 Nondeterministic and Randomized Boolean Hierarchies

In the following definitions, restrict $\mathcal{C}$ to be either NP or RP. We will use two different but equivalent definitions of the constituent levels of the Nondeterministic and Randomized Boolean Hierarchies. Our "official" definition is in terms of the following "decision list functions" (also known as "odd-max-bit"):

▶ **Definition 5.** $\Delta_q \colon \{0,1\}^q \to \{0,1\}$ *is defined inductively as follows:*
- $\Delta_1(z_1) \coloneqq z_1$.
- *If q is odd,* $\Delta_q(z_1,\ldots,z_{q-1},z_q) \coloneqq \Delta_{q-1}(z_1,\ldots,z_{q-1}) \vee z_q$.
- *If q is even,* $\Delta_q(z_1,\ldots,z_{q-1},z_q) \coloneqq \Delta_{q-1}(z_1,\ldots,z_{q-1}) \wedge (\neg z_q)$.

In other words, letting $\oplus \colon \mathbb{N} \to \{0,1\}$ denote the parity function, we have $\Delta_q(z) \coloneqq \oplus(i)$ where i is the greatest index such that $z_i = 1$ (or $i = 0$ if z is all zeros).

▶ **Definition 6.** *A $\mathcal{C}(q)^{\mathsf{cc}}$ protocol is an ordered list of q many $\mathcal{C}^{\mathsf{cc}}$ protocols $\Pi = (\Pi_1,\ldots,\Pi_q)$. Given an input (x,y), the output of the protocol is $\Pi(x,y) \coloneqq \Delta_q(\Pi_1(x,y),\ldots,\Pi_q(x,y))$. The cost of a $\mathcal{C}(q)^{\mathsf{cc}}$ protocol is the sum of the costs of the component $\mathcal{C}^{\mathsf{cc}}$ protocols.*

See Figure 2 for an example of such a protocol. The Nondeterministic Boolean Hierarchy is $\bigcup_{\text{constant } q} \mathsf{NP}(q)^{\mathsf{cc}}$ and the Randomized Boolean Hierarchy is $\bigcup_{\text{constant } q} \mathsf{RP}(q)^{\mathsf{cc}}$. We are also interested in the complement classes $\mathsf{coNP}(q)^{\mathsf{cc}}$ and $\mathsf{coRP}(q)^{\mathsf{cc}}$. As is standard, when we write $\mathsf{co}\mathcal{C}(q)^{\mathsf{cc}}$ we refer to the class $\mathsf{co}(\mathcal{C}(q)^{\mathsf{cc}})$ (that is, functions that are the negations of functions in $\mathcal{C}(q)^{\mathsf{cc}}$) as opposed to $(\mathsf{co}\mathcal{C})(q)^{\mathsf{cc}}$ (that is, where the component protocols are $\mathsf{co}\mathcal{C}^{\mathsf{cc}}$ protocols).

There are analogous definitions in query complexity:

▶ **Definition 7.** *A $\mathcal{C}(q)^{\mathsf{dt}}$ decision tree is an ordered list of q many $\mathcal{C}^{\mathsf{dt}}$ decision trees $T = (T_1,\ldots,T_q)$. Given an input z, define the output as $T(z) \coloneqq \Delta_q(T_1(z),\ldots,T_q(z))$. The cost of a $\mathcal{C}(q)^{\mathsf{dt}}$ decision tree is the sum of the costs of the component $\mathcal{C}^{\mathsf{dt}}$ decision trees.*

Our alternative definition of the Nondeterministic and Randomized Boolean Hierarchies simply replaces Δ_q with the parity function $\oplus_q \colon \{0,1\}^q \to \{0,1\}$.

▶ **Lemma 8.** *For $\mathcal{C} \in \{\mathsf{NP}, \mathsf{RP}\}$, if the definitions of $\mathcal{C}(q)^{\mathsf{cc}}$ and $\mathcal{C}(q)^{\mathsf{dt}}$ are changed to use $\oplus_q$ in place of Δ_q, it only affects the complexity measures $\mathcal{C}(q)^{\mathsf{cc}}(F)$ and $\mathcal{C}(q)^{\mathsf{dt}}(f)$ by a constant factor (depending on q).*

Wagner [32] showed that these alternative characterizations are equivalent for the classical Nondeterministic Boolean Hierarchy, and Halstenberg and Reischuk [17] observed the same (up to a constant factor) in communication complexity. This latter proof uses only the property that $\mathsf{NP}^{\mathsf{cc}}$ is closed under intersection and union; that is, if $\mathsf{NP}^{\mathsf{cc}}(F_1), \mathsf{NP}^{\mathsf{cc}}(F_2) \leq k$, then $\mathsf{NP}^{\mathsf{cc}}(F_1 \wedge F_2)$ and $\mathsf{NP}^{\mathsf{cc}}(F_1 \vee F_2)$ are both $O(k)$. Observe that since this property also holds for $\mathsf{RP}^{\mathsf{cc}}$, $\mathsf{NP}^{\mathsf{dt}}$, and $\mathsf{RP}^{\mathsf{dt}}$, their proof works for these models of computation as well. In fact, in all of these models, the cost of the intersection or union of i cost-k computations is at most ik.

We use both definitions in this paper. We found that the Δ_q definition makes it easier to prove the lifting theorems, and the $\oplus_q$ definition makes it easier to prove concrete upper and lower bounds.

2.2 Parallel queries

In the following definitions, restrict $\mathcal{C}$ to be either NP or RP.

▶ **Definition 9.** *A $\mathsf{P}_{\|}^{\mathcal{C}[q]\mathsf{cc}}$ protocol consists of a deterministic protocol Π_{det} that maps an input (x, y) to two things: a function $\mathsf{out}\colon \{0,1\}^q \to \{0,1\}$ and an ordered list of q many $\mathcal{C}^{\mathsf{cc}}$ protocols $(\Pi_1, \ldots, \Pi_q)$. The output is then $\mathsf{out}(\Pi_1(x,y), \ldots, \Pi_q(x,y))$. The cost of a $\mathsf{P}_{\|}^{\mathcal{C}[q]\mathsf{cc}}$ protocol is the communication cost (depth) of Π_{det} plus the maximum over (x,y) of the sum of the costs of the $\mathcal{C}^{\mathsf{cc}}$ protocols produced by $\Pi_{\text{det}}(x,y)$.*

▶ **Definition 10.** *A $\mathsf{P}_{\|}^{\mathcal{C}[q]\mathsf{dt}}$ decision tree consists of a deterministic decision tree T_{det} that maps an input z to two things: a function $\mathsf{out}\colon \{0,1\}^q \to \{0,1\}$ and an ordered list of q many $\mathcal{C}^{\mathsf{dt}}$ decision trees $(T_1, \ldots, T_q)$. The output is then $\mathsf{out}(T_1(z), \ldots, T_q(z))$. The cost of a $\mathsf{P}_{\|}^{\mathcal{C}[q]\mathsf{dt}}$ decision tree is the query cost (depth) of T_{det} plus the maximum over z of the sum of the costs of the $\mathcal{C}^{\mathsf{dt}}$ decision trees produced by $T_{\text{det}}(z)$.*

The following lemma states that at each leaf of Π_{det} or T_{det}, we can replace the q "$\mathcal{C}$ oracle queries" with one "$\mathcal{C}(q)$ oracle query" (where some leaves may output the oracle's answer, while other leaves output the negation of it). This was shown in classical time-bounded complexity using the so-called "mind-change argument" [5], and this argument can be translated directly to communication and query complexity. For example, [17] used this method to show that $\mathsf{P}_{\|}^{\mathsf{NP}[q]\mathsf{cc}} \subseteq \mathsf{NP}(q+1)^{\mathsf{cc}} \cap \mathsf{coNP}(q+1)^{\mathsf{cc}}$. We will only need to use the result for $\mathcal{C} = \mathsf{NP}$.

▶ **Lemma 11.** *For $\mathcal{C} \in \{\mathsf{NP}, \mathsf{RP}\}$, we have $\mathsf{P}_{\|}^{\mathcal{C}[q]\mathsf{cc}}(F) = \Theta(\mathsf{P}^{\mathcal{C}(q)[1]\mathsf{cc}}(F))$ and $\mathsf{P}_{\|}^{\mathcal{C}[q]\mathsf{dt}}(f) = \Theta(\mathsf{P}^{\mathcal{C}(q)[1]\mathsf{dt}}(f))$ for every constant q.*

3 Separations

In this section we prove our separation results (Theorem 1 and Theorem 3).

3.1 Proof of Theorem 1

▶ **Theorem** (Restatement of Theorem 1). *For total functions, $\mathsf{coRP}(q)^{\mathsf{cc}} \not\subseteq \mathsf{NP}(q)^{\mathsf{cc}}$ for every constant q.*

Fix any constant q. Let $\oplus_q\textsc{NonEq}\colon (\{0,1\}^n)^q \times (\{0,1\}^n)^q \to \{0,1\}$ be the two-party total function where Alice's and Bob's inputs are divided into q blocks $x = (x_1,\dots,x_q)$ and $y = (y_1,\dots,y_q)$ with each $x_i, y_i \in \{0,1\}^n$, and which is defined by $\oplus_q\textsc{NonEq}(x,y) \coloneqq 1$ iff there are an odd number of blocks i such that $x_i \neq y_i$. Note that $\mathsf{RP}(q)^{\mathsf{cc}}(\oplus_q\textsc{NonEq}) = O(1)$ by Lemma 8 and the standard fact that $\mathsf{RP}^{\mathsf{cc}}(\textsc{NonEq}) = O(1)$. Thus $\overline{\oplus_q\textsc{NonEq}} \in \mathsf{coRP}(q)^{\mathsf{cc}}$. We will now prove that $\mathsf{NP}(q)^{\mathsf{cc}}(\overline{\oplus_q\textsc{NonEq}}) = \Omega(n)$.

Suppose for contradiction $\overline{\oplus_q\textsc{NonEq}}$ has an $\mathsf{NP}(q)^{\mathsf{cc}}$ protocol of cost $k \le n/2^q$, say $\Pi = (\mathcal{R}_1,\dots,\mathcal{R}_q)$ where each $\mathcal{R}_i$ is a nonempty set of rectangles. By Lemma 8 we may assume the protocol outputs 1 iff the input is contained in an odd number of the rectangle unions $\bigcup_{R\in\mathcal{R}_i} R$ for $i \in [q]$, in other words, $\Pi(x,y) \coloneqq \oplus_q(\mathcal{R}_1(x,y),\dots,\mathcal{R}_q(x,y))$. Note that, assuming $\mathcal{R}_i$ has cost k_i, the total number of rectangles in these unions is at most $\sum_i |\mathcal{R}_i| \le \prod_i(|\mathcal{R}_i|+1) \le \prod_i 2^{k_i} = 2^k$.

We will iteratively construct a sequence of rectangles Q^j for $j = 0,\dots,q$ such that *(i)* there are at least j many values of i for which $Q^j \subseteq \bigcup_{R\in\mathcal{R}_i} R$, and *(ii)* Q^j contains an input where exactly j blocks are unequal. We obtain the contradiction when $j = q$: by *(ii)* some input $(x,y) \in Q^q$ has $x_i \neq y_i$ for all i and thus $\overline{\oplus_q\textsc{NonEq}}(x,y) = 1 - \oplus(q)$, yet by *(i)* we have $\mathcal{R}_i(x,y) = 1$ for all i and thus $\Pi(x,y) = \oplus(q)$, contradicting the supposed correctness of Π.

We will actually maintain stronger invariants than the above *(i)* and *(ii)*: For *(i)*, we will actually have for some j values of i – we assume they are $1,\dots,j$ for notational convenience – some individual rectangle $R_i \in \mathcal{R}_i$ contains Q^j. For *(ii)*, Q^j will actually have the following form: for some fixed strings $a^j = (a_1,\dots,a_j) \in (\{0,1\}^n)^j$ and $b^j = (b_1,\dots,b_j) \in (\{0,1\}^n)^j$ such that $a_i \neq b_i$ for all $i \in [j]$, and for some nonempty set $S^j \subseteq (\{0,1\}^n)^{q-j}$, we have $Q^j \coloneqq \{a^j s : s \in S^j\} \times \{b^j s : s \in S^j\}$, which we abbreviate as $a^jS^j \times b^jS^j$. Defining a *diagonal* input in Q^j to be one of the form (a^js, b^js) for any particular $s \in S^j$, we see that each diagonal input has exactly j unequal blocks, as needed for *(ii)*.

In fact, we will maintain not just that S^j is nonempty, but that it is sufficiently large. Specifically, the *deficiency* of S^j, defined as $\mathbf{D}_\infty(S^j) \coloneqq n(q-j) - \log|S^j|$, will be at most $(2^j-1)(2k+1)$. At the end, since $(2^q-1)(2k+1) < \infty$, this guarantees that S^q will contain at least one element from $(\{0,1\}^n)^{q-q}$. The latter set only has one element, namely the empty tuple, so this means Q^q will contain the single input (a^q, b^q), which has all blocks unequal.

We start with $S^0 = (\{0,1\}^n)^q$, which indeed has $\mathbf{D}_\infty(S^0) = 0 = (2^0-1)(2k+1)$, and thus Q^0 is the rectangle of all possible inputs. Now supposing we already have a^j, b^j, and S^j satisfying all the properties from the previous two paragraphs, we explain how to obtain $a_{j+1}, b_{j+1} \in \{0,1\}^n$ and $S^{j+1} \subseteq (\{0,1\}^n)^{q-(j+1)}$ so these properties again hold (with $a^{j+1} \coloneqq a^ja_{j+1}$ and $b^{j+1} \coloneqq b^jb_{j+1}$).

We first observe that each diagonal input in Q^j must be contained in at least one rectangle from $\mathcal{R}_{j+1} \cup \dots \cup \mathcal{R}_q$. This is because such an input (x,y) is already contained in the rectangles $R_1 \in \mathcal{R}_1,\dots,R_j \in \mathcal{R}_j$, and these cannot be the only values of i such that $\mathcal{R}_i(x,y) = 1$ since otherwise we would have $\Pi(x,y) = \oplus(j)$ while $\overline{\oplus_q\textsc{NonEq}}(x,y) = 1 - \oplus(j)$, contradicting the supposed correctness of Π. Now pick one of the (at most 2^k) rectangles from $\mathcal{R}_{j+1} \cup \dots \cup \mathcal{R}_q$ that contains the largest fraction (at least $1/2^k$) of diagonal inputs from Q^j, and assume this rectangle is $R_{j+1} \in \mathcal{R}_{j+1}$ for notational convenience. Defining $\tilde{S}^j \coloneqq \{s \in S^j : (a^js, b^js) \in R_{j+1}\}$, we see that $\mathbf{D}_\infty(\tilde{S}^j) \le \mathbf{D}_\infty(S^j) + k \le (2^j-1)(2k+1) + k$.

Since R_{j+1} is a rectangle, it must in fact contain the entire rectangle $a^j\tilde{S}^j \times b^j\tilde{S}^j$. Since $a^j\tilde{S}^j \times b^j\tilde{S}^j \subseteq a^jS^j \times b^jS^j = Q^j$, by assumption it is also contained in each of $R_1,\dots,R_j$. In the end, we will ensure Q^{j+1} is a subrectangle of $a^j\tilde{S}^j \times b^j\tilde{S}^j$, which will maintain property *(i)*: Q^{j+1} is contained in each of $R_1,\dots,R_{j+1}$.

To maintain *(ii)*, we will find some $a_{j+1} \neq b_{j+1}$ and then define $S^{j+1} \coloneqq \{s : a_{j+1}s \in \tilde{S}^j \text{ and } b_{j+1}s \in \tilde{S}^j\}$. Then $a_{j+1}S^{j+1} \subseteq \tilde{S}^j$ and $b_{j+1}S^{j+1} \subseteq \tilde{S}^j$ ensure that $Q^{j+1} \coloneqq a^{j+1}S^{j+1} \times b^{j+1}S^{j+1}$ is indeed a subrectangle of $a^j\tilde{S}^j \times b^j\tilde{S}^j$, as we needed for *(i)*. The fact that this can be done with a not-too-small S^{j+1} is encapsulated in the following technical lemma, which we prove shortly:

▶ **Lemma 12.** *Consider any bipartite graph with left nodes U and right nodes V, and suppose $1 \geq \varepsilon \geq 2/|U|$. If an ε fraction of all possible edges are present in the graph, then there exist distinct nodes $u, u' \in U$ that have at least $(\varepsilon^2/2) \cdot |V|$ common neighbors.*

Specifically, take $U \coloneqq \{0,1\}^n$ and $V \coloneqq (\{0,1\}^n)^{q-(j+1)}$ (so $|V| = 1$ if $j = q-1$, but that is fine), put an edge between $u \in U$ and $v \in V$ iff $uv \in \tilde{S}^j$, and let $\varepsilon \coloneqq |\tilde{S}^j|/2^{n(q-j)} = 1/2^{\mathbf{D}_\infty(\tilde{S}^j)}$. Notice that $\varepsilon \geq 2/|U|$ holds since $\mathbf{D}_\infty(\tilde{S}^j) \leq (2^j - 1)(2k+1) + k \leq 2^{j+1}k - 1 \leq n - 1$ follows from our assumption that $k \leq n/2^q$. Thus Lemma 12 guarantees we can pick strings $a_{j+1} \neq b_{j+1}$ (corresponding to the nodes u, u') such that S^{j+1} (the set of common neighbors) has size at least $(\varepsilon^2/2) \cdot 2^{n(q-(j+1))}$. Thus

$$\begin{aligned}\mathbf{D}_\infty(S^{j+1}) \;&\coloneqq\; n(q-(j+1)) - \log|S^{j+1}| \;\leq\; \log(2/\varepsilon^2) \;=\; 2\mathbf{D}_\infty(\tilde{S}^j) + 1 \\ &\leq\; 2\big((2^j - 1)(2k+1) + k\big) + 1 \;=\; \big(2(2^j-1)+1\big)(2k+1) \\ &=\; (2^{j+1} - 1)(2k+1)\end{aligned}$$

as we needed for *(ii)*. This finishes the proof of Theorem 1.

Proof of Lemma 12. Let d_u and d_v denote the degrees of nodes $u \in U$ and $v \in V$, and let $d_{u,u'}$ denote the number of common neighbors of $u, u' \in U$. Summing over ordered pairs u, u' of not-necessarily-distinct left nodes, we have

$$\sum_{u,u' \in U} d_{u,u'} \;=\; \sum_{v \in V} d_v^2 \;\geq\; \Big(\sum_{v \in V} d_v\Big)^2 / |V| \;=\; \varepsilon^2 \cdot |U|^2 \cdot |V|$$

by Cauchy–Schwarz and the assumption $\sum_{v \in V} d_v = \varepsilon \cdot |U| \cdot |V|$. Now sampling u, u' independently uniformly at random from U, we have

$$\varepsilon^2 \cdot |V| \;\leq\; \mathop{\mathrm{E}}_{u,u'}[d_{u,u'}] \;\leq\; \mathop{\mathrm{E}}_{u,u'}[d_{u,u'} \mid u \neq u'] + \mathop{\mathrm{E}}_{u}[d_u] \cdot \Pr_{u,u'}[u = u']$$

(the conditioning is valid by the assumption $|U| \geq 2$). Since $\mathrm{E}_u[d_u] = \varepsilon \cdot |V|$ and $\Pr_{u,u'}[u = u'] = 1/|U|$, rearranging gives

$$\mathop{\mathrm{E}}_{u,u'}[d_{u,u'} \mid u \neq u'] \;\geq\; \varepsilon^2 \cdot |V| - \varepsilon \cdot |V|/|U| \;\geq\; (\varepsilon^2/2) \cdot |V|$$

where the last inequality holds by the assumption $1/|U| \leq \varepsilon/2$. Thus there must be some $u \neq u'$ such that $d_{u,u'}$ is at least this large. ◀

3.2 Proof of Theorem 3

▶ **Theorem** (Restatement of Theorem 3). *For partial functions,* $\mathsf{RP}(q+1)^{\mathsf{cc}} \cap \mathsf{coRP}(q+1)^{\mathsf{cc}} \not\subseteq \mathsf{P}_{\|}^{\mathsf{NP}[q]\mathsf{cc}}$ *for every constant q.*

To prove this result, we require the query complexity separation $\mathsf{coRP}(q)^{\mathsf{dt}} \not\subseteq \mathsf{NP}(q)^{\mathsf{dt}}$.

▶ **Definition 13.** *Fix any constant q. Let $\oplus_q\textsc{GapOr}\colon (\{0,1\}^n)^q \to \{0,1\}$ be the partial function where the input is divided into q blocks $z = (z_1, \ldots, z_q)$ having the promise that each $z_i \in \{0,1\}^n$ is either all zeros or at least half ones (call such an input* valid*), and which is defined by $\oplus_q\textsc{GapOr}(z) \coloneqq 1$ iff an odd number of blocks i are such that z_i is nonzero.*

Note that $\mathsf{RP}(q)^{\mathsf{dt}}(\oplus_q\textsc{GapOr}) = O(1)$ by Lemma 8 and the fact that $\mathsf{RP}^{\mathsf{dt}}(\textsc{GapOr}) = 1$. The full version of this paper [27] contains a proof of the following lemma:

▶ **Lemma 14.** $\mathsf{NP}(q)^{\mathsf{dt}}(\overline{\oplus_q\textsc{GapOr}}) = \Omega(n)$.

We now proceed to the proof of Theorem 3.

Fix any constant q. Let $\textsc{Which}\oplus_{q+1}\textsc{GapOr}\colon (\{0,1\}^{2n})^{2(q+1)} \to \{0,1\}$ be the following partial function: The input is divided into two halves $z = (z^0, z^1)$, and each half is divided into $q+1$ blocks $z^h = (z^h_1, \dots, z^h_{q+1})$ having the promise that each $z^h_i \in \{0,1\}^{2n}$ is either all zeros or at least a *quarter* ones, and moreover it is promised that the number of nonzero blocks in z^0 has the opposite parity as the number of nonzero blocks in z^1 (call such an input *valid*). The partial function is defined by

$$\textsc{Which}\oplus_{q+1}\textsc{GapOr}(z) = \begin{cases} 1 & \text{if the number of nonzero blocks is odd in } z^0 \text{ and even in } z^1 \\ 0 & \text{if the number of nonzero blocks is even in } z^0 \text{ and odd in } z^1 \end{cases}$$

We henceforth abbreviate $\textsc{Which}\oplus_{q+1}\textsc{GapOr}$ as f. Note that $\mathsf{RP}(q+1)^{\mathsf{dt}}(f) = O(1)$ by applying the $\mathsf{RP}(q+1)^{\mathsf{dt}}$ decision tree for $\oplus_{q+1}\textsc{GapOr}$ on z^0 (adapted for the different block length and different threshold for fraction of ones in a block). By symmetry (focusing on z^1), we also have $\mathsf{RP}(q+1)^{\mathsf{dt}}(\overline{f}) = O(1)$. Letting $g\colon [m] \times \{0,1\}^m \to \{0,1\}$ be the index gadget with $m := N^{20}$ where $N := 4(q+1)n$, this implies that

$$\mathsf{RP}(q+1)^{\mathsf{cc}}(f \circ g^N) = O(\log n) \qquad \text{and} \qquad \mathsf{coRP}(q+1)^{\mathsf{cc}}(f \circ g^N) = O(\log n)$$

(by the "easy direction" of $\mathsf{RP}(q+1)$ lifting) and thus $f \circ g^N \in \mathsf{RP}(q+1)^{\mathsf{cc}} \cap \mathsf{coRP}(q+1)^{\mathsf{cc}}$. We will now prove that $\mathsf{P}_{\parallel}^{\mathsf{NP}[q]\mathsf{dt}}(f) = \Omega(n)$, which by Theorem 4.*(ii)* implies that $\mathsf{P}_{\parallel}^{\mathsf{NP}[q]\mathsf{cc}}(f \circ g^N) = \Omega(n \log n)$.

We show this by reduction from Lemma 14. We henceforth abbreviate $\overline{\oplus_q\textsc{GapOr}}$ as f'. Supposing f has a $\mathsf{P}_{\parallel}^{\mathsf{NP}[q]\mathsf{dt}}$ decision tree T of cost $k \le n/2$, say with deterministic phase T_{det}, we will use it to construct an $\mathsf{NP}(q)^{\mathsf{dt}}$ decision tree T' of cost at most k for f'.

By Lemma 11 we may assume that each leaf of T_{det} produces a single $\mathsf{NP}(q)^{\mathsf{dt}}$ decision tree and chooses whether to output the same or opposite answer as that decision tree. Follow the root-to-leaf path in T_{det} where all queries are answered with zero. Let $\rho \in (\{0,*\}^{2n})^{2(q+1)}$ be the partial assignment with at most $k \le n/2$ zeros that records these queries (so an input leads to this leaf iff it is consistent with ρ). Let $T_{\text{leaf}} = (\Phi_1, \dots, \Phi_q)$ be the $\mathsf{NP}(q)^{\mathsf{dt}}$ decision tree of cost at most k produced at this leaf, where each Φ_i is a DNF. By symmetry, we assume (without loss of generality) that this leaf chooses to output the same answer as T_{leaf}.

Given any valid input z' to f', we show how to map it to a valid input z to f such that *(i)* $f'(z') = f(z)$, *(ii)* z is consistent with ρ, and *(iii)* each bit of z either is fixed or is some preselected bit of z'. Since *(iii)* implies that $T_{\text{leaf}}(z)$ can be viewed as an $\mathsf{NP}(q)^{\mathsf{dt}}$ decision tree $T'(z')$ by substituting a constant or variable of z' in for each variable of z (which does not increase the width of any conjunction), and T' would correctly compute f' since

$$f'(z') \;=\; f(z) \;=\; T(z) \;=\; T_{\text{leaf}}(z) \;=\; T'(z')$$

by *(i)*, correctness of T, *(ii)*, and *(iii)* respectively, this would show that $\mathsf{NP}(q)^{\mathsf{dt}}(f') \le k \le n/2$, which we know is false from Lemma 14.

To define z, we start with ρ (that is, we place zeros everywhere ρ requires, thus ensuring *(ii)*). Since ρ has at most n zeros in each block (indeed, at most $n/2$ zeros total), we can then place more zeros in such a way that each block now has exactly n zeros and n stars.

Next we replace the stars in z^0_{q+1} with ones and replace the stars in z^1_{q+1} with zeros. Finally, for each $i \in [q]$, we fill in the stars of z^0_i with a copy of z'_i and fill in the stars of z^1_i with another copy of z'_i. This construction satisfies *(iii)*.

To verify *(i)*, first observe that since each block of z' is either all zeros or at least half $(n/2)$ ones, this ensures each block of z is either all zeros or at least a quarter $(2n/4)$ ones. Furthermore, if z' has exactly ℓ nonzero blocks then the number of nonzero blocks is $\ell + 1$ in z^0 (since z^0_{q+1} is nonzero) and ℓ in z^1 (since z^1_{q+1} is all zeros). Hence if $f'(z') = 1$ (ℓ is even) then $f(z) = 1$ (since $\ell + 1$ is odd and ℓ is even), and if $f'(z') = 0$ (ℓ is odd) then $f(z) = 0$ (since $\ell + 1$ is even and ℓ is odd). Thus $f'(z') = f(z)$, and this finishes the proof of Theorem 3.

4 Total function collapse

▶ **Theorem** (Restatement of Theorem 2). *For total functions,*

$$\mathsf{P}_{\|}^{\mathsf{NP}[q]\mathsf{cc}} = \mathsf{NP}(q+1)^{\mathsf{cc}} \cap \mathsf{coNP}(q+1)^{\mathsf{cc}} \qquad \textit{and} \qquad \mathsf{P}_{\|}^{\mathsf{RP}[q]\mathsf{cc}} = \mathsf{RP}(q+1)^{\mathsf{cc}} \cap \mathsf{coRP}(q+1)^{\mathsf{cc}}$$

for every constant q.

In this section we will present intuition for the proof of the nondeterministic case of Theorem 2. For the complete proof, see the full version of this paper [27].

This proof is similar to the argument for the $q = 0$ case (that is, for total functions, $\mathsf{P}^{\mathsf{cc}} = \mathsf{NP}^{\mathsf{cc}} \cap \mathsf{coNP}^{\mathsf{cc}}$). In that proof, Alice and Bob can use the fact that the rectangles in the $\mathsf{NP}^{\mathsf{cc}}$ protocol's 1-monochromatic covering of F are disjoint from the rectangles in the $\mathsf{coNP}^{\mathsf{cc}}$ protocol's 0-monochromatic covering. Specifically, if $F(x, y) = 1$, then (x, y) is in some 1-rectangle, which is row-disjoint or column-disjoint from each 0-rectangle. (If a 1-rectangle and 0-rectangle shared a row and a column, they would intersect, which is not possible for a total function.) Therefore, Alice and Bob can repeatedly eliminate from consideration at least half of the remaining 0-rectangles, by identifying a 1-rectangle that either has x in its row set but is row-disjoint from at least half the remaining 0-rectangles, or has y in its column set but is column-disjoint from at least half the remaining 0-rectangles. If (x, y) is indeed in a 1-rectangle, then this process can always continue until there are no 0-rectangles left. If (x, y) is in a 0-rectangle, then this process will never eliminate that rectangle, so the process will halt with a nonempty set of 0-rectangles.

We repeat a similar argument, but using the "top level" of the $\mathsf{NP}(q+1)^{\mathsf{cc}}$ and the $\mathsf{coNP}(q+1)^{\mathsf{cc}}$ protocols for F as our monochromatic rectangle sets. Here we think of a $\mathsf{coNP}(q+1)^{\mathsf{cc}}$ protocol as computing F by applying $\overline{\Delta}_{q+1}$ (with negations pushed to the leaves) to the indicators for $q + 1$ rectangle unions. Depending on the parity of q, the rectangle union $\mathcal{R}_{q+1}$ queried at depth 1 of the $\mathsf{NP}(q+1)^{\mathsf{cc}}$ protocol will correspond to either 1-monochromatic rectangles or 0-monochromatic rectangles for F. The rectangle union $\mathcal{R}'_{q+1}$ queried at depth 1 of the $\mathsf{coNP}(q+1)^{\mathsf{cc}}$ protocol will be the opposite color of monochromatic rectangles. Crucially, this means that no input is in a rectangle from both of these sets (as we are assuming F is total). See Figure 3 for an illustration.

A key observation is that a deterministic protocol similar to the one used in the $q = 0$ case, ran using these top-level rectangle sets, will return the correct answer *under the promise that (x, y) is in one of these rectangles.* Say, for example, that (x, y) is in some rectangle in the 1-monochromatic top-level set. Then the deterministic protocol will successfully eliminate all 0-rectangles from the other top-level set, and will announce that the answer is 1. If (x, y) was in one of the 0-rectangles, then that rectangle will never be eliminated, and so the protocol would announce that the answer is 0.

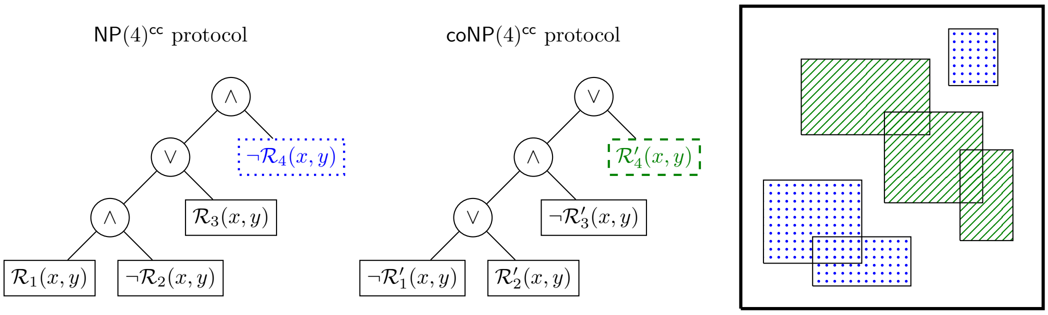


Figure 3 If a total function has an $\mathsf{NP}(4)^{\mathsf{cc}}$ protocol and a $\mathsf{coNP}(4)^{\mathsf{cc}}$ protocol, then the rectangle unions from the $\mathsf{NP}^{\mathsf{cc}}$ functions at depth one of each protocol are disjoint.

If (x, y) is in the top-level rectangle union for one of the protocols, then (x, y) is not in the top-level rectangle union of the *other* protocol, so $F(x, y)$ can be computed by the other protocol but where the top level is skipped (resulting in only q many $\mathsf{NP}^{\mathsf{cc}}$ oracle queries). This boils down to the observation that $\Delta_{q+1}(z_1, \ldots, z_q, 0) = \Delta_q(z_1, \ldots, z_q)$.

What if (x, y) is in neither top-level rectangle union? Then we can make no guarantees about the behavior of the deterministic protocol – it might answer 0 or 1 (which we interpret as merely a "guess" for $F(x, y)$). However, in this case *both* protocols correctly compute $F(x, y)$ even if the top level is skipped. Therefore, we will still get the correct answer no matter which guess is produced by the deterministic protocol.

5 Query-to-communication lifting theorems

▶ **Theorem** (Restatement of Theorem 4.). *For every partial function $f\colon \{0,1\}^n \to \{0,1\}$ and every constant q,*

(i) $\mathsf{NP}(q)^{\mathsf{cc}}(f \circ g^n) = \mathsf{NP}(q)^{\mathsf{dt}}(f) \cdot \Theta(\log n)$

(ii) $\mathsf{P}_{\parallel}^{\mathsf{NP}[q]\mathsf{cc}}(f \circ g^n) = \mathsf{P}_{\parallel}^{\mathsf{NP}[q]\mathsf{dt}}(f) \cdot \Theta(\log n)$

where $g\colon [m] \times \{0,1\}^m \to \{0,1\}$ is the index gadget defined by $g(x, y) = y_x$ with $m := n^{20}$.

In this section we present the proof for Theorem 4.*(i)*, as well as the necessary background. The full version of this paper [27] contains the proof of Theorem 4.*(ii)*. The high-level idea of the latter proof is to convert a $\mathsf{P}^{\mathsf{NP}(q)[1]\mathsf{cc}}$ protocol for $f \circ g^n$ into a $\mathsf{P}^{\mathsf{NP}(q)[1]\mathsf{dt}}$ decision tree for f by using the P lifting theorem of Raz and McKenzie [28, 14] to handle the deterministic phase, followed by our $\mathsf{NP}(q)$ lifting theorem to handle the single $\mathsf{NP}(q)$ oracle query.

5.1 Decision lists

The reason we call Δ_q a "decision list function" is that it highlights the connection between the Boolean Hierarchy classes and the *decision list* models of computation:

▶ **Definition 15.** *A rectangle decision list $\mathcal{L}_{\mathrm{R}}$ is an ordered list of pairs $(R_1, \ell_1), (R_2, \ell_2), \ldots$ where each R_i is a combinatorial rectangle, $\ell_i \in \{0,1\}$ is a label, and the final rectangle in the list contains all inputs in the domain. For an input (x, y), the output $\mathcal{L}_{\mathrm{R}}(x, y)$ is ℓ_i where i is the first index for which $(x, y) \in R_i$. The cost of $\mathcal{L}_{\mathrm{R}}$ is the* log *of the length of the list.*

▶ **Definition 16.** *A conjunction decision list $\mathcal{L}_{\mathrm{C}}$ is an ordered list of pairs $(C_1, \ell_1), (C_2, \ell_2), \ldots$ where each C_i is a conjunction, $\ell_i \in \{0,1\}$ is a label, and the final conjunction in the list accepts all inputs in the domain. For an input z, the output $\mathcal{L}_{\mathrm{C}}(z)$ is ℓ_i where i is the first index for which $C_i(z) = 1$. The cost of $\mathcal{L}_{\mathrm{C}}$ is the maximum width of any conjunction in the list.*

Note that the restriction on the final rectangle/conjunction is without loss of generality. The complexity measures $\mathsf{DL}^{\mathsf{cc}}(F)$ and $\mathsf{DL}^{\mathsf{dt}}(f)$ are the minimum cost of any rectangle/conjunction decision list computing F or f, and the classes $\mathsf{DL}^{\mathsf{cc}}$ and $\mathsf{DL}^{\mathsf{dt}}$ contain those functions with complexity at most $\text{polylog}(n)$.

We now define *q-alternating decision lists* to have the additional restriction that the sequence of output labels $\ell_1, \ell_2, \ldots$ only flips between 0 and 1 at most q times, and furthermore the last label is 0. This restriction partitions the list into contiguous *levels* where all labels in the same level are equal; without loss of generality the last level consists only of the final "catch-all" entry. For convenience, in the list entries we replace the labels with the level numbers themselves.

▶ **Definition 17.** *A q-alternating rectangle decision list $\mathcal{L}_{\mathrm{R}}$ is an ordered list of pairs $(R_1, \ell_1), (R_2, \ell_2), \ldots$ where each R_i is a combinatorial rectangle, $\ell_i \in \{0, 1, \ldots, q\}$ is a level such that $\ell_i \geq \ell_{i+1}$ for all i, and the final rectangle in the list contains all inputs in the domain and is the only rectangle at level 0. For an input (x, y), the output $\mathcal{L}_{\mathrm{R}}(x, y)$ is $\oplus(\ell_i)$ where i is the first index for which $(x, y) \in R_i$. The cost of $\mathcal{L}_{\mathrm{R}}$ is the* log *of the length of the list.*

▶ **Definition 18.** *A q-alternating conjunction decision list $\mathcal{L}_{\mathrm{C}}$ is an ordered list of pairs $(C_1, \ell_1), (C_2, \ell_2), \ldots$ where each C_i is a conjunction, $\ell_i \in \{0, 1, \ldots, q\}$ is a level such that $\ell_i \geq \ell_{i+1}$ for all i, and the final conjunction in the list accepts all inputs in the domain and is the only conjunction at level 0. For an input z, the output $\mathcal{L}_{\mathrm{C}}(z)$ is $\oplus(\ell_i)$ where i is the first index for which $C_i(z) = 1$. The cost of $\mathcal{L}_{\mathrm{C}}$ is the maximum width of any conjunction in the list.*

The complexity measures $\mathsf{DL}(q)^{\mathsf{cc}}(F)$ and $\mathsf{DL}(q)^{\mathsf{dt}}(f)$ are the minimum cost of any q-alternating rectangle/conjunction decision list computing F or f, and the classes $\mathsf{DL}(q)^{\mathsf{cc}}$ and $\mathsf{DL}(q)^{\mathsf{dt}}$ contain those functions with complexity at most $\text{polylog}(n)$.

It turns out that q-alternating decision lists are equivalent to $\mathsf{NP}(q)$ in both communication and query complexity. As this follows almost immediately from the definition of Δ_q, we omit the proof here.

▶ **Lemma 19.** $\mathsf{DL}(q)^{\mathsf{cc}}(F) = \Theta(\mathsf{NP}(q)^{\mathsf{cc}}(F))$ *and* $\mathsf{DL}(q)^{\mathsf{dt}}(f) = \Theta(\mathsf{NP}(q)^{\mathsf{dt}}(f))$ *for every constant q. Thus,* $\mathsf{DL}(q)^{\mathsf{cc}} = \mathsf{NP}(q)^{\mathsf{cc}}$ *and* $\mathsf{DL}(q)^{\mathsf{dt}} = \mathsf{NP}(q)^{\mathsf{dt}}$ *for partial functions.*

This can be contrasted with the lemma from [10] stating that $\mathsf{DL}^{\mathsf{cc}} = \mathsf{P}^{\mathsf{NPcc}}$ and $\mathsf{DL}^{\mathsf{dt}} = \mathsf{P}^{\mathsf{NPdt}}$ for partial functions.

5.2 Query-to-communication lifting for $\mathsf{NP}(q)$

The big-O direction of Theorem 4.*(i)* follows immediately from the same fact for NP: for every f, $\mathsf{NP}^{\mathsf{cc}}(f \circ g^n) = \mathsf{NP}^{\mathsf{dt}}(f) \cdot O(\log n)$ holds by replacing each of the $n^{O(k)}$ conjunctions in a width-k DNF with m^k rectangles (each of which contains inputs where the gadget outputs satisfy the conjunction), for a total of $n^{O(k)} m^k = 2^{k \cdot O(\log n)}$ rectangles. In the rest of this section we prove the big-Ω direction. By Lemma 19 it suffices to show

$$\mathsf{DL}(q)^{\mathsf{cc}}(f \circ g^n) = \mathsf{DL}(q)^{\mathsf{dt}}(f) \cdot \Omega(\log n).$$

5.2.1 Technical preliminaries

Our proof is closely related to the P^{NP} lifting theorem of Göös, Kamath, Pitassi, and Watson [10], so we start by recalling some definitions and lemmas that were used in that work. We will need to tweak some of the statements and parameters, though.

Define $G\colon [m]^n \times (\{0,1\}^m)^n \to \{0,1\}^n$ as $G \coloneqq g^n$. This partitions the input domain into 2^n slices $G^{-1}(z) = \{(x,y) : g(x_i,y_i) = z_i \text{ for all } i \in [n]\}$, one for each $z \in \{0,1\}^n$. For a set $Z \subseteq \{0,1\}^n$, let $G^{-1}(Z) \coloneqq \bigcup_{z\in Z} G^{-1}(z)$. Consider sets $A \subseteq [m]^n$ and $B \subseteq (\{0,1\}^m)^n$. For $I \subseteq [n]$, we let $A_I \coloneqq \{x_I : x \in A\}$ and $B_I \coloneqq \{y_I : y \in B\}$ be the *projections* onto the coordinates of I. The *min-entropy* of a random variable $\boldsymbol{x}$ is $\mathbf{H}_\infty(\boldsymbol{x}) \coloneqq \min_x \log(1/\Pr[\boldsymbol{x} = x])$. We say A is *δ-dense* if the uniform random variable $\boldsymbol{x}$ over A satisfies the following: for every nonempty $I \subseteq [n]$, $\mathbf{H}_\infty(\boldsymbol{x}_I) \geq \delta|I|\log m$ (that is, the min-entropy of the marginal distribution of $\boldsymbol{x}$ on coordinates I is at least a δ fraction of the maximum possible for a distribution over $[m]^I$). The *deficiency* of B is $\mathbf{D}_\infty(B) \coloneqq mn - \log|B|$.

▶ **Lemma 20** ([10, Lemma 11]). *If $A \subseteq [m]^n$ is 0.8-dense and $B \subseteq (\{0,1\}^m)^n$ has deficiency at most n^4, then $G(A \times B) = \{0,1\}^n$, that is, for every $z \in \{0,1\}^n$ there are $x \in A$ and $y \in B$ with $G(x,y) = z$.*

Here the density parameter is $\delta = 0.8$ and the deficiency is $\mathbf{D}_\infty(B) \leq n^4$, instead of $\delta = 0.9$ and $\mathbf{D}_\infty(B) \leq n^2$ as in the original. Lemma 20 still holds because our gadget size has increased: we use $m \coloneqq n^{20}$, whereas [10] used $m \coloneqq n^4$. This can be verified by a simple substitution in the proof.

The next lemma is altered somewhat from the original. For a proof, see the full version [27].

▶ **Lemma 21** (A more general version of [10, Claim 12]). *Let $\mathcal{X} \subseteq [m]^n$ be 0.85-dense. If $A' \subseteq \mathcal{X}$ satisfies $|A'| \geq |\mathcal{X}|/2^{k+1}$ then there exist an $I \subseteq [n]$ of size $|I| < 20(k+1)/\log m$ and an $A \subseteq A'$ such that A is fixed on coordinates I and 0.8-dense on all other coordinates.*

5.2.2 The simulation

We exhibit an algorithm that takes a q-alternating rectangle decision list $\mathcal{L}_{\mathrm{R}}$ for $f \circ g^n$ of cost k, and converts it to a q-alternating conjunction decision list $\mathcal{L}_{\mathrm{C}}$ for f of cost $O(k/\log n)$. The argument from [10] does exactly this except without preserving the bound on the number of alternations. In [10] the argument is formulated using a "dual" characterization of $\mathsf{DL}^{\mathsf{dt}}$, but it has the effect of building $\mathcal{L}_{\mathrm{C}}$ in order, obtaining each conjunction by "extracting" it from one of the rectangles in $\mathcal{L}_{\mathrm{R}}$. The trouble is that the rectangles are not necessarily "extracted from" in order: after extracting a conjunction from some rectangle, the *next* conjunction that gets put in $\mathcal{L}_{\mathrm{C}}$ may be extracted from a rectangle that is *earlier* in $\mathcal{L}_{\mathrm{R}}$. Thus $\mathcal{L}_{\mathrm{C}}$ may end up with more alternations than $\mathcal{L}_{\mathrm{R}}$.

To fix this, we convert the argument to a "primal" form and argue that it still works when we force the rectangles to be extracted from in order. The high-level view is that we iterate through the rectangles of $\mathcal{L}_{\mathrm{R}}$ in order, and for each we extract as many conjunctions as we can until the rectangle becomes "exhausted", at which time we remove the remaining "error" portion of the rectangle (by deleting few rows and columns) and move on to the next rectangle. With this modification, the rest of the technical details from [10] continue to work, and it now preserves the number of alternations.

At any step of this process, we let $X \times Y$ be the remaining rows and columns (after having removed the error portion of all previous rectangles in $\mathcal{L}_{\mathrm{R}}$), and we let $Z \subseteq \{0,1\}^n$ be the remaining inputs to f (which have not been accepted by any previous conjunctions we put in $\mathcal{L}_{\mathrm{C}}$). Suppose (R_i, ℓ_i) is our current entry in $\mathcal{L}_{\mathrm{R}}$. The goal is to find a subrectangle $A \times B \subseteq R_i \cap (X \times Y)$ that is "conjunction-like" in the sense that $G(A \times B)$ is exactly the inputs accepted by some small-width conjunction C, and such that among all remaining inputs $z \in Z$, C only accepts those with $f(z) = \oplus(\ell_i)$. These properties would ensure it is safe to put (C, ℓ_i) next in $\mathcal{L}_{\mathrm{C}}$.

Algorithm 1 Simulation algorithm.

```
     In: $\mathcal{L}_{\mathrm{R}} = (R_1, \ell_1), \ldots, (R_{2^k}, \ell_{2^k})$ and $\mathcal{X} \subseteq [m]^n$, $\mathcal{Y} \subseteq (\{0,1\}^m)^n$
     Out: $\mathcal{L}_{\mathrm{C}}$
 1:  initialize $X \leftarrow \mathcal{X}$, $Y \leftarrow \mathcal{Y}$, $Z \leftarrow$ domain of $f$, $\mathcal{L}_{\mathrm{C}} \leftarrow$ empty list
 2:  for $i = 1$ to $2^k$ do
 3:      while $Z \neq \emptyset$ do
 4:          for each $x \in X$, let $Y_x := \{y \in Y : (x,y) \in R_i \cap G^{-1}(Z)\}$
 5:          let $A' := \{x \in X : |Y_x| \geq 2^{mn-n^4}\}$
 6:          if $|A'| \leq |\mathcal{X}|/2^{k+1}$ then
 7:              update $X \leftarrow X \smallsetminus A'$
 8:              update $Y \leftarrow Y \smallsetminus \bigcup_{x \in X \smallsetminus A'} Y_x$
 9:              break out of inner loop
10:          else $|A'| > |\mathcal{X}|/2^{k+1}$
11:              let $A \subseteq A'$, $I \subseteq [n]$, $\alpha \in [m]^I$ be such that:
12:                  $|I| = O(k/\log n)$, $A_I$ is fixed to $\alpha$, and $A_{[n] \smallsetminus I}$ is 0.8-dense
13:              pick any $x' \in A$ and choose $\beta \in (\{0,1\}^m)^I$ to maximize $|\{y \in Y_{x'} : y_I = \beta\}|$
14:              let $C$ be the conjunction "$z_I = g^I(\alpha, \beta)$"
15:              update $\mathcal{L}_{\mathrm{C}}$ by appending $(C, \ell_i)$ to it
16:              update $Z \leftarrow Z \smallsetminus C^{-1}(1)$
```

Combining Lemma 20 and Lemma 21 (using $\mathcal{X} = [m]^n$) suggests an approach for finding a conjunction-like subrectangle: If A' is not too small, we can restrict it to A that is fixed on few coordinates I and dense on the rest (by Lemma 21). If B is also not too small (low deficiency) and fixed on coordinates I, then $G(A \times B)$ is fixed on I and takes on all possible values on the remaining coordinates (by Lemma 20, which still works with $[n] \smallsetminus I$ in place of $[n]$). In other words, $G(A \times B) = C^{-1}(1)$ for a small-width conjunction C, as desired.

The other property we needed to ensure is that if this C is the first conjunction in $\mathcal{L}_{\mathrm{C}}$ to accept a particular z, then $f(z) = \oplus(\ell_i)$ (so $\mathcal{L}_{\mathrm{C}}$ is correct). This will follow if we know there is some $(x,y) \in G^{-1}(z)$ such that R_i is the first rectangle in $\mathcal{L}_{\mathrm{R}}$ to contain (x,y), as that guarantees $f(z) = f(G(x,y)) = (f \circ g^n)(x,y) = \oplus(\ell_i)$. It turns out this will hold automatically if $A \times B \subseteq R_i \cap (X \times Y)$, because $A \times B$ touches the slice of every z that is accepted by C, and all inputs $(x,y) \in G^{-1}(Z)$ that were in some R_j with $j < i$ have already been removed from $X \times Y$.

Our algorithm for building $\mathcal{L}_{\mathrm{C}}$ from $\mathcal{L}_{\mathrm{R}}$ is shown in Algorithm 1. It is described as starting from some arbitrary initial rectangle $\mathcal{X} \times \mathcal{Y}$. For the purpose of proving Theorem 4.*(i)* we only need to take $\mathcal{X} = [m]^n$ and $\mathcal{Y} = (\{0,1\}^m)^n$, but when we invoke this as a component in the proof of Theorem 4.*(ii)* we will need to start from some $\mathcal{X} \times \mathcal{Y}$ that is merely "dense × large" rather than the full input domain, so we state this more general version now.

▶ **Lemma 22.** *If $\mathcal{L}_{\mathrm{R}}$ computes $f \circ g^n$ on $\mathcal{X} \times \mathcal{Y}$ and has cost k, and if $\mathcal{X}$ is 0.85-dense and $\mathbf{D}_\infty(\mathcal{Y}) \leq n^3$, then $\mathcal{L}_{\mathrm{C}}$ produced by Algorithm 1 computes f and has cost $O(k/\log n)$. Moreover, if $\mathcal{L}_{\mathrm{R}}$ is q-alternating then so is $\mathcal{L}_{\mathrm{C}}$.*

Proof. To verify the cost, just note that lines 11 and 12 always succeed by Lemma 21 (since $\mathcal{X}$ is 0.85-dense and $|A'| \geq |\mathcal{X}|/2^{k+1}$), so when a conjunction is added to $\mathcal{L}_{\mathrm{C}}$ on lines 14 and 15, it has width $|I| < 20(k+1)/\log m = O(k/\log n)$. On line 13, defining $B := \{y \in Y_{x'} : y_I = \beta\}$ we have $|B| \geq |Y_{x'}|/2^{m|I|} \geq 2^{mn-n^4-m|I|} = 2^{m(n-|I|)-n^4}$ (since $x' \in A \subseteq A'$) and therefore

$\mathbf{D}_\infty(B_{[n]\smallsetminus I}) \le n^4$ (relative to $(\{0,1\}^m)^{[n]\smallsetminus I}$). Thus by applying Lemma 20 to $A_{[n]\smallsetminus I}$ (which is 0.8-dense) and $B_{[n]\smallsetminus I}$ we have $g^{[n]\smallsetminus I}(A_{[n]\smallsetminus I} \times B_{[n]\smallsetminus I}) = \{0,1\}^{n-|I|}$ and therefore $G(A \times B) = C^{-1}(1)$. (Lemma 20 works with the same parameters even though the sets are now on fewer than n coordinates.)

The algorithm terminates because Z always shrinks on line 16: for any $y \in B$ we have $G(x', y) \in Z$ (from the definition of $Y_{x'}$) and $C(G(x', y)) = 1$ (since $x'_I = \alpha$ and $y_I = \beta$ and thus $G(x', y)_I = g^I(\alpha, \beta)$).

The algorithm maintains the invariant that for all $j < i$, $R_j \cap (X \times Y) \cap G^{-1}(Z) = \emptyset$. This vacuously holds at the beginning, and is clearly maintained in the **else** case because i stays the same and nothing gets added to X, Y, or Z. Lines 7 and 8 maintain the invariant in the **if** case because the removed rows and columns cover all of $R_i \cap (X \times Y) \cap G^{-1}(Z)$ and i goes up by 1.

Next we argue that when the algorithm terminates, Z must be empty. In each iteration of the outer loop, we throw out at most $|\mathcal{X}|/2^{k+1}$ rows and at most $|\mathcal{X}| \cdot 2^{mn-n^4} \le m^n \cdot 2^{mn-n^4} \le 2^{mn-n^3}/2^{k+1} \le |\mathcal{Y}|/2^{k+1}$ columns. (We throw out columns in Y_x for $x \notin A'$, all of these Y_x had the property $|Y_x| < 2^{mn-n^4}$, we do this for at most $|\mathcal{X}|$ values of x, and $n^4 - n \log m \ge n^3 + k + 1$.) Since the outer loop executes 2^k times, by the end at most half the rows of $\mathcal{X}$ and half the columns of $\mathcal{Y}$ have been discarded, so $|X| \ge |\mathcal{X}|/2$ and $|Y| \ge |\mathcal{Y}|/2$. This means X is essentially as dense as $\mathcal{X}$ (only a -1 loss in any $\mathbf{H}_\infty(\boldsymbol{x}_I)$) and Y is essentially as low-deficiency as $\mathcal{Y}$ (only a $+1$ loss in $\mathbf{D}_\infty$). Thus Lemma 20 (with a tiny perturbation of the parameters, which does not affect the result) shows that $G(X \times Y) = \{0,1\}^n$. However, the last rectangle that is processed, R_{2^k}, contains all of $\mathcal{X} \times \mathcal{Y}$ by definition (since we assume $\mathcal{L}_\mathrm{R}$ is correct on $\mathcal{X} \times \mathcal{Y}$). So, the invariant guarantees $(X \times Y) \cap G^{-1}(Z) = \emptyset$ at termination. This can only happen if $G^{-1}(Z) = \emptyset$ and thus $Z = \emptyset$ (since $G(X \times Y) = \{0,1\}^n$).

We now argue that $\mathcal{L}_\mathrm{C}$ is correct. Consider any z in the domain of f. Since Z is empty at termination, z must be accepted by some conjunction in $\mathcal{L}_\mathrm{C}$. Let (C, ℓ_i) be the first entry such that $C(z) = 1$, so $z \in Z$ during the iteration of the inner loop when this entry was added. Since in this iteration we have $G(A \times B) = C^{-1}(1)$ and $z \in C^{-1}(1)$, there is some $(x, y) \in A \times B$ with $G(x, y) = z$. Since $A \times B \subseteq R_i$ we have $(x, y) \in R_i$. Since $A \times B \subseteq X \times Y$, we have $(x, y) \in (X \times Y) \cap G^{-1}(Z)$ and thus (x, y) cannot be in R_j for any $j < i$ since $R_j \cap (X \times Y) \cap G^{-1}(Z) = \emptyset$ by the invariant. In summary, R_i is the first rectangle in $\mathcal{L}_\mathrm{R}$ that contains (x, y). By correctness of $\mathcal{L}_\mathrm{R}$ on $\mathcal{X} \times \mathcal{Y}$, we have $\oplus(\ell_i) = (f \circ g^n)(x, y) = f(G(x, y)) = f(z)$. Thus $\mathcal{L}_\mathrm{C}$ also correctly outputs $\oplus(\ell_i)$ on input z.

The "moreover" part is straightforward to verify: the levels assigned to conjunctions in $\mathcal{L}_\mathrm{C}$ come from the levels assigned to rectangles in $\mathcal{L}_\mathrm{R}$ (namely $\{0, \ldots, q\}$), in the same order (which is non-increasing). ◀

References

1 Josh Alman and Ryan Williams. Probabilistic rank and matrix rigidity. In *Proceedings of the 49th Symposium on Theory of Computing (STOC)*, pages 641–652. ACM, 2017. doi:10.1145/3055399.3055484.

2 László Babai, Peter Frankl, and Janos Simon. Complexity classes in communication complexity theory. In *Proceedings of the 27th Symposium on Foundations of Computer Science (FOCS)*, pages 337–347. IEEE, 1986. doi:10.1109/SFCS.1986.15.

3 László Babai, Noam Nisan, and Mario Szegedy. Multiparty protocols and logspace-hard pseudorandom sequences. In *Proceedings of the 21st Symposium on Theory of Computing (STOC)*, pages 1–11. ACM, 1989. doi:10.1145/73007.73008.

4 Ziv Bar-Yossef, T.S. Jayram, Ravi Kumar, and D. Sivakumar. An information statistics approach to data stream and communication complexity. *Journal of Computer and System Sciences*, 68(4):702–732, 2004. doi:10.1016/j.jcss.2003.11.006.

5 Richard Beigel. Bounded queries to SAT and the Boolean hierarchy. *Theoretical Computer Science*, 84(2):199–223, 1991. doi:10.1016/0304-3975(91)90160-4.

6 Alberto Bertoni, Danilo Bruschi, Deborah Joseph, Meera Sitharam, and Paul Young. Generalized Boolean hierarchies and Boolean hierarchies over RP. In *Proceedings of the 7th International Conference on Fundamentals of Computation Theory (FCT)*, pages 35–46. Springer, 1989. doi:10.1007/3-540-51498-8_4.

7 Jin-Yi Cai and Lane Hemachandra. The Boolean hierarchy: Hardware over NP. In *Proceedings of the 1st Structure in Complexity Theory Conference (STRUCTURES)*, pages 105–124. Springer, 1986. doi:10.1007/3-540-16486-3_93.

8 Arkadev Chattopadhyay, Shachar Lovett, and Marc Vinyals. Equality alone does not simulate randomness. In *Proceedings of the 34th Computational Complexity Conference (CCC)*, pages 14:1–14:11. Schloss Dagstuhl, 2019. doi:10.4230/LIPIcs.CCC.2019.14.

9 Mika Göös. Lower bounds for clique vs. independent set. In *Proceedings of the 56th Symposium on Foundations of Computer Science (FOCS)*, pages 1066–1076. IEEE, 2015. doi:10.1109/FOCS.2015.69.

10 Mika Göös, Pritish Kamath, Toniann Pitassi, and Thomas Watson. Query-to-communication lifting for P^{NP}. *Computational Complexity*, 28(1):113–144, 2019. doi:10.1007/s00037-018-0175-5.

11 Mika Göös, Shachar Lovett, Raghu Meka, Thomas Watson, and David Zuckerman. Rectangles are nonnegative juntas. *SIAM Journal on Computing*, 45(5):1835–1869, 2016. doi:10.1137/15M103145X.

12 Mika Göös, Toniann Pitassi, and Thomas Watson. Zero-information protocols and unambiguity in Arthur–Merlin communication. *Algorithmica*, 76(3):684–719, 2016. doi:10.1007/s00453-015-0104-9.

13 Mika Göös, Toniann Pitassi, and Thomas Watson. Query-to-communication lifting for BPP. In *Proceedings of the 58th Symposium on Foundations of Computer Science (FOCS)*, pages 132–143. IEEE, 2017. doi:10.1109/FOCS.2017.21.

14 Mika Göös, Toniann Pitassi, and Thomas Watson. Deterministic communication vs. partition number. *SIAM Journal on Computing*, 47(6):2435–2450, 2018. doi:10.1137/16M1059369.

15 Mika Göös, Toniann Pitassi, and Thomas Watson. The landscape of communication complexity classes. *Computational Complexity*, 27(2):245–304, 2018. doi:10.1007/s00037-018-0166-6.

16 Bernd Halstenberg and Rüdiger Reischuk. Relations between communication complexity classes. In *Proceedings of the 3rd Structure in Complexity Theory Conference (STRUCTURES)*, pages 19–28. IEEE, 1988. doi:10.1109/SCT.1988.5259.

17 Bernd Halstenberg and Rüdiger Reischuk. Relations between communication complexity classes. *Journal of Computer and System Sciences*, 41(3):402–429, 1990. doi:10.1016/0022-0000(90)90027-I.

18 Bernd Halstenberg and Rüdiger Reischuk. Different modes of communication. *SIAM Journal on Computing*, 22(5):913–934, 1993. doi:10.1137/0222057.

19 Stasys Jukna. *Boolean Function Complexity: Advances and Frontiers*, volume 27 of *Algorithms and Combinatorics*. Springer, 2012. doi:10.1007/978-3-642-24508-4.

20 Jim Kadin. The polynomial time hierarchy collapses if the Boolean hierarchy collapses. *SIAM Journal on Computing*, 17(6):1263–1282, 1988. doi:10.1137/0217080.

21 Bala Kalyanasundaram and Georg Schnitger. The probabilistic communication complexity of set intersection. *SIAM Journal on Discrete Mathematics*, 5(4):545–557, 1992. doi:10.1137/0405044.

22 Hartmut Klauck. Rectangle size bounds and threshold covers in communication complexity. In *Proceedings of the 18th Conference on Computational Complexity (CCC)*, pages 118–134. IEEE, 2003. doi:10.1109/CCC.2003.1214415.

23 Johannes Köbler, Uwe Schöning, and Klaus Wagner. The difference and truth-table hierarchies for NP. *Theoretical Informatics and Applications*, 21(4):419–435, 1987. doi:10.1051/ita/1987210404191.

24 Eyal Kushilevitz and Noam Nisan. *Communication Complexity*. Cambridge University Press, 1997.

25 Nati Linial, Toniann Pitassi, and Adi Shraibman. On the communication complexity of high-dimensional permutations. In *Proceedings of the 10th Innovations in Theoretical Computer Science Conference (ITCS)*, pages 54:1–54:20. Schloss Dagstuhl, 2019. doi:10.4230/LIPIcs.ITCS.2019.54.

26 Periklis Papakonstantinou, Dominik Scheder, and Hao Song. Overlays and limited memory communication. In *Proceedings of the 29th Conference on Computational Complexity (CCC)*, pages 298–308. IEEE, 2014. doi:10.1109/CCC.2014.37.

27 Toniann Pitassi, Morgan Shirley, and Thomas Watson. Nondeterministic and randomized boolean hierarchies in communication complexity. Technical Report TR19-043, Electronic Colloquium on Computational Complexity (ECCC), 2019. URL: https://eccc.weizmann.ac.il/report/2019/043.

28 Ran Raz and Pierre McKenzie. Separation of the monotone NC hierarchy. *Combinatorica*, 19(3):403–435, 1999. doi:10.1007/s004930050062.

29 Alexander Razborov. On rigid matrices. Technical report, Steklov Mathematical Institute, 1989. In Russian.

30 Alexander Razborov. On the distributional complexity of disjointness. *Theoretical Computer Science*, 106(2):385–390, 1992. doi:10.1016/0304-3975(92)90260-M.

31 Ronald Rivest. Learning decision lists. *Machine Learning*, 2(3):229–246, 1987. doi:10.1007/BF00058680.

32 Klaus Wagner. Bounded query computations. In *Proceedings of the 3rd Structure in Complexity Theory Conference (STRUCTURES)*, pages 260–277. IEEE, 1988. doi:10.1109/SCT.1988.5286.

33 Thomas Watson. A $\mathrm{ZPP}^{\mathrm{NP}[1]}$ lifting theorem. In *Proceedings of the 36th International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 59:1–59:16. Schloss Dagstuhl, 2019. doi:10.4230/LIPIcs.STACS.2019.59.

34 Gerd Wechsung. On the Boolean closure of NP. In *Proceedings of the 5th International Conference on Fundamentals of Computation Theory (FCT)*, pages 485–493. Springer, 1985. doi:10.1007/BFb0028832.

35 Andrew Yao. Some complexity questions related to distributive computing. In *Proceedings of the 11th Symposium on Theory of Computing (STOC)*, pages 209–213. ACM, 1979. doi:10.1145/800135.804414.

36 Stathis Zachos and Hans Heller. A decisive characterization of BPP. *Information and Control*, 69(1-3):125–135, 1986. doi:10.1016/S0019-9958(86)80044-4.

A Spectral Bound on Hypergraph Discrepancy

Aditya Potukuchi

Department of Computer Science, Rutgers University, New Brunswick, NJ, USA
`https://www.adityapotukuchi.com/`
aditya.potukuchi@cs.rutgers.edu

Abstract

Let $\mathcal{H}$ be a t-regular hypergraph on n vertices and m edges. Let M be the $m \times n$ incidence matrix of $\mathcal{H}$ and let us denote $\lambda = \max_{v \in \mathbf{1}^\perp} \frac{1}{\|v\|}\|Mv\|$. We show that the discrepancy of $\mathcal{H}$ is $O(\sqrt{t} + \lambda)$. As a corollary, this gives us that for every t, the discrepancy of a random t-regular hypergraph with n vertices and $m \geq n$ edges is almost surely $O(\sqrt{t})$ as n grows. The proof also gives a polynomial time algorithm that takes a hypergraph as input and outputs a coloring with the above guarantee.

2012 ACM Subject Classification Mathematics of computing → Discrete mathematics

Keywords and phrases Hypergraph discrepancy, Spectral methods, Beck-Fiala conjecture

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.93

Category Track A: Algorithms, Complexity and Games

Funding *Aditya Potukuchi*: Research supported in part by NSF grants CCF-1540634 and CCF-1814409.

Acknowledgements I am extremely thankful to Jeff Kahn for suggesting the problem and to Shachar Lovett for the discussions that led to a refinement over a previous version. I would also like to thank Huseyin Acan and Cole Franks for the very helpful discussions. I would also like to thank the anonymous ICALP referees for the numerous detailed comments and suggestions.

1 Introduction

The main aim of this paper is to give a spectral condition that is sufficient for the discrepancy of a regular hypergraph to be small. This is proved via the partial coloring approach while using some combinatorial properties of the hypergraph that are given by this spectral condition. This immediately implies, via an old proof technique of Kahn and Szemerédi, that for every t, the discrepancy of a random t-regular hypergraph on n vertices and $m \geq n$ edges is almost surely $O\left(\sqrt{t}\right)$. Previously, a result of this form was proved by Ezra and Lovett [11] who show that the discrepancy of a random t-regular hypergraph on n vertices and $m \geq n$ edges is $O(\sqrt{t \log t})$ almost surely as t grows. More recently, Bansal and Meka [3] showed that for random t-regular hypergraphs on n vertices and m edges, the discrepancy is $O\left(\sqrt{t}\right)$ almost surely provided $t = \Omega\left((\log\log m)^2\right)$. To state our result formally, we make some definitions.

Let $\mathcal{H} = (V, E)$ be a hypergraph, with V as the set of vertices, and $E \subseteq 2^V$ as the set of (hyper)edges. Let $\mathcal{X} = \{\chi : V \to \{\pm 1\}\}$, be the set of ± 1 colorings of V, and for $\chi \in \mathcal{X}$, and $e \in E$, denote $\chi(e) := \sum_{v \in e} \chi(v)$. The discrepancy of $\mathcal{H}$, denoted by $\text{disc}(\mathcal{H})$ is defined as:

$$\text{disc}(\mathcal{H}) := \min_{\chi \in \mathcal{X}} \max_{e \in E} |\chi(e)|.$$

We call a hypergraph t-*regular* if every vertex is present in exactly t hyperedges. These will be the main focus of this paper. For a hypergraph $\mathcal{H}$, let $M = M(\mathcal{H})$ be the $|E| \times |V|$ incidence matrix of $\mathcal{H}$, i.e., M has rows indexed by E, columns indexed by V, and entries are $M(e, v) = 1$ if $v \in e$ and 0 otherwise. We will use $\|\cdot\|$ to denote the Euclidean norm throughout the paper. Our main result is the following:

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 93; pp. 93:1–93:14
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

▶ **Theorem 1.** *Let $\mathcal{H}$ be a t-regular hypergraph on n vertices and m edges with M as its incidence and let $\lambda = \max_{v \perp \bar{1}, \|v\|=1} \|Mv\|$. Then*

$$\operatorname{disc}(\mathcal{H}) = O\left(\sqrt{t} + \lambda\right).$$

Moreover, there is an $\tilde{O}((\max\{n, m\})^7)$ time algorithm that takes the hypergraph $\mathcal{H}$ as input and outputs the coloring with the above guarantee.

1.1 Background

The study of hypergraph discrepancy, which seems to have been first defined in a paper of Beck [4], has led to some very interesting results with diverse applications (see, for example [18, 9]). One of the most interesting open problems in discrepancy theory is what is commonly known as the Beck-Fiala conjecture, regarding the discrepancy of general t-regular hypergraphs.

▶ **Conjecture 2** (Beck-Fiala conjecture). *For a t-regular hypergraph $\mathcal{H}$, we have*

$$\operatorname{disc}(\mathcal{H}) = O(\sqrt{t}).$$

Although this conjecture is usually stated for *bounded degree* hypergraphs (as opposed to regular ones), this is not really an issue. One can always add hyperedges containing just a single vertex and make it regular, which increases the discrepancy of the original hypergraph by at most one. Beck and Fiala [5] also proved that for any t-regular hypergraph $\mathcal{H}$,

$$\operatorname{disc}(\mathcal{H}) \leq 2t - 1.$$

This is more commonly known as the Beck-Fiala theorem. Essentially the same proof can be done a bit more carefully to get a bound of $2t - 3$ (see [6]). Given Conjecture 2, it is perhaps surprising that the best upper bound, due to Bukh [8], is "stuck at" $2t - \log^* t$ for large enough t.

It is possible that one of the reasons that the discrepancy upper bounds are so far away from the conjectured bound (assuming it's true) is our inability to handle many "large" hyperedges. Indeed, if one is offered the restriction that each hyperedge is also of size $O(t)$ (regular and "almost uniform"), then a folklore argument using the Lovász Local Lemma shows that the discrepancy is bounded by $O(\sqrt{t \log t})$. The proof of Theorem 1 also relies on being able to avoid dealing with large edges (which are few, if any, in number).

1.2 Discrepancy in random settings

Motivated by the long-standing open problem of bounding discrepancy of general t-regular hypergraphs, Ezra and Lovett [11] initiated the study of discrepancy of *random* t-regular hypergraphs. By random t-regular hypergraph, we mean the hypergraph sampled by the following procedure: We fix n vertices V and m (initially empty) hyperedges E. Each vertex in V chooses t (distinct) hyperedges in E uniformly and independently to be a part of. They showed that if $m \geq n$, then the discrepancy of such a hypergraph is almost surely $O(\sqrt{t \log t})$ as t grows. The proof idea is the following: First observe that most of the hyperedges have size $O(t)$. For the remaining large edges, one can delete one vertex from every hyperedge and make them pairwise disjoint. This allows one to apply a folklore Lovász Local Lemma based argument, but with a slight modification which makes sure that the large edges have discrepancy at most 2. More recently, Bansal and Meka [3] reduced the discrepancy bound to

$O(\sqrt{t})$ almost surely as long as $t = \Omega\left((\log\log n)^2\right)$ for all m and n. A corollary of Theorem 1 states that one can get the bound of $O(\sqrt{t})$ for every (not necessarily growing) $t = t(n)$ as n grows and $m \geq n$. More formally,

▶ **Corollary 3.** *There is an absolute constant $C > 0$ such that the following holds: Let $\mathcal{H}_t$ be a random t-regular hypergraph on n vertices and $m \geq n$ hyperedges where $t = o(\sqrt{m})$. Then,*

$$\mathbb{P}\left(\operatorname{disc}(\mathcal{H}_t) \leq C\sqrt{t}\right) \geq 1 - o(1)$$

The theorem that implies Corollary 3 from Theorem 1 is the following:

▶ **Theorem 4.** *Let M be the incidence matrix of a random t-regular set system on n vertices, where $t = o(\sqrt{m})$, and $m \geq n$ edges. Then with probability at least $1 - n^{\Omega(1)}$,*

$$\max_{v \perp \bar{1}, \|v\|=1} \|Mv\| = O\left(\sqrt{t}\right).$$

A couple of remarks here: First, observe that it suffices to prove Theorem 4 for $m = n$. Indeed, let M and N be random $m \times m$ and $m \times n$ random matrices (m > n) respectively distributed by choosing t random 1's in each column independently. Notice that the distribution of N is exactly the same as that of the first n columns of M. Then, setting M_n to be the matrix consisting of the first n columns of M, we observe that $\lambda(M_n) \leq \lambda(M)$. Second, we point out that $t = o(\sqrt{m})$ is just a limitation of the proof technique in [13] (also see [7]) that we use to prove this theorem. Although we believe that Theorem 4 should hold for all $t < m$, we do not make any attempt to verify this, especially since the result of Bansal and Meka [3] already takes care of the discrepancy of random hypergraphs in this case. Although many variations of Theorem 4 are known and standard, one needs to verify it for our setting too. It should come as no surprise that the proof follows that of Kahn and Szemerédi's [1] in [13], which is postponed to Section 3.2.

1.3 The partial coloring approach

Most of the bounds and algorithms on hypergraph discrepancy proceed via a *partial coloring approach*. In general, a partial coloring approach [4] works by coloring a fraction of the (still uncolored) vertices in each step, while ensuring that no edge has discrepancy more than the desired bound. Perhaps the most famous successful application of this is Spencer's celebrated "six standard deviations" result [22], which gives a bound of $6\sqrt{n}$ for any hypergraph on n vertices and n edges. The original proof of Spencer was not algorithmic, i.e., it did not give an obvious way to take as input a hypergraph on n vertices and n edges, and efficiently output a coloring that achieves discrepancy $O(\sqrt{n})$. In fact, Alon and Spencer([1], §14.5) suggested that such an algorithm is not possible. However, this was shown to be incorrect by Bansal [2] who showed an efficient algorithm to do the same task. However, the analysis of this algorithm still relied on the (non-algorithmic) discrepancy bound of $6\sqrt{n}$. Later, Lovett and Meka [17] gave a "truly constructive" proof of the fact that the discrepancy is $O(\sqrt{n})$. This proof did not rely on any existing discrepancy bounds and the novel and simple analysis proved to be extremely influential. The proof of Theorem 1 will rely on a somewhat technical feature of the main partial coloring from this work. More recently, a result due to Rothvoss [21] gives a simpler proof of the same $O(\sqrt{n})$ bound, which is also constructive, and more general.

[1] [13] is combination of two papers that prove the same result upto a constant factor: one by Friedman using the so-called trace method, and the other by Kahn and Szemerédi using a more combinatorial approach which is flexible enough to be easily adapted here.

1.4 Proof sketch

The proof of Theorem 1 is proved via the aforementioned partial coloring approach. The main source of inspiration is a later paper of Spencer [23], which computes the discrepancy of the projective plane (i.e., the hypergraph where the vertices are the points and the hyperedges are the lines of $\mathrm{PG}(2,q)$) upto a constant factor. A more general bound was also obtained by Matoušek [19], who upper bounds the discrepancy of set systems of bounded VC-dimension (note that the projective plane has VC-dimension 2).

We also use the aforementioned result of Lovett and Meka [17] heavily, in particular, the partial coloring theorem. Informally, this says that one can "color" roughly an α fraction of the hypergraph with real numbers in $[-1,1]$ so that (1) at least half the vertices get colors 1 or -1 and (2) every edge e has discrepancy $O(\sqrt{e})$. We now sketch the proof.

Consider the following "dream approach" using partial coloring: In every step, one colors an α fraction of vertices. Suppose that at the start, every edge has size $O(t)$ and that each step of partial coloring colors exactly an α fraction of the remaining uncolored vertices (i.e., these vertices are colored from $\{-1,1\}$). Then the discrepancy of an edge e is at most $O\left(\sum_i \sqrt{\alpha^i|e|}\right) = O(\sqrt{t})$. Of course, this is too much to hope for, since some edges can potentially be large, and more importantly, there is no guarantee on how much of each edge gets colored in this partial coloring procedure.

This is precisely where the spectral condition on M saves us. One can establish standard combinatorial "pseudorandomness" properties of $\mathcal{H}$ in terms of λ. In particular, if λ is small, then an α fraction of $V(\mathcal{H})$ take up an α fraction of most edges. This means, intuitively, that in the partial coloring approach, if one colors an α fraction of the vertices, then most of the edge sizes will have also reduced by an α fraction. The partial coloring method of Lovett and Meka (and, curiously, none of the older ones) also allows one to color in such a way that $\Omega(n)$ edges can be made to have discrepancy zero in each step. This allows one to maintain that in every round of the partial coloring, the edges that don't behave according to the "dream approach", i.e., those that are too large (i.e., $\Omega(t)$) or don't reduce by an α fraction can be made to have discrepancy *zero* in the next step. Thus, most other edges reduce in size by an α fraction. This lets one not have to deal with the discrepancy of these "bad" edges until they become small.

2 Proof of Theorem 1

2.1 Preliminaries and notation

We will need the aforementioned partial coloring theorem due to Lovett and Meka:

▶ **Theorem 5** ([17]). *Given a family of sets $M_1,\dots,M_m \subseteq [n]$, a vector $x_0 \in [-1,1]^n$, positive real numbers $c_1,\dots,c_m$ such that $\sum_{i\in[m]} \exp\left(-c_i^2/16\right) \le n/16$, and a real number $\delta \in [0,1]$, there is a vector $x \in [-1,1]^n$ such that:*

1. *For all $i \in [m]$, $\langle x - x_0, \mathbb{1}_{M_i}\rangle \le c_i\sqrt{|M_i|}$.*
2. *$|x_i| \ge 1-\delta$ for at least $n/2$ values of i.*

Moreover, this vector x can be found in $\tilde{O}((m+n)^3\delta^{-2})$ time.

Lovett and Meka initially gave a randomized algorithm for the above. It has since been made deterministic [16].

2.1.1 A technical remark

The reason we use the Lovett-Meka partial coloring, as opposed to Beck's partial coloring is not just the algorithmic aspect that the former offers, but also because it also offers the technical condition:

$$\sum_{i\in[m]} \exp\left(-c_i^2/16\right) \leq n/16.$$

This means one can set $\Omega(n)$ edges to have discrepancy 0. To compare, we first state Beck's partial coloring lemma (for reference, see [18]):

▶ **Theorem 6** (Beck's partial coloring lemma). *Given a family of sets $M_1, \ldots, M_m \subseteq [n]$, and positive real numbers $c_1, \ldots, c_m$ such that $\sum_{i\in[m]} g(c_i) \leq n/5$, where*

$$g(x) = \begin{cases} e^{-x^2/9} & x > 0.1 \\ \ln(1/x) & x \leq 0.1 \end{cases}$$

there is a vector $x \in \{-1, 0, 1\}^n$ such that:

1. *For all $i \in [m]$, $\langle x, \mathbb{1}_{M_i}\rangle \leq c_i\sqrt{|M_i|}$.*
2. *$|x_i| = 1$ for at least $n/2$ values of i.*

If one ignores the algorithmic aspect, Beck's partial coloring, while assigning vertices to $\{-1, 1, 0\}$ (instead of $[-1, 1]$, thus making it a "partial coloring" in the true sense) only guarantees that $\Omega\left(\frac{n}{\log t}\right)$ edges can be made to have discrepancy 0. Although [17] did not really need this particular advantage, they do mention that this feature could potentially be useful elsewhere. This seemingly subtle advantage turns out to be crucial in the proof of Theorem 1, where we set $\Omega(n)$ edges (that will be called "bad" and "dormant" edges) to have discrepancy 0.

Henceforth, let V and E denote the vertices and edges of our hypergraph respectively. We will need a "pseudorandomness" lemma that informally states that an α fraction of vertices takes up around an α fraction of most edges:

▶ **Lemma 7.** *For any $S \subseteq V$ with $|S| = \alpha n$ where $\alpha \in (0, 1)$ and a positive real number K, there is a subset $E' \subset E$ of size at most $K^{-2} \cdot \alpha n$ such that for every $e \notin E'$, we have $||e \cap S| - \alpha|e|| \leq K\lambda$, where $\lambda = \max_{v\perp\bar{\mathbf{1}}, \|v\|=1} \|Mv\|$.*

Proof. Consider a vector $v \in \mathbb{R}^n$ where $v(i) = 1 - \alpha$ for $i \in S$ and $-\alpha$ otherwise. Clearly, $v \in \mathbf{1}^\perp$ and so

$$\|Mv\|^2 \leq \lambda^2 \cdot \|v\|^2 = \lambda^2\alpha(1-\alpha)n. \tag{1}$$

On the other hand, $Mv(e) = (1-\alpha)|e \cap S| - \alpha|e \setminus S| = |e \cap S| - \alpha|e|$, and so

$$\|Mv\|^2 = \sum_e (|e \cap S| - \alpha|e|)^2. \tag{2}$$

Putting (1) and (2) together, we get that there at most $K^{-2} \cdot \alpha n$ edges e such that $||e \cap S| - \alpha|e|| \geq K\lambda$. ◀

Since this proof is via partial coloring, let us use i to index the steps of the partial coloring. For a partial coloring $\chi : V \to [-1,1]$, we call the set of vertices u for which $|\chi(u)| < 1$ as *uncolored.* Let us use V^i to denote the still uncolored vertices at step i and for an edge $e \in E$, let us denote $e^i := e \cap V^i$. In every step, we invoke Theorem 5 setting $\delta = \frac{1}{n}$ to get the partial coloring, so will have $|V^i| \leq 2^{-i}n$. Let $t' := \max\{t, \lambda\}$ [2].

We call an edge *dormant* at step i if $|e^i| > 100t'$. Let us call an edge *bad* in step i if $\big||e^i| - 2^{-i}|e|\big| \geq 10\lambda$. Edges that are neither dormant not bad are called *good.* Finally, we say that e is *dead* in step i if $|e^i| \leq 100\lambda$.

Informally, the roles of these sets are as follows: In the partial coloring step i, we ensure that an edge e edges only get nonzero discrepancy if it is good, i.e., if $|e^i|$ is close to what is expected and is not too large. Even dead edges can be good or bad, and we will not distinguish them while coloring the vertices. However, in the analysis we will break the total discrepancy accumulated by e into two parts: Before it is dead and after. The main point is to bound the discrepancy gained before it becomes dead. After it becomes dead, we simply bound the discrepancy incurred since by its remaining size, i.e., at most 100λ.

First, we make two easy observations:

▷ Claim 8. If $|V^i| = 2^{-i}n$, then at step i, the number of dormant edges is at most $\frac{1}{100}2^{-i}n$.

Proof. This is just Markov's inequality, using the fact that the average edge size is $\frac{|V^i|t}{m} \leq \frac{|V^i|t'}{m}$. ◁

▷ Claim 9. If $|V^i| = 2^{-i}n$, then at step i, the number of bad edges is at most $\frac{1}{100}2^{-i}n$.

Proof. This is by setting $K = 10$ and $\alpha = 2^{-i}$ in Lemma 7. ◁

2.2 Partial coloring using Lemma 7

Proof of Theorem 1. Setting $V^0 = V$, we proceed by partial coloring that colors exactly half the remaining uncolored vertices at each stage. For a step $i \geq 0$, suppose that $|V^i| = 2^{-i}n$. We will describe a partial coloring given by $\chi_i : V^i \to [-1,1]$ that colors half the vertices of V^i.

For $\ell \geq 1$, let $A_\ell := \{e \in E \mid |e| \in [100 \cdot 2^\ell t', 100 \cdot 2^{\ell+1}t')\}$, and $A_0 := \{e \in E \mid |e| < 200t'\}$. Observe that the edges in A_ℓ for $\ell \geq 1$ are either bad or dormant in steps $i < \ell$. Also observe that $|A_\ell| \leq \frac{2^{-\ell}}{100}n$ for $\ell \geq 1$, Define constants $\{c_e\}_{e \in E}$ as follows:

$$c_e = \begin{cases} 4\sqrt{2\ln\left(\frac{1}{2^{\ell-i}}\right)} & \text{if } e \in A_\ell \text{ for } \ell \geq 1 \text{ is good} \\ 4\sqrt{\ln\left(\frac{200t'}{2^{-i}|e|}\right)} & \text{if } e \in A_0 \text{ is good} \\ 0 & \text{otherwise.} \end{cases}$$

[2] In fact, we may assume w.l.o.g. that $\lambda \leq t$ and so $t' = t$ since in the other case, the Beck-Fiala Theorem gives us that the discrepancy is $O(t) = O(\lambda)$. However, this is not needed and the techniques here also handle this case with this minor change.

Let $\mathcal{B} = \mathcal{B}^i$ and $\mathcal{D} = \mathcal{D}^i$ denote the bad and dormant edges respectively. We handle the edges in A_0 and $E \setminus A_0$ separately. For edges in $E \setminus A_0$, we have:

$$\begin{aligned}\sum_{e \in E \setminus A_0} e^{-\frac{c_e^2}{16}} &\leq \sum_{e \in E \setminus (\mathcal{B} \cup \mathcal{D} \cup A_0)} e^{-\frac{c_e^2}{16}} + |\mathcal{B}| + |\mathcal{D}| \\ &\leq \sum_{1 \leq \ell \leq i} \sum_{e \in A_\ell} e^{2 \ln(2^{\ell - i})} + \frac{2^{-i} n}{50} \\ &= \sum_{1 \leq \ell \leq i} |A_\ell| 2^{2(\ell - i)} + \frac{2^{-i} n}{50} \\ &\leq \frac{n}{100} \sum_{\ell \leq i} 2^{-\ell} \cdot 2^{2\ell - 2i} + \frac{2^{-i} n}{50} \\ &= \frac{2^{-i} n}{100} \sum_{\ell \leq i} 2^{\ell - i} + \frac{2^{-i} n}{50} \\ &\leq \frac{2^{-i} n}{25}.\end{aligned}$$

The second inequality above follows from Claim 8 and Claim 9. For the other case, we have

$$\sum_{e \in A_0} e^{-\frac{c_e^2}{16}} \leq \sum_{e \in A_0} e^{\ln\left(\frac{2^{-i}|e|}{200t}\right)} = \frac{2^{-i}}{200} \sum_{e \in E} \frac{|e|}{t'} = \frac{2^{-i} n}{200}.$$

Here we have used the fact that since the hypergraph is t-regular, we have $\sum_{e \in E} |e| = nt \leq nt'$. Putting these together, we have

$$\sum_{e \in E} e^{-\frac{c_e^2}{16}} \leq \frac{2^{-i} n}{200} + \frac{2^{-i} n}{50} \leq \frac{|V^i|}{20}.$$

Therefore, Theorem 5 guarantees that there is a fractional coloring $\chi_i : V^i \to [-1, 1]$ such that

1. $|\chi_i(v)| \geq 1 - \frac{1}{n}$ for at least half of V^i.
2. All the bad and dormant edges get discrepancy 0.
3. A good and live edge e gets discrepancy at most $c_e \sqrt{|e^i|}$.

Finally, we pick an arbitrary subset of all the vertices v such that $|\chi_i(v)| \geq 1 - \frac{1}{n}$ of size exactly $(1/2) \cdot |V^i|$ and round them to the nearest integer. It is easy to see that since every edge has size at most n, this rounding, over all the steps of the partial coloring adds discrepancy of at most 1 for every edge. This completes step i of the partial coloring and we are left with $2^{-(i+1)} n$ uncolored vertices for the next step.

For an edge e, let i be a round where e had incurred non-zero discrepancy and e^i was not dead. Since only good edges incur nonzero discrepancy, $|e^i| = 2^{-i}|e| \pm 10\lambda$. Since e is also not dead at step i, we must have that $|e^i| \geq 100\lambda$. This gives us that $2^{-i}|e| \geq 90\lambda$ and therefore $(1/2) \cdot 2^{-i}|e| \leq |e_i| \leq 2 \cdot 2^{-i}|e|$. So, if $e \in A_\ell$ where $\ell \geq 1$, the total discrepancy incurred by e at step i without the rounding step is at most

$$4\sqrt{2 \ln(1/2^{j-i}) e^i} \leq 8\sqrt{200 \ln(1/2^{\ell - i}) \cdot (2^{\ell - i}) \cdot t'}.$$

Here, we have used the fact that $|e| \leq 100 \cdot 2^{\ell+1} t'$. If $e \in A_0$, the discrepancy incurred by e at step i without the rounding is at most

$$4\sqrt{2\ln\left(\frac{200t'}{2^{-i}|e|}\right)e^i} \leq 8\sqrt{200\ln(1/2^{-i})\cdot(2^{-i})\cdot t'}.$$

Therefore, the discrepancy of an edge $e \in A_\ell$ for $\ell \geq 0$ until it becomes dead is at most

$$\sum_{i\geq\ell} 8\sqrt{200\ln(1/2^{\ell-i})\cdot(2^{\ell-i})\cdot t'} = O(\sqrt{t'}) = O(\sqrt{t}+\lambda).$$

Here we have used the fact that $t' = \max\{t,\lambda\}$. Finally, rounding the color of every vertex to its nearest integer increases the discrepancy by at most 1. When the edge becomes dead, we simply bound its discrepancy by its size $O(\lambda)$.

It remains to check that each of the $O(\log n)$ stages of partial coloring can be done in time $\tilde{O}((m+n)^3 n^2)$, and the constants $\{c_e\}_{e\in E}$ take $\tilde{O}(mn)$ time to compute at each stage, thus establishing the algorithmic part. ◀

3 Proof of Theorem 4

3.1 A martingale inequality

We will state a martingale inequality that we will use in the proof of Theorem 4. A sequence of random variables $X_0, X_1, \ldots, X_n$ martingale with respect to another sequence of random variables $Z_0, Z_1, \ldots, Z_n$ such that for all $i \in [n-1]$, we have $X_i = f_i(Z_1, \ldots Z_i)$ for some function f_i, and $\mathbf{E}[X_{i+1}|Z_i, \ldots, Z_1] = X_i$.

A martingale is said to have the C-bounded difference property if $|X_{i+1} - X_i| \leq C$.

The variance of a martingale is the quantity:

$$\sigma^2 = \sum_{i\in[n-1]} \sup_{(Z_1,\ldots,Z_i)} \mathbf{E}[(X_{i+1}-X_i)^2|Z_1,\ldots,Z_i].$$

We get good large deviation inequalities for martingales with bounded differences and variances (see, for example, [10], Theorem 6.3 and Theorem 6.5). For a martingale $X_0, X_1, \ldots, X_n$ with respect to $Z_0, Z_1, \ldots, Z_n$, with the C-bounded difference property and variance σ^2, we have

$$\mathbb{P}(|X_n - X_0| \geq \lambda) \leq e^{-\frac{t^2}{2(\sigma^2+C\lambda/3)}}. \tag{3}$$

3.2 Proof of Theorem 4

We shall now prove Theorem 4. Recall that we only need to prove the case where $m = n$, As mentioned before, we adapt the proof technique of Kahn and Szemerédi for our random model (also see [7]). We have that the regularity is $t \ll m^{1/2}$.

We shall prove that for every x, and y such that $\|x\| = \|y\| = 1$ and $x \perp \bar{1}$, we have that $|y^t M x| \leq O(\sqrt{t})$. First, we "discretize" our problem by restricting x to belong to the ϵ-net

$$T := \left\{x \in \left(\frac{\epsilon}{\sqrt{m}}\mathbb{Z}\right)^m \mid \|x\| \leq 1 \text{ and } x \perp \bar{1}\right\}$$

and y belonging to

$$T' := \left\{y \in \left(\frac{\epsilon}{\sqrt{m}}\mathbb{Z}\right)^m \mid \|y\| \leq 1\right\}$$

for a small enough constant ϵ.

▷ Claim 10 ([13], Proposition 2.1). If for every $x \in T$, and $y \in T'$, we have that $\|y^t Mx\| \le \alpha$, then we have that for every $z \in \mathbb{R}^m$ such that $\|z\| = 1$, we have that $\|Mz\| \le (1-3\epsilon)^{-1}\alpha$.

Proof. Let $z = \operatorname{argmax}_{\|z\|=1} \|Mz\|$. We shall use the fact that there are $x \in T$, and $y \in T'$ such that $\|x - z\| \le \epsilon$, and $\left\| y - \frac{Mz}{\|Mz\|} \right\| \le \epsilon$. With this in mind, we have:

$$\begin{aligned}\|Mz\| &= \left\langle \frac{Mz}{\|Mz\|}, Mz \right\rangle = \langle y + w_1, M(x + w_2)\rangle \\ &= y^t Mx + \langle w_1, Mx\rangle + \langle y, Mw_2\rangle + \langle w_1, Mw_2\rangle.\end{aligned}$$

Where $|w_1|, |w_2| \le \epsilon$. We note that each of the terms $\langle w_1, Mx\rangle$ and $\langle y, Mw_2\rangle$, and $\langle w_1, Mw_2\rangle$ are upper bounded by $\epsilon\|Mz\|$, and $\langle w_1, Mw_2\rangle \le \epsilon^2\|Mz\|$. Combining this, and using the fact that $\epsilon^2 \le \epsilon$, we have

$$\|Mz\| \le (1-3\epsilon)^{-1} y^t Mx \le (1-3\epsilon)^{-1}\alpha. \qquad \triangleleft$$

So now, will need to only union bound over $T \cup T'$. It is not hard to see that each of these has size at most $|T|, |T'| \le \left(\frac{C_v}{\epsilon}\right)^m$ for some absolute constant C_v.

Indeed, we have:

$$\begin{aligned}|T| &\le \left(\frac{\sqrt{m}}{\epsilon}\right)^m \mathrm{Vol}\,\{x \in \mathbb{R}^m \mid \|x\| \le 1 + \epsilon\} \\ &\le \left(\frac{\sqrt{m}}{\epsilon}\right)^m \cdot \frac{1}{\sqrt{\pi m}} \left(\frac{2\pi e}{m}\right)^{m/2} (1+\epsilon)^m \\ &\le \left(\frac{C_v}{\epsilon}\right)^m\end{aligned}$$

for some constant C_v.

We split the pairs $[m] \times [m] = L \cup \overline{L}$ where $L := \{(u, v) \mid |x_u y_v| \ge \sqrt{t}/m\}$, which we will call "large entries" and write our quantity of interest:

$$\sum_{(u,v)\in[m]\times[m]} x_u M_{u,v} y_v = \sum_{(u,v)\in L} x_u M_{u,v} y_v + \sum_{(u,v)\in\overline{L}} x_u M_{u,v} y_v.$$

For the large entries: For a set of vertices $A \subset [m]$ and a set of edges $B \subset [m]$, let us denote $I(A, B)$ to be the number of vertex-edge incidences in A and B. Let us use $\mu(A, B) := \mathbf{E}[|I(A, B)|]$.

▶ **Lemma 11.** *There is a constant C such that, for every set A of vertices and every set B of hyperedges where $|A| \le |B|$, we have that with probability at least $1 - m^{-\Omega(1)}$, $I := |I(A, B)|$ and $\mu := \mu(A, B)$ satisfy at least one of the following:*

1. $I \le C\mu$
2. $I \log(I/\mu) \le C|B| \log(m/|B|)$.

This lemma is sufficient to show that the large pairs do not contribute too much, as shown by the following lemma, which is the main part of the proof of Kahn and Szemerédi.

▶ **Lemma 12** ([13], Lemma 2.6, [7], Lemma 17). *If the conditions given in Lemma 11 are satisfied, then $\sum_{(u,v)\in L} |x_u M_{u,v} y_v| = O(\sqrt{t})$ for all $x, y \in T$.*

Notice that since we are bounding $\sum_{(u,v)\in L} |x_u M_{u,v} y_v| = O(\sqrt{t})$, which is much stronger than what we really need, it is okay to consider both x and y from T.

Proof of Lemma 11. First, we observe that it is enough to consider $|B| \le m/2$, since otherwise, $|I(A,B)| \le d|A| \le 2\mu(A,B)$. Let $\mathcal{B}_i(a,b)$ denote the event that there is an A of size a and a B of size b which do not satisfy either of the conditions (with a fixd constant C to be specified later) and $|I(A,B)| = i$. Before, we prove the lemma, let us make some observations, which (in hindsight) help us compute the probabilities much easier. Let A be a set of a vertices and B be a collection of b edges, such that $a \le b \le m/2$.

The point here is that we basically want to evaluate the sum:

$$\begin{aligned}\mathbb{P}\left(\bigcup_{a,b,i}\mathcal{B}_i(a,b)\right) &\le \sum_i \mathbb{P}\left(\bigcup_{a,b}\mathcal{B}_i(a,b)\right)\\ &= \sum_{i\le\log^2 m}\mathbb{P}\left(\bigcup_{a,b}\mathcal{B}_i(a,b)\right) + \sum_{i\ge\log^2 m}\mathbb{P}\left(\bigcup_{a,b}\mathcal{B}_i(a,b)\right).\end{aligned}$$

The first observation is that every term in the second sum is small. Towards this, we have the straightforward claim.

▷ Claim 13. For a set of vertices A and edges B and a set of possible incidences $J \subset A \times B$, we have that $\mathbb{P}(I(A,B) = J) \le \left(\frac{2t}{m}\right)^{|J|}$.

Proof. W.L.O.G, let $A = \{1,\dots,a\}$, and for $i \in A$, let $t_i = I(\{i\},B)$. We have that:

$$\mathbb{P}(I(A,B)=J) = \prod_{i\in A}\frac{\binom{m-b}{t-t_i}}{\binom{m}{t}} \le \prod_{i\in A} 2\frac{(m-b)^{t-t_i}}{(t-t_i)!}\frac{t!}{m^t} \le \prod_{i\in A}\left(\frac{2t}{m}\right)^{t_i} \le \left(\frac{2t}{m}\right)^{|J|}. \qquad \triangleleft$$

Here, the first inequality uses the fact that $t = o(\sqrt{m})$. Therefore, we have:

$$\mathbb{P}\left(\mathcal{B}_i(a,b)\right) \le \binom{m}{a}\binom{m}{b}\binom{ab}{i}\left(\frac{2t}{m}\right)^i \le \binom{m}{b}^2\left(e\frac{abt}{mi}\right)^i \le \binom{m}{b}^2\left(\frac{\mu}{i}\right)^i (e)^i.$$

If $i \ge 2e\mu$ and $i \ge \log^2 m$, this probability is at most $2^{2m} \cdot 2^{-\log^2 m} \ll m^{-\Omega(\log m)}$. Thus

$$\sum_{i\ge\log^2 m}\mathbb{P}\left(\bigcup_{a,b}\mathcal{B}_i(a,b)\right) \le \sum_{a,b}\sum_{i\ge\log^2 m}\mathbb{P}\left(\mathcal{B}_i(a,b)\right) \le m^{-\Omega(\log m)}.$$

It remains to deal with the sum $\sum_{i\le\log^2 m}\Pr\left(\bigcup_{a,b}\mathcal{B}_i(a,b)\right)$. For these summands, we have that if $|I(A,B)| \le \log^2 m$ and $I\log(I/\mu) > Cb\log(m/b)$, then

$$I\log m \ge I\log(I/\mu) > Cb\log(m/b) \ge Cb.$$

and so $Cb \le \log^3 m$. The first inequality above comes from the observation that $I \le ab$ and so $I/\mu \le m/t \le m$. Now, using that $I\log m \ge Cb\log(m/\log^3 m)$, we have that $I \ge Cb/2$.

Therefore, we only need to evaluate the sum:

$$\begin{aligned}\sum_{i=Cb/2}^{\log^2 m}\mathbb{P}\left(\mathcal{B}_i(a,b)\right) &\le \binom{m}{a}\binom{m}{b}\sum_{i=Cb/2}^{\log^2 m}\binom{ab}{i}\left(\frac{10et}{m}\right)^i \le \binom{m}{b}^2\sum_{i=Cb/2}^{\log^2 m}\left(\frac{10e^2abt}{im}\right)^i\\ &\le \log^2 m\left(\frac{em}{b}\right)^{2b}\left(\frac{20e^2at}{Cm}\right)^{Cb/2}\\ &= m^{2b-Cb/2}b^{-2b}a^{Cb/2}t^{Cb/2}(20e^2)^{2b}\\ &\le m^{2b-Cb/4}b^{Cb/2-2b}(20e^2)^{2b}\\ &= m^{-\Omega(b)}.\end{aligned}$$

We have used the fact that $t = o(\sqrt{m})$, $b \ge a$ and $b \le \log^3 m$. Thus union bounding over $\log^3 m$ many values of a and b, we have $\sum_{a,b\le\log^3 m}\sum_{i\le\log^2 m}\mathbb{P}\left(\bigcup_{a,b}\mathcal{B}_i(a,b)\right) = m^{-\Omega(1)}$. ◀

For the small entries: Bounding the contribution from the small entries is much easier. The analysis given here is slightly different to the one given in [13] and [7]. However, it does not make much of a difference, and is still, essentially, the same large deviation inequality. We will first compute the expected value of the quantity of interest using the following claim:

▷ Claim 14. We have that:

$$\left|\sum_{(u,v)\in\overline{L}} x_u y_v\right| \leq \frac{m}{\sqrt{t}}.$$

Proof. Since $\sum x_i = 0$, we have $\left(\sum x_i\right)\left(\sum y_i\right) = \sum_{(u,v)\in L} x_u y_v + \sum_{(u,v)\in\overline{L}} x_u y_v = 0$ or

$$\left|\sum_{(u,v)\in\overline{L}} x_u y_v\right| = \left|\sum_{(u,v)\in L} x_u y_v\right|.$$

To bound this, we note that

$$1 = \left(\sum x_u^2\right)\left(\sum y_u^2\right) \geq \sum_{(u,v)\in L} x_u^2 y_v^2 \geq \frac{\sqrt{t}}{m}\left|\sum_{(u,v)\in L} x_u y_v\right| = \frac{\sqrt{t}}{m}\left|\sum_{(u,v)\in\overline{L}} x_u y_v\right|.$$

which gives us what we want. ◁

Given Claim 14 above, we can easily compute the expectation:

$$\mathbf{E}\left[\sum_{(u,v)\in\overline{L}} x_u M_{u,v} y_v\right] = \frac{t}{m}\sum_{(u,v)\in\overline{L}} x_u y_v \in [-\sqrt{t}, \sqrt{t}].$$

▷ Claim 15. We have that with high probability, $\sum_{(u,v)\in\overline{L}} x_u M_{u,v} y_v = O(\sqrt{t})$.

Proof. We set up a martingale and use the method of bounded variances. Let us write the quantity that we wish to estimate as

$$X := \sum_{(u,v)\in B} x_u M_{u,v} y_v.$$

We imagine M being sampled one column at a time, and in each column, t entries are sampled. For column i, let us denote these by $e_{i,1},\ldots,e_{i,t}$. Clearly, $X = X(e_{1,1},\ldots,e_{m,t})$. Denote $X_{i,j} := \mathbf{E}[X|e_{1,1},\ldots,e_{i,j}]$. For distinct $k,k'\in[m]$, it is easy to see that we have the "Lipschitz property":

$$|\mathbf{E}[X|e_{1,1},\ldots,e_{i,j-1},e_{i,j}=k] - \mathbf{E}[X|e_{1,1},\ldots,e_{i,j-1},e_{i,j}=k']| \leq |x_i y_k| + |x_i y_{k'}|.$$

Therefore, we have a bounded difference property on $|X_{i,j} - X_{i,j-1}|$ as follows:

$$\begin{aligned}|X_{i,j} - X_{i,j-1}| = \Bigg|&\mathbf{E}[X|e_{1,1},\ldots,e_{i,j-1},e_{i,j}] \\ &- \frac{1}{m-j+1}\sum_{k'\in[m]\setminus\{e_{i,1},\ldots,e_{i,j-1}\}} \mathbf{E}[X|e_{1,1},\ldots,e_{i,j-1},e_{i,j}=k']\Bigg| \\ \leq |x_{e_j}||y_i| &+ \frac{1}{m-j+1}\sum_{k'\in[m]\setminus\{e_{i,1},\ldots,e_{i,j-1}\}} \mathbb{1}[(k',i)\in\overline{L}]|x_{k'}y_i|\end{aligned}$$

We will use that the above quantity is bounded by $\frac{2\sqrt{t}}{m}$ since we only consider $|x_{e_j} y_i|$ where $(e_j, i) \in \overline{L}$. However, another way to upper bound the above is by using

$$\begin{aligned}
&\frac{1}{m-j+1} \sum_{k' \in [m] \setminus \{e_{i,1}, \ldots, e_{i,j-1}\}} \mathbb{1}[(k', i) \in \overline{L}] |x_{k'} y_i| \\
&\leq \frac{1}{m-j+1} \sum_{k' \in [m] \setminus \{e_{i,1}, \ldots, e_{i,j-1}\}} |x_{k'} y_i| \\
&\leq \frac{|y_i|}{n-j+1} \sum_{k' \in [m]} |x_{k'}| \\
&\leq \frac{2|y_i|}{\sqrt{m}}.
\end{aligned}$$

Using this, we now compute the variance of the martingale:

$$\begin{aligned}
\operatorname{Var}(X_{i,j} - X_{i,j-1} | e_{1,1}, \ldots, e_{i,j-1}) &\leq \frac{1}{m-j+1} \sum_{k \in [m]} \left(|x_k y_i| + \frac{2|y_i|}{\sqrt{m}} \right)^2 \\
&\leq \frac{2}{m-j+1} \sum_{k \in [m]} \left(|x_k y_i|^2 + \frac{4y_i^2}{m} \right) \\
&\leq \frac{10 y_i^2}{m-j+1}.
\end{aligned}$$

Where the last inequality uses that $\sum_k x_k^2 \leq 1$. Therefore, the variance of the martingale is at most$t \cdot \frac{10}{m-t} \sum_i y_i^2 \leq \frac{20t}{m} =: \sigma^2$. This is because $\sum_i y_i^2 \leq 1$. Therefore, by the bounded variance martingale inequality (3), using $|X_i - X_{i-1}| \leq \frac{2\sqrt{t}}{m} =: C$:

$$\mathbb{P}(X \geq (D+1)\sqrt{t}) \leq \exp\left\{ -\frac{D^2 t}{2\sigma^2 + tC/3} \right\} \leq \exp\left\{ -\frac{D^2 t}{\frac{40t}{m} + \frac{2t}{3m}} \right\} \leq \exp\left\{ -\Omega(D^2 m) \right\}.$$

For a large enough constant D, this lets us union bound over all $x, y \in T$, whose number can be bounded by $\left(\frac{C_v}{\epsilon}\right)^m$. ◁

4 Conclusion

We have given an upper bound on t-regular hypergraph discrepancy in terms of t and a spectral property of the incidence matrix. However, when one restricts attention to random t-regular hypergraphs, the $O(\sqrt{t})$ bound is achieved only when $m = \Omega(n)$. In the case where $m = o(n)$, one can replace λ in Theorem 1 by λ' where

$$\lambda'(\mathcal{H}) := \max_{\substack{U \subset V \\ |U| = 16m}} \max_{\substack{v \perp \overline{1}, \\ \|v\|=1, \\ \operatorname{supp}(v) \subseteq U}} \|Mv\|$$

and the proof would remain the same. This is because using the partial coloring theorem (Theorem 5), one may assign colors to all but at most $16m$ vertices while maintaining that the discrepancy of *every* edge is 0. However, when $\mathcal{H}$ is a random t regular hypergraph with n vertices and $m = o(n)$ edges, we need not have $\lambda'(\mathcal{H}) = O\left(\sqrt{t}\right)$ (in fact, the guess would be $O(\sqrt{tn/m})$). The problem is that Claim 15 (In Section 3.2) does not extend. However, in this regime, we believe that with high probability, the discrepancy is much *lower* than $\sqrt{t}$ (in contrast to λ growing).

Recently, Franks and Saks [12] showed that for $n = \tilde{\Omega}(m^3)$, the discrepancy is $O(1)$ almost surely. Independently, Hoberg and Rothvoss [14] considered a different model of random hypergraphs with n vertices and m edges and each vertex-edge-incidence is an i.i.d. $\mathrm{Ber}(p)$ random variable. They show that if $n = \tilde{\Omega}(m^2)$, the discrepancy is $O(1)$ almost surely. Both [12] and [14] used similar Fourier analytic techniques inspired by [15]. Moreover, it was an open question in [14] whether the hypergraph with i.i.d $\mathrm{Ber}(1/2)$ incidences where $n = O(m \log m)$ almost surely has discrepancy $O(1)$. This was shown to be true by the author [20].

We argue that this is an interesting regime for random regular hypergraphs, as this kind of discrepancy bound is not implied by the Beck-Fiala conjecture. The case where $n = \Omega(m \log m)$, is of particular interest, since we believe there is a phase transition for constant discrepancy at this point. On the one hand, we do not know if the discrepancy bound given by Corollary 3 is the truth, and on the other hand, we do not know if random regular hypergraphs with, for example, $n = \Theta(m^{1.5})$ almost surely has discrepancy $O(1)$. We conclude with a conjecture, building on an open problem (open problem 1) in [12]:

▶ **Conjecture 16.** *There is an absolute constant $K > 0$ such that the following holds. Let $t > 0$ be any integer and $\mathcal{H}$ be a random t-regular hypergraph on n vertices and $K\frac{n}{\log n}$ edges. Then with high probability,*

$$\mathrm{disc}(\mathcal{H}) = O(1).$$

References

1 Noga Alon and Joel H. Spencer. The probabilistic method, 2000.

2 Nikhil Bansal. Constructive algorithms for discrepancy minimization. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 3–10, 2010. doi:10.1109/FOCS.2010.7.

3 Nikhil Bansal and Raghu Meka. On the discrepancy of random low degree set systems. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 2557–2564, 2019. doi:10.1137/1.9781611975482.157.

4 József Beck. Roth's estimate of the discrepancy of integer sequences is nearly sharp. *Combinatorica*, 1(4):319–325, 1981. doi:10.1007/BF02579452.

5 József Beck and Tibor Fiala. "integer-making" theorems. *Discrete Applied Mathematics*, 3(1):1–8, 1981. doi:10.1016/0166-218X(81)90022-6.

6 Debe Bednarchak and Martin Helm. A note on the beck-fiala theorem. *Combinatorica*, 17(1):147–149, March 1997. doi:10.1007/BF01196138.

7 Andrei Z. Broder, Alan M. Frieze, Stephen Suen, and Eli Upfal. Optimal construction of edge-disjoint paths in random graphs. *SIAM J. Comput.*, 28(2):541–573, 1998. doi:10.1137/S0097539795290805.

8 Boris Bukh. An improvement of the beck-fiala theorem. *Combinatorics, Probability and Computing*, 25(3):380?398, 2016. doi:10.1017/S0963548315000140.

9 Bernard Chazelle. *The Discrepancy Method: Randomness and Complexity.* Cambridge University Press, New York, NY, USA, 2000.

10 Fan Chung and Linyuan Lu. Concentration inequalities and martingale inequalities: a survey. *Internet Math.*, 3(1):79–127, 2006. URL: https://projecteuclid.org:443/euclid.im/1175266369.

11 Esther Ezra and Shachar Lovett. On the beck-fiala conjecture for random set systems. In *APPROX-RANDOM*, 2015.

12 C. Franks and M. Saks. On the Discrepancy of Random Matrices with Many Columns. *ArXiv e-prints*, July 2018. arXiv:1807.04318.

13 J. Friedman, J. Kahn, and E. Szemerédi. On the second eigenvalue of random regular graphs. In *Proceedings of the Twenty-first Annual ACM Symposium on Theory of Computing*, STOC '89, pages 587–598, New York, NY, USA, 1989. ACM. doi:10.1145/73007.73063.

14 Rebecca Hoberg and Thomas Rothvoss. A fourier-analytic approach for the discrepancy of random set systems. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 2547–2556. SIAM, 2019. doi:10.1137/1.9781611975482.156.

15 Greg Kuperberg, Shachar Lovett, and Ron Peled. Probabilistic existence of rigid combinatorial structures. In Howard J. Karloff and Toniann Pitassi, editors, *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 1091–1106. ACM, 2012. doi:10.1145/2213977.2214075.

16 Avi Levy, Harishchandra Ramadas, and Thomas Rothvoss. Deterministic discrepancy minimization via the multiplicative weight update method. In *Integer Programming and Combinatorial Optimization - 19th International Conference, IPCO 2017, Waterloo, ON, Canada, June 26-28, 2017, Proceedings*, pages 380–391, 2017. doi:10.1007/978-3-319-59250-3_31.

17 Shachar Lovett and Raghu Meka. Constructive discrepancy minimization by walking on the edges. *SIAM J. Comput.*, 44(5):1573–1582, 2015. doi:10.1137/130929400.

18 J. Matousek. *Geometric Discrepancy: An Illustrated Guide.* Algorithms and Combinatorics. Springer Berlin Heidelberg, 1999. URL: https://books.google.com/books?id=BKvXj1GisPOC.

19 Jiří Matoušek. Tight upper bounds for the discrepancy of half-spaces. *Discrete & Computational Geometry*, 13:593–601, 1995. doi:10.1007/BF02574066.

20 Aditya Potukuchi. Discrepancy in random hypergraph models, 2018. arXiv:1811.01491.

21 Thomas Rothvoss. Constructive discrepancy minimization for convex sets. *SIAM J. Comput.*, 46(1):224–234, 2017. doi:10.1137/141000282.

22 Joel Spencer. Six standard deviations suffice. *Transactions of the American Mathematical Society*, 289(2):679–706, 1985. doi:10.1090/S0002-9947-1985-0784009-0.

23 Joel Spencer. Coloring the projective plane. *Discrete Mathematics*, 73(1):213–220, 1988. doi:10.1016/0012-365X(88)90150-1.

Faster Dynamic Range Mode

Bryce Sandlund
Cheriton School of Computer Science, University of Waterloo, Canada
bcsandlund@gmail.com

Yinzhan Xu
CSAIL, Massachusetts Institute of Technology, Cambridge, MA, USA
xyzhan@mit.edu

Abstract

In the dynamic range mode problem, we are given a sequence a of length bounded by N and asked to support element insertion, deletion, and queries for the most frequent element of a contiguous subsequence of a. In this work, we devise a deterministic data structure that handles each operation in worst-case $\tilde{O}(N^{0.655994})$ time, thus breaking the $O(N^{2/3})$ per-operation time barrier for this problem. The data structure is achieved by combining the ideas in Williams and Xu (SODA 2020) for batch range mode with a novel data structure variant of the Min-Plus product.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis

Keywords and phrases Range Mode, Min-Plus Product

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.94

Category Track A: Algorithms, Complexity and Games

Acknowledgements We would like to thank Virginia Vassilevska Williams for her valuable comments on a draft of this paper. We would like to thank the anonymous reviewers for their helpful suggestions.

1 Introduction

Given a sequence of elements $a_1, a_2, \dots, a_n$, the dynamic range mode problem asks to support queries for the most frequent element in a specified subsequence $a_l, a_{l+1}, \dots, a_r$ while also supporting insertion or deletion of an element at a given index i. The mode of a sequence of elements is one of the most basic data statistics, along with the median and the mean. It is frequently computed in data mining, information retrieval, and data analytics.

The range mode problem seeks to answer multiple queries on distinct intervals of the data sequence without having to recompute each answer from scratch. Its study in the data structure community has shown that the mode is a much more challenging data statistic to maintain than other natural range queries: while range sum, min or max, median, and majority all support linear space dynamic data structures with poly-logarithmic or better time per operation [22, 7, 16, 12, 10], the current fastest dynamic range mode data structure prior to this paper requires a stubborn $O(n^{2/3})$ time per operation [9]. Indeed, range mode is one of few remaining classical range queries to which our currently known algorithms may be far from optimal. As originally stated by Brodal et al. [4] and mentioned by Chan et al. [6] in 2011 and 2014, respectively, "The problem of finding the most frequent element within a given array range is still rather open."

The current best conditional lower bound, by Chan et al. [6], reduces multiplication of two $\sqrt{n} \times \sqrt{n}$ boolean matrices to n range mode queries on a fixed array of size $O(n)$. This indicates that if the current algorithm for boolean matrix multiplication is optimal, then answering n range mode queries on an array of size $O(n)$ cannot be performed in time better than $O(n^{3/2-\epsilon})$ time for $\epsilon > 0$ with combinatorial techniques, or $O(n^{\omega/2-\epsilon})$ time for $\epsilon > 0$ in general, where $\omega < 2.373$ [25, 13] is the square matrix multiplication exponent. This reduction can be strengthened for dynamic range mode by reducing from the online

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 94; pp. 94:1–94:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

matrix-vector multiplication problem [17]. Using $O(n)$ dynamic range mode operations on a sequence of length $O(n)$, we can multiply a $\sqrt{n} \times \sqrt{n}$ boolean matrix with $\sqrt{n}$ boolean vectors given one at a time. This indicates that a dynamic range mode data structure taking $O(n^{1/2-\epsilon})$ time per operation for $\epsilon > 0$ is not possible with current knowledge.

Previous attempts indicate the higher $O(n^{2/3})$ per operation cost as the bound to beat [6, 9]. Indeed, $\tilde{O}(n^{2/3})$ time per operation[1] can be achieved with a variety of techniques, but crossing the $O(n^{2/3})$ barrier appears much harder.

Progress towards this goal has been established with the recent work of Williams and Xu [26]. They show that by appealing to Min-Plus product of structured matrices, n range mode queries on an array of size n can be answered in $\tilde{O}(n^{1.4854})$ time, thus beating the combinatorial lower bound for batch range mode. This result also shows a separation between batch range mode and dynamic range mode: while batch range mode can be completed in $O(n^{1/2-\epsilon})$ time per operation, such a result for dynamic range mode would imply a breakthrough in the online matrix-vector multiplication problem.

Range mode is not the first problem shown to be closely related to the Min-Plus product problem. It is well-known that the all-pairs shortest paths (APSP) problem is asymptotically equivalent to Min-Plus product [11], in the sense that a $T(n)$ time algorithm to compute the Min-Plus product of two $n \times n$ matrices implies an $O(T(n))$ time algorithm for APSP in n-node graphs and vice versa. Although it is not known how to perform Min-Plus product of two arbitrary $n \times n$ matrices in time $O(n^{3-\epsilon})$ for $\epsilon > 0$, several problems reduce to Min-Plus products of matrices A and B which have nice structures that can be exploited. The simplest examples result by restricting edge weights in APSP problems [23, 24, 28, 5, 27]. Bringmann et al. [3] show Language Edit Distance, RNA-folding, and Optimum Stack Generation can be reduced to Min-Plus product where matrix A has small difference between adjacent entries in each row and column. Finally, the recent work of Williams and Xu [26] reduces APSP in certain geometric graphs, batch range mode, and the maximum subarray problem with entries bounded by $O(n^{0.62})$ to a more general structured Min-Plus product, extending the result of Bringmann et al. All of the above structured Min-Plus products are solvable in truly subcubic $O(n^{3-\epsilon})$ time for $\epsilon > 0$, improving algorithms in the problems reduced to said product.

The connection and upper bound established by Williams and Xu [26] of batch range mode to Min-Plus product suggest other versions of the range mode problem may be amenable to similar improvements. In particular, the ability to efficiently compute a batch of range mode queries via reducing to a structured Min-Plus product suggests that one might be able to improve the update time of dynamic range mode in a similar way.

1.1 Our Results

In this paper, we break the $O(n^{2/3})$ time per operation barrier for dynamic range mode. We do so by adapting the result of Williams and Xu [26]. Specifically, we define the following new type of data structure problem on the Min-Plus product that can be applied to dynamic range mode, which may be of independent interest. Then we combine this data structure problem with the algorithm of Williams and Xu.

▶ **Problem 1** (*Min-Plus-Query* problem)**.** *During initialization, we are given two matrices A, B. For each query, we are given three parameters i, j, S, where i, j are two integers, and S is a set of integers. The query asks* $\min_{k \notin S}\{A_{i,k} + B_{k,j}\}$.

[1] We use the $\tilde{O}(\cdot)$ notation to hide poly-logarithmic factors.

Our performance theorem is the following.

▶ **Theorem 2.** *There exists a deterministic data structure for dynamic range mode on a sequence $a_1, \ldots, a_n$ that supports query, insertion, and deletion in worst-case $\tilde{O}(N^{0.655994})$ time per operation, where N is the maximum size of the sequence at any point in time. The space complexity of the data structure is $\tilde{O}(N^{1.327997})$.*

Our result shows yet another application of the Min-Plus product to an independently-studied problem, ultimately showing a dependence of the complexity of dynamic range mode on the complexity of fast matrix multiplication. Further, in contrast to many other reductions to Min-Plus in which we must assume a structured input on the original problem [23, 24, 28, 5, 27, 26], our algorithm works on the fully general dynamic range mode problem. In this sense, our result is perhaps most directly comparable to the batch range mode reduction of Williams and Xu [26] and the Language Edit Distance, RNA-folding, and Optimum Stack Generation reductions of Bringmann et al. [3].

1.2 Discussion of Technical Difficulty

Despite the new $\tilde{O}(n^{1.4854})$ time algorithm for batch range mode [26], we cannot directly apply the result to dynamic range mode. The main issue is the element deletion operation. In the range mode algorithm of Williams and Xu (and in many other range mode algorithms), critical points are chosen evenly distributed in the array, and the algorithm precomputes the range mode of intervals between every pair of critical points. In [26], the improvement is achieved via a faster precomputation algorithm, which uses a Min-Plus product algorithm for structured matrices. However, if element deletion is allowed, the results stored in the precomputation will not be applicable. For example, an interval between two critical points could contain x copies of element a, $x-1$ copies of element b, and many other elements with frequencies less than $x-1$. During precomputation, the range mode of this interval would be a. However, if we delete two copies of a, there is no easy way to determine that the mode of this interval has now changed to b.

We overcome this difficulty by introducing the Min-Plus-Query problem, as defined in Section 1.1. Intuitively, in the Min-Plus-Query problem, a large portion of the work of the Min-Plus product is put off until the query. It also supports more flexible queries. Using the Min-Plus-Query problem as a subroutine, we will be able to query the most frequent element excluding a set S of forbidden elements. For instance, in the preceding example, we would be able to query the most frequent element that is not a. This is the main technical contribution of the paper.

Another major difference between our algorithm for dynamic range mode and the batch range mode algorithm of Williams and Xu [26] is the need for rectangular matrix multiplication. In our algorithm, we treat elements that appear more than about $N^{2/3}$ times differently from the rest (a similar treatment is given in the dynamic range mode algorithm of Hicham et al. [9]). However, the number of critical points we use is about $N^{1/3}$; thus the number of critical points and frequent elements differ. This contrasts with batch range mode, where elements that appear more than about $\sqrt{n}$ times are considered frequent and the number of critical points used coincides with the number of frequent elements. The consequence of this difference is that a rectangular matrix product is required for dynamic range mode, while a square matrix product sufficed in [26].

2 Related Work

The range mode problem was first studied formally by Krizanc et al. [18]. They study space-efficient data structures for static range mode, achieving a time-space tradeoff of $O(n^{2-2\epsilon})$ space and $O(n^\epsilon \log n)$ query time for any $0 < \epsilon \leq 1/2$. They also give a solution occupying $O(n^2 \log\log n / \log n)$ space with $O(1)$ time per query.

Chan et al. [6] also study static range mode, focusing on linear space solutions. They achieve a linear space data structure supporting queries in $O(\sqrt{n})$ time via clever use of arrays, which can be improved to $O(\sqrt{n/\log n})$ time via bit-packing tricks. Their paper also introduces the conditional lower bound which reduces multiplication of two $\sqrt{n} \times \sqrt{n}$ boolean matrices to n range mode queries on an array of size $O(n)$. As mentioned, combined with the presumed hardness of the online matrix vector problem [17], this result indicates a dynamic range mode data structure must take greater than $O(n^{1/2-\epsilon})$ for $\epsilon > 0$ time per operation. Finally, Chan et al. [6] also give the first data structure for dynamic range mode. At linear space, their solution achieves $O(n^{3/4} \log n / \log\log n)$ worst-case time per query and $O(n^{3/4} \log\log n)$ amortized expected time update, and at $O(n^{4/3})$ space, their solution achieves $O(n^{2/3} \log n / \log\log n)$ worst-case time query and amortized expected update time.

Recently, Hicham et al. [9] improved the runtime of dynamic range mode to worst-case $O(n^{2/3})$ time per operation while simultaneously improving the space usage to linear. Prior to this paper, this result was the fastest data structure for dynamic range mode.

A cell-probe lower bound for static range mode has been devised by Greve et al. [15]. Their result states that any range mode data structure that uses S memory cells of w-bit words needs $\Omega(\frac{\log n}{\log(Sw/n)})$ time to answer a query.

Via reduction to a structured Min-Plus product, Williams and Xu [26] recently showed that n range mode queries on a fixed array of size n can be answered in $\tilde{O}(n^{1.4854})$ time. Williams and Xu actually show how to compute the *frequency* of the mode for each query. We can adapt this method to find the element that is mode using the following binary search. For query $[l, r]$, we ask the frequency of the mode in range $[l, (l + r)/2]$. If it is the same, we repeat the search with right endpoint in range $[(l + r)/2, r]$; if it is not, we repeat the search with right endpoint in range $[l, (l + r)/2]$. Using this method, we can binary search until we determine when the frequency of the mode changes, thus finding the element that is mode in an additional $O(\log n)$ queries. The algorithm of Williams and Xu can also be used to speed up the preprocessing time of the $O(n)$ space, $O(\sqrt{n})$ query time static range mode data structure to $\tilde{O}(n^{1.4854})$ time.

Both static and dynamic range mode have been studied in approximate settings [2, 15, 8].

3 Preliminaries

We formally define the Min-Plus product problem and the dynamic range mode problem.

▶ **Problem 3** (Min-Plus product). *The Min-Plus product of an $m \times n$ matrix A and an $n \times p$ matrix B is the $m \times p$ matrix $C = A \star B$ such that $C_{i,j} = \min_k\{A[i, k] + B[k, j]\}$.*

▶ **Problem 4** (Dynamic Range Mode). *In the dynamic range mode problem, we are given an initially empty sequence and must support the following operations:*

- *Insert an element at a given position of the sequence.*
- *Delete one element of the sequence.*
- *Query the most frequent element of any contiguous subsequence. If there are multiple answers, output any.*

It is guaranteed that the size of the array does not exceed N at any point in time.

We use ω to denote the square matrix multiplication exponent, i.e. the smallest real number such that two $n \times n$ matrices can be multiplied in $n^{\omega+o(1)}$ time. The current bound on ω is $2 \leq \omega < 2.373$ [13, 25]. In this work, we will use fast rectangular matrix multiplication. Analogous to the square case, we use $\omega(k)$ to denote the exponent of rectangular matrix multiplication, i.e., the smallest real number such that an $n \times n^k$ matrix and an $n^k \times n$ matrix can be multiplied in $n^{\omega(k)+o(1)}$ time. Le Gall and Urrutia [14] computed smallest upper bounds to date for various values of k. In this work, we are mostly interested in values of $\omega(k)$ listed in Figure 1.

k	Upper Bound on $\omega(k)$
1.75	3.021591
2	3.251640

Figure 1 Upper bounds for the exponent of multiplying an $n \times n^k$ matrix and an $n^k \times n$ matrix [14].

It is known that the function $\omega(k)$ is convex for $k > 0$ (see e.g. [19], [20]), so we can use values of $\omega(p)$ and $\omega(q)$ to give upper bounds for $\omega(k)$ as long as $p \leq k \leq q$.

▶ **Fact 5.** *When $0 < p \leq k \leq q$, $\omega(k) \leq \frac{k-p}{q-p}\omega(q) + \frac{q-k}{q-p}\omega(p)$.*

Combining Figure 1 and Fact 5, we obtain the following bound on $\omega(k)$ when $k \in [1.75, 2]$.

▶ **Corollary 6.** *When $1.75 \leq k \leq 2$, $\omega(k) \leq 0.920196k + 1.41125$.*

4 Main Algorithm

A main technical component for our dynamic range mode algorithm is the use of the *Min-Plus-Query* problem, which is formally defined in Section 1. We are given two matrices A, B. For each query, we are given three parameters i, j, S, and we need to compute $\min_{k \notin S}\{A_{i,k} + B_{k,j}\}$.

If we just use the Min-Plus-Query problem, we can only compute the frequency of the range mode. Although we can binary search for the most frequent element as described in Section 2, we are also able to return the witness from the Min-Plus-Query problem organically. This construction may be of independent interest.

▶ **Problem 7** (*Min-Plus-Query-Witness* **problem**)**.** *During initialization, we are given two matrices A, B. For each query, we are given three parameters i, j, S, where i, j are two integers, and S is a set of integers. We must output an index $k^* \notin S$ such that $A_{i,k^*} + B_{k^*,j} = \min_{k \notin S}\{A_{i,k} + B_{k,j}\}$.*

If A is an $n \times n^s$ matrix and B is an $n^s \times n$ matrix, then the naive algorithm for Min-Plus-Query just enumerates all possible indices k for each query, which takes $O(n^s)$ time per query. In order to get a faster algorithm for dynamic range mode, we need to achieve $\tilde{O}(n^{2+s-\epsilon})$ preprocessing time and $\tilde{O}(n^{s-\epsilon} + |S|)$ query time, for some $\epsilon > 0$, where A, B are some special matrices generated by the range mode instance. Specifically, matrix B meets the following two properties:

1. Each row of B is non-increasing;
2. The difference between the sum of elements in the j-th column and the sum of elements in the $(j+1)$-th column is at most n^s, for any j.

Williams and Xu [26] give a faster algorithm for multiplying an arbitrary matrix A with such matrix B, which leads to a faster algorithm for static range mode. We will show that nontrivial data structures exist for the Min-Plus-Query problem for such input matrices A and B. Such a data structure will lead to a faster algorithm for dynamic range mode.

In the following lemma, we show a data structure for the Min-Plus-Query problem when both input matrices have integer weights small in absolute value.

▶ **Lemma 8.** *Let $s \geq 1$ be a constant. Let A and B be two integer matrices of dimension $n \times n^s$ and $n^s \times n$, respectively, with entries in $\{-W, \ldots, W\} \cup \{\infty\}$ for some $W \geq 1$. Then we can solve the Min-Plus-Query problem of A and B in $\tilde{O}(Wn^{\omega(s)})$ preprocessing time and $\tilde{O}(|S|)$ query time. The space complexity is $\tilde{O}(Wn^2 + n^{1+s})$.*

Proof. The algorithm uses the idea by Alon, Galil and Margalit in [1], which computes the Min-Plus product of A, B in $\tilde{O}(Wn^{\omega(s)})$ time.

In their algorithm, they first construct matrix A' defined by

$$A'_{i,k} = \begin{cases} (n^s+1)^{A_{i,k}+W} & \text{if } A_{i,k} \neq \infty, \\ 0 & \text{otherwise.} \end{cases}$$

We can define B' similarly. Then the product $A'B'$ captures some useful information about the Min-Plus product of A and B. Namely, for each entry $(A'B')_{i,j}$, we can uniquely write it as $\sum_{t \geq 0} r_t^{i,j}(n^s+1)^t$ for integers $0 \leq r_t^{i,j} \leq n^s$. Note that $r_t^{i,j}$ exactly equals the number of k such that $A_{i,k} + B_{k,j} = t - 2W$. Thus, we can use $A'B'$ to compute the Min-Plus Product of A and B.

In our algorithm, we use a range tree to maintain the sequence $r_t^{i,j}$ for each pair of i, j. The preprocessing takes $\tilde{O}(Wn^{\omega(s)})$ time, which is the time to compute $A'B'$ and the sequences $r_t^{i,j}$.

During each query, we are given i, j, S. We enumerate each $k \in S$, and decrement $r^{i,j}_{A_{i,k}+B_{k,j}+2W}$ in the range tree if $A_{i,j} + B_{k,j} < \infty$. After we do this for every $k \in S$, we query the range tree for the smallest t such that $r_t^{i,j} \neq 0$, so $t - 2W$ is the answer to the Min-Plus-Query query. After each query, we need to restore the values of $r^{i,j}$, which can also be done efficiently. The query time is $\tilde{O}(|S|)$, since each update and each query of range tree takes $\tilde{O}(1)$ time. The space complexity should be clear from the algorithm. ◀

In the previous lemma, the data structure only answers the Min-Plus-Query problem. In all subsequent lemmas, the data structure will be able to handle the Min-Plus-Query-Witness problem.

In the next lemma, we use Lemma 8 as a subroutine to show a data structure for the Min-Plus-Query-Witness problem when only matrix A has small integer weights in absolute value.

▶ **Lemma 9.** *Let $s \geq 1$ be a constant. Let A and B be two integer matrices of dimension $n \times n^s$ and $n^s \times n$, respectively, where A has entries in $\{-W, \ldots, W\} \cup \{\infty\}$ for some $W \geq 1$, and B has arbitrary integer entries represented by* polylog n *bit numbers. Then for every integer $1 \leq P \leq n^s$, we can solve the Min-Plus-Query-Witness problem of A and B in $O(\frac{n^s}{P}Wn^{\omega(s)})$ preprocessing time and $O(|S| + P)$ query time. The space complexity is $\tilde{O}(\frac{Wn^{2+s}}{P} + \frac{n^{1+2s}}{P})$.*

Proof. For simplicity, assume P is a factor of n^s. We sort each column of matrix B and put entries whose rank is between $(\ell - 1)P + 1$ and ℓP into the ℓ-th bucket. We use $K_{j,\ell}$ to denote the set of row indices of entries in the ℓ-th bucket of the column j. We use $L_{j,\ell}$ to denote the smallest entry value of the bucket $K_{j,\ell}$, and use $H_{j,\ell}$ to denote the largest entry value. Formally,

$$L_{j,\ell} = \min_{k \in K_{j,\ell}} B_{k,j} \quad \text{and} \quad H_{j,\ell} = \max_{k \in K_{j,\ell}} B_{k,j}.$$

For each $\ell \in [n^s/P]$, we do the following[2]. We create an $n^s \times n$ matrix B^ℓ and initialize all its entries to ∞. Then for each column j, if $H_{j,\ell} - L_{j,\ell} \leq 2W$ (we will call it a small bucket), we set $B^\ell_{k,j} := B_{k,j} - L_{j,\ell} - W$ for all $k \in K_{j,\ell}$. We will handle the case $H_{j,\ell} - L_{j,\ell} > 2W$ (large bucket) later. Clearly, all entries in B^ℓ have values in $\{-W, \dots, W\} \cup \{\infty\}$, so we can use the algorithm in Lemma 8 to preprocess A and B^ℓ and store the data structure in D^ℓ. Also, for each pair (i,j), we create a range tree $\mathcal{T}^{i,j}_{\text{small}}$ on the sequence $(A \star B^1)_{i,j}, (A \star B^2)_{i,j}, (A \star B^3)_{i,j}, \dots, (A \star B^{n^s/P})_{i,j}$, which stores the optimal Min-Plus values when k is from a specific small bucket. This part takes $\tilde{O}(\frac{n^s}{P} W n^{\omega(s)})$ time. The space complexity is $\frac{n^s}{P}$ times more than the space complexity of Lemma 8, so space complexity of this part is $\tilde{O}(\frac{Wn^{2+s}}{P} + \frac{n^{1+2s}}{P})$.

We also do the following preprocessing for buckets where $H_{j,\ell} - L_{j,\ell} > 2W$. We first create a 0/1 matrix $\bar{A}$ where $\bar{A}_{i,k} = 1$ if and only if $A_{i,k} \neq \infty$. Then for each $\ell \in [n^s/P]$, we create a 0/1 matrix $\bar{B}^\ell$ such that $\bar{B}^\ell_{k,j} = 1$ if and only if $k \in K_{j,\ell}$ and $H_{j,\ell} - L_{j,\ell} > 2W$. Then we use fast matrix multiplication to compute the product $\bar{A}\bar{B}^\ell$. If $K_{j,\ell}$ is a large bucket, the (i,j)-th entry of $\bar{A}\bar{B}^\ell$ is the number of $k \in K_{j,\ell}$ such that $A_{i,k} < \infty$; if $K_{j,\ell}$ is a small bucket, the (i,j)-th entry is 0. For each pair (i,j), we create a range tree $\mathcal{T}^{i,j}_{\text{large}}$ on the sequence $(\bar{A}\bar{B}^1)_{i,j}, (\bar{A}\bar{B}^2)_{i,j}, (\bar{A}\bar{B}^3)_{i,j}, \dots, (\bar{A}\bar{B}^{n^s/P})_{i,j}$. This part takes $\tilde{O}(\frac{n^s}{P} n^{\omega(s)})$ time, which is dominated by the time for small buckets. The space complexity is also dominated by the data structures for small buckets.

Now we describe how to handle a query (i,j,S). First consider small buckets. In $O(|S|)$ time, we can compute the set of small buckets $K_{j,\ell}$ that intersect with S. For each such $K_{j,\ell}$, we can query the data structure D^ℓ with input $(i, j, S \cap K_{j,\ell})$ to get the optimum value when $k \in K_{j,\ell}$. For each small bucket that intersects with S, we can set its corresponding value in the range tree $\mathcal{T}^{i,j}_{\text{small}}$ to ∞, then we can compute the optimum value of all small buckets that do not intersect with S by querying the minimum value of the range tree $\mathcal{T}^{i,j}_{\text{small}}$. After this query, we need to restore all values in the range tree. It takes $\tilde{O}(|S|)$ time to handle small buckets on query.

Now consider large buckets. Intuitively, we want to enumerate indices in all large buckets $K_{j,\ell}$ such that there exists an index $k \in K_{j,\ell} \cap ([n^s] \setminus S)$ where $A_{i,k} < \infty$. However, doing so would be prohibitively expensive. We will show that we only need two such buckets. Consider three large buckets $l_1 < l_2 < l_3$. Pick any $k_1 \in T_{j,l_1}, k_3 \in T_{j,l_3}$ such that $A_{i,k_1} < \infty$. Since

$$A_{i,k_1} + B_{k_1,j} \leq W + L_{j,l_2} < W + H_{j,l_2} - 2W < A_{i,k_3} + B_{k_3,j},$$

k_3 can never be the optimum. Thus, it suffices to find the smallest two buckets such that there exists an index $k \in K_{j,\ell} \cap ([n^s] \setminus S)$ where $A_{i,k} < \infty$, and then enumerate all indices in these two buckets. To find such two buckets, we can enumerate over all indices $k \in S$, and if $A_{i,k} < \infty$ we can decrement the corresponding value in the range tree $\mathcal{T}^{i,j}_{\text{large}}$. Thus, we can compute the two smallest buckets by querying the two earliest nonzero values in the range tree. We also need to restore the range tree after the query. The range tree part takes $\tilde{O}(|S|)$ time and scanning the two large buckets requires $O(P)$ time. Thus, this step takes $\tilde{O}(|S| + P)$ time.

At this point, we will know the bucket that contains the optimum index k^*. Thus, we can iterate all indices in this bucket to actually get the witness for the Min-Plus-Query-Witness query. It takes $O(P)$ time to do so.

In summary, the preprocessing time, query time, and space complexity meet the promise in the lemma statement. ◀

[2] We use $[n]$, with n integer, to denote the set $\{1, 2, \dots, n\}$.

In the following lemma, we show a data structure for the Min-Plus-Query-Witness problem when the matrix B has the *bounded difference* property, which means that nearby entries in each row have close values. The proof adapts the strategy of [26].

▶ **Lemma 10.** *Let $s \geq 1$ be a constant. Let A be an $n \times n^s$ integer matrix, and let B be an $n^s \times n$ integer matrix. It is guaranteed that there exists $1 \leq \Delta \leq \min\{n, W\}$, such that for every k, $|B_{k,j_1} - B_{k,j_2}| \leq W$ as long as $\lceil j_1/\Delta \rceil = \lceil j_2/\Delta \rceil$. Then for every $L = \Omega(\Delta)$, we can solve the Min-Plus-Query-Witness problem of A and B in $\tilde{O}(\Delta^2 \frac{n^s}{L} W n^{\omega(s)} + \frac{n^{2+s}}{\Delta})$ preprocessing time and $\tilde{O}(L)$ query time, when $|S| < L$. The space complexity is $\tilde{O}(\frac{\Delta^2 W n^{2+s}}{L} + \frac{\Delta^2 n^{1+2s}}{L} + \frac{n^{2+s}}{\Delta})$.*

Proof. Preprocessing Step 1: Create an Estimation Matrix

First, we create a matrix $\hat{B}$, where $\hat{B}_{k,j} = B_{k,\lceil j/\Delta \rceil \Delta}$. By the property of matrix B, $|\hat{B}_{k,j} - B_{k,j}| \leq W$ for every k, j. For each pair (i,j), we compute the L-th smallest value of $A_{i,k} + \hat{B}_{k,j}$ among all $1 \leq k \leq n^s$, and denote this value by $\hat{C}^L_{i,j}$. Notice that $\hat{C}^L_{i,j} = \hat{C}^L_{i,\lceil j/\Delta \rceil \Delta}$, so it suffices to compute $\hat{C}^L_{i,j}$ when j is a multiple of Δ, and we can infer other values correctly. It takes $O(n^s)$ time to compute each $\hat{C}^L_{i,j}$, so this step takes $O(n^{2+s}/\Delta)$ time.

If we similarly define $C^L_{i,j}$ as the L-th smallest value of $A_{i,k} + B_{k,j}$ among all $1 \leq k \leq n^s$, then $|C^L_{i,j} - \hat{C}^L_{i,j}| \leq W$ by the following claim, whose proof is omitted for space constraint.

▷ Claim. Given two sequences $(a_k)_{k=1}^m$ and $(b_k)_{k=1}^m$ such that $|a_k - b_k| \leq W$, then the L-th smallest element of a and the the L-th smallest element of b differ by at most W.

Also, in $\tilde{O}(n^{2+s}/\Delta)$ time, we can compute a sorted list $\mathcal{L}^{i,j}_{\text{small}}$ of indices k sorted by the value $A_{i,k} + \hat{B}_{k,j} - \hat{C}^L_{k,j}$, for every i, and every j that is a multiple of Δ.

The space complexity in this step is not dominating.

Preprocessing Step 2: Perform Calls to Lemma 9

For some integer $\rho \geq 1$, we will perform ρ rounds of the following algorithm. At the r-th round for some $1 \leq r \leq \rho$, we randomly sample $j^r \in [n]$, and let $A^r_{i,k} := A_{i,k} + B_{k,j^r} - \hat{C}^L_{i,j^r}$ and $B^r_{k,j} := B_{k,j} - B_{k,j^r}$. Clearly, $A^r_{i,k} + B^r_{k,j} = A_{i,k} + B_{k,j} - \hat{C}^L_{i,j^r}$. For each pair (i,k), we find the smallest r such that $|A^r_{i,k}| \leq 3W$. We keep these entries as they are and replace all other entries by ∞. For every (i,k), there exists at most one r such that $A^r_{i,k} \neq \infty$. Then we use Lemma 9 to preprocess A^r and B^r for every $1 \leq r \leq \rho$. Thus, this part takes $O(\rho \frac{n^s}{P} W n^{\omega(s)})$ time, for some integer P to be determined later. Note that this parameter also affects the query time. This step stores ρ copies of the data structure from Lemma 9, so the space complexity is $\tilde{O}(\rho \frac{W n^{2+s}}{P} + \rho \frac{n^{1+2s}}{P})$.

Note that this step is the only step that uses randomization. We can use the method of [26], Appendix A, to derandomize it. We omit the details for simplicity.

Preprocessing Step 3: Handling Uncovered Pairs

For a pair (i,k), if $A^r_{i,k} \neq \infty$ for any r, we call (i,k) *covered*; otherwise, we call the pair (i,k) *uncovered.* For each pair (i,j), we enumerate all k such that $|A_{i,k} + \hat{B}_{k,j} - \hat{C}^L_{i,j}| \leq 2W$ and (i,k) is uncovered. Notice that since $A_{i,k} + \hat{B}_{k,j} - \hat{C}^L_{i,j} = A_{i,k} + \hat{B}_{k,\lceil j/\Delta \rceil \Delta} - \hat{C}^L_{i,\lceil j/\Delta \rceil \Delta}$, we only need to exhaustively enumerate all $k \in [n^s]$ when j is a multiple of Δ. Thus, if the total number of (i,k,j) where $|A_{i,k} + \hat{B}_{k,j} - \hat{C}^L_{i,j}| \leq 2W$ and (i,k) is uncovered is X, then we can enumerate all such triples (i,k,j) in $O(X + n^{2+s}/\Delta)$ time.

It remains to bound the total number of triples that satisfy the condition. Fix an arbitrary pair (i,k), and suppose the number of j such that $|A_{i,k} + \hat{B}_{k,j} - \hat{C}^L_{i,j}| \leq 2W$ is at least $(10+s)n \ln n/\rho$. Then with probability at least $1 - (1 - \frac{(10+s)\ln n}{\rho})^\rho \geq 1 - \frac{1}{n^{10+s}}$, we pick a j^r where $|A_{i,k} + \hat{B}_{k,j^r} - \hat{C}^L_{i,j^r}| \leq 2W$. Therefore,

$$|A^r_{i,k}| = \left|A_{i,k} + B_{k,j^r} - \hat{C}^L_{i,j^r}\right| \le \left|A_{i,k} + \hat{B}_{k,j^r} - \hat{C}^L_{i,j^r}\right| + \left|\hat{B}_{k,j^r} - B_{k,j^r}\right| \le 3W,$$

which means (i,k) is covered. Therefore, with high probability, all pairs of (i,k) where the number of j such that $|A_{i,k} + \hat{B}_{k,j} - \hat{C}^L_{i,j}| \le 2W$ is at least $(10+s)n \ln n/\rho$ will be covered. In other words, $X = O(n^{1+s} \cdot n \ln n/\rho) = \tilde{O}(n^{2+s}/\rho)$.

For each pair (i,j), if we enumerate more than L indices k, we only keep the L values of k that give the smallest values of $A_{i,k} + B_{k,j}$. We call this list $\mathcal{L}^{i,j}_{\text{triple}}$. From previous discussion, the time cost in this step is $\tilde{O}(n^{2+s}/\rho + n^{2+s}/\Delta)$. Since we need to store all the triples, the space complexity is $O(n^{2+s}/\rho)$.

Handling Queries

Now we discuss how to handle queries. For each query (S,i,j), let $k^* = \arg\min_{k \notin S} A_{i,k} + B_{k,j}$ be the optimum index. Consider two cases:

- (i,k^*) is covered. By definition of being covered, there exists a round r such that $A^r_{i,k^*} = A_{i,k^*} + B_{k^*,j^r} - \hat{C}^L_{i,j^r}$, so $A^r_{i,k^*} + B^r_{k^*,j} = A_{i,k^*} + B_{k^*,j} - \hat{C}^L_{i,j^r}$. Therefore, we can query the data structure in Lemma 9 for every A^r and B^r and denote b^r as the result. The answer is given by the smallest value of $b^r + \hat{C}^L_{i,j^r}$ over all r. The witness is given by the data structure of Lemma 9.

  Note that when querying A^r and B^r, we only need to pass the set $\{k \in S : A^r_{i,k} \ne \infty\}$. For every $k \in S$, there is at most one r such that $A^r_{i,k} \ne \infty$, so the total size of the sets passing to the data structure of Lemma 9 is $|S|$. Thus, this case takes $O(|S| + \rho P)$ time.
- (i,k^*) is uncovered. There are still two possibilities to consider in this case.
  - **Possibility I**: $A_{i,k^*} + \hat{B}_{k^*,j} - \hat{C}^L_{i,j} < -2W$. In this case,

    $$A_{i,k^*} + B_{k^*,j} \le A_{i,k^*} + \hat{B}_{k^*,j} + W < \hat{C}^L_{i,j} - W,$$

    so the optimum value is smaller than $\hat{C}^L_{i,j}$. By reading the list $\mathcal{L}^{i,\lceil j/\Delta\rceil\Delta}_{\text{small}}$, we can effectively find all such k where $A_{i,k} + \hat{B}_{k,j} - \hat{C}^L_{i,j} < -2W$ in time linear to the number of such k. The number of such k is at most L, by the definition of $\hat{C}^L_{i,j}$. Thus, this part takes $O(L)$ time.
  - **Possibility II**: $A_{i,k^*} + \hat{B}_{k^*,j} - \hat{C}^L_{i,j} \ge -2W$. In fact, in this case, we further have

    $$A_{i,k^*} + \hat{B}_{k^*,j} - \hat{C}^L_{i,j} \le A_{i,k^*} + B_{k^*,j} - C^L_{i,j} + 2W \le 2W,$$

    where $A_{i,k^*} + B_{k^*,j} - C^L_{i,j} \le 0$ because $|S| < L$. Therefore, in this case, we have $|A_{i,k^*} + \hat{B}_{k^*,j} - \hat{C}^L_{i,j}| \le 2W$, so we can enumerate all indices in $\mathcal{L}^{i,j}_{\text{triple}}$ and take the best choice. This takes $O(L)$ time.

Time and Space Complexity

In summary, the preprocessing time is

$$\tilde{O}\left(\rho \frac{n^s}{P} W n^{\omega(s)} + n^{2+s}/\Delta + n^{2+s}/\rho\right),$$

and the query time is $\tilde{O}(L + \rho P)$. To balance the terms, we can set $\rho = \Delta$ and $P = \frac{L}{\Delta}$ to achieve a $\tilde{O}(\Delta^2 \frac{n^s}{L} W n^{\omega(s)} + \frac{n^{2+s}}{\Delta})$ preprocess time and a $\tilde{O}(L)$ query time. Note that since we need $P \ge 1$, we must have $L = \Omega(\Delta)$.

From the preprocessing steps, the space complexity is $\tilde{O}(\rho\frac{Wn^{2+s}}{P}+\rho\frac{n^{1+2s}}{P}+n^{2+s}/\rho)$. Plugging in $\rho=\Delta$ and $P=\frac{L}{\Delta}$ reduces this to

$$\tilde{O}\left(\frac{\Delta^2 W n^{2+s}}{L}+\frac{\Delta^2 n^{1+2s}}{L}+\frac{n^{2+s}}{\Delta}\right),$$

as given in the statement of the lemma. ◀

The next lemma is our last data structure for Min-Plus-Query-Witness problems.

▶ **Lemma 11.** *Let $s \geq 1$ be a constant. Let A be an $n \times n^s$ integer matrix and B be an $n^s \times n$ integer matrix. Suppose matrix B satisfies*

1. *Each row of B is non-increasing;*
2. *The difference between the sum of elements in the j-th column and the sum of elements in the $(j+1)$-th column is at most n^s, for any j.*

Then for every positive integer $L=\Omega(n^{\omega(s)-2})$, we can solve the Min-Plus-Query-Witness problem of A and B in $\tilde{O}(n^{\frac{8}{5}+s+\frac{1}{5}\omega(s)}L^{-\frac{1}{5}})$ preprocessing time and $\tilde{O}(L)$ query time, when $|S|<L$. The space complexity is $\tilde{O}(L^{-\frac{1}{5}}n^{\frac{18}{5}+s-\frac{4}{5}\omega(s)}+L^{-\frac{3}{5}}n^{\frac{9}{5}+2s-\frac{2}{5}\omega(s)}+L^{-\frac{1}{5}}n^{\frac{8}{5}+s+\frac{1}{5}\omega(s)})$.

Proof. Let $\Delta, W \geq 1$ be small polynomials in n to be fixed later. Define $I(j)$ to be the interval $[j-\Delta+1, j]$.

Let j' be any multiple of Δ. By property **2** of matrix B, $\sum_{k=1}^{n^s} B_{k,j}-\sum_{k=1}^{n^s} B_{k,j+1} \leq n^s$ for any $j \in I(j')$. Thus, we have

$$\sum_{k=1}^{n^s} B_{k,j'-\Delta+1}-\sum_{k=1}^{n^s} B_{k,j'} \leq \Delta n^s.$$

By averaging, there are at most $\Delta n^s/W$ indices $k \in [n^s]$ such that $B_{k,j'-\Delta+1}-B_{k,j'}>W$. We create a new matrix $\hat{B}$, initially the same as matrix B. For each k such that $B_{k,j'-\Delta+1}-B_{k,j'}>W$, and for each $j \in I(j')$, we set $\hat{B}_{k,j}$ as M, where M is some large enough integer. After this replacement, $\hat{B}_{k,j'-\Delta+1}-\hat{B}_{k,j'} \leq W$ for any k and any j' multiple of Δ. Also, since $\hat{B}_{k,j'-\Delta+1} \geq \hat{B}_{k,j} \geq \hat{B}_{k,j'}$ for any $j \in I(j')$, we have that $|\hat{B}_{k,j_1}-\hat{B}_{k,j_2}| \leq W$ as long as $\lceil j_1/\Delta\rceil=\lceil j_2/\Delta\rceil$. Therefore, we can use Lemma 10 to preprocess A and $\hat{B}$ in $O(\Delta^2\frac{n^s}{L}Wn^{\omega(s)}+\frac{n^{2+s}}{\Delta})$ time. The space complexity is $\tilde{O}(\frac{\Delta^2 W n^{2+s}}{L}+\frac{\Delta^2 n^{1+2s}}{L}+\frac{n^{2+s}}{\Delta})$.

On the other hand, note that $\hat{B}$ differs with B on at most $n^{1+s}\Delta/W$ entries, so we need to do some extra preprocessing to handle those entries. For each pair (i,j), we initialize a range tree $\mathcal{T}^{(i,j)}$ whose elements are all ∞ (it takes $\tilde{O}(1)$ time to initialize each range tree if we implement it carefully). Then for every k such that $B_{k,j} \neq \hat{B}_{k,j}$, we set the k-th element in $\mathcal{T}^{(i,j)}$ as $A_{i,k}+B_{k,j}$. The total number of operations we perform in all the range trees are $O(n^{2+s}\Delta/W)$, so this part takes $\tilde{O}(n^{2+s}\Delta/W)$ time. The space complexity is also $\tilde{O}(n^{2+s}\Delta/W)$.

During a query (S,i,j), we first query the data structure in Lemma 10 on matrix A and $\hat{B}$ with parameters (S,i,j). Then we query the minimum value from the range tree $\mathcal{T}^{(i,j)}$ after setting all $A_{i,k}+B_{k,j}$ as ∞ for $k \in S$. Taking the minimum of these two queries gives the answer. The optimum index k^* is either given by the data structure of Lemma 10 or can be obtained from the range tree.

Thus, the preprocessing time of the algorithm is

$$\tilde{O}(\Delta^2\frac{n^s}{L}Wn^{\omega(s)}+\frac{n^{2+s}}{\Delta}+n^{2+s}\Delta/W),$$

and the query time is $\tilde{O}(L)$. We get the desired preprocessing time by setting $\Delta = L^{1/5}n^{\frac{2}{5}-\frac{1}{5}\omega(s)}$ and $W = \Delta^2$. Since we need $\Delta \geq 1$, we require that $L = \Omega(n^{\omega(s)-2})$. In Lemma 10, we also requires that $L = \Omega(\Delta)$, but this is always true when $L = \Omega(n^{\omega(s)-2})$.

By previous discussion, the space complexity is $\tilde{O}(\frac{\Delta^2 W n^{2+s}}{L} + \frac{\Delta^2 n^{1+2s}}{L} + \frac{n^{2+s}}{\Delta} + n^{2+s}\Delta/W)$. Plugging in the value for Δ and W simplifies the complexity to

$$\tilde{O}(L^{-1/5}n^{18/5+s-4\omega(s)/5} + L^{-3/5}n^{9/5+2s-2\omega(s)/5} + L^{-1/5}n^{8/5+s+\omega(s)/5}). \qquad ◀$$

Finally, we can apply the data structure of Lemma 11 to prove Theorem 2.

Proof of Theorem 2. For clarity, we will use *element* to refer to a specific item a_i of the sequence and use *value* to refer to all elements of a given type. Given a pointer to an element of the sequence a_i, we assume the ability to look up its index i in the sequence in $\tilde{O}(1)$ time by storing all elements of the sequence in a balanced binary search tree with worst-case time guarantees (e.g. a red-black tree). Thus we can go from index i to element a_i and back via appropriate rank and select queries on the balanced binary search tree. We may also add or remove an element a_i from the sequence, and thus the binary search tree, in $\tilde{O}(1)$ time.

Let T_1, T_2, T_3 be three parameters of the algorithm. Parameter T_1 is a threshold that controls the number of "frequent" colors, T_2 controls how frequently the data structure is rebuilt, and T_3 represents the size of blocks in the algorithm.

We call values that appear more than N/T_1 times *frequent* and all other values *infrequent*. Thus, there are at most T_1 frequent values at any point in time. Note that a fixed value can change from frequent to infrequent, or from infrequent to frequent, via a deletion or insertion.

Infrequent Values

First, we discuss how to handle infrequent values. We maintain $\frac{N}{T_1}$ balanced search trees $\mathcal{BST}_1, \ldots, \mathcal{BST}_{\frac{N}{T_1}}$. For balanced search tree $\mathcal{BST}_k$, we prepare the key/value pairs in the following way. Fix a given value of the sequence. Say all its occurrences are at indices $i_1, i_2, \ldots, i_t$. Then we insert the key/value pairs (i_x, i_{x+k-1}) to $\mathcal{BST}_k$ for every $1 \leq x \leq t-k+1$. However, the indices themselves would need updating when sequence a is updated. Instead of inserting the indices themselves, we insert corresponding pointers to the nodes of the binary search tree that holds sequence a. That way we can perform all comparisons using binary search tree operations in $\tilde{O}(1)$ time, without needing to update indices when sequence a changes. We also augment each balanced search tree $\mathcal{BST}_i$ so that every subtree stores the smallest value y of any pair (x, y) in the subtree. After an insertion or deletion, we need to update a total of $O((\frac{N}{T_1})^2)$ pairs. Thus, we can maintain these balanced search trees in $\tilde{O}((\frac{N}{T_1})^2)$ time per operation.

During a query $[l, r]$, we iterate through all the balanced search trees $\mathcal{BST}_1, \ldots, \mathcal{BST}_{\frac{N}{T_1}}$. If there exists a pair $(i_1, i_2) \in \mathcal{BST}_k$ such that $l \leq i_1 \leq i_2 \leq r$, then the range mode is at least k. Thus, if the range mode is an infrequent value, we can find its frequency and corresponding value by querying the balanced search trees. The query time is $\tilde{O}(\frac{N}{T_1})$, which is not the dominating term.

Newly Modified Values

We now consider how to handle frequent values. We handle newly modified values and unmodified values differently. We will rebuild our data structure after every T_2 operations, and call values that are inserted or deleted after the last rebuild *newly modified values.*

For every value, we maintain a balanced search tree of occurrences of this value in the sequence. It takes $\tilde{O}(1)$ time per operation to maintain such balanced search trees. Thus, given an interval $[l, r]$, it takes $\tilde{O}(1)$ time to query the number of occurrences of a particular value in the interval. We use this method to query the number of occurrences of each newly modified value. Since there can be at most T_2 such values, this part takes $\tilde{O}(T_2)$ time per operation.

Data Structure Rebuild

It remains to handle the frequent, not newly modified values during each rebuild. In this case, we will assume we can split the whole array roughly equally into a left half and right half. We can recursively build the data structure on these two halves so that we may assume a range mode query interval has left endpoint in the left half and right endpoint in the right half. The recursive construction adds only a poly-logarithmic factor to the complexity.

We split the left half and the right half into consecutive segments of length at most T_3, so that there are $O(N/T_3)$ segments. We call the segments $P_1, P_2, \ldots, P_m$ in the left half and $Q_1, Q_2, \ldots, Q_m$ in the right half, where segments with a smaller index are closer to the middle of the sequence.

Let $v_1, v_2, \ldots, v_l$ be the frequent values during the rebuild. We create a matrix A such that $A_{i,k}$ equals the negation of the number of occurrences of v_k in segments $P_1, \ldots, P_i$; similarly, we create a matrix B such that $B_{k,j}$ equals the negation of the number of occurrences of v_k in segments $Q_1, \ldots, Q_j$. Note that the negation of the value $A_{i,k} + B_{k,j}$ is the frequency of value v_k in the interval from P_i to Q_j. It is not hard to verify that matrix B satisfies the requirement of Lemma 11. We take the negation here since Lemma 11 handles $(\min, +)$-product instead of $(\max, +)$-product. Then we use the preprocessing part of Lemma 11 with matrices A, B, and $L = T_2$. If we let $T_1 = N^{t_1}, T_2 = N^{t_2}, T_3 = N^{t_3}$, then in the notation of Lemma 11, $n = m = O(N/T_3) = O(N^{1-t_3})$ and $n^s = O(T_1) = O(N^{t_1})$, so $s = \frac{t_1}{1-t_3}$ and $L = N^{t_2}$. Thus, by Lemma 11 the rebuild takes

$$\tilde{O}(N^{(1-t_3)(\frac{8}{5}+\frac{t_1}{1-t_3}+\frac{1}{5}\omega(\frac{t_1}{1-t_3}))-\frac{1}{5}t_2})$$

time. Since we perform the rebuild every T_2 operations, the amortized cost of rebuild is

$$\tilde{O}(N^{(1-t_3)(\frac{8}{5}+\frac{t_1}{1-t_3}+\frac{1}{5}\omega(\frac{t_1}{1-t_3}))-\frac{6}{5}t_2})$$

per operation.

Now we discuss how to handle queries for frequent, unmodified elements. For a query interval $[l, r]$, we find all the segments inside the interval $[l, r]$. The set of such segments must have the form $P_1, \cup \cdots \cup P_i \cup Q_1 \cup \cdots \cup Q_j$ for some i, j. We scan through all elements in $[l, r] \setminus (P_1 \cup \cdots \cup P_i \cup Q_1 \cup \cdots \cup Q_j)$, and use their frequency to update the answer. Since the size of segments is $O(T_3)$, the time complexity to do so is $\tilde{O}(T_3)$.

For the segments $P_1, \ldots, P_i, Q_1, \ldots, Q_j$, we query the data structure in Lemma 11 with S being the set of newly modified elements. The answer will be the most frequent element in the interval from P_i to Q_j that is not newly modified. By Lemma 11 this takes $O(L) = O(N^{t2})$ time per operation.

Time and Space Complexity

In summary, the amortized cost per operation is

$$\tilde{O}(N^{2-2t_1} + N^{t_2} + N^{t_3} + N^{(1-t_3)(\frac{8}{5}+\frac{t_1}{1-t_3}+\frac{1}{5}\omega(\frac{t_1}{1-t_3}))-\frac{6}{5}t_2}).$$

To balance the terms, we set $t_1 = 1 - \frac{1}{2}t_2$, and $t_3 = t_2$. The time complexity thus becomes

$$\tilde{O}(N^{t_2} + N^{(1-t_2)(\frac{8}{5}+\frac{1-0.5t_2}{1-t_2}+\frac{1}{5}\omega(\frac{1-0.5t_2}{1-t_2}))-\frac{6}{5}t_2}).$$

By observation, we can note that the optimum value of $\frac{1-0.5t_2}{1-t_2}$ lies in $[1.75, 2]$. Thus, we can plug in Corollary 6 and use $t_2 = 0.655994$ to balance the two terms. This gives an $\tilde{O}(N^{0.655994})$ *amortized* time per operation algorithm.

The space usage has two potential bottlenecks. The first is the space to store $\mathcal{BST}_1, \ldots, \mathcal{BST}_{\frac{N}{T_1}}$ for handling infrequent elements, which is $\tilde{O}(\frac{N^2}{T_1})$. The second is the space used by Lemma 11, which is

$$\tilde{O}(N^{-t_2/5}N^{(1-t_3)(18/5+s-4\omega(s)/5)}+N^{-3t_2/5}N^{(1-t_3)(9/5+2s-2\omega(s)/5)}+N^{-t_2/5}N^{(1-t_3)(8/5+s+\omega(s)/5)}).$$

By plugging in the values for t_2, t_3 and s, the space complexity becomes $\tilde{O}(N^{1.327997})$, with the $\tilde{O}(\frac{N^2}{T_1})$ term being the dominating term.

Worst-Case Time Complexity

By applying the *global rebuilding* of Overmars [21], we can achieve a worst-case time bound. The basic idea is that after T_2 operations, we don't immediately rebuild the Min-Plus-Query-Witness data structure. Instead, we rebuild the data structure during the next T_2 operations, spreading the work evenly over each operation. To answer queries during these T_2 operations, we use the previous build of the Min-Plus-Query-Witness data structure. By this technique, the per-operation runtime can be made worst-case. ◀

References

1 Noga Alon, Zvi Galil, and Oded Margalit. On the exponent of the all pairs shortest path problem. *J. Comput. Syst. Sci.*, 54(2):255–262, 1997.

2 Prosenjit Bose, Evangelos Kranakis, Pat Morin, and Yihui Tang. Approximate range mode and range median queries. In *Annual Symposium on Theoretical Aspects of Computer Science*, 2005.

3 Karl Bringmann, Fabrizio Grandoni, Barna Saha, and Virginia Williams. Truly sub-cubic algorithms for language edit distance and rna folding via fast bounded-difference min-plus product. *SIAM Journal on Computing*, 48, July 2017. doi:10.1137/17M112720X.

4 Gerth Stølting Brodal, Beat Gfeller, Allan Jørgensen, and Peter Sanders. Towards optimal range medians. In *Theoretical Computer Science*, 2011.

5 Timothy M. Chan. More algorithms for all-pairs shortest paths in weighted graphs. *SIAM J. Comput.*, 39(5):2075–2089, 2010.

6 Timothy M. Chan, Stephane Durocher, Kasper Green Larsen, Jason Morrison, and Bryan T. Wilkinson. Linear-space data structures for range mode query in arrays. *Theory of Computing Systems*, 55:719–741, 2014.

7 Timothy M Chan and Konstantinos Tsakalidis. Dynamic orthogonal range searching on the ram, revisited. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 77. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

8 Hicham El-Zein, Meng He, J. Ian Munro, Yakov Nekrich, and Bryce Sandlund. On approximate range mode and range selection. In *30th International Symposium on Algorithms and Computation (ISAAC 2019)*, 2019.

9 Hicham El-Zein, Meng He, J. Ian Munro, and Bryce Sandlund. Improved time and space bounds for dynamic range mode. In *Proceedings of the 26th Annual European Symposium on Algorithms*, pages 25:1–25:13, 2018.

10 Amr Elmasry, Meng He, J Ian Munro, and Patrick K Nicholson. Dynamic range majority data structures. In *International Symposium on Algorithms and Computation*, pages 150–159. Springer, 2011.

11 Michael J Fischer and Albert R Meyer. Boolean matrix multiplication and transitive closure. In *n 12th Annual Symposium on Switching and Automata Theory*, pages 129–131. IEEE, 1971.

12 Travis Gagie, Meng He, and Gonzalo Navarro. Compressed dynamic range majority data structures. In *Data Compression Conference (DCC), 2017*, pages 260–269. IEEE, 2017.

13 François Le Gall. Powers of tensors and fast matrix multiplication. In *International Symposium on Symbolic and Algebraic Computation, ISSAC 2014, Kobe, Japan, July 23-25, 2014*, pages 296–303, 2014.

14 François Le Gall and Florent Urrutia. Improved rectangular matrix multiplication using powers of the coppersmith-winograd tensor. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1029–1046. SIAM, 2018.

15 Mark Greve, Allan Grønlund Jørgensen, Kasper Dalgaard Larsen, and Jakob Truelsen. Cell probe lower bounds and approximations for range mode. In *International Colloquium on Automata, Languages, and Programming*, 2010.

16 Meng He, J. Ian Munro, and Patrick K. Nicholson. Dynamic range selection in linear space. In *International Symposium on Algorithms and Computation*, 2011.

17 Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing*, page 21–30, 2015.

18 Danny Krizanc, Pat Morin, and Michiel Smid. Range mode and range median queries on lists and trees. *Nordic Journal of Computing*, 2005.

19 François Le Gall. Faster algorithms for rectangular matrix multiplication. In *2012 IEEE 53rd annual symposium on foundations of computer science*, pages 514–523. IEEE, 2012.

20 Grazia Lotti and Francesco Romani. On the asymptotic complexity of rectangular matrix multiplication. *Theoretical Computer Science*, 23(2):171–185, 1983.

21 Mark H Overmars. *The design of dynamic data structures*, volume 156. Springer Science & Business Media, 1983.

22 Mihai Pătraşcu and Emanuele Viola. Cell-probe lower bounds for succinct partial sums. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 117–122. Society for Industrial and Applied Mathematics, 2010.

23 Raimund Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. *J. Comput. Syst. Sci.*, 51(3):400–403, 1995.

24 Avi Shoshan and Uri Zwick. All pairs shortest paths in undirected graphs with integer weights. In *40th Annual IEEE Symposium on Foundations of Computer Science*, pages 605–615, 1999.

25 Virginia Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 887–898, 2012.

26 Virginia Vassilevska Williams and Yinzhan Xu. Truly subcubic min-plus product for less structured matrices, with applications. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms*, 2020.

27 Raphael Yuster. Efficient algorithms on sets of permutations, dominance, and real-weighted apsp. In *20th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 950–957, 2009.

28 Uri Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *J. ACM*, 49(3):289–317, 2002.

An FPT-Algorithm for Recognizing k-Apices of Minor-Closed Graph Classes

Ignasi Sau
LIRMM, Université de Montpellier, CNRS, France
ignasi.sau@lirmm.fr

Giannos Stamoulis
Department of Informatics and Telecommunications,
National and Kapodistrian University of Athens, Greece
Inter-university Postgraduate Programme "Algorithms, Logic, and Discrete Mathematics" (ALMA),
Athens, Greece
giannosstam@di.uoa.gr

Dimitrios M. Thilikos
LIRMM, Université de Montpellier, CNRS, France
sedthilk@thilikos.info

Abstract

Let $\mathcal{G}$ be a graph class. We say that a graph G is a k-*apex* of $\mathcal{G}$ if G contains a set S of at most k vertices such that $G \setminus S$ belongs to $\mathcal{G}$. We prove that if $\mathcal{G}$ is minor-closed, then there is an algorithm that either returns a set S certifying that G is a k-apex of $\mathcal{G}$ or reports that such a set does not exist, in $2^{\mathsf{poly}(k)}n^3$ time. Here poly is a polynomial function whose degree depends on the maximum size of a minor-obstruction of $\mathcal{G}$, i.e., the minor-minimal set of graphs not belonging to $\mathcal{G}$. In the special case where $\mathcal{G}$ excludes some apex graph as a minor, we give an alternative algorithm running in $2^{\mathsf{poly}(k)}n^2$ time.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Graph modification problems, irrelevant vertex technique, graph minors, parameterized algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.95

Category Track A: Algorithms, Complexity and Games

Related Version The full version of this paper is permanently available at `https://arxiv.org/abs/2004.12692`.

Funding *Ignasi Sau*: Supported by projects DEMOGRAPH (ANR-16-CE40-0028) and ESIGMA (ANR-17-CE23-0010).
Dimitrios M. Thilikos: Supported by projects DEMOGRAPH (ANR-16-CE40-0028) and ESIGMA (ANR-17-CE23-0010). Supported by the Research Council of Norway and the French Ministry of Europe and Foreign Affairs, via the Franco-Norwegian project PHC AURORA 2019.

Acknowledgements We wish to thank the anonymous reviewers for their comments and remarks that improved the presentation of this paper. Moreover, we wish to thank Dániel Marx for his valuable comments and advise.

1 Introduction

Graph modification problems are fundamental in algorithmic graph theory. Typically, such a problem is determined by a graph class $\mathcal{G}$ and some prespecified set $\mathcal{M}$ of *local* modifications, and the question is, given a graph G and an integer k, whether it is possible to transform G to a graph in $\mathcal{G}$ by applying k modification operations from $\mathcal{M}$. A plethora of graph

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 95; pp. 95:1–95:20
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

problems can be formulated for different instantiations of $\mathcal{G}$ and $\mathcal{M}$. Applications span diverse topics such as computational biology, computer vision, machine learning, networking, and sociology [24]. As reported by Roded Sharan in [48], already in 1979, Garey and Johnson mentioned 18 different types of modification problems [25, Section A1.2]. For more on graph modification problems, see [9, 24], as well as the running survey in [13]. In this paper we focus our attention on the vertex deletion operation. We say that a graph G is a *k-apex* of a graph class $\mathcal{G}$ if there is a set $S \subseteq V(G)$ of size at most k such that the removal of S from G results in a graph in $\mathcal{G}$. In other words, we consider the following meta-problem.

Vertex Deletion to $\mathcal{G}$
Input: A graph G and a non-negative integer k.
Question: Find, if exists, a set $S \subseteq V(G)$ certifying that G is a k-apex of $\mathcal{G}$.

To illustrate the expressive power of Vertex Deletion to $\mathcal{G}$, if $\mathcal{G}$ is the class of edgeless (resp. acyclic, planar, bipartite, (proper) interval, chordal) graphs, we obtain the Vertex Cover (resp. Feedback Vertex Set, Vertex Planarization, Odd Cycle Transversal, (proper) Interval Vertex Deletion, Chordal Vertex Deletion) problem.

By the classical result of Lewis and Yannakakis [39], Vertex Deletion to $\mathcal{G}$ is NP-hard for every non-trivial graph class $\mathcal{G}$. To circumvent its intractability, we study it from the parameterized complexity point of view and we parameterize it by the number k of vertex deletions. In this setting, the most desirable behavior is the existence of an algorithm running in time $f(k) \cdot n^{\mathcal{O}(1)}$, where f is a function depending only on k. Such an algorithm is called *fixed-parameter tractable*, or FPT-algorithm for short, and a parameterized problem admitting an FPT-algorithm is said to belong to the parameterized complexity class FPT. Also, the function f is called *parametric dependence* of the corresponding FPT-algorithm, and the challenge is to design FPT-algorithms with small parametric dependencies [14, 17, 20, 42].

Unfortunately, we cannot hope for the existence of FPT-algorithms for every graph class $\mathcal{G}$. Indeed, the problem is W-hard[1] for some classes $\mathcal{G}$ that are closed under induced subgraphs [40] or, even worse, NP-hard, for $k = 0$, for every class $\mathcal{G}$ whose recognition problem is NP-hard, such as some classes closed under subgraphs or induced subgraphs (for instance 3-colorable graphs), edge contractions [11], or induced minors [18].

On the positive side, a very relevant subset of classes of graphs *does* allow for FPT-algorithms. These are classes $\mathcal{G}$ that are closed under minors[2], or *minor-closed*. To see this, we define $\mathcal{G}_k$ as the class of the k-apices of $\mathcal{G}$, i.e., the yes-instances of Vertex Deletion to $\mathcal{G}$, and observe that if $\mathcal{G}$ is minor-closed then the same holds for $\mathcal{G}_k$, for every k. This, in turn, implies that for every k, $\mathcal{G}_k$ can be characterized by a set $\mathcal{F}_k$ of minor-minimal graphs not in $\mathcal{G}_k$; we call these graphs the *obstructions* of $\mathcal{G}_k$ and we know that they are finite because of the Robertson and Seymour theorem [46]. In other words, we know that the obstruction set of $\mathcal{G}_k$ is bounded by some function of k. Then one can decide whether a graph G belongs to $\mathcal{G}_k$ by checking whether G excludes all members of the obstruction set of $\mathcal{G}_k$, and this can be checked by using the FPT-algorithm in [45] (see also [19]).

As the Robertson and Seymour theorem [46] does not construct $\mathcal{F}_k$, the aforementioned argument is not constructive, i.e., it is not able to construct the claimed FPT-algorithm. An important step towards the constructibility of such an FPT-algorithm was done by Adler et al. [2], who proved that the parametric dependence of the above FPT-algorithm is indeed a constructible function.

[1] Implying that an FPT-algorithm would result in an unexpected complexity collapse; see [17].
[2] A graph H is a *minor* of a graph G if it can be obtained from a subgraph of G by contracting edges.

The task of specifying (or even optimizing) this parametric dependence for different instantiations of $\mathcal{G}$ occupied a considerable part of research in parameterized algorithms. The most general result in this direction says that, for every t, there is some c such that if the graphs in $\mathcal{G}$ have treewidth at most t, then Vertex Deletion to $\mathcal{G}$ admits an FPT-algorithm that runs in $c^k \cdot n^{\mathcal{O}(1)}$ time [22,34]. Reducing the constant c in this running time has attracted research on particular problems such as Vertex Cover [12] (with $c = 1.2738$), Freedback Vertex Set [36] (with $c = 3.619$), Apex-Pseudoforest [10] (with $c = 3$), Pathwidth 1 Vertex Deletion (with $c = 4.65$) [15] (see also [29] for further related results). The first step towards a parameterized algorithm for Vertex Deletion to $\mathcal{G}$ for cases where $\mathcal{G}$ has unbounded treewidth was done in [41] and later in [30] for the Planarization problem, and the best parameterized dependence for this problem is $2^{\mathcal{O}(k \cdot \log k)} \cdot n$, achieved by Jansen et al. [28]. These results were later extended by Kociumaka and Marcin Pilipczuk [37], who proved that if $\mathcal{G}_g$ is the class of graphs of Euler genus at most g, then Vertex Deletion to $\mathcal{G}_g$ admits a $2^{\mathcal{O}_g(k^2 \log k)} \cdot n^{\mathcal{O}(1)}$ step[3] algorithm.

Our results. In this paper we give an explicit FPT-algorithm for Vertex Deletion to $\mathcal{G}$ for *every* minor-closed graph class $\mathcal{G}$. In particular, our main results are the following:

▶ **Theorem 1.** *If $\mathcal{G}$ is a minor-closed graph class, then Vertex Deletion to $\mathcal{G}$ admits an algorithm of time $2^{\mathsf{poly}(k)} \cdot n^3$, for some polynomial* poly *whose degree depends on $\mathcal{G}$.*

We say that a graph H is an *apex* graph if it is a 1-apex of the class of planar graphs.

▶ **Theorem 2.** *If $\mathcal{G}$ is a minor-closed graph class excluding some apex graph, then Vertex Deletion to $\mathcal{G}$ admits an algorithm of time $2^{\mathsf{poly}(k)} \cdot n^2$, for some polynomial* poly *whose degree depends on $\mathcal{G}$.*

Very recently, Fomin et al. [23] gave an FPT-algorithm of $\mathcal{O}_{s,k}(n^4)$ time for the following problem: for a fixed finite family of graphs $\mathcal{F}$, each on at most s vertices, decide whether an n-vertex input graph G contains a k-apex of the class of graphs that exclude the graphs in $\mathcal{F}$ as *topological minors*[4]. For every graph H, there is a finite set $\mathcal{H}$ of graphs such that a graph G contains H as a minor if and only if G contains a graph in $\mathcal{H}$ as a topological minor. Based on this observation, the result of Fomin et al. [23] implies that for every minor-closed graph class $\mathcal{G}$, Vertex Deletion to $\mathcal{G}$ admits an $\mathcal{O}(h(k,s) \cdot n^4)$ time FPT-algorithm,where s is the maximum size of an obstruction of $\mathcal{G}$. Notice that this implication is a solid improvement on Vertex Deletion to $\mathcal{G}$ with respect to the result of [2], where only the computability of h is proved. However, as mentioned in [23], even for fixed values of s, the dependence of h on k is humongous. Therefore, Theorem 1 can be seen as orthogonal to the result of [23].

Our techniques. We provide here just a very succinct enumeration of the techniques that we use in order to achieve Theorem 1 and Theorem 2; a more detailed description with the corresponding definitions is provided, along with the algorithms, in the next sections.

Our starting point to prove Theorem 1 is to use the standard iterative compression technique of Reed et al. [44]. This allows us to assume that we have at hand a slightly too large set $S \subseteq V(G)$ such that $G \setminus S \in \mathcal{G}$. We then run the algorithm of Lemma 11 (whose proof uses [1,3,31,43]) that either reports that we have a no-instance, or concludes that the

[3] Given a tuple $\mathbf{t} = (x_1, \ldots, x_\ell) \in \mathbb{N}^\ell$ and two functions $\chi, \psi : \mathbb{N} \to \mathbb{N}$. We write $\chi(n) = \mathcal{O}_{\mathbf{t}}(\psi(n))$ in order to denote that there exists a computable function $\phi : \mathbb{N}^\ell \to \mathbb{N}$ such that $\chi(n) = \mathcal{O}(\phi(\mathbf{t}) \cdot \psi(n))$.

[4] The definition is as minors, except that only edges incident to degree-two vertices are contracted.

treewidth of G is polynomially bounded by k, or finds a large wall R in G. In the second case, we use the main algorithmic result of Baste et al. [5] (Proposition 3) to solve the problem parameterized by treewidth, achieving the claimed running time. In the latter case, we apply Proposition 12 (whose proof uses [8, 32, 33]) to find in R a large *flat* wall W together with an apex set A. We find in W a packing of an appropriate number of pairwise disjoint large enough subwalls. Two possible scenarios may occur. If the "interior" of each of these subwalls has enough neighbors in the set apex $S \cup A$, we apply a combinatorial result (Lemma 15) that guarantees that every possible solution should intersect $S \cup A$, and we can branch on it. On the other hand, if there exists a subwall whose interior W has few neighbors in $S \cup A$, we argue that we can define from it a *flat* wall in which we can apply the irrelevant vertex technique of Robertson and Seymour [45] (Lemma 14). We stress that this flat subwall is not precisely a subwall of W but a tiny "tilt" of a subwall of W, a new concept that is necessary for our proofs. The application of the irrelevant vertex technique requires a lot of technical care. For this, we use and enhance some of the ingredients introduced by Baste et al. [5].

In order to achieve the improved running time claimed in Theorem 2, we do *not* use iterative compression. Instead, we directly invoke Lemma 11. If the treewidth is small, we proceed as above. If a large wall is found, we apply Proposition 12 and we now distinguish two cases. If a large flat wall whose flaps have bounded treewidth is found, we find an irrelevant vertex using Lemma 14. Otherwise, inspired by an idea of Marx and Schlotter [41], we exploit the fact that $\mathcal{G}$ excludes an apex graph, and we use flow techniques to either find a vertex that should belong to the solution, or to conclude that we are dealing with a no-instance.

Organization. We provide in Section 2 some definitions and preliminary results. In Section 3 we state several algorithmic and combinatorial results that will be used when finding an irrelevant vertex and when applying the branching argument discussed above. In Section 4 we present the algorithms claimed in Theorem 1 and Theorem 2. We conclude in Section 5 with some directions for further research. All the missing proofs are available in the full version of the paper [47].

2 Definitions and preliminary results

Before we explain our techniques, we give some necessary definitions. They concern fundamental tools from the Graph Minors series of Robertson and Seymour that are heavily used in our algorithms and proofs. But first, we restate the problem in a more convenient way.

2.1 Restating the problem

Let $\mathcal{F}$ be a finite non-empty collection of non-empty graphs. We use $\mathcal{F} \leq_{\mathsf{m}} G$ to denote that some graph in $\mathcal{F}$ is a minor of G.

Let $\mathcal{G}$ be a minor-closed graph class and $\mathcal{F}$ be the set of its minor-obstructions. Clearly, Vertex Deletion to $\mathcal{G}$ is the same problem as asking, given a graph G and some $k \in \mathbb{N}$, whether G contains a vertex set S of at most k vertices such that $\mathcal{F} \not\leq_{\mathsf{m}} G \setminus S$. Following the terminology of [4–7, 22, 23, 34, 35], we call this problem $\mathcal{F}$-M-Deletion. In order to prove Theorem 1, we apply the iterative compression technique (introduced in [44]; see also [14]) and we give a $2^{\mathsf{poly}(k)} \cdot n^2$ time algorithm for the following problem.

$\mathcal{F}$-M-Deletion-Compression
Input: A graph G, a $k \in \mathbb{N}$, and a set S of size $k+1$ such that $\mathcal{F} \not\leq_{\mathsf{m}} G \setminus S$.
Objective: Find, if exists, a set $S' \subseteq V(G)$ of size at most k such that $\mathcal{F} \not\leq_{\mathsf{m}} G \setminus S'$.

Some conventions. In what follows we always denote by $\mathcal{F}$ the obstruction set of the minor-closed class $\mathcal{G}$ of the instantiation of VERTEX DELETION TO $\mathcal{G}$ that we consider. Also, given a graph G, we define its *apex number* to be the smallest integer a for which G is an a-apex of the class of planar graphs. We define three constants depending on $\mathcal{F}$ that will be used throughout the paper whenever we consider such a collection $\mathcal{F}$. We define $a_{\mathcal{F}}$ as the minimum apex number of a graph in $\mathcal{F}$, we set $s_{\mathcal{F}} = \max\{|V(H)| \mid H \in \mathcal{F}\}$, and we set $\ell_{\mathcal{F}} = \max\{|E(H)| + |V(H)| \mid H \in \mathcal{F}\}$. We also agree that n is the size of the input graph G. We can always assume that G has $\mathcal{O}_{s_{\mathcal{F}}}(k \cdot n)$ edges, otherwise we can directly conclude that (G, k) is a no-instance (for this, use the fact that $\mathcal{F}$-minor free graphs are sparse [38, 50]).

We present here the main result of [5]. We will use this in order to solve $\mathcal{F}$-M-DELETION on instances of bounded treewidth.

▶ **Proposition 3.** *Let $\mathcal{F}$ be a finite collection of graphs. There exists an algorithm that given a triple (G, tw, k) where G is a graph on n vertices and of treewidth at most tw, and k is a non-negative integer, it outputs, if it exists, a vertex set S of G of size at most k such that $\mathcal{F} \not\leq_m G \setminus S$. This algorithm runs in $2^{\mathcal{O}_{s_{\mathcal{F}}}(\mathsf{tw} \log \mathsf{tw})} \cdot n$ time.*

2.2 Definitions

We give here a minimal set of definitions and concepts that are necessary to support the description of our results. Some of them are given precisely, and for some of them we just provide enough intuition.

Renditions. Let Δ be a closed disk, i.e., a set homeomorphic to the set $\{(x, y) \in \mathbb{R}^2 \mid x^2 + y^2 \leq 1\}$. Given a subset X of Δ, we denote its closure by $\bar{X}$ and its boundary by $\mathbf{bor}(X)$. A Δ-*painting* is a pair $\Gamma = (U, N)$ where N is a finite set of points of Δ, $N \subseteq U \subseteq \Delta$, $U \setminus N$ has finitely many arcwise-connected components, called *cells*, such that, for every cell c, $\bar{c}$ is a closed disk, $\mathbf{bor}(c) \cap \Delta \subseteq N$, and $|\mathbf{bor}(c) \cap N| \leq 3$. We use the notation $U(\Gamma) := U$, $N(\Gamma) := N$ and denote the set of cells of Γ by $C(\Gamma)$. Notice that, given a Δ-painting Γ, the pair $(N(\Gamma), \{c \cap N \mid c \in C(\Gamma)\})$ is a hypergraph whose hyperedges have cardinality at most three, and Γ can be seen as a plane embedding of this hypergraph in Δ. Let G be a graph, and let Ω be a cyclic permutation of a subset of $V(G)$ that we denote by $V(\Omega)$. By an Ω-*rendition* of G we mean a triple (Γ, σ, π), where (a) Γ is a Δ-painting for some closed disk Δ, (b) $\pi : N(\Gamma) \to V(G)$ is an injection, and (c) σ assigns to each cell $c \in C(\Gamma)$ a subgraph $\sigma(c)$ of G, such that

(1) $G = \bigcup_{c \in C(\Gamma)} \sigma(c)$,
(2) for distinct $c, c' \in C(\Gamma)$, $\sigma(c)$ and $\sigma(c')$ are edge-disjoint,
(3) for every cell $c \in C(\Gamma)$, $\pi(c \cap N) \subseteq V(\sigma(c))$,
(4) for every cell $c \in C(\Gamma)$, $V(\sigma(c)) \cap \bigcup_{c' \in C(\Gamma) \setminus \{c\}} V(\sigma(c')) \subseteq \pi(c \cap N)$, and
(5) $\pi(N(\Gamma) \cap \mathbf{bor}(\Delta)) = V(\Omega)$, such that the points in $N(\Gamma) \cap \mathbf{bor}(\Delta)$ appear in $\mathbf{bor}(\Delta)$ in the same ordering as their images, via π, in Ω.

We say that an Ω-rendition (Γ, σ, π) of G is *tight* if the following conditions are satisfied:

(1) For every $c \in C(\Gamma)$, the graph $\sigma(c) \setminus \pi(c \cap N)$, when non-null, is connected and the neighborhood of its vertex set in G is $\pi(c \cap N)$.
(2) For every $c \in C(\Gamma)$ there are $|c \cap N|$ vertex-disjoint paths in G from $\pi(c \cap N)$ to the set $V(\Omega)$.
(3) If there are two points x, y of N such that $e = \{\pi(x), \pi(y)\} \in E(G)$, then there is a $c \in C(\Gamma)$ such that $\sigma(c)$ is the two-vertex connected graph $(e, \{e\})$.

It is easy to see that given an Ω-rendition of a graph G where $V(\Omega)$ contains at least three vertices that are in a cycle of G, a tight Ω-rendition of G can be constructed in $\mathcal{O}(n + m)$ steps.

Walls. We avoid here the detailed definition of an r-wall. As an intuitive alternative we provide the graph G in Figure 1 where an *elementary* 7*-wall* $\hat{W}$ is the graph with red and green vertices that has as edges the vertical and horizontal segments between a red and a green vertex. The 7*-wall* W is the spanning subgraph of G that is a subdivision of $\hat{W}$ with the black vertices as subdivision vertices. The *pegs* of $\hat{W}$ are depicted by the squared vertices, while the *corners* of $\hat{W}$ are the endpoints of the highest and the lowest horizontal path of $\hat{W}$, depicted by the fat squared vertices (the corners are also pegs). Notice that a wall W can occur in several ways from the elementary wall $\hat{W}$, depending on the way the vertices in the perimeter of $\hat{W}$ are subdivided. Each of them gives a different selection (P, C) of pegs and corners of W. We insist that, for every r-wall, the number r is always odd: for this, whenever an r-wall appears with r even, we agree to round it up to the next odd $r + 1$.

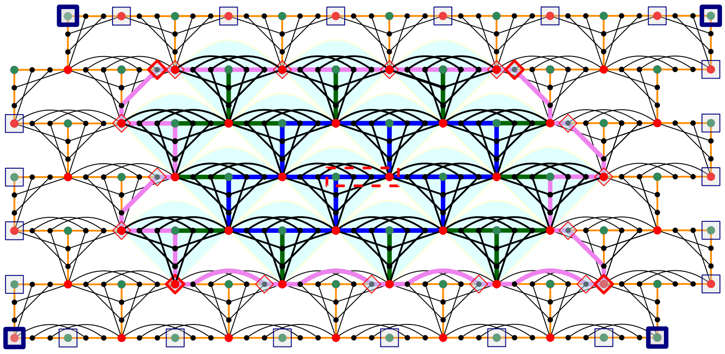


Figure 1 A graph G with a 7-wall and a flat 5-wall in it.

Flat walls. Given a graph G, we say that a pair $(L, R) \in 2^{V(G)} \times 2^{V(G)}$ is a *separation* of G if $L \cup R = V(G)$ and there is no edge in G between $L \setminus R$ and $R \setminus L$. An r-wall has a planar embedding where the boundary/ies of its external/internal face/es define its *perimeter* and its *bricks*. The *center* of the wall W is the path between a red and a green vertex depicted in the dashed red rectangle in Figure 1. Let G be a graph and let W be an r-wall of G. We say that W is a *flat* r*-wall* of G if there is a separation (X, Y) of G and a choice (P, C) of pegs and corners for W such that:

- $V(W) \subseteq Y$,
- $P \subseteq X \cap Y \subseteq V(D(W))$, and
- if Ω is the cyclic ordering of the vertices $X \cap Y$ as they appear in $D(W)$, then there exists an Ω-rendition (Γ, σ, π) of $G[Y]$.

A subwall of W is every subgraph W' of W that is located "orthocanonically" in W and the wall $W^{(r)}$ is the unique r-subwall of W with the same center as W. A subwall is called *internal* if it does not intersect the perimeter of W. In Figure 1, W contains only one internal 5-subwall $W' = W^{(5)}$ and many internal 3-subwalls, among them the wall $W''' = W^{(3)}$ (depicted in blue). Of course, the graph G in Figure 1 contains also other walls as subgraphs such as the wall W'' consisting of the purple, green, and blue edges.

Compass and flaps. Given a flat wall W of a graph G as above, we call $G[Y]$ *the compass* of W in G, denoted by $\mathsf{compass}(W)$. We call (X, Y) the *separation certifying the flat wall* W and $X \cap Y$ is called the *frontier* of W. The *ground set* of W is $\mathsf{ground}(W) := \pi(N(\Gamma))$. We clarify that $\mathsf{ground}(W)$ may consist of vertices of the compass of W that are not necessarily vertices of W (this is not the case in the simple example of Figure 1). We also call the

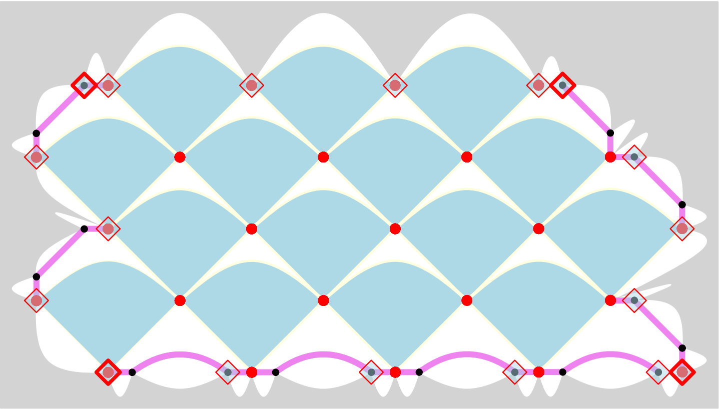


Figure 2 The rendition of the compass of the flat 5-wall W'' of Figure 1. Trivial flaps are depicted in purple.

graphs in $\mathsf{flaps}(W) := \{\sigma(c) \mid c \in C(\Gamma)\}$ *flaps* of the wall W. For each flap $F \in \mathsf{flaps}(W)$ we define its *base* as the set $\partial F := V(F) \cap \mathsf{ground}(W)$. A flap $F \in \mathsf{flaps}(W)$ is *trivial* if $|\partial F| = 2$ and it consists of one edge between the two vertices in ∂F. As an example, the wall W'' in Figure 1, formed by all the fat edges (purple, green, and blue), is a flat wall. The pegs are the diamond vertices, the corners are the fat diamond vertices, and the rendition has two types of flaps: those whose base has three vertices, that are inside the light-blue disks, and those that are trivial flaps and are the purple fat edges that are outside of the light-blue disks (see also Figure 2 for the rendition of W''). Notice that *none* of the internal subwalls of W is a flat wall.

Tilts. Given a wall W', we define its *inpegs* as the vertices of its perimeter that are incident to edges of W that are not in its perimeter. The *interior* of W' is the subgraph of W' induced by the union $V(W' \setminus V(P))$ and its inpegs. We say that a wall W'' is a *tilt* of a wall W' if W'' and W' have identical interiors. For instance, in Figure 1 the wall W'' is a tilt of $W' = W^{(5)}$.

Partially disk-embedded graphs. We say that a graph G is *partially disk-embedded in some closed disk* Δ, if there is some subgraph K of G that is embedded in Δ such that $(V(G) \cap \Delta, V(G) \setminus \mathsf{int}(\Delta))$ is a separation of G, where int is used to denote the interior of a subset of the plane. From now on, we use the term *partially* Δ*-embedded graph* G to denote that a graph G is partially disk-embedded in some closed disk Δ. We call the graph $K = G \cap \Delta$ *compass* of the Δ-embedded graph G and we assume that G is accompanied by an embedding of its compass in Δ, that is the set $G \cap \Delta$. We say that G is a Δ-embedded graph if it is partially Δ-embedded graph and $G \subseteq \Delta$ (the whole G is embedded in Δ).

Levelings. Let W be a flat wall of a graph G. Following [5], we define the *leveling* of W in G, denoted by $\tilde{W}$, as the bipartite graph where one part is the ground set of W, the other part is the set of flaps of W, and, given a pair $(v, F) \in \mathsf{ground}(W) \times \mathsf{flaps}(W)$, the set $\{v, F\}$ is an edge of $\tilde{W}$ if and only if $v \in \partial F$. Again, keep in mind that $\tilde{W}$ may contain (many) vertices that are not in W. Notice that the incidence graph of the plane hypergraph $(N(\Gamma), \{c \cap N \mid c \in C(\Gamma)\})$ is isomorphic to $\tilde{W}$ via an isomorphism that extends π and, moreover, bijectively corresponds cells to flaps. This permits us to treat $\tilde{W}$ as a Δ-embedded graph where $\mathbf{bor}(\Delta) \cap \tilde{W}$ is the frontier of W. We call the vertices of $\mathsf{ground}(W)$ (resp. $\mathsf{flaps}(W)$) *ground-vertices* (resp. *flap-vertices*) of $\tilde{W}$. See Figure 3 for an example of a leveling.

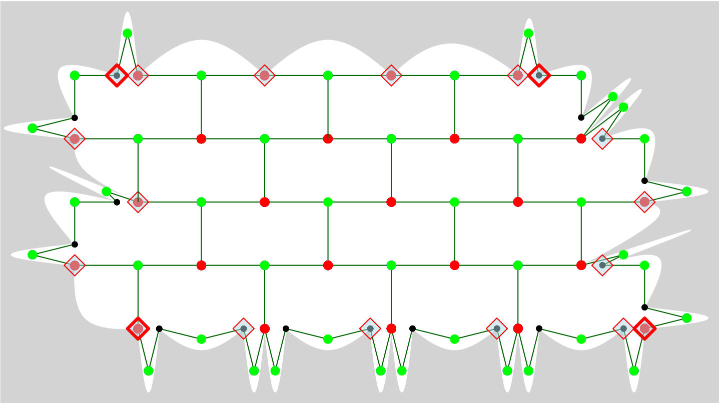


Figure 3 The leveling of the flat 5-wall W'' of Figure 1. The green vertices are the flap-vertices and the non-green vertices are the ground-vertices.

Recall that each edge e of $\mathsf{compass}(W)$ belongs to exactly one flap of W. If both of the endpoints of e are in the boundary of this flap, then this flap should be a trivial one and we say that e is a *short* edge of $\mathsf{compass}(W)$. We define the graph $W^\bullet$ as the graph obtained from W if we subdivide once every short edge in W. The next observation is a consequence of the following three facts: flap-vertices of $\tilde{W}$ have degree at most three, all the vertices of a wall have degree at most three, and every separation (A, B) of order at most three of a wall is trivial.

▶ **Observation 4.** *If W is a flat wall of a graph G, then the leveling $\tilde{W}$ of W in G contains a subgraph W^R that is isomorphic to some subdivision of $W^\bullet$ via an isomorphism that maps each ground vertex to itself.*

We call the graph W^R as in Observation 4 *representation* of the flat wall W in the Δ-embedded graph $\tilde{W}$, and therefore we can see it as a Δ-embedded subgraph of $\tilde{W}$. Notice that the above observation permits to bijectively map each cycle of W to a cycle of W^R that is also a cycle of $\tilde{W}$. That way, each cycle C of W corresponds to a cycle C of W^R denoted by C^R and we call C^R the *representation* of C in $\tilde{W}$. From now on, we reserve the superscript "^{R}"-notation to denote the correspondence between W (resp. C) and W^R (resp. C^R).

We define the function $\mathsf{flaps} : \mathcal{C}(W) \to 2^{\mathsf{flaps}(W)}$ so that, for each cycle C of W, $\mathsf{flaps}(C)$ contains each flap F of W that, when seen as a flap-vertex of the Δ-embedded graph $\tilde{W}$, belongs to the closed disk bounded by C^R. The following result is very similar to [33, Lemma 6.1]. The proof is strongly based on the notion of levelings.

▶ **Lemma 5.** *Let $a, r, r' \in \mathbb{N}$, where $r > r' \geq 3$. Also, let G be a graph, let (A, W) be an (a, r)-apex wall pair of G, and let W' be an internal r'-subwall of W. Then W' has a tilt W'' such that (A, W'') is an (a, r')-apex wall pair of G. Moreover,*

1. *the compass of W'' in $G \setminus A$ is a subgraph of the compass of W in $G \setminus A$ and*
2. *if P' is the perimeter of W', then the vertex set of the compass of W'' in $G \setminus A$ is a subset of $\bigcup \mathsf{flaps}(P')$.*

Morever, given G, (A, W), and W', the (a, r')-apex wall pair (A, W'') can be constructed in $\mathcal{O}(n)$ time.

From now on we refer to an (A, W'') as in Lemma 5 as *an* (a, r')-apex wall pair *generated by* the internal r'-subwall W' of W and we keep in mind that the compasses of all such flat walls W'' may differ *only* on their perimeter. The proof of Lemma 5 also implies the following.

▶ **Observation 6.** *Let W be a flat wall of a graph G, and W^R be the representation of W in the leveling $\tilde{W}$ of W in G. Then for every internal subwall $\bar{W}$ of W^R there exist an internal subwall W' of W and a tilt W'' of W' such that*

- *$\bar{W}$ is the representation of W' in the leveling $\tilde{W}$ of W,*
- *W'' is a flat wall, and*
- *the vertex set of the compass of W'' in G is a subset of $\bigcup \mathsf{flaps}(P')$, where P' is the perimeter of W'.*

Moreover, given G, W, and $\bar{W}$, the flat wall W'' can be constructed in $\mathcal{O}(n)$ steps.

Boundaried graphs. Let $t \in \mathbb{N}$. A *t-boundaried graph* is a triple $\mathbf{G} = (G, B, \rho)$ where G is a graph, $B \subseteq V(G)$, $|B| = t$, and $\rho : B \to [t]$ is a bijection. We say that $\mathbf{G}_1 = (G_1, B_1, \rho_1)$ and $\mathbf{G}_2 = (G_2, B_2, \rho_2)$ are *isomorphic* if there is an isomorphism from G_1 to G_2 that extends the bijection $\rho_2^{-1} \circ \rho_1$. The triple (G, B, ρ) is a *boundaried graph* if it is a t-boundaried graph for some $t \in \mathbb{N}$. As in [45], we define the *detail* of a boundaried graph $\mathbf{G} = (G, B, \rho)$ as $\mathsf{detail}(\mathbf{G}) := \max\{|E(G)|, |V(G) \setminus B|\}$. We denote by $\mathcal{B}^{(t)}$ the set of all (pairwise non-isomorphic) t-boundaried graphs. We also set $\mathcal{B} = \bigcup_{t \in \mathbb{N}} \mathcal{B}^{(t)}$.

Folios. We say that a t-boundaried graph $\mathbf{G}_1 = (G_1, B_1, \rho_1)$ is a *minor* of a t-boundaried graph $\mathbf{G}_2 = (G_2, B_2, \rho_2)$, denoted by $\mathbf{G}_1 \preceq_{\mathsf{m}} \mathbf{G}_2$, if there is a sequence of removals of non-boundary vertices, edge removals, and edge contractions in G_2, disallowing contractions of edges with both endpoints in B_2, that transforms $\mathbf{G}_2$ to a boundaried graph that is isomorphic to $\mathbf{G}_1$ (during edge contractions, boundary vertices prevail). Note that this extends the usual definition of minors in graphs without boundary.

We say that (M, T) is a *tm-pair* if M is a graph, $T \subseteq V(M)$, and all vertices in $V(M) \setminus T$ have degree two. We denote by $\mathsf{diss}(M, T)$ the graph obtained from M by dissolving all vertices in $V(M) \setminus T$. A *tm-pair* of a graph G is a *tm-pair* (M, T) where if M is a subgraph of G. We call the vertices in T *branch* vertices of (M, T).

If $\mathbf{M} = (M, B, \rho) \in \mathcal{B}$ and $T \subseteq V(M)$ with $B \subseteq T$, we call $(\mathbf{M}, T)$ a *btm-pair* and we define $\mathsf{diss}(\mathbf{M}, T) = (\mathsf{diss}(M, T), B, \rho)$. Note that we do not permit dissolution of boundary vertices, as we consider all of them to be branch vertices. If $\mathbf{G} = (G, B, \rho)$ is a boundaried graph and (M, T) is a tm-pair of G where $B \subseteq T$, then we say that $(\mathbf{M}, T)$, where $\mathbf{M} = (M, B, \rho)$, is a *$\mathsf{btm}$-pair* of $\mathbf{G} = (G, B, \rho)$. Let $\mathbf{G}_i = (G_i, B_i, \rho_i), i \in [2]$. We say that $\mathbf{G}_1$ is a *topological minor* of $\mathbf{G}_2$, denoted by $\mathbf{G}_1 \preceq_{\mathsf{tm}} \mathbf{G}_2$, if $\mathbf{G}_2$ has a btm-pair $(\mathbf{M}, T)$ such that $\mathsf{diss}(\mathbf{M}, T)$ is isomorphic to $\mathbf{G}_1$. We define the *ℓ-folio* of $\mathbf{G} = (G, B, \rho) \in \mathcal{B}$ as $\ell\text{-}\mathsf{folio}(\mathbf{G}) = \{\mathbf{G}' \in \mathcal{B} \mid \mathbf{G}' \preceq_{\mathsf{tm}} \mathbf{G}$ and $\mathbf{G}'$ has detail at most $\ell\}$.

Homogeneous walls. Let G be a graph and W be a flat wall of G. Let also (Γ, σ, π) be a rendition of the compass of W in G. Recall that $\Gamma = (U, N)$ is a Δ-painting for some closed disk Δ. Given a flap F, we denote by $\Omega(F)$ the counter-clockwise ordering of the vertices of ∂F as they appear in the corresponding cell of $C(\Gamma)$. Notice that as $|\partial F| \leq 3$, this cyclic ordering is significant only when $|\partial F| = 3$, in the sense that (x_1, x_2, x_3) remains invariant under shifting, i.e., $(v_1, v_2, v_3) \equiv (v_2, v_3, v_1)$ but not under inversion, i.e., $(v_1, v_2, v_3) \not\equiv (v_3, v_2, v_1)$.

Given a graph G, we say that the pair (A, W) is an *(r, a)-apex wall pair* of G if A is a subset of a vertices from G and W is a flat r-wall of $G \setminus A$. Let G be a graph and let (A, W) be an (a, r)-apex wall pair of G. For each cell $F \in \mathsf{flaps}(W)$ with $t_F = |\partial F|$, we fix $\rho_F : \partial F \to [a + 1, a + t_F]$ such that $(\rho_F^{-1}(a + 1), \ldots, \rho_F^{-1}(a + t_F)) \equiv \Omega(c)$. We also fix a bijection $\rho_A : A \to [a]$. For each flap $F \in \mathsf{flaps}(W)$, we define the boundaried graph $\mathbf{F}^A := (G[A \cup F], A \cup \partial F, \rho_A \cup \rho_F)$ and we denote by F^A the underlying graph of $\mathbf{F}^A$. We call $\mathbf{F}^A$ *augmented flap* of (A, W). Notice that $G[V(\mathsf{compass}(W)) \cup A] = \bigcup_{F \in \mathsf{flaps}(W)} F^A$.

Given some $\ell \in \mathbb{N}$, we say that two flaps $F_1, F_2 \in \mathsf{flaps}(W)$ are *(A,ℓ)-equivalent*, denoted by $F_1 \sim_{A,\ell} F_2$, if $\ell\text{-}\mathsf{folio}(\mathbf{F}_1^A) = \ell\text{-}\mathsf{folio}(\mathbf{F}_2^A)$. For each $F \in \mathsf{flaps}(W)$, we define the *(a,ℓ)-color* of F, denoted by $(a,\ell)\text{-}\mathsf{color}(F)$, as the equivalence class of $\sim_{A,\ell}$ to which $\mathbf{F}^A$ belongs.

Let $\tilde{W}$ be the leveling of W in $G \setminus A$ and let W^R be the representation of W in $\tilde{W}$. Recall that $\tilde{W}$ is a Δ-embedded graph. For each cycle C of W, we define the *(a,ℓ)-palette* of C, denoted by $(a,\ell)\text{-}\mathsf{palette}(C)$, as the set of all the (a,ℓ)-colors of the flaps that appear as vertices of $\tilde{W}$ in the closed disk bounded by C^R in Δ (recall that by C^R we denote the representation of C in $\tilde{W}$).

Let $a, \ell, r, q \in \mathbb{N}$, where $r > q \geq 3$ and let (A, W) be an (a,r)-apex wall pair of a graph G. We say that (A, W) is an *(ℓ, q)-homogeneous* (a,r)-apex wall pair of G if every internal brick B of W that is not a brick of $W^{(q)}$ has the same (a,ℓ)-palette (seen as a cycle of W). If we drop the demand that "B is not a brick of $W^{(q)}$" then we simply say that (A, W) is an *ℓ-homogeneous* (a,r)-apex wall pair of G.

The following observation is a consequence of the fact that, given a wall W and an internal subwall W' of W, every internal brick of a tilt W'' of W' is also an internal brick of W.

▶ **Observation 7.** *Let $a, r, r' \in \mathbb{N}$, where $r > r' \geq 3$. Also, let G be a graph, let (A, W) be an (a,r)-apex wall pair of G, and let W' be an internal r'-subwall of W. If (A, W) is (ℓ, q)-homogeneous for some $\ell, q \in \mathbb{N}$ where $r > q \geq 3$, then every (a, r')-apex wall pair (A, W'') generated by W' is (ℓ, q)-homogeneous.*

The following result is from [5, Lemma 4.3] and implies that if the wall of an apex wall pair is polynomially large on r, then its compass contains a homogeneous flat r-wall.

▶ **Proposition 8.** *There is a function $f_1 : \mathbb{N}^3 \to \mathbb{N}$ such that if $\ell, r, a \in \mathbb{N}$, where $r \geq 3$, G is a graph, and (A, W) is an $(a, f_1(\ell, r, a))$-apex wall pair of G, then W has a r-subwall W' such that every (a,r)-apex wall pair of G that is generated by W' is ℓ-homogeneous. Moreover, it holds that $f_1(\ell, r, a) = \mathcal{O}(r^{c_{a,\ell}})$, for some constant $c_{a,\ell}$ depending on a and ℓ.*

We refer to the constant $c_{a,\ell}$ of Proposition 8, when $a = a_{\mathcal{F}}$ and $\ell = \ell_{\mathcal{G}}$ as the *palette-variety* of $\mathcal{F}$. This constant reflects the *price of homogeneity*: the degree of the polynomial overhead that we need to pay in order to force homogeneity in a flat wall.

Treewidth. A *tree decomposition* of a graph G is a pair (T, χ) where T is a tree and $\chi : V(T) \to 2^{V(G)}$ such that (1) $\bigcup_{t \in V(T)} \chi(t) = V(G)$, (2) for every edge e of G there is a $t \in V(T)$ such that $\chi(t)$ contains both endpoints of e, and (3) for every $v \in V(G)$, the subgraph of T induced by $\{t \in V(T) \mid v \in \chi(t)\}$ is connected. The *width* of (T, χ) is defined as $\mathbf{w}(T, \chi) := \max\big\{\, |\chi(t)| - 1 \;\big|\; t \in V(T)\big\}$. The *treewidth of G* is defined as $\mathsf{tw}(G) := \min\big\{\, \mathbf{w}(T, \chi) \;\big|\; (T, \chi) \text{ is a tree decomposition of } G\big\}$.

The following is the main result of [8]. We will use it to compute a tree decomposition of a graph of bounded treewidth.

▶ **Proposition 9.** *There is an algorithm that, given an graph G on n vertices and an integer k, it outputs either a report that $\mathsf{tw}(G) > k$, or a tree decomposition of G of width at most $5k + 4$. Moreover, this algorithm runs in $2^{\mathcal{O}(k)} \cdot n$ steps.*

The following result is derived from [1]. We will use it in order to find a wall in a bounded treewidth graph, given a tree decomposition of it.

▶ **Proposition 10.** *There is an algorithm that, given a graph G on m edges, a graph H on h edges without isolated vertices, and a tree decomposition of G of width at most k, it outputs, if it exists, a minor of G isomorphic to H. Moreover, this algorithm runs in $2^{\mathcal{O}(k \log k)} \cdot h^{\mathcal{O}(k)} \cdot 2^{\mathcal{O}(h)} \cdot m$ steps.*

3 Auxiliary algorithmic and combinatorial results

3.1 Two algorithmic results

We are now in position to state some results that will support our algorithms.

▶ **Lemma 11.** *There exist a function $f_2 : \mathbb{N} \to \mathbb{N}$ and an algorithm as follows:*
`Find-Wall`(G, r, k)
Input: *A graph G, an odd $r \in \mathbb{N}_{\geq 3}$, and a $k \in \mathbb{N}$.*
Output: *One of the following:*

- *Either a report that G has treewidth at most $f_2(s_{\mathcal{F}}) \cdot r + k$, or*
- *an r-wall W of G, or*
- *a report that (G, k) is a* no*-instance of $\mathcal{F}$-M-Deletion.*

Moreover, this algorithm runs in $2^{\mathcal{O}_{s_{\mathcal{F}}}(r^2+(k+r)\log(k+r))} \cdot n$ steps.

The proof of Lemma 11 combines the algorithm of Perkovič and Reed in [43] for computing the treewidth of a graph, as well as the excellent analysis of the algorithm provided in [3]. We also use the upper bound for the treewidth of an K_h-minor free graph without an r-wall by [31], the dynamic programming algorithm of [1] for finding a wall in a graph of bounded treewidth, and the single-exponential FPT-approximation algorithm for treewidth in [8].

The next result follows from [33, Theorem 1.9] and the proof of [32, Theorem 5.2].

▶ **Proposition 12.** *There are functions $f_3 : \mathbb{N} \to \mathbb{N}$, $f_4 : \mathbb{N} \to \mathbb{N}$ and an algorithm as follows:*
`Clique-Or-Flat-Wall`(G, r, t, W)
Input: *A graph G on n vertices and m edges, an odd integer $r \geq 3$, a $t \in \mathbb{N}_{\geq 1}$, and an R-wall W in G, where $R = f_3(t) \cdot r$.*
Output: *Either a minor of G isomorphic to K_t, or*

(1) *a set $A \subseteq V(G)$ of size at most $12288t^{24}$,*
(2) *a flat r-wall $\bar{W}$ of $G \setminus A$ such that $V(\bar{W}) \cap A = \emptyset$, and*
(3) *a separation (X, Y) of $G \setminus A$ that certifies that $\bar{W}$ is a flat wall and an Ω-rendition of $G[Y]$ with flaps of treewidth at most $f_4(t) \cdot r$, where Ω is a cyclic ordering of $X \cap Y$ determined by the order on the perimeter of $\bar{W}$.*

Moreover, this algorithm runs in $2^{\mathcal{O}_t(r)}(m + n)$ time.

Proposition 12, without the bound on the treewidth of the flaps, has been proven in [33, Theorem 1.9]. However, in [33, Theorem 1.9] $\bar{W}$ is a tilt of some r-subwall of W with a different function $f_3{}'$ for the relation between R and r and has running time $\mathcal{O}(t^{24}(n+m))$.

In Proposition 12 the bound on the treewidth of the flaps is obtained if we plug [33, Theorem 1.9] in the proof of [32, Theorem 5.2], taking into account the linear dependence between R and r. The parameterized dependence $2^{\mathcal{O}_t(r)}$ of the algorithm follows because of the use of the linear FPT-approximation algorithm for treewidth in [8] so as to compute the tree decompositions of the flaps. Another stronger version of Proposition 12, where no R-wall W is given in the input, appears in [23, Lemma 3.2], running in $2^{\mathcal{O}_t(r^{58})} \cdot n \log^2 n$ time. We do not need this stronger version as, for our problem, the R-wall W will be found by the algorithm of Lemma 11.

3.2 Finding an irrelevant vertex

The *irrelevant vertex technique* was introduced in [45] for providing an FPT-algorithm for the Disjoint Paths problem. Moreover, this technique has appeared to by quite versatile and is now a standard tool of parameterized algorithm design (see e.g., [14, 49]). The applicability of this technique for $\mathcal{F}$-M-Deletion is materialized by the algorithm of Lemma 14.

Given a graph G, a set $A \subseteq V(G), |A| = a$, and an r-wall W of G, we say that (A, W) is an (a, r)-*apex-wall* pair if W is a flat r-wall of $G \setminus A$.

▶ **Lemma 13.** *There is a function $f_5 : \mathbb{N}^4 \to \mathbb{N}$ such that if $a, \ell, q, k \in \mathbb{N}$, $q \geq 3$, G is a graph, and (A, W) is an ℓ-homogeneous $(a, f_5(a, \ell, q, k))$-apex wall pair of G, then for every (a, q)-apex wall pair $(A, \hat{W})$ generated by $W^{(q)}$, it holds that (G, k) and $(G \setminus V(\mathsf{compass}(\hat{W})), k)$ are equivalent instances of $\mathcal{F}$-M-Deletion. Moreover, $f_6(a, \ell, q, k) = O_{a,\ell,q}(k)$.*

Sketch of the proof. The proof considers $k + 1$ subwalls of W which, in turn, generate $k + 1$ flat walls (by taking their tilts) whose compasses are all disjoint except at some territory T of constant size containing the center of W that is common for all these walls. If now S is a solution to $\mathcal{F}$-M-Deletion for the instance (G, k) then one, say W_i, of these $k + 1$ flat walls will have a compass that does not intersect S (except from some constant size territory $T_i$around the center of W_i that contains T). We then claim that $S' = S \setminus T_i$ is a new solution of $\mathcal{F}$-M-Deletion that avoids the center of W. To prove that this is the case, we assume to the contrary that some graph L in $\mathcal{F}$ appears as a minor in $G' = G \setminus S'$. But this means that part of the realization of H in G' meets some vertex in the center of W and therefore it traverses $\mathsf{compass}(W_i)$. We arrive to a contradiction by providing an alternative realization of L that is routed away from T_i. For this rerouting we use the main combinatorial result of [5] that guarantees that there is a function $f_6 : \mathbb{N}^3 \to \mathbb{N}$ such that, for every $a, \ell, q \in \mathbb{N}$ and every graph G, if (A, W) is an (ℓ, q)-homogeneous $(a, f_6(a, \ell, q))$-apex wall pair of G, there is an (a, q)-apex wall pair of G generated by $W^{(q)}$ whose compass is irrelevant. For this it is enough to pick $f_5(a, \ell, q, k) = \mathcal{O}(k \cdot f_6(a, \ell, q) + q)$. ◀

▶ **Lemma 14.** *There exist a function $f_7 : \mathbb{N}^4 \to \mathbb{N}$ and an algorithm with the following specifications:*

`Find-Irrelevant-Wall`(G, q, k, b, A, W)

***Input:** A graph G on n vertices, two integers $k, q \in \mathbb{N}$, a $b \in \mathbb{N}_{\geq 3}$, and an $(a, f_7(a, \ell_{\mathcal{F}}, b, k))$-apex wall pair (A, W) of G whose all flaps have treewidth at most q.*

***Output:** A flat b-wall $\hat{W}$ of $G \setminus A$ such that (G, k) and $(G \setminus V(\mathsf{compass}(\hat{W})), k)$ are equivalent instances of $\mathcal{F}$-M-Deletion.*

Moreover, $f_7(a, \ell_{\mathcal{F}}, b, k) = \mathcal{O}_{a,\ell_{\mathcal{F}}}((k + b)^{c_{a,\ell_{\mathcal{F}}}})$ for some constant $c_{a,\ell_{\mathcal{F}}}$ depending on a and $\ell_{\mathcal{F}}$. This algorithm runs in $2^{\mathcal{O}_{a,\ell_{\mathcal{F}}}(q \log q + (k+b) \log(k+b))} \cdot n$ time.

Proof. We set $f_7(a, \ell_{\mathcal{F}}, b, k) := f_1(\ell_{\mathcal{F}}, r, a)$, where $r = f_5(a, \ell, b, k)$. The algorithm considers each one of the $\binom{f_7(a,\ell_{\mathcal{F}},b,k)}{r}$ internal r-subwalls W' of W and constructs an (a, r)-wall pair (A, W'') generated by W'. This can be done in $\mathcal{O}(n)$ time because of Lemma 5.

From Proposition 8 there is a choice of W' such that (A, W'') is $\ell_{\mathcal{F}}$-homogeneous. To check whether (A, W'') is $\ell_{\mathcal{F}}$-homogeneous we do the following. Let $\mathcal{B}$ be the set of all flaps of W'' that, when seen as flap vertices of $\tilde{W}$, appear in the closed disk bounded by the representation of B in $\tilde{W}$, where B is an internal brick of W'' that is not a brick of $W''^{(b)}$. For every flap $F \in \mathcal{B}$, we consider the boundaried graph $\mathbf{F}^A$. Using the fact that $\mathsf{tw}(F^A) \leq q + a$, we apply the algorithm of Proposition 9 which outputs a tree decomposition of F^A of width at most $5(q + a) + 4$. Then by applying the algorithm of Proposition 10, we compute $\ell_{\mathcal{F}}$-$\mathsf{folio}(\mathbf{F}^A)$ in $2^{\mathcal{O}_{a,\ell_{\mathcal{F}}}(q \log q)}$ time. Then, it is easy to check in linear time whether (A, W'') is $\ell_{\mathcal{F}}$-homogeneous.

After we find W', we again use Lemma 5 in order to construct a flat b-wall $\hat{W}$ of $G \setminus A$ generated by $W'^{(b)}$. The algorithm outputs $\hat{W}$, and this is correct because of Lemma 13. ◀

3.3 Combinatorial results for branching

We now give a combinatorial result that will justify a branching step of our algorithm, i.e., its recursive application on a set of $\mathcal{O}_s(k)$ vertices. Given a graph G and a set $A \subseteq V(G)$, we say that a graph H is an *A-fixed minor of G* if H can be obtained from a subgraph G' of G where $A \subseteq V(G')$, after contracting edges without endpoints in A (see Figure 4 for the definition of an r-grid and its central subgrids that we will need later). A graph H is an *A-apex r-grid* if it can be obtained by an r-grid Γ after adding a set A of new vertices and some edges between the vertices of A and $V(\Gamma)$. We call Γ *underlying grid* of H.

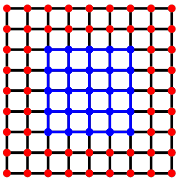


Figure 4 A 9-grid and its central 5-subgrid.

Next we identify a combinatorial structure that guarantees the existence of a set of $q = \mathcal{O}_s(k)$ vertices that intersects every solution S of $\mathcal{F}$-M-Deletion on imput (G, k). This will permit branching on q simpler instances of the form $(G', k-1)$.

▶ **Lemma 15.** *There exist three functions $f_8, f_9, f_{10} : \mathbb{N}^2 \to \mathbb{N}$, such that if $\mathcal{F}$ is a finite set of graphs, G is a graph, $k \in \mathbb{N}$, and $A \subseteq V(G)$, $|A| = a_{\mathcal{F}}$, such that G contains as an A-fixed minor an A-apex $f_8(s_{\mathcal{F}}, k)$-grid H where each vertex $v \in A$ has at least $f_9(s_{\mathcal{F}}, k)$ neighbors in the central $(f_8(s_{\mathcal{F}}, k) - f_{10}(s_{\mathcal{F}}, k))$-grid of $H \setminus A$, then for every solution S of $\mathcal{F}$-M-Deletion for the instance (G, k), it holds that $S \cap A \neq \emptyset$. Moreover, $f_8(s_{\mathcal{F}}, k) = \mathcal{O}_{s_{\mathcal{F}}}(k^{3/2})$, $f_9(s_{\mathcal{F}}, k) = \mathcal{O}_{s_{\mathcal{F}}}(k^3)$, and $f_{10}(s_{\mathcal{F}}, k) = \mathcal{O}_{s_{\mathcal{F}}}(k)$.*

We conjecture that Lemma 15 is tight, in the sense that it cannot be proved for some $f_9(s_{\mathcal{F}}, k) = \mathcal{O}_{s_{\mathcal{F}}}(k^{3-\epsilon})$.

Notice that in the special case where $a_{\mathcal{F}} = 1$, then Lemma 15 can be improved by using the main combinatorial result of [16]. In particular [16, Lemma 3.1] easily implies that, in this case, $f_8(s_{\mathcal{F}}, k) = \mathcal{O}_{s_{\mathcal{F}}}(k)$, $f_9(s_{\mathcal{F}}, k) = \mathcal{O}_{s_{\mathcal{F}}}(k^2)$, and $f_{10}(s_{\mathcal{F}}, k) = \mathcal{O}_{s_{\mathcal{F}}}(\sqrt{k})$. In [47], we prove that these bounds can be improved to $f_8(s_{\mathcal{F}}, k) = \mathcal{O}_{s_{\mathcal{F}}}(\sqrt{k})$, $f_9(s_{\mathcal{F}}, k) = \mathcal{O}_{s_{\mathcal{F}}}(k)$, and $f_{10}(s_{\mathcal{F}}, k) = \mathcal{O}_{s_{\mathcal{F}}}(1)$. We will use these improved bounds for the proof of Theorem 2.

We conclude this subsection with one additional definition that will be useful for the application of Lemma 15 in our main algorithm.

Canonical partitions. We define the *canonical partition* of an r-wall W to be a collection $\mathcal{Q} = \{Q_{\text{ext}}, Q_1, \ldots, Q_q\}$ of $(r-2)^2 + 1$ connected subgraphs of W such that their vertex sets form a partition of $V(W)$ as indicated in Figure 5.

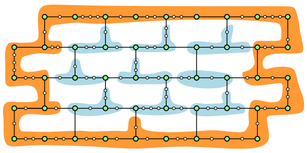


Figure 5 A 5-wall and its canonical partition $\mathcal{Q}$. The orange bag is the external bag Q_{ext}.

Let (A, W) be an (a, r)-apex wall pair of a graph G and let $\tilde{W}$ be the Δ-embedded graph that is the leveling $\tilde{W}$ of W in $G \setminus A$. Let also W^R be the representation of W in $\tilde{W}$. Consider a canonical partition $\mathcal{Q}$ of W^R. We enhance the graphs of $\mathcal{Q}$ so to include in them all the vertices of $\tilde{W}$ by applying the following procedure. We set $\tilde{\mathcal{Q}} := \mathcal{Q}$ and, as long as there is a vertex $x \in V(\tilde{W}) \setminus V(\bigcup \tilde{\mathcal{Q}})$[5] that is adjacent to a vertex of a graph $Q \in \tilde{\mathcal{Q}}$, update $\tilde{\mathcal{Q}} := \tilde{\mathcal{Q}} \setminus \{Q\} \cup \{\tilde{Q}\}$, where $\tilde{Q} = \tilde{W}[\{x\} \cup V(Q)]$. We just defined a partition of the vertices of $\tilde{W}$ into subsets inducing connected graphs. We call such a partition *canonical partition* of $\tilde{W}$. Notice that a canonical partition of $\tilde{W}$ is not unique (since the sets in $\mathcal{Q}$ can be "expanded" arbitrarily when introducing the vertex x).

4 The algorithms

4.1 The general algorithm

▶ **Lemma 16.** *Let $\mathcal{F}$ be a finite collection of graphs. There is an algorithm solving $\mathcal{F}$-*M-Deletion-Compression *in $2^{\mathcal{O}_s(k^{2(c_{\mathcal{F}}+2)}\log k)} \cdot n^2$ time.*

Proof. For simplicity, in this proof, we use c instead of $c_{\mathcal{F}}$, s instead of $s_{\mathcal{F}}$, ℓ instead of $\ell_{\mathcal{F}}$, a instead of $a_{\mathcal{F}}$, and remember that $\ell = \mathcal{O}_s(1)$ and $a = \mathcal{O}_s(1)$. We set

$$\begin{array}{lll} b = f_7(k, a, \ell) = \mathcal{O}(k^c), & x = f_9(s, k), & l = (12288 s^{24} + k + 1) \cdot x, \\ m = f_8(s, k), & p = f_{10}(s, k), & h = \max\{m - p, \lceil \sqrt{l+1} \cdot b \rceil\}, \\ r = h + p + 2, & R = f_3(s) \cdot r, & \text{and notice that } R = \mathcal{O}_s(k^{c+2}). \end{array}$$

Recall that, in the definition of R, the constant c is the palette-variety of $\mathcal{F}$. We present the algorithm `Solve-Compression`, whose input is a quadruple (G, k', k, S) where G is a graph, k' and k are non-negative integers where $k' \leq k$, and S is a subset of $V(G)$ such that $|S| = k$ and $\mathcal{F} \not\leq_{\mathsf{m}} G \setminus S$. The algorithm returns, if it exists, a solution for $\mathcal{F}$-M-Deletion on (G, k'). Certainly, we may assume that $k' < k$, otherwise S is already a solution and we are done. The steps of the algorithm are the following:

Step 1. Run the algorithm `Find-Wall` of Lemma 11 with input $(G \setminus S, R, 0)$. This outputs, in $2^{\mathcal{O}_s(k^{2(c+2)})} \cdot n$ time, either a report that $\mathsf{tw}(G \setminus S) \leq f_2(s) \cdot R$, or an R-wall W_0 of $G \setminus S$. Notice that `Find-Wall`$(G \setminus S, R, s)$ never outputs the third case, since $(G \setminus S, 0)$ is a yes-instance of $\mathcal{F}$-M-Deletion. In the first possible output, we know that $\mathsf{tw}(G) \leq f_2(s) \cdot R + k = \mathcal{O}_s(k^{c+2})$, and we call the algorithm of Proposition 3 with input $(G, f_2(s) \cdot R + k, k')$ and return a correct answer in $2^{\mathcal{O}_s(k^{c+2}\log k)} \cdot n$ steps. In the second possible output, the algorithm moves to the second step.

Step 2. Call `Clique-Or-Flat-Wall` of Proposition 12 on $(G \setminus S, r, s, W_0)$. Since $\mathcal{F} \not\leq_{\mathsf{m}} G \setminus S$ and $\mathcal{F} \leq_{\mathsf{m}} K_s$, the algorithm outputs, in time $2^{\mathcal{O}_s(r)} \cdot (m + n) = 2^{\mathcal{O}_s(k^{c+2})} \cdot n$, the following:

- A set $A \subseteq V(G) \setminus S$ of size at most $12288 s^{24}$,
- a flat r-wall W of $G \setminus (S \cup A)$ such that $V(W) \cap A = \emptyset$, and
- a separation (X, Y) of $G \setminus A$ that certifies that W is a flat wall, and an Ω-rendition of $G[Y]$ with flaps of treewidth at most $q = f_4(s) \cdot r$, where Ω is a cyclic ordering of $X \cap Y$ determined by the order on the outer cycle of W.

Let $\tilde{W}$ be the leveling of W and let W^R be the representation of W in the Δ-embedded graph $\tilde{W}$ ($\tilde{W}$ and W^R can straighforwardly be constructed in $\mathcal{O}(n)$ steps using the Ω-rendition of $G[Y]$). Let also $\bar{W} = (W^R)^{(r-p)}$, i.e., $\bar{W}$ is the central $(r-p)$-subwall of W^R.

[5] If $\mathcal{S}$ is a collection of objects where the operation $\cup$ is defined, then we denote $\bigcup \mathcal{S} = \bigcup_{X \in \mathcal{S}} X$.

Consider a family $\bar{\mathcal{W}} = \{\bar{W}_1, \ldots, \bar{W}_{l+1}\}$ of $l+1$ internal b-subwalls of $\bar{W}$, such that if D_i is the closed disk in Δ bounded by the perimeter, denoted by $\bar{P}_i$, of $\bar{W}_i$, then $D_i \cap D_j = \emptyset$, for $i \neq j$. We are allowed to do this since $r - p - 2 \geq \lceil \sqrt{l+1} \cdot b \rceil$. By $\mathsf{flaps}(D_i)$ we denote all the flaps corresponding to flap-vertices of $\tilde{W}$ that are inside D_i in the embedding of $\tilde{W}$ in Δ. For every $i \in [l+1]$, we compute, in $\mathcal{O}(n)$ time, the set $A_i = \big\{v \in S \cup A \mid v \text{ is adjacent in } G \text{ to a vertex of } \bigcup \mathsf{flaps}(D_i)\big\}$ and we proceed to the last step.

Step 3. The algorithm examines two cases:

Case A: For every $i \in [l+1]$, it holds that $|A_i| > a$. In this case the algorithm recursively calls `Solve-Compression` with input $(G \setminus x, k' - 1, |S \setminus x|, S \setminus x)$, for every $x \in S \cup A$, and if one of these new instances is a yes-instance, certified by a set $\bar{S}$, then return $\bar{S} \cup \{x\}$, otherwise return that (G, k') is a no-instance.

We now prove that the above branching step of the algorithm is correct. Let $\tilde{W}$ be the leveling of W in $G \setminus (S \cup A)$. We define $\tilde{G}$ as the graph obtained from $G \setminus (S \cup A)$ if we remove all the vertices of the compass of W and take the union of the resulting graph with $\tilde{W}$. Notice that $\tilde{G}$ is *partially* Δ*-embedded* in the sense that the part of G that is embedded in Δ is $\tilde{W}$. Notice that $\tilde{G}$ is not necessarily a contraction of $G \setminus (S \cup A)$, and this is because the trivial flaps appear in $\tilde{W}$ as induced paths of length two instead of edges. Therefore, if $\breve{G}$ is the graph obtained from $\tilde{G}$ after dissolving each flap-vertex corresponding to a trivial flap, then

- $\breve{G}$ is a contraction of $G \setminus (S \cup A)$,
- $\breve{G}$ is a partially Δ-embedded graph whose compass is a dissolution of $\tilde{W}$, and
- $\breve{G}$ contains an r-wall $\breve{W}$ that is a dissolution of W^R and $\breve{W}$ is embedded in Δ so that its perimeter is a dissolution of the perimeter of W^R.

Consider a canonical partition $\tilde{\mathcal{Q}}$ of $\tilde{W}$. Let $\breve{\mathcal{Q}}$ be the collection of connected subgraphs of $\breve{G}$ that are obtained if we apply to the graphs in $\tilde{\mathcal{Q}}$ the same dissolutions that we used to transform $\tilde{G}$ to $\breve{G}$ (we just take care that the edge contracted during each dissolution has both endpoints in some bag). Moreover, we enhance $\breve{\mathcal{Q}}$ by adding in its external bag all the vertices of $\breve{G}$ that are not points of Δ. Notice that the vertex sets of the graphs in this new $\breve{\mathcal{Q}}$ define a partition of $V(\breve{G})$. Let now $\breve{G}^+$ be the graph obtained if we apply in G the same contractions that transform $G \setminus (S \cup A)$ to its contraction $\breve{G}$. Let $a^* = |S \cup A| \leq 12288 s^{24} + k + 1$. We now construct a minor of $\breve{G}^+$ by contracting all edges of each member of $\breve{\mathcal{Q}}$ to a single vertex and removing the vertex to which the external bag was contracted. We denote the resulting graph by $\bar{G}$ and we observe that $\bar{G}$ contains as a spanning subgraph an $S \cup A$-apex $(r-2)$-grid Γ. Recall that Γ is a $S \cup A$-fixed minor of G. Let $\bar{\Gamma}$ be the underlying grid of Γ and let $\bar{\Gamma}_1, \ldots, \bar{\Gamma}_{l+1}$ be the "packing" of the h-central grid $\bar{\Gamma}'$ of $\bar{\Gamma}$, corresponding to the walls in $\bar{\mathcal{W}}$, where each $\bar{\Gamma}_i$ is a b-grid. We can assume the existence of this packing because $h \geq \lceil \sqrt{l+1} \cdot b \rceil$. The initial assumption that $|A_i| > a$, for $i \in [l+1]$, implies that $\forall i \in [l+1]$, there are more than a apices of Γ that are adjacent to vertices of $\bar{\Gamma}_i$.

For every $v \in S \cup A$, let N_v be the set of neighbors of v in $\bar{\Gamma}'$ and let $\bar{N} = \bigcup_{v \in S \cup A} N_v$. Let A^* be the set of vertices of $S \cup A$ with $|N_v| \geq x$. We claim that $|A^*| \geq a$. Suppose to the contrary that $|A^*| < a$. This implies that the vertices in $(S \cup A) \setminus A^*$ are adjacent to at most $x \cdot |(S \cup A) \setminus A^*| \leq l$ vertices in $\bar{N}$. This, in turn implies that there is an $i \in [l+1]$ such that there are no vertices in $(S \cup A) \setminus A^*$ adjacent to vertices of $\bar{\Gamma}_i$. Thus, for this i, there are at most a apex vertices of $\bar{G}$ that are adjacent to vertices of $\bar{\Gamma}_i$, a contradiction to the conclusion of the previous paragraph. We arbitrarily remove vertices from A^* so that $|A^*| = a$.

Consider now the A^*-apex $(r-2)$-grid $H = \Gamma \setminus ((S \cup A) \setminus A^*)$, and as each vertex in A^* has at least x neighbors in $V(\bar{\Gamma}')$ and $r - 2 \geq m$, Lemma 15 can be applied for k', A^*, H. This implies that (G, k') is a yes-instance of $\mathcal{F}$-M-Deletion if and only if there is some

$v \in A^*$ such that $(G \setminus v, k' - 1)$ is a yes-instance of $\mathcal{F}$-M-Deletion. This completes the correctness of the branching step in Case A.

Case B: there is an $i \in [l+1]$ such that $|A_i| \leq a$. Since $\bar{W}_i$ is an internal b-subwall of W^R, there is a subgraph of $\mathsf{compass}(W)$ that is a flat b-wall W_i'' of $G \setminus (S \cup A)$ such that the set of vertices of the compass of W_i'' is a subset of $\bigcup \mathsf{flaps}(D_i)$ (as we argued in Subsection 3.2, W_i'' is a tilt of the subwall of W represented by $\bar{W}_i$ and can be found in $\mathcal{O}(n)$ time). This implies that if A_i'' is the set of vertices from $S \cup A$ that are adjacent with vertices of the compass of W_i'' in $G \setminus (S \cup A)$, then $A_i'' \subseteq A_i$. Thus (A'', W_i'') is an $(|A_i''|, b)$-apex wall pair in G.

We now apply `Find-Irrelevant-Wall` of Lemma 14 for $(G, k, q, 3, A_i'', W_i'')$ and pick a vertex v in the center of the obtained 3-wall. According to Lemma 14, it holds that (G, k) and $(G \setminus v, k)$ are equivalent instances of $\mathcal{F}$-M-Deletion, v can be detected in $2^{\mathcal{O}_{a,\ell}(k \log k)} \cdot n$ time, and the algorithm correctly calls recursively `Solve-Compression` with input $(G \setminus v, k', k, S)$.

Recall that $|S \cup A| \leq k + 1 + 12288 s^{24} = \mathcal{O}_s(k)$. Therefore, if $T(n, k', k)$ is the running time of the above algorithm, then $T(n, k', k) \leq 2^{\mathcal{O}_s(k^{2(c+2)})} n + \max\{T(n-1, k', k), \mathcal{O}_s(k) \cdot T(n, k-1, k)\}$ which, given that $k' \leq k$, implies that $T(n, k', k) = 2^{\mathcal{O}_s(k^{2(c+2)})} n^2$.

Notice now that the output of `Solve-Compression` on $(G, k, k+1, S)$ gives a solution for $\mathcal{F}$-M-Deletion-Compression on this instance. ◀

4.2 The apex-minor free case

In this subsection we prove that, in the case where $a_{\mathcal{F}} = 1$, there is an algorithm that solves $\mathcal{F}$-M-Deletion in time $2^{\mathcal{O}_{s_{\mathcal{F}}}(k^{2(c+1)})} \cdot n^2$, where $c = c_{a, \ell_{\mathcal{F}}}$ and $a = 12288 (s_{\mathcal{F}})^{24}$. The existence of such an algorithm implies Theorem 2.

Let G be graph and let W be an r-wall in G. The *drop*, denoted by $D_{W'}$, of a q-subwall W' of W, where $q \leq r$, is defined as follows: Contract in G the perimeter of W to a single vertex v. $D_{W'}$ is the unique biconnected component of the resulting graph that contains the interior of W'. We call the vertex v the *pole* of the drop $D_{W'}$.

The algorithm. Our algorithm avoids iterative compression in the fashion that this is done by Marx and Schlotter in [41] for the Planarization problem. The algorithm has three main steps. We first set $s = s_{\mathcal{F}}$, $a = 12288 s^{24}$, $b' = f_7(a, k, \ell_{\mathcal{F}}) = \mathcal{O}_s(k^{c_{a,\ell_{\mathcal{F}}}})$, and we define

$$b = f_3(s) \cdot 2b' + 2 \qquad l = f_9(s,k) \cdot k \qquad p = f_{10}(s,k)$$
$$h = \max\{f_8(s,k) - p, b \cdot \sqrt{l+1}\} \qquad r = h + p + 2 \qquad R = f_3{}'(s) \cdot r + k = \mathcal{O}_s(k^{c+1}).$$

Step 1. Run the algorithm `Find-Wall` of Lemma 11 with input (G, R, k) and, in $2^{\mathcal{O}_s(k^{2(c+1)})} \cdot n$ time, either report a no-answer, or conclude that $\mathsf{tw}(G) \leq f_2(s) \cdot R + k$ and solve $\mathcal{F}$-M-Deletion in $\mathcal{O}(2^{\mathcal{O}_s(k^{c+1}) \log k} \cdot n)$ time using the algorithm of Proposition 3, or obtain an R-wall $W^\bullet$ of G. In the third case, consider all the $\binom{R}{b} = 2^{\mathcal{O}_s(k^c \log k)}$ b-subwalls of W and for each one of them, say W', construct its drop $D_{W'}$, consider in $D_{W'}$ the central $(b-2)$-subwall $\bar{W}$ of W', and run the algorithm `Clique-Or-Flat-Wall` of Proposition 12 with input $D_{W'}$, $2b'$, s, and $\bar{W}$. This takes time $2^{\mathcal{O}_s(k^c)} \cdot n$. If for some of these drops the result is an $(|A'|, 2b')$-apex wall pair (W'', A') where $|A'| \leq a$ and its flaps have treewidth at most $q = f_4(2b') \cdot r$, then apply Step 2, otherwise apply Step 3.

Step 2. Consider the leveling $\tilde{W}$ of W'' and, in the representation W^R of W'' in $\tilde{W}$, pick a b'-wall $\bar{W}$ whose flap vertices do not correspond to a flap containing the pole of $D_{W'}$. Then use $\bar{W}$ in order to find an $(|A'|, b')$-apex wall pair (W''', A') of $D_{W'}$ whose compass does not contain the pole of $D_{W'}$. Notice that (A', W''') is also an $(|A'|, b')$-apex wall pair of G, therefore the algorithm can apply `Find-Irrelevant-Vertex` of Lemma 14 for $(G, k, q, 3, A', W''')$ and obtain, in $2^{\mathcal{O}_s(k \log k)} \cdot n$ time, an "irrelevant" flat 3-wall and a vertex v in its center such that (G, k) and $(G \setminus v, k)$ are equivalent instances of $\mathcal{F}$-M-Deletion. Then the algorithm runs recursively on the equivalent instance $(G \setminus v, k)$.

Step 3. Consider all the $\binom{R}{r} = 2^{\mathcal{O}_s(k^c \log k)}$ r-subwalls of $W^\bullet$, and for each one W' of them, consider its central h-subwall $\bar{W}$ and compute the canonical partition $\mathcal{Q}$ of $\bar{W}$. Then for each internal bag Q of $\mathcal{Q}$ add a new vertex v_Q and make it adjacent with all vertices in Q, then add a new vertex x_{all} and make it adjacent with all x_Q's, and in the resulting graph, for every vertex y of G that is not in the union of the internal bags of $\mathcal{Q}$, check, in time $\mathcal{O}(k \cdot m) = \mathcal{O}_s(k \cdot n)$ (using standard flow techniques), if there there are $f_9(s, k)$ internally vertex-disjoint paths from x_{all} to y. If this is indeed the case for some y, then y should belong to every solution of $\mathcal{F}$-M-Deletion for the instance (G, k) and the algorithm runs recursively on the equivalent instance $(G \setminus y, k - 1)$. If no such y exists, then report that (G, k) is a no-instance of $\mathcal{F}$-M-Deletion.

Notice that the third step of the algorithm, when applied takes time $2^{\mathcal{O}_s(k^c \log k)} \cdot n^2$. However, it cannot be applied more than k times during the course of the algorithm. As the first step runs in time $2^{\mathcal{O}_s(k^{2(c+1)}) \log k} \cdot n$, and the second step runs in time $2^{\mathcal{O}_s(k \log k)} \cdot n$, they may be applied at most n times, and the claimed time complexity follows.

5 Concluding remarks

Limitations of the irrelevant vertex technique. An intriguing open question is whether Vertex Deletion to $\mathcal{G}$ admits an algorithm in time $2^{\mathcal{O}_{s_\mathcal{F}}(k^c)} \cdot n^{\mathcal{O}(1)}$ for some universal constant c (i.e., not depending on the class $\mathcal{G}$). Clearly, this is not the case of the algorithms of Theorem 1 and Theorem 2, running in time $2^{\mathcal{O}_{s_\mathcal{F}}(k^{2(c+2)} \log k)} \cdot n^3$ and $2^{\mathcal{O}_{s_\mathcal{F}}(k^{2(c+1)} \log k)} \cdot n^2$, respectively, where c is the *the palette-variety* of the minor-obstruction set $\mathcal{F}$ of $\mathcal{G}$ which, from the corresponding proofs, is estimated to be $c = 2^{\mathcal{O}({s_\mathcal{F}}^2 \cdot \log s_\mathcal{F})}$ and $c = 2^{\mathcal{O}({s_\mathcal{F}}^{24} \cdot \log s_\mathcal{F})}$, respectively (recall that $s_\mathcal{F}$ is the maximum size of a minor-obstruction of $\mathcal{G}$). We tend to believe that this dependence is unavoidable if we want to use the *irrelevant vertex technique*, as it reflects the *price of homogeneity*, as we mentioned in the end of Subsection 2.2. Having homogeneous walls is critical for the application of this technique when $\mathcal{G}$ is more general than surface embeddable graphs (in the bounded genus case, all subwalls are already homogeneous). Is there a way to prove that this behavior is unavoidable subject to some complexity assumption? An interesting result of this flavor concerning the existence of polynomial kernels for Vertex Deletion to $\mathcal{G}$ was given by Giannopoulou et al. [26] who proved that, even for minor-closed families $\mathcal{G}$ that exclude a planar graph, the dependence on $\mathcal{G}$ on the degree of the polynomial kernel, which exists because of [22], is unavoidable subject to reasonable complexity assumptions.

Variants. Our approach can be easily modified so to deal with several variants of the Vertex Deletion to $\mathcal{G}$ problem such as the *annotated version* of the problem where the input comes with a set of vertices that we are permitted to remove, the *weighted version* where the vertices carry positive weights, the *counting version* where we are asked to count the number of (minimal) solutions, or the *colored version* where the vertices of the input graph are partitioned into k parts and we are requested to pick one vertex from each part. The details can be found in the full version of the paper [47].

Other modification operations. Another direction is to consider graph modification to a minor-closed graph class for different modification operations. Our approach becomes just simpler in the case where the modification operation is edge removal or edge contraction. In these two cases, we immediately get rid of the branching part of our algorithms, and only the irrelevant vertex part needs to be applied. Another challenge is to combine all aforementioned modifications. This is more complicated (and tedious) but not more complex. What is really more complex is to additionally consider edge additions. We leave it as an open research challenge (a first step was done for the case of planar graphs [21]).

Lower bounds. Concerning lower bounds for VERTEX DELETION TO $\mathcal{G}$ under the Exponential Time Hypothesis [27], we are not aware of any lower bound stronger than $2^{o(k)} \cdot n^{\mathcal{O}(1)}$ for *any* minor-closed class $\mathcal{G}$. This lower bound already applies when $\mathcal{F} = \{K_2\}$, i.e., for the VERTEX COVER problem [7, 27].

References

1 Isolde Adler, Frederic Dorn, Fedor V. Fomin, Ignasi Sau, and Dimitrios M. Thilikos. Faster parameterized algorithms for minor containment. *Theoretical Computer Science*, 412(50):7018–7028, 2011. doi:10.1016/j.tcs.2011.09.015.

2 Isolde Adler, Martin Grohe, and Stephan Kreutzer. Computing excluded minors. In *Proc. of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 641–650, 2008. URL: http://dl.acm.org/citation.cfm?id=1347082.1347153.

3 Ernst Althaus and Sarah Ziegler. Optimal Tree Decompositions Revisited: A Simpler Linear-Time FPT Algorithm. *CoRR*, abs/1912.09144, 2019. arXiv:1912.09144.

4 Julien Baste, Ignasi Sau, and Dimitrios M. Thilikos. Optimal algorithms for hitting (topological) minors on graphs of bounded treewidth. In *Proc. of the 12th International Symposium on Parameterized and Exact Computation (IPEC)*, volume 89 of *LIPIcs*, pages 4:1–4:12, 2017. doi:10.4230/LIPIcs.IPEC.2017.4.

5 Julien Baste, Ignasi Sau, and Dimitrios M. Thilikos. A complexity dichotomy for hitting connected minors on bounded treewidth graphs: the chair and the banner draw the boundary. In *Proc. of the 31st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 951–970, 2020. doi:10.1137/130947374.

6 Julien Baste, Ignasi Sau, and Dimitrios M. Thilikos. Hitting minors on bounded treewidth graphs. II. single-exponential algorithms. *Theoretical Computer Science*, 814:135–152, 2020. doi:10.1016/j.tcs.2020.01.026.

7 Julien Baste, Ignasi Sau, and Dimitrios M. Thilikos. Hitting minors on bounded treewidth graphs. III. lower bounds. *Journal of Computer and System Sciences*, 109:56–77, 2020. doi:10.1016/j.jcss.2019.11.002.

8 Hans L. Bodlaender, Pål Grønås Drange, Markus S. Dregi, Fedor V. Fomin, Daniel Lokshtanov, and Michal Pilipczuk. A $c^k n$ 5-Approximation Algorithm for Treewidth. *SIAM Journal on Computing*, 45(2):317–378, 2016. doi:10.1137/130947374.

9 Hans L. Bodlaender, Pinar Heggernes, and Daniel Lokshtanov. Graph modification problems (dagstuhl seminar 14071). *Dagstuhl Reports*, 4(2):38–59, 2014. doi:10.4230/DagRep.4.2.38.

10 Hans L. Bodlaender, Hirotaka Ono, and Yota Otachi. A faster parameterized algorithm for pseudoforest deletion. *Discrete Applied Mathematics*, 236:42–56, 2018. doi:10.1016/j.dam.2017.10.018.

11 Andries E. Brouwer and Henk Jan Veldman. Contractibility and NP-completeness. *Journal of Graph Theory*, 11(1):71–79, 1987. doi:10.1002/jgt.3190110111.

12 Jianer Chen, Iyad A. Kanj, and Ge Xia. Improved upper bounds for vertex cover. *Theoretical Computer Science*, 411:3736–3756, September 2010. doi:10.1016/j.tcs.2010.06.026.

13 Christophe Crespelle, Pål Grønås, Drange, Fedor V. Fomin, and Petr A. Golovach. A survey of parameterized algorithms and the complexity of edge modification. *CoRR*, abs/2001.06867, 2013. `arXiv:2001.06867`.

14 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. `doi:10.1007/978-3-319-21275-3`.

15 Marek Cygan, Marcin Pilipczuk, Michal Pilipczuk, and Jakub Onufry Wojtaszczyk. An improved FPT algorithm and a quadratic kernel for pathwidth one vertex deletion. *Algorithmica*, 64(1):170–188, 2012. `doi:10.1007/s00453-011-9578-2`.

16 Erik D. Demaine, Fedor V. Fomin, Mohammadtaghi Hajiaghayi, and Dimitrios M. Thilikos. Bidimensional parameters and local treewidth. *SIAM Journal on Discrete Mathematics*, 18(3):501–511, 2004. `doi:10.1137/S0895480103433410`.

17 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. `doi:10.1007/978-1-4471-5559-1`.

18 Michael R. Fellows, Jan Kratochvíl, Matthias Middendorf, and Frank Pfeiffer. The complexity of induced minors and related problems. *Algorithmica*, 13(3):266–282, 1995. `doi:10.1007/BF01190507`.

19 Michael R. Fellows and Michael A. Langston. Nonconstructive tools for proving polynomial-time decidability. *Journal of the ACM*, 35(3):727–739, 1988. `doi:10.1145/44483.44491`.

20 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006. `doi:10.1007/3-540-29953-X`.

21 Fedor V. Fomin, Petr A. Golovach, and Dimitrios M. Thilikos. Modification to planarity is fixed parameter tractable. In *Proc. of the 36th International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 126 of *LIPIcs*, pages 28:1–28:17, 2019. `doi:10.4230/LIPIcs.STACS.2019.28`.

22 Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Planar $\mathcal{F}$-Deletion: Approximation, Kernelization and Optimal FPT Algorithms. In *Proc. of the 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 470–479, 2012. `doi:10.1109/FOCS.2012.62`.

23 Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, Saket Saurabh, and Meirav Zehavi. Hitting Topological Minors is FPT. *CoRR*, abs/1904.02944, 2019. To appear in *Proc. of the 52nd Annual ACM Symposium on Theory of Computing (STOC 2020)*. `arXiv:1904.02944`.

24 Fedor V. Fomin, Saket Saurabh, and Neeldhara Misra. Graph modification problems: A modern perspective. In *Proc. of the 9th International Frontiers in Algorithmics Workshop (FAW)*, volume 9130 of *LNCS*, pages 3–6, 2015. `doi:10.1007/978-3-319-19647-3_1`.

25 Michael R. Garey and David S. Johnson. *Computers and intractability. A guide to the theory of NP-completeness*. W. H. Freeman and Co., San Francisco, 1979.

26 Archontia C. Giannopoulou, Bart M. P. Jansen, Daniel Lokshtanov, and Saket Saurabh. Uniform kernelization complexity of hitting forbidden minors. *ACM Transactions on Algorithms*, 13(3):35:1–35:35, 2017. `doi:10.1145/3029051`.

27 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001. `doi:10.1006/jcss.2001.1774`.

28 Bart M. P. Jansen, Daniel Lokshtanov, and Saket Saurabh. A near-optimal planarization algorithm. In *Proc. of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1802–1811, 2014. `doi:10.1137/1.9781611973402.130`.

29 Gwenaël Joret, Christophe Paul, Ignasi Sau, Saket Saurabh, and Stéphan Thomassé. Hitting and harvesting pumpkins. *SIAM Journal on Discrete Mathematics*, 28(3):1363–1390, 2014. `doi:10.1137/120883736`.

30 Ken-ichi Kawarabayashi. Planarity allowing few error vertices in linear time. In *Proc. of the 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 639–648, 2009. `doi:10.1109/FOCS.2009.45`.

31 Ken-ichi Kawarabayashi and Yusuke Kobayashi. Linear min-max relation between the treewidth of H-minor-free graphs and its largest grid. In *Proc. of the 29th International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 14 of *LIPIcs*, pages 278–289, 2012. doi:10.4230/LIPIcs.STACS.2012.278.

32 Ken-ichi Kawarabayashi, Yusuke Kobayashi, and Bruce Reed. The disjoint paths problem in quadratic time. *Journal of Combinatorial Theory, Series B*, 2011. doi:10.1016/j.jctb.2011.07.004.

33 Ken-ichi Kawarabayashi, Robin Thomas, and Paul Wollan. A new proof of the flat wall theorem. *Journal of Combinatorial Theory, Series B*, 129:204–238, 2018. doi:10.1016/j.jctb.2017.09.006.

34 Eun Jung Kim, Alexander Langer, Christophe Paul, Felix Reidl, Peter Rossmanith, Ignasi Sau, and Somnath Sikdar. Linear kernels and single-exponential algorithms via protrusion decompositions. *ACM Transactions on Algorithms*, 12(2):21:1–21:41, 2016. doi:10.1145/2797140.

35 Eun Jung Kim, Maria J. Serna, and Dimitrios M. Thilikos. Data-compression for parametrized counting problems on sparse graphs. In *Proc. of the 29th International Symposium on Algorithms and Computation (ISAAC)*, volume 123 of *LIPIcs*, pages 20:1–20:13, 2018. doi:10.4230/LIPIcs.ISAAC.2018.20.

36 Tomasz Kociumaka and Marcin Pilipczuk. Faster deterministic feedback vertex set. *Information Processing Letters*, 114(10):556–560, 2014. doi:10.1016/j.ipl.2014.05.001.

37 Tomasz Kociumaka and Marcin Pilipczuk. Deleting Vertices to Graphs of Bounded Genus. *Algorithmica*, 81(9):3655–3691, 2019. doi:10.1007/s00453-019-00592-7.

38 Alexandr V. Kostochka. Lower bound of the Hadwiger number of graphs by their average degree. *Combinatorica*, 4:307–316, 1984. doi:10.1007/BF02579141.

39 John M. Lewis and Mihalis Yannakakis. The node-deletion problem for hereditary properties is NP-complete. *Journal of Computer and System Sciences*, 20(2):219–230, 1980. doi:10.1016/0022-0000(80)90060-4.

40 Daniel Lokshtanov. Wheel-Free Deletion Is W[2]-Hard. In *Proc. of the 3rd International Workshop on Parameterized and Exact Computation (IWPEC)*, volume 5018 of *LNCS*, pages 141–147, 2008. doi:10.1007/978-3-540-79723-4_14.

41 Dániel Marx and Ildikó Schlotter. Obtaining a planar graph by vertex deletion. *Algorithmica*, 62(3-4):807–822, 2012. doi:10.1007/s00453-010-9484-z.

42 Rolf Niedermeier. *Invitation to fixed parameter algorithms*, volume 31. Oxford University Press, 2006. doi:10.1093/ACPROF:OSO/9780198566076.001.0001.

43 Ljubomir Perkovic and Bruce Reed. An Improved Algorithm for Finding Tree Decompositions of Small Width. *International Journal of Foundations of Computer Science*, 11:365–371, January 2000. doi:10.1142/S0129054100000247.

44 Bruce Reed, Kaleigh Smith, and Adrian Vetta. Finding odd cycle transversals. *Operations Research Letters*, 32(4):299–301, 2004. doi:10.1016/j.orl.2003.10.009.

45 Neil Robertson and Paul D. Seymour. Graph Minors. XIII. The Disjoint Paths Problem. *Journal of Combinatorial Theory, Series B*, 63(1):65–110, 1995. doi:10.1006/jctb.1995.1006.

46 Neil Robertson and Paul D. Seymour. Graph Minors. XX. Wagner's conjecture. *Journal of Combinatorial Theory, Series B*, 92(2):325–357, 2004. doi:10.1016/j.jctb.2004.08.001.

47 Ignasi Sau, Gianos Stamoulis, and Dimitrios M. Thilikos. An FPT-algorithm for recognizing k-apices of minor-closed graph classes. *CoRR*, abs/2004.12692, 2020. arXiv:2004.12692.

48 Roded Sharan. *Graph Modification Problems and their Applications to Genomic Research.* PhD thesis, Sackler Faculty of Exact Sciences, School of Computer Science, 2002.

49 Dimitrios M. Thilikos. Graph minors and parameterized algorithm design. In *The Multivariate Algorithmic Revolution and Beyond - Essays Dedicated to Michael R. Fellows on the Occasion of His 60th Birthday*, pages 228–256, 2012.

50 Andrew Thomason. The extremal function for complete minors. *Journal of Combinatorial Theory, Series B*, 81(2):318–338, 2001. doi:10.1006/jctb.2000.2013.

Contraction: A Unified Perspective of Correlation Decay and Zero-Freeness of 2-Spin Systems

Shuai Shao
Department of Computer Sciences, University of Wisconsin-Madison, Madison, WI, USA
sh@cs.wisc.edu

Yuxin Sun
Department of Computer Sciences, University of Wisconsin-Madison, Madison, WI, USA
yxsun@cs.wisc.edu

Abstract

We study complex zeros of the partition function of 2-spin systems, viewed as a multivariate polynomial in terms of the edge interaction parameters and the uniform external field. We obtain new zero-free regions in which all these parameters are complex-valued. Crucially based on the zero-freeness, we are able to extend the existence of correlation decay to these complex regions from real parameters. As a consequence, we obtain an FPTAS for computing the partition function of 2-spin systems on graphs of bounded degree for these parameter settings. We introduce the *contraction* property as a *unified* sufficient condition to devise FPTAS via either Weitz's algorithm or Barvinok's algorithm. Our main technical contribution is a very simple but general approach to extend any *real* parameter of which the 2-spin system exhibits correlation decay to its *complex* neighborhood where the partition function is zero-free and correlation decay still exists. This result formally establishes the inherent connection between two distinct notions of phase transition for 2-spin systems: the existence of correlation decay and the zero-freeness of the partition function via a unified perspective, contraction.

2012 ACM Subject Classification Mathematics of computing → Approximation algorithms

Keywords and phrases 2-Spin system, Correlation decay, Zero-freeness, Phase transition, Contraction

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.96

Category Track A: Algorithms, Complexity and Games

Related Version A full version of this paper is available at `https://arxiv.org/abs/1909.04244`.

Funding The authors are supported by NSF CCF-1714275.

Acknowledgements We want to thank Professor Jin-Yi Cai, the advisor of the first author, for many inspiring discussions and valuable comments on a preliminary version of this paper. Despite his support, he generously declined our invitation for co-authorship. We also want to thank Professor Alex Scott, Dr. Jingcheng Liu and anonymous reviewers for their helpful comments.

1 Introduction

Spin systems originated from statistical physics to model interactions between neighbors on graphs. In this paper, we focus on 2-state spin (2-spin) systems. Such a system is specified by two edge interaction parameters β and γ, and a uniform external field λ. An instance is a graph $G = (V, E)$. A configuration σ is a mapping $\sigma : V \to \{+, -\}$ which assigns one of the two spins $+$ and $-$ to each vertex in V. The weight $w(\sigma)$ of a configuration σ is given by $w(\sigma) = \beta^{m_+(\sigma)}\gamma^{m_-(\sigma)}\lambda^{n_+(\sigma)}$, where $m_+(\sigma)$ denotes the number of $(+, +)$ edges under the configuration σ, $m_-(\sigma)$ denotes the number of $(-, -)$ edges, and $n_+(\sigma)$ denotes the number of vertices assigned to spin $+$. The partition function $Z_G(\beta, \gamma, \lambda)$ of the system parameterized by (β, γ, λ) is defined to be the sum of weights over all configurations, i.e.,

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 96; pp. 96:1–96:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

$$Z_G(\beta,\gamma,\lambda) = \sum_{\sigma:V\to\{+,-\}} w(\sigma).$$

It is a sum-of-product computation. If a 2-spin system is restricted to graphs of degree bounded by Δ, we say such a system is Δ-bounded.

In classical statistical mechanics the parameters (β,γ,λ) are usually non-negative real numbers, and such 2-spin systems are divided into the *ferromagnetic* case ($\beta\gamma > 1$) and the *antiferromagnetic* case ($\beta\gamma < 1$). The case $\beta\gamma = 1$ is degenerate. When (β,γ,λ) are non-negative numbers and they are not all zero, the partition function can be viewed as the normalizing factor of the Gibbs distribution, which is the distribution where a configuration σ is drawn with probability $\Pr_{G;\beta,\gamma,\lambda}(\sigma) = \frac{w(\sigma)}{Z_G(\beta,\gamma,\lambda)}$. However, it is meaningful to consider parameters of complex values. By analyzing the location of *complex* zeros of the partition function, the phenomenon of *phase transitions* was discovered by statistical physicists. One of the first and also the best known result is the Lee-Yang theorem [21] for the *Ising model*, a special case of 2-spin systems. This result was later extended to more general models by several people [1, 34, 37, 29, 24]. In this paper, we view the partition function $Z_G(\beta,\gamma,\lambda)$ as a *multivariate* polynomial over these three *complex* parameters (β,γ,λ). We study the zeros of this polynomial and the relation to the approximation of the partition function.

Partition functions encode rich information about the macroscopic properties of 2-spin systems. They are not only of significance in statistical physics, but also are well-studied in computer science. Computing the partition function of 2-spin systems given an input graph G can be viewed as the most basic case of Counting Graph Homomorphisms (#GH) [11, 5, 14, 8] and Counting Constraint Satisfaction Problems (#CSP) [10, 9, 6, 12, 7], which are two very well studied frameworks for counting problems. Many natural combinatorial problems can be formulated as 2-spin systems. For example, when $\beta = \gamma$, such a system is the famous *Ising model.* And when $\beta = 0$ and $\gamma = 1$, $Z_G(0,1,\lambda)$ is the independence polynomial of the graph G (also known as the *hard-core model* in statistical physics); it counts the number of independent sets of the graph G when $\lambda = 1$.

Related work

For exact computation of $Z_G(\beta,\gamma,\lambda)$, the problem is proved to be #P-hard for all complex valued parameters but a few very restricted trivial settings [2, 8, 9]. So the main focus is to approximate $Z_G(\beta,\gamma,\lambda)$. This is an area of active research, and many inspiring algorithms are developed. The pioneering algorithm developed by Jerrum and Sinclair gives a *fully polynomial-time randomized approximation scheme* (FPRAS) for the ferromagnetic Ising model [19]. This FPRAS is based on the *Markov Chain Monte Carlo* (MCMC) method which devises approximation counting algorithms via random sampling. Later, it was extended to general ferromagnetic 2-spin systems [15, 26]. The MCMC method can only handle non-negative parameters as it is based on probabilistic sampling.

The *correlation decay* method developed by Weitz [43] was originally used to devise *deterministic fully polynomial-time approximation schemes* (FPTAS) for the hardcore model up to the uniqueness threshold of the infinite regular tree. It turns out to be a very powerful tool for devising FPTAS for antiferromagnetic 2-spin systems [44, 22, 23, 39]. Combining with hardness results [40, 13], an exact threshold of computational complexity transition of antiferromagnetic 2-spin systems is identified and the only remaining case is at the critical point. On the other hand, for ferromagnetic 2-spin systems, limited results [44, 17] have been obtained via the correlation decay method. Although correlation decay is usually analyzed in 2-spin systems of non-negative parameters, it can be adapted to complex parameters. An FPTAS was obtained for the hard-core model in the Shearer's region (a disc in the complex plane) via correlation decay in [18].

Recently, a new method developed by Barvinok [3], and extended by Patel and Regts [30] is the *Taylor polynomial interpolation* method that turns complex zero-free regions of the partition function into FPTAS of corresponding complex parameters. Suppose that the partition function $Z_G(\beta,\gamma,\lambda)$ has no zero in a complex region containing an easy computing point, e.g., $\lambda = 0$. It turns out that, probably after a change of coordinates, $\log Z_G(\beta,\gamma,\lambda)$ is well approximated in a slightly smaller region by a low degree Taylor polynomials which can be efficiently computed. This method connects the long-standing study of complex zeros to algorithmic studies of the partition function of physical systems. Motivated by this, more recently some complex zero-free regions have been obtained for hard-core models [4, 32], Ising models [27, 31], and general 2-spin systems [16].

Our contribution

In this paper, we obtain new zero-free regions of the partition function of 2-spin systems. Crucially based on the zero-freeness, we are able to extend the existence of correlation decay to these *complex* regions from real parameters. As a consequence, we obtain an FPTAS for computing the partition function of bounded 2-spin systems for these parameter settings. Our result gives the first zero-free regions in which all three parameters (β,γ,λ) are complex-valued and new correlation decay results for bounded ferromagnetic 2-spin systems. Our main technical contribution is a very simple but general approach to extend any real parameter of which the bounded 2-spin system exhibits correlation decay to its complex neighborhood where the partition function is zero-free and correlation decay still exists. We show that for bounded 2-spin systems, the *real contraction*[1] property that ensures correlation decay exists for certain real parameters directly implies the zero-freeness and the existence of correlation decay of corresponding complex neighborhoods.

We formally describe our main result. We use $\boldsymbol{\zeta} \in \mathbb{C}^3$ to denote the parameter vector (β,γ,λ). Since the case $\beta = \gamma = 0$ is trivial, by symmetry we always assume $\gamma \neq 0$.

▶ **Theorem 1.** *Fix $\Delta \in \mathbb{N}$. If $\boldsymbol{\zeta}_0 \in \mathbb{R}^3$ satisfies real contraction for Δ, then there exists a $\delta > 0$ such that for any $\boldsymbol{\zeta} \in \mathbb{C}^3$ where $\|\boldsymbol{\zeta} - \boldsymbol{\zeta}_0\|_\infty < \delta$, we have*

- *$Z_G(\boldsymbol{\zeta}) \neq 0$ for every graph[2] G of degree at most Δ;*
- *the Δ-bounded 2-spin system specified by $\boldsymbol{\zeta}$ exhibits correlation decay.*

As a consequence, there is an FPTAS for computing $Z_G(\boldsymbol{\zeta})$.

This result formally establishes the inherent connection between two distinct notions of phase transition for bounded 2-spin systems: the existence of correlation decay and the zero-freeness of the partition function, via a unified perspective, *contraction*. The connection from the existence of correlation decay of real parameters to the zero-freeness of corresponding complex neighborhoods was already observed for the hard-core model [32] and the Ising model without external field [27]. In this paper, we extend it to general 2-spin systems, and furthermore we establish the connection from the zero-freeness of complex neighborhoods back to the existence of correlation decay of such complex regions.

Now, we give our zero-free regions. We first identify the sets of real parameters of which bounded 2-spin systems exhibit correlation decay.

[1] See Definition 11. In many cases, the existence of correlation decay boils down to this property.

[2] This is true even if G contains some vertices pinned by a *feasible configuration* (Definition 7).

▶ **Definition 2.** *Fix integer $\Delta \geq 3$. We define the following four real correlation decay sets.*

1. $\mathcal{S}_1^\Delta = \{\boldsymbol{\zeta} \in \mathbb{R}^3 \mid \frac{\Delta-2}{\Delta} < \sqrt{\beta\gamma} < \frac{\Delta}{\Delta-2}, \beta, \gamma > 0$ *and* $\lambda \geq 0\}$*,*
2. $\mathcal{S}_2^\Delta = \{\boldsymbol{\zeta} \in \mathbb{R}^3 \mid \beta\gamma < 1, \beta \geq 0, \gamma > 0, \lambda \geq 0,$ *and* $\boldsymbol{\zeta}$ *is up-to-*Δ *unique (Definition 14)*$\}$*,*
3. $\mathcal{S}_3^\Delta = \{\boldsymbol{\zeta} \in \mathbb{R}^3 \mid \beta\gamma > \frac{\Delta}{\Delta-2}, \beta, \gamma > 0$ *and* $0 \leq \lambda < \frac{\gamma}{t^{\Delta-1}[(\Delta-2)\beta\gamma-\Delta]}\}$ *where* $t = \max\{1, \beta\}$*,*
4. *and* $\mathcal{S}_4^\Delta = \{\boldsymbol{\zeta} \in \mathbb{R}^3 \mid \beta\gamma > \frac{\Delta}{\Delta-2}, \beta, \gamma > 0$ *and* $\lambda > \frac{(\Delta-2)\beta\gamma-\Delta}{\beta r^{\Delta-1}}\}$ *where* $r = \min\{1, 1/\gamma\}$*.*

When context is clear, we omit the superscript Δ*.*

The set $\mathcal{S}_1^\Delta$ was given in [44] and $\mathcal{S}_2^\Delta$ was given in [23]. To our best knowledge, $\mathcal{S}_1^\Delta$ and $\mathcal{S}_2^\Delta$ cover all *non-negative* parameters of which *bounded* 2-spin systems are known to exhibit correlation decay. The sets $\mathcal{S}_3^\Delta$ and $\mathcal{S}_4^\Delta$ are obtained in this paper[3]. They give new correlation decay results and hence FPTAS for bounded ferromagnetic 2-spin systems. When $\beta < \gamma$ and λ is sufficiently large, it is known that approximating the partition function of ferromagnetic 2-spin systems over general graphs is #BIS-hard [26]. Our result $\mathcal{S}_4^\Delta$ shows that there is an FPTAS for such a problem when restricted to graphs of bounded degree. When $\beta < 1 < \gamma$, the FPTAS obtained from $\mathcal{S}_3^\Delta$ is covered by [17].

▶ **Theorem 3.** *Fix integer $\Delta \geq 3$. For every $\boldsymbol{\zeta}_0 \in \mathcal{S}_i^\Delta$ $(i \in [4])$, there exists a $\delta > 0$ such that for any $\boldsymbol{\zeta} \in \mathbb{C}^3$ where $\|\boldsymbol{\zeta} - \boldsymbol{\zeta}_0\|_\infty < \delta$, we have*

- $Z_G(\boldsymbol{\zeta}) \neq 0$ *for every graph G of degree at most Δ; (G may contain a feasible configuration.)*
- *the Δ-bounded 2-spin system specified by $\boldsymbol{\zeta}$ exhibits correlation decay.*

Then via either Weitz's algorithm or Barvinok's algorithm, there is an FPTAS for computing the partition function $Z_G(\boldsymbol{\zeta})$.

▶ Remark 4. The choice of δ does not depend on the size of the graph, only on Δ and $\boldsymbol{\zeta}_0$.

Organization

This paper is organized as follows. In Section 2, we briefly describe Weitz's algorithm [43]. We introduce real contraction as a sufficient condition for the existence of correlation decay of real parameters, and we show that sets $\mathcal{S}_i^\Delta (i \in [4])$ satisfy it. In Section 3, we briefly describe Barvinok's algorithm [3]. We introduce complex contraction as a generalization of real contraction, and we show that it gives a unified sufficient condition for both the zero-freeness of the partition function and the existence of correlation decay of complex parameters. Finally, in Section 4, we prove our main result that real contraction implies complex contraction. This finishes the proof of Theorem 3. We use the following diagram (Figure 1) to summarize our approach to establish the connection between correlation decay and zero-freeness. We expect it to be further explored for other models.

Independent work

After a preliminary version [36] of this manuscript was posted, we learned that based on similar ideas, Liu simplified the proofs of [32] and [27], and generalized them to antiferromagnetic Ising models ($\beta = \gamma < 1$) in chapter 3 of his Ph.D. thesis [25], where similar zero-freeness results (a complex neighborhood of $\mathcal{S}_2^\Delta$ restricted to $\beta = \gamma$) were obtained. We mention that by using the unique analytic continuation and the inverse function theorem, our main technical result (Theorem 24) is generic; it does not rely on a particularly chosen potential function. Thus, in our approach we can work with *any* existing potential function based

[3] Since we do *not* assume $\beta \leqslant \gamma$ or $\beta \geqslant \gamma$, $\mathcal{S}_3^\Delta$ and $\mathcal{S}_4^\Delta$ are essentially the same by swapping β and γ and replacing λ with $1/\lambda$. However, if one restrict to $\beta \leqslant \gamma$, then $\mathcal{S}_3^\Delta$ is no longer the same as $\mathcal{S}_4^\Delta$.

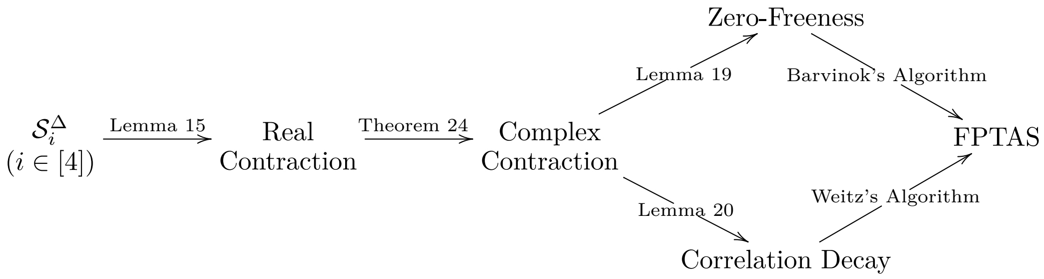


Figure 1 The structure of our approach.

argument for correlation decay even if the potential function does not have an explicit expression, for instance, the one used in [23] when $\beta \neq \gamma$. Furthermore, we mention also that based on the zero-freeness, we obtain new correlation decay results for complex parameters (Lemma 20). Note that Barvinok's algorithm requires an entire region in which the partition function is zero-free and there is an easy computing point in this region. However, our correlation decay results show that one can always devise an FPTAS for these parameter settings via Weitz's algorithm, even if Barvinok's algorithm fails.

2 Weitz's Algorithm

In this section, we describe Weitz's algorithm and introduce real contraction. We first consider positive parameters $\boldsymbol{\zeta} \in \mathbb{R}_+^3$. An obvious but important fact about $\boldsymbol{\zeta}$ being positive is that $Z_G(\boldsymbol{\zeta}) \neq 0$ for any graph G. This is true even if G contains arbitrary number of vertices pinned to spin $+$ or $-$. Then, the partition function can be viewed as the normalizing factor of the Gibbs distribution.

2.1 Notations and definitions

Let $\boldsymbol{\zeta} \in \mathbb{R}_+^3$. We use $p_v(\boldsymbol{\zeta})$ to denote the marginal probability of v being assigned to spin $+$ in the Gibbs distribution, i.e., $p_v(\boldsymbol{\zeta}) = \frac{Z^+_{G,v}(\boldsymbol{\zeta})}{Z_G(\boldsymbol{\zeta})}$, where $Z^+_{G,v}(\boldsymbol{\zeta})$ is the contribution to $Z_G(\boldsymbol{\zeta})$ over all configurations with v being assigned to spin $+$. We know that $p_v(\boldsymbol{\zeta})$ is well-defined since $Z_G(\boldsymbol{\zeta}) \neq 0$. (Later, we will extend the definition of $p_v(\boldsymbol{\zeta})$ to complex parameters $\boldsymbol{\zeta}$.)

Let $\sigma_\Lambda \in \{0,1\}^\Lambda$ be a configuration of some subset $\Lambda \subseteq V$. We allow Λ to be the empty set. We call vertices in Λ *pinned* and other vertices *free*. We use $p_v^{\sigma_\Lambda}(\boldsymbol{\zeta})$ to denote the marginal probability of a free vertex v ($v \notin \Lambda$) being assigned to spin $+$ conditioning on the configuration σ_Λ of Λ, i.e., $p_v^{\sigma_\Lambda}(\boldsymbol{\zeta}) = \frac{Z^{\sigma_\Lambda,+}_{G,v}(\boldsymbol{\zeta})}{Z^{\sigma_\Lambda}_G(\boldsymbol{\zeta})}$, where $Z^{\sigma_\Lambda}_G(\boldsymbol{\zeta})$ is the weight over all configurations where vertices in Λ are pinned by the configuration σ_Λ, and $Z^{\sigma_\Lambda,+}_{G,v}(\boldsymbol{\zeta})$ is the contribution to $Z^{\sigma_\Lambda}_G(\boldsymbol{\zeta})$ with v being assigned to spin $+$. Correspondingly, we can define $Z^{\sigma_\Lambda,-}_{G,v}(\boldsymbol{\zeta})$. Let $R^{\sigma_\Lambda}_{G,v}(\boldsymbol{\zeta}) := \frac{Z^{\sigma_\Lambda,+}_{G,v}(\boldsymbol{\zeta})}{Z^{\sigma_\Lambda,-}_{G,v}(\boldsymbol{\zeta})} = \frac{p_v^{\sigma_\Lambda}(\boldsymbol{\zeta})}{1-p_v^{\sigma_\Lambda}(\boldsymbol{\zeta})}$ be the ratio between the two probabilities that the free vertex v is assigned to spin $+$ and $-$, while imposing some condition σ_Λ. Since $Z_G(\boldsymbol{\zeta}) \neq 0$ for any graph G with arbitrary number of pinned vertices, both $p_v^{\sigma_\Lambda}(\boldsymbol{\zeta})$ and $R^{\sigma_\Lambda}_{G,v}(\boldsymbol{\zeta})$ are well-defined. When context is clear, we write $p_v(\boldsymbol{\zeta})$, $p_v^{\sigma_\Lambda}(\boldsymbol{\zeta})$ and $R^{\sigma_\Lambda}_{G,v}(\boldsymbol{\zeta})$ as p_v, $p_v^{\sigma_\Lambda}$ and $R^{\sigma_\Lambda}_{G,v}$ for convenience.

Since computing the partition function of 2-spin systems is self-reducible, if one can compute p_v for any vertex v, then the partition function can be computed via telescoping [20]. The goal of Weitz's algorithm is to estimate $p_v^{\sigma_\Lambda}$, which is equivalent to estimating $R_{G,v}^{\sigma_\Lambda}$. For the case that the graph is a tree T, $R_{T,v}^{\sigma_\Lambda}$ can be computed by recursion. Suppose that a free vertex v has d children, and s_1 of them are pinned to $+$, s_2 are pinned to $-$, and k are free ($s_1 + s_2 + k = d$). We denote these k free vertices by $v_i (i \in [k])$ and let T_i be the corresponding subtree rooted at v_i. We use σ_Λ^i to denote the configuration σ_Λ restricted to T_i. Since all subtrees are independent, it is easy to get the following recurrence relation,

$$R_{T,v}^{\sigma_\Lambda} = \frac{Z_{T,v}^{\sigma_\Lambda,+}(\boldsymbol{\zeta})}{Z_{T,v}^{\sigma_\Lambda,-}(\boldsymbol{\zeta})} = \frac{\lambda^{1+s_1}\beta^{s_1}\prod_{i=1}^{k}\left(\beta Z_{T_i,v_i}^{\sigma_\Lambda^i,+}(\boldsymbol{\zeta}) + Z_{T_i,v_i}^{\sigma_\Lambda^i,-}(\boldsymbol{\zeta})\right)}{\lambda^{s_1}\gamma^{s_2}\prod_{i=1}^{k}\left(Z_{T_i,v_i}^{\sigma_\Lambda^i,+}(\boldsymbol{\zeta}) + \gamma Z_{T_i,v_i}^{\sigma_\Lambda^i,-}(\boldsymbol{\zeta})\right)} = \frac{\lambda\beta^{s_1}}{\gamma^{s_2}}\prod_{i=1}^{k}\left(\frac{\beta R_{T_i,v_i}^{\sigma_\Lambda^i} + 1}{R_{T_i,v_i}^{\sigma_\Lambda^i} + \gamma}\right).$$

▶ **Definition 5** (Recursion function). *Let* $\mathbf{s} = (s_1, s_2, k) \in \mathbb{N}^3$ *(including 0). A recursion function* $F_{\mathbf{s}}$ *for 2-spin systems is defined to be*

$$F_{\mathbf{s}}(\boldsymbol{\zeta}, \mathbf{x}) := \lambda\beta^{s_1}\gamma^{-s_2}\prod_{i=1}^{k}\left(\frac{\beta x_i + 1}{x_i + \gamma}\right),$$

where $\boldsymbol{\zeta} = (\beta, \gamma, \lambda) \in \mathbb{C} \times (\mathbb{C}\backslash\{0\}) \times \mathbb{C}$ *and* $\mathbf{x} = (x_1, \ldots, x_k) \in (\mathbb{C}\backslash\{-\gamma\})^k$*. We define* $F_{\boldsymbol{\zeta},\mathbf{s}}(\mathbf{x}) := F_{\mathbf{s}}(\boldsymbol{\zeta}, \mathbf{x})$ *for fixed* $\boldsymbol{\zeta}$ *with* $\gamma \neq 0$*, and* $F_{\mathbf{x},\mathbf{s}}(\boldsymbol{\zeta}) := F_{\mathbf{s}}(\boldsymbol{\zeta}, \mathbf{x})$ *for fixed* $\mathbf{x}$*.*

▶ Remark 6. Every recursion function is analytic on its domain.

For a general graph G, Weitz reduced computing $R_{G,v}^{\sigma_\Lambda}$ to that in a tree T, called the self-avoiding walk (SAW) tree, and Weitz's theorem [43] states that $R_{G,v}^{\sigma_\Lambda} = R_{T,v}^{\sigma_\Lambda}$. (Please see [43], [17] or the full paper for more details.) We want to generalize Weitz's theorem to complex parameters $\boldsymbol{\zeta} \in \mathbb{C}^3$. First, we need to make sure that $R_{G,v}^{\sigma_\Lambda}$ and $p_v^{\sigma_\Lambda}$ are well-defined for any vertex $v \notin \Lambda$. This requires that $Z_G^{\sigma_\Lambda}(\boldsymbol{\zeta}) \neq 0$ for any graph G and any configuration σ_Λ. Now, $p_v^{\sigma_\Lambda}$ no longer has a probabilistic meaning. It is just a ratio of two complex numbers. However, one can easily observe that for some special parameters, there are trivial configurations such that $Z_{G,v}^{\sigma_\Lambda}(\boldsymbol{\zeta}) = 0$. We will rule these cases out as they are *infeasible.*

▶ **Definition 7** (Feasible configuration). *Let* $\boldsymbol{\zeta} \in \mathbb{C}^3$*. Given a graph* $G = (V, E)$ *of the 2-spin system specified by* $\boldsymbol{\zeta}$*, a configuration* σ_Λ *on some vertices* $\Lambda \subseteq V$ *is feasible if*

- σ_Λ *does not assign any vertex in* G *to spin* $+$ *when* $\lambda = 0$*, and*
- σ_Λ *does not assign any two adjacent vertices in* G *both to spin* $+$ *when* $\beta = 0$*.*

▶ Remark 8. Let σ_Λ be a feasible configuration. If we further pin one vertex $v \notin \Lambda$ to spin $-$, and get the configuration $\sigma_{\Lambda'}$ on $\Lambda' = \Lambda \cup \{v\}$, then $\sigma_{\Lambda'}$ is still a feasible configuration. Thus, given $\boldsymbol{\zeta} \in \mathbb{C}^3$, if $Z_G^{\sigma_\Lambda}(\boldsymbol{\zeta}) \neq 0$ for any graph G and any arbitrary feasible configuration σ_Λ on G, then both $p_v^{\sigma_\Lambda}$ and $R_{G,v}^{\sigma_\Lambda}$ are well-defined.

Given $R_{G,v}^{\sigma_\Lambda}$ is well-defined for some $\boldsymbol{\zeta} \in \mathbb{C}^3$, we can still compute it by recursion via SAW tree. We first consider the case that $\lambda \neq 0$. Let σ_Λ be a feasible configuration. It is easy to verify that the corresponding configuration on the SAW tree is also feasible and Weitz's theorem still holds. For the case that $\lambda = 0$, it is obvious that $R_{G,v}^{\sigma_\Lambda} \equiv 0$ for any graph G, any free vertex v and any feasible configuration σ_Λ. This is equal to the value of recursion functions $F_{\mathbf{s}}(\boldsymbol{\zeta}, \mathbf{x})$ at $\lambda = 0$. We agree that $R_{G,v}^{\sigma_\Lambda}$ can be computed by recursion functions when $\lambda = 0$, although Weitz's theorem does not hold for this case. For the case that $\beta = 0$, we have $R_{G,v}^{\sigma_\Lambda} = 0$ if one of the children of v is pinned to $+$. Then, we may view v as it is pinned to $-$. Thus, for $\beta = 0$, we only consider recursion functions $F_{\mathbf{s}}$ where $s_1 = 0$.

$$\begin{tikzcd}
\mathbf{y} \arrow[r, shift left, "\varphi"] \arrow[d, "F"] & \mathbf{x} \arrow[l, shift left, "\varphi^{-1}"] \arrow[d, "F^{\varphi}"] \\
F(\mathbf{y}) \arrow[r, shift left, "\varphi"] & F^{\varphi}(\mathbf{x}) \arrow[l, shift left, "\varphi^{-1}"]
\end{tikzcd}$$

Figure 2 Commutative diagram between F and F^{φ}.

2.2 Correlation decay and real contraction

The SAW tree may be exponentially large in size of G. In order to get a polynomial time approximation algorithm, we may run the tree recursion at logarithmic depth and hence in polynomial time, and plug in some arbitrary values at the truncated boundary. We have the following notion of *strong spatial mixing* (SSM) to bound the error caused by arbitrary guesses. It was originally introduced for non-negative parameters. Here, we extend it to complex parameters.

▶ **Definition 9** (Strong spatial mixing). *A 2-spin system specified by $\boldsymbol{\zeta} \in \mathbb{C}^3$ on a family $\mathcal{G}$ of graphs is said to exhibit strong spatial mixing if for any graph $G = (V, E) \in \mathcal{G}$, any $v \in V$, and any feasible configurations $\sigma_{\Lambda_1} \in \{0,1\}^{\Lambda_1}$ and $\tau_{\Lambda_2} \in \{0,1\}^{\Lambda_2}$ where $v \notin \Lambda_1 \cup \Lambda_2$, we have*

1. $Z_G^{\sigma_{\Lambda_1}}(\boldsymbol{\zeta}) \neq 0$ *and* $Z_G^{\tau_{\Lambda_2}}(\boldsymbol{\zeta}) \neq 0$*, and*
2. $\left|p_v^{\sigma_{\Lambda_1}} - p_v^{\tau_{\Lambda_2}}\right| \leq \exp(-\Omega(\mathrm{dist}(v, S)))$.

Here, $S \subseteq \Lambda_1 \cup \Lambda_2$ is the subset on which σ_{Λ_1} and τ_{Λ_2} differ (If a vertex v is free in one configuration but pinned in the other, we say that these two configurations differ at v), and $\mathrm{dist}_G(v, S)$ is the shortest distance from v to any vertex in S.

▶ Remark 10. When $\boldsymbol{\zeta} \in \mathbb{R}_+^3$, condition 1 is always satisfied. Condition 2 is a stronger form of SSM of real parameters (see Definition 5 of [23]). For real values, by monotonicity one need to consider only the case that $\Lambda_1 = \Lambda_2$ (the two configurations are on the same set of vertices). Here, we need to consider the case that $\Lambda_1 \neq \Lambda_2$. This is *necessary* to extend Weitz's algorithm to complex parameters.

In statistical physics, SSM implies correlation decay. If SSM holds, then the error caused by arbitrary boundary guesses at logarithmic depth of the SAW tree is polynomially small. Hence, Weitz's algorithm gives an FPTAS. A main technique that has been widely used to establish SSM is the *potential method* [33, 22, 23, 38, 17]. Instead of bounding the rate of decay of recursion functions directly, we use a potential function $\varphi(x)$ to map the original recursion to a new domain (See Figure 2 for the commutative diagram).

Let $F_{\mathbf{s}}(\boldsymbol{\zeta}, \mathbf{y})$ be a recursion function ($\mathbf{s} = (s_1, s_2, k) \in \mathbb{N}^3$). We use $F_{\mathbf{s}}^{\varphi}(\boldsymbol{\zeta}, \mathbf{x})$ to denote the composition $\varphi(F_{\mathbf{s}}(\boldsymbol{\zeta}, \boldsymbol{\varphi}^{-1}(\mathbf{x})))$ where $\mathbf{y} = \boldsymbol{\varphi}^{-1}(\mathbf{x})$ denotes the vector $(\varphi^{-1}(x_1), \ldots, \varphi^{-1}(x_k))$. Correspondingly, we define $F_{\boldsymbol{\zeta},\mathbf{s}}^{\varphi}(\mathbf{x})$ for fixed $\boldsymbol{\zeta}$, and $F_{\mathbf{x},\mathbf{s}}^{\varphi}(\boldsymbol{\zeta})$ for fixed $\mathbf{x}$. We will specify the domain on which $F_{\mathbf{s}}^{\varphi}$ is well-defined per each φ that will be used. For positive $\boldsymbol{\zeta}$, a sufficient condition for the bounded 2-spin system of $\boldsymbol{\zeta}$ exhibiting SSM is that there exists a "good" potential function φ such that $F_{\boldsymbol{\zeta},\mathbf{s}}^{\varphi}$ satisfies the following contraction property.

▶ **Definition 11** (Real contraction). *Fix $\Delta \in \mathbb{N}$. We say that $\boldsymbol{\zeta} \in \mathbb{R}^3$ satisfies real contraction for Δ if there is a real compact interval $J \subseteq \mathbb{R}$ where $\lambda \in J$, $-\gamma \notin J$ and $-1 \notin J$, and a real analytic function $\varphi : J \to I$ where $\varphi'(x) \neq 0$ for all $x \in J$, such that*

1. *$F_{\boldsymbol{\zeta},\mathbf{s}}(J^k) \subseteq J$ for every $\mathbf{s}$ with $\|\mathbf{s}\|_1 \leq \Delta - 1$ and $-1 \notin F_{\boldsymbol{\zeta},\mathbf{s}}(J^k)$ for every $\mathbf{s}$ with $\|\mathbf{s}\|_1 = \Delta$;*
2. *there exists $\eta > 0$ s.t. $\left\|\nabla F^{\varphi}_{\boldsymbol{\zeta},\mathbf{s}}(\mathbf{x})\right\|_1 \leq 1 - \eta$ for every $\mathbf{s}$ with $\|\mathbf{s}\|_1 \leq \Delta - 1$ and all $\mathbf{x} \in I^k$.*

We say φ defined on J is a good potential function for $\boldsymbol{\zeta}$.

▶ Remark 12. Since φ is analytic and $\varphi'(x) \neq 0$ for all $x \in J$, the function φ is invertible and the inverse $\varphi^{-1} : I \to J$ is also analytic by the inverse function theorem (Theorem 22). Also for every $\mathbf{s}$ with $\|\mathbf{s}\|_1 \leq \Delta - 1$, since $F_{\boldsymbol{\zeta},\mathbf{s}}(J^k) \subseteq J$ and $-\gamma \notin J$, the function $F_{\boldsymbol{\zeta},\mathbf{s}}(\mathbf{x})$ is analytic on J^k. Thus, $F^{\varphi}_{\boldsymbol{\zeta},\mathbf{s}}(\mathbf{x})$ is well-defined and analytic on I^k, and then $\nabla F^{\varphi}_{\boldsymbol{\zeta},\mathbf{s}}(\mathbf{x})$ is well-defined on I^k. Note that I is also a real compact interval since J is a real compact interval and φ is a real analytic function.

Since $-1 \notin J$, $F_{\boldsymbol{\zeta},\mathbf{s}}(J^k) \subseteq J$ implies that $-1 \notin F_{\boldsymbol{\zeta},\mathbf{s}}(J^k)$. Thus, real contraction implies that $-1 \notin F_{\boldsymbol{\zeta},\mathbf{s}}(J^k)$ for all $\|\mathbf{s}\|_1 \leq \Delta$. The reason why we require $F_{\boldsymbol{\zeta},\mathbf{s}}(J^k) \subseteq J$ for $\|\mathbf{s}\|_1 \leq \Delta - 1$, but only require $-1 \notin F_{\boldsymbol{\zeta},\mathbf{s}}(J^k)$ for $\|\mathbf{s}\|_1 = \Delta$ is that in a tree of degree at most Δ, only the root node may have Δ many children, while other nodes have at most $\Delta - 1$ many children.

▶ **Lemma 13.** *If $\boldsymbol{\zeta} \in \mathbb{R}^3_+$ satisfies real contraction for Δ, then the Δ-bounded 2-spin system of $\boldsymbol{\zeta}$ exhibits SSM. Thus there is an FPTAS for computing the partition function $Z_G(\boldsymbol{\zeta})$.*

Proof. The proof directly follows from the argument of the potential method, see [23, 17]. The FPTAS follows from Weitz's algorithm. ◀

Now, we give the sets of non-negative parameters that satisfy real contraction.

▶ **Definition 14** (Uniqueness condition [23]). *Let $\boldsymbol{\zeta} \in \mathbb{R}^3$ be antiferromagnetic ($\beta\gamma < 1$) with $\beta \geq 0$, $\gamma > 0$ and $\lambda \geq 0$, and $f_d(x) = \lambda \left(\frac{\beta x + 1}{x + \gamma}\right)^d$. We say $\boldsymbol{\zeta}$ is up-to-Δ unique, if $\lambda = 0$ or $\lambda > 0$ and there exists a constant $0 < c < 1$ such that for every integer $1 \leq d \leq \Delta - 1$,*

$$|f'_d(\hat{x}_d)| = \frac{d(1 - \beta\gamma)\hat{x}_d}{(\beta\hat{x}_d + 1)(\hat{x}_d + \gamma)} \leq c,$$

where $\hat{x}_d$ is the unique positive fixed point of the function $f_d(x)$.

Let $\mathcal{S}^{\Delta}_i (i \in [4])$ be the correlation decay sets defined in Definition 2. The set $\mathcal{S}^{\Delta}_1$ was given in [44] and $\mathcal{S}^{\Delta}_2$ was given in [23]. Directly following their proofs, it is easy to verify that both sets satisfy real contraction . The sets $\mathcal{S}^{\Delta}_3$ and $\mathcal{S}^{\Delta}_4$ are obtained in this paper, and we show that they also satisfy real contraction.

▶ **Lemma 15.** *Fix $\Delta \geq 3$. For every $\boldsymbol{\zeta} \in \mathcal{S}^{\Delta}_i (i \in [4])$, it satisfies real contraction for Δ.*

Proof. We only give a proof for sets $\mathcal{S}^{\Delta}_3$ and $\mathcal{S}^{\Delta}_4$. For a proof of sets $\mathcal{S}^{\Delta}_1$ and $\mathcal{S}^{\Delta}_2$, please refer to the full paper. The case that $\lambda = 0$ is easy to check. We only consider that $\lambda \neq 0$.

Since $\beta\gamma > \frac{\Delta}{\Delta - 2} > 1$, we have $1/\gamma < \beta$. We pick the interval $J = [\lambda r^{\Delta-1}, \lambda t^{\Delta-1}]$ where $r = \min\{1, 1/\gamma\}$ and $t = \max\{1, \beta\}$, and the potential function $\varphi = \log(x)$. Clearly, φ is analytic on J and $\varphi'(x) \neq 0$ for all $x \in J$. Also, we know that $\lambda \in J$, $-\gamma \notin J$ and $-1 \notin J$, and $-1 \notin F_{\boldsymbol{\zeta},\mathbf{s}}(J^k)$ for every $\|\mathbf{s}\|_1 = \Delta$. Since $\beta > 0$ and $\gamma > 0$, for all $x > 0$,

$$r \leq \min\{\beta, 1/\gamma\} \leq \frac{\beta x + 1}{x + \gamma} \leq \max\{\beta, 1/\gamma\} \leq t.$$

Thus, for all $\mathbf{x} \in J^k$,

$$F_{\boldsymbol{\zeta},\mathbf{s}}(\mathbf{x}) = \lambda\beta^{s_1}\gamma^{-s_2}\prod_{i=1}^{k}\left(\frac{\beta x_i+1}{x_i+\gamma}\right) \in \left[\lambda r^{\|\mathbf{s}\|_1}, \lambda t^{\|\mathbf{s}\|_1}\right] \subseteq \left[\lambda r^{\Delta-1}, \lambda t^{\Delta-1}\right].$$

Hence, $F_{\boldsymbol{\zeta},\mathbf{s}}(J^k) \subseteq J$ for every $\|\mathbf{s}\|_1 \leq \Delta - 1$. Condition 1 of real contraction is satisfied.

Let $I = \varphi(J)$. Consider the gradient $\nabla F^{\varphi}_{\boldsymbol{\zeta},\mathbf{s}}(\mathbf{x})$ for every $\|\mathbf{s}\|_1 \leq \Delta - 1$ and all $\mathbf{x} \in I^k$. Note that $F^{\varphi}_{\boldsymbol{\zeta},\mathbf{s}}(\mathbf{x}) = \log\lambda + s_1\log\beta - s_2\log\gamma + \sum_{i=1}^{k}\log\left(\frac{\beta e^{x_i}+1}{e^{x_i}+\gamma}\right)$, and $e^{x_i} = \varphi^{-1}(x_i) \in J$ and $e^{-x_i} \in [\frac{1}{\lambda t^{\Delta-1}}, \frac{1}{\lambda r^{\Delta-1}}]$ when $x_i \in I$.

If $\boldsymbol{\zeta} \in \mathcal{S}_3^{\Delta}$, then $(\Delta-2)\beta\gamma - \Delta < \frac{\gamma}{\lambda t^{\Delta-1}}$. Thus,

$$\left|\frac{\partial F^{\varphi}_{\boldsymbol{\zeta},\mathbf{s}}}{\partial x_i}\right| = \frac{\beta\gamma-1}{\beta e^{x_i}+\gamma e^{-x_i}+1+\beta\gamma} \leq \frac{\beta\gamma-1}{\frac{\gamma}{\lambda t^{\Delta-1}}+1+\beta\gamma} < \frac{\beta\gamma-1}{(\Delta-2)\beta\gamma-\Delta+1+\beta\gamma} = \frac{1}{\Delta-1}.$$

Otherwise, $\boldsymbol{\zeta} \in \mathcal{S}_4^{\Delta}$ and then $\lambda\beta r^{\Delta-1} > (\Delta-2)\beta\gamma - \Delta$. Thus,

$$\left|\frac{\partial F^{\varphi}_{\boldsymbol{\zeta},\mathbf{s}}}{\partial x_i}\right| = \frac{\beta\gamma-1}{\beta e^{x_i}+\gamma e^{-x_i}+1+\beta\gamma} \leq \frac{\beta\gamma-1}{\beta\lambda r^{\Delta-1}+1+\beta\gamma} < \frac{\beta\gamma-1}{(\Delta-2)\beta\gamma-\Delta+1+\beta\gamma} = \frac{1}{\Delta-1}.$$

Thus, in both cases, there exists some $\eta > 0$ such that $\left\|\nabla F^{\varphi}_{\boldsymbol{\zeta},\mathbf{s}}(\mathbf{x})\right\|_1 \leq 1 - \eta$ for every $\|\mathbf{s}\|_1 \leq \Delta - 1$ and all $\mathbf{x} \in I^k$. Condition 2 of real contraction is satisfied. ◀

In order to generalize the correlation decay technique to complex parameters, we need to ensure that the partition function is zero-free. Now, let us first take a detour to Barvinok's algorithm which crucially relies on the zero-free regions of the partition function. After we carve out our new zero-free regions, we will come back to the existence of correlation decay of complex parameters.

3 Barvinok's Algorithm

In this section, we describe Barvinok's algorithm and introduce complex contraction. Let $I = [0, t]$ be a closed real interval. We define the δ-strip of I to be $\{z \in \mathbb{C} \mid |z - z_0| < \delta, z_0 \in I\}$, denoted by I_δ. It is a complex neighborhood of I. Suppose a graph polynomial $P(z) = \sum_{i=0}^{n} a_i z^i$ of degree n is zero-free in I_δ. Barvinok's method [3] roughly states that for any $z \in I_\delta$, $P(z)$ can be $(1 \pm \varepsilon)$-approximated using coefficients $a_0, \ldots, a_k$ for some $k = O(e^{\Theta(1/\delta)}\log(n/\varepsilon))$, via truncating the Taylor expansion of the logarithm of the polynomial. For the partition function of 2-spin systems, these coefficients can be computed in polynomial time [30, 28]. For the purpose of obtaining an FPTAS, we will view the partition function as a univariate polynomial $Z_{G;\beta,\gamma}(\lambda)$ in λ and fix β and γ. The following result is known.

▶ **Lemma 16.** *Fix $\beta, \gamma \in \mathbb{C}$ and $\Delta \in \mathbb{N}$. Let G be a graph of degree at most Δ. If $Z_{G;\beta,\gamma}(\lambda) \neq 0$ lies in a δ-strip I_δ of $I = [0, t]$, then there is an FPTAS for computing $Z_{G;\beta,\gamma}(\lambda)$ for $\lambda \in I_\delta$.*

Proof. This lemma is a generalization of Lemma 4 in [16], where β and γ are both real. The generalization to complex valued parameters directly follows from the argument in [28]. ◀

3.1 Zero-freeness and complex contraction

With Lemma 16 in hand, the main effort is to obtain zero-free regions of the partition function. For this purpose, we will still view $Z_G(\boldsymbol{\zeta})$ as a multivariate polynomial in (β, γ, λ). A main and widely used approach to obtain zero-free regions is the *recursion* method [41, 35, 4, 32, 27]. This method is related to the correlation decay method.

Assuming $Z^-_{G,v}(\boldsymbol{\zeta}) \neq 0$ for some vertex v, then $Z_G(\boldsymbol{\zeta}) \neq 0$ is equivalent to $R_{G,v} = \frac{Z^+_{G,v}(\boldsymbol{\zeta})}{Z^-_{G,v}(\boldsymbol{\zeta})} \neq -1$. As pointed above, the ratio $R_{G,v}$ can be computed by recursion via the SAW tree in which v is the root. Roughly speaking, the key idea of the recursion method is to construct a *contraction* region $Q \subseteq \mathbb{C}$ where $\lambda \in Q$ and $-1 \notin Q$ such that for all recursion functions $F_{\boldsymbol{\zeta},\mathbf{s}}$ with $\|\mathbf{s}\|_1 \leq \Delta - 1$, $F_{\boldsymbol{\zeta},\mathbf{s}}(Q^k) \subseteq Q$, and for all $F_{\boldsymbol{\zeta},\mathbf{s}}$ with $\|\mathbf{s}\|_1 = \Delta$, $-1 \notin F_{\boldsymbol{\zeta},\mathbf{s}}(Q^k)$. This condition guarantees that with the initial value $R_{G,v_\ell} = \lambda$ where v_ℓ is a free leaf node in the SAW tree of which the degree is bounded by Δ, the recursion will never achieve -1. Hence, we have $Z_G(\boldsymbol{\zeta}) \neq 0$ by induction. Again, we may use a potential function $\varphi : Q \to P$ to change the domain, and we prove $F^{\varphi}_{\boldsymbol{\zeta},\mathbf{s}}(P^k) \subseteq P$.

Now, we introduce the following *complex* contraction property as a generalization of real contraction. This property gives a sufficient condition for the zero-freeness of the partition function.

▶ **Definition 17** (Complex contraction). *Fix $\Delta \in \mathbb{N}$. We say that $\boldsymbol{\zeta} \in \mathbb{C}^3$ satisfies complex contraction for Δ if there is a closed and bounded complex region $Q \subseteq \mathbb{C}$ where $\lambda \in Q$, $-\gamma \notin Q$ and $-1 \notin Q$, and an analytic and invertible function $\varphi : Q \to P$ where the inverse $\varphi^{-1} : P \to Q$ is also analytic and P is convex, such that*

1. *$F_{\boldsymbol{\zeta},\mathbf{s}}(Q^k) \subseteq Q$ for every $\mathbf{s}$ with $\|\mathbf{s}\|_1 \leq \Delta - 1$ and $-1 \notin F_{\boldsymbol{\zeta},\mathbf{s}}(Q^k)$ for every $\mathbf{s}$ with $\|\mathbf{s}\|_1 = \Delta$;*
2. *there exists $\eta > 0$ s.t. $\left\|\nabla F^{\varphi}_{\boldsymbol{\zeta},\mathbf{s}}(\mathbf{x})\right\|_1 \leq 1 - \eta$ for every $\mathbf{s}$ with $\|\mathbf{s}\|_1 \leq \Delta - 1$ and all $\mathbf{x} \in P^k$.*

▶ Remark 18. Similar to the remark of Definition 11, the function $F^{\varphi}_{\boldsymbol{\zeta},\mathbf{s}}(\mathbf{x})$ is well-defined and analytic on P^k. Here, we directly assume that the inverse φ^{-1} is analytic instead of $\varphi'(x) \neq 0$ for the sake of simplicity of our proof.

▶ **Lemma 19.** *If $\boldsymbol{\zeta} \in \mathbb{C}^3$ satisfies complex contraction for Δ, then $Z^{\sigma_\Lambda}_G(\boldsymbol{\zeta}) \neq 0$ for any graph G of degree at most Δ and any feasible configuration σ_Λ.*

Please refer to the full paper for a proof of Lemma 19. Such a proof only uses condition 1 of complex contraction. However, condition 2 combining with the zero-freeness result of Lemma 19 gives a sufficient condition for bounded 2-spin systems of *complex* parameters exhibiting correlation decay. This is a generalization of Lemma 13. Also, please refer to the full paper for a proof.

▶ **Lemma 20.** *If $\boldsymbol{\zeta} \in \mathbb{C}^3$ satisfies complex contraction for Δ, then the Δ-bounded 2-spin system of $\boldsymbol{\zeta}$ exhibits SSM. Thus, there is an FPTAS for computing $Z_G(\boldsymbol{\zeta})$ via Weitz's algorithm.*

4 From Real Contraction to Complex Contraction

In this section, we prove our main result. We first give some preliminaries in complex analysis. The main tools are the unique analytic continuation and the inverse function theorem. Here, we slightly modify the statements to fit for our settings. Please refer to [42] for the proofs.

▶ **Theorem 21** (Unique analytic continuation). *Let $f(x)$ be a (real) analytic function defined on a compact real interval $I \subseteq \mathbb{R}$. Then, there exists a complex neighborhood $\widetilde{I} \subseteq \mathbb{C}$ of I, and a (complex) analytic function $\widetilde{f}(x)$ defined on $\widetilde{I}$ such that $\widetilde{f}(x) \equiv f(x)$ for all $x \in I$. Moreover, if there is another (complex) analytic function $\widetilde{g}(x)$ also defined on $\widetilde{I}$ such that $\widetilde{g}(x) \equiv \widetilde{f}(x)$ for all $x \in I$ and the measure $\mathfrak{m}(I) \neq 0$, then $\widetilde{g}(x) \equiv \widetilde{f}(x)$ for all $x \in \widetilde{I}$. We call $\widetilde{f}(x)$ the unique analytic continuation of $f(x)$ on $\widetilde{I}$.*

▶ **Theorem 22** (Inverse function theorem). *For a real analytic function φ defined on a real interval $J \subseteq \mathbb{R}$, if $\varphi'(x) \neq 0$ for all $x \in J$, then φ is invertible on J and the inverse φ^{-1} is also analytic on $\varphi(J)$. For a complex analytic function ψ defined on $U \subseteq \mathbb{C}$, if $\psi'(z) \neq 0$ for some $z \in U$, then there exists a complex neighborhood D of z such that ψ is invertible on D and the inverse is also analytic.*

Combining the above theorems, we have the following result.

▶ **Lemma 23.** *Let $\varphi : J \to I$ be a real analytic function, and $\varphi'(x) \neq 0$ for all $x \in J$ where J and I are real compact intervals. Then, there exists an analytic continuation $\widetilde{\varphi}$ on a complex neighborhood $\widetilde{J}$ of J such that $\widetilde{\varphi}$ is invertible on $\widetilde{J}$ and the inverse $\widetilde{\varphi}^{-1}$ is also analytic.*

Proof. If $\mathfrak{m}(J) = 0$, i.e., $J = \{x\}$, then by Theorem 21 there exists an analytic continuation $\widetilde{\varphi}$ of φ. Since $\widetilde{\varphi}'(x) = \varphi'(x) \neq 0$, by Theorem 22, there is a neighborhood of x on which $\widetilde{\varphi}$ is invertible and the inverse $\widetilde{\varphi}^{-1}$ is analytic.

Otherwise, $\mathfrak{m}(J) \neq 0$. Since $\varphi(x)$ is analytic and $\varphi'(x) \neq 0$ for all $x \in J$, the function φ is invertible and by Theorem 22, the inverse $\varphi^{-1} : I \to J$ is analytic on I. By Theorem 21, there exists an analytic continuation $\widetilde{\varphi^{-1}}$ of φ^{-1} defined on a neighborhood $\widetilde{I_1}$ of I. Similarly, there exists an analytic continuation $\widetilde{\varphi}$ of φ defined on a neighborhood $\widetilde{J}$ of J. We use $\widetilde{I}$ to denote the image $\widetilde{\varphi}(\widetilde{J})$. Since $\widetilde{\varphi}$ is analytic, by the open mapping theorem $\widetilde{I}$ is an open set in the complex plane. Clearly, we have $\varphi(J) = I \subseteq \widetilde{I}$. We can pick $\widetilde{J}$ small enough while still keeping $J \subseteq \widetilde{J}$ such that the image $\widetilde{I} = \widetilde{\varphi}(\widetilde{J}) \subseteq \widetilde{I_1}$ and still $I \subseteq \widetilde{I}$. Thus, the composition $\widetilde{\varphi^{-1}} \circ \widetilde{\varphi}$ is a well-defined analytic function on $\widetilde{J}$. Clearly, we have that

$$\widetilde{\varphi^{-1}} \circ \widetilde{\varphi}(x) = \varphi^{-1} \circ \varphi(x) \equiv x \text{ for all } x \in J.$$

Since $\mathfrak{m}(J) \neq 0$, by Theorem 21, we have that $\widetilde{\varphi^{-1}} \circ \widetilde{\varphi}(x) \equiv x$ for all $x \in \widetilde{J}$.

Thus, $\widetilde{\varphi}$ is invertible on $\widetilde{J}$ and the inverse $\widetilde{\varphi}^{-1} = \widetilde{\varphi^{-1}}$ is analytic. ◀

Now, we are ready to prove our main result.

▶ **Theorem 24.** *If $\boldsymbol{\zeta}_0$ satisfies real contraction for Δ, then there exists a $\delta > 0$ such that for every $\boldsymbol{\zeta} \in \mathbb{C}^3$ with $\|\boldsymbol{\zeta} - \boldsymbol{\zeta}_0\|_\infty < \delta$, $\boldsymbol{\zeta}$ satisfies complex contraction for Δ.*

Proof. Let $\varphi : J \to I$ be a good potential function for $\boldsymbol{\zeta}_0$. By Definition 11 and Lemma 23, there exists a neighborhood $\widetilde{J}$ of J such that the analytic continuation $\widetilde{\varphi} : \widetilde{J} \to \widetilde{I}$ of φ on $\widetilde{J}$ is invertible. Here $\widetilde{I} = \widetilde{\varphi}(\widetilde{J})$ is a neighborhood of I, and the inverse $\widetilde{\varphi}^{-1}$ is also analytic on $\widetilde{I}$. We use $\mathcal{B}_\delta := \{\mathbf{z} \in \mathbb{C}^3 \mid \|\mathbf{z} - \boldsymbol{\zeta}_0\|_\infty < \delta\}$ to denote the 3-dimensional complex ball around $\boldsymbol{\zeta}_0$ of radius δ in terms of the infinity norm. Recall that we define $I_\varepsilon = \{z \in \mathbb{C} \mid |z - z_0| < \varepsilon, z_0 \in I\}$. Given a set $U \subseteq \mathbb{C}^k$, we use $\overline{U}$ to denote its closure.

We first show that we can pick a pair of $(\delta_1, \varepsilon_1)$ such that for every $\mathbf{s}$ with $\|\mathbf{s}\|_1 \leq \Delta - 1$, the composition

$$F_{\mathbf{s}}^{\widetilde{\varphi}}(\boldsymbol{\zeta}, \mathbf{x}) = \widetilde{\varphi}(F_{\mathbf{s}}(\boldsymbol{\zeta}, \widetilde{\boldsymbol{\varphi}}^{-1}(\mathbf{x}))) \text{ is well-defined and analytic on } \mathcal{B}_{\delta_1} \times I_{\varepsilon_1}^k.$$

Given some $\mathbf{s}$ with $\|\mathbf{s}\|_1 \leq \Delta - 1$, we consider the function $F_{\mathbf{s}}(\boldsymbol{\zeta}, \mathbf{x})$. We know that it is analytic on a neighborhood of $\{\boldsymbol{\zeta}_0\} \times J^k$ and by real contraction we have $F_{\mathbf{s}}(\boldsymbol{\zeta}_0, J^k) \subseteq J$. Then, we can pick some $\delta_{\mathbf{s}}$ and a neighborhood $\widetilde{J_{\mathbf{s}}}$ of J that are small enough such that $F_{\mathbf{s}}(\boldsymbol{\zeta}, \mathbf{x})$ is analytic on $\mathcal{B}_{\delta_{\mathbf{s}}} \times \widetilde{J}_{\mathbf{s}}^k$, and $F_{\mathbf{s}}(\mathcal{B}_{\delta_{\mathbf{s}}}, \widetilde{J}_{\mathbf{s}}^k) \subseteq \widetilde{J}$. Let

$$\delta_1 = \min_{\|\mathbf{s}\|_1 \leq \Delta - 1} \{\delta_{\mathbf{s}}\} \quad \text{and} \quad \widetilde{J_1} = \bigcap_{\|\mathbf{s}\|_1 \leq \Delta - 1} \widetilde{J_{\mathbf{s}}}.$$

Since there is only a finite number of $\mathbf{s}$ with $\|\mathbf{s}\|_1 \leq \Delta - 1$, we have that $\delta_1 > 0$, and $\widetilde{J_1}$ is open and it is a neighborhood of J. Then, $F_{\mathbf{s}}(\mathcal{B}_{\delta_1}, \widetilde{J_1}) \subseteq \widetilde{J}$ for every $\mathbf{s}$ with $\|\mathbf{s}\|_1 \leq \Delta - 1$. Since $\widetilde{\varphi}^{-1}$ is analytic on $\widetilde{I}$ and $\widetilde{\varphi}^{-1}(I) = J$, similarly we can pick a small enough neighborhood $\widetilde{I_1}$ of I where $\widetilde{I_1} \subseteq \widetilde{I}$ such that $\widetilde{\varphi}^{-1}(\widetilde{I_1}) \subseteq \widetilde{J_1}$. For every $z_0 \in I$, we can pick an ε_{z_0} such that the disc $B_{z_0,\varepsilon_{z_0}} := \{z \in \mathbb{C} \mid |z - z_0| < \varepsilon_{z_0}\}$ is in $\widetilde{I_1}$. Recall that I is a compact real interval, by the finite cover theorem, we can uniformly pick an ε_1 such that $I \subseteq I_{\varepsilon_1} \subseteq \widetilde{I_1}$. Thus, $F^{\widetilde{\varphi}}_{\mathbf{s}}(\boldsymbol{\zeta}, \mathbf{x})$ is well-defined and analytic on $\mathcal{B}_{\delta_1} \times I^k_{\varepsilon_1}$ for every $\mathbf{s}$ with $\|\mathbf{s}\|_1 \leq \Delta - 1$. In fact, $F^{\widetilde{\varphi}}_{\mathbf{s}}$ is a (multivariate) analytic continuation of $F^{\varphi}_{\mathbf{s}}$. Since I is a compact interval, in the following when we pick a neighborhood $\widetilde{I}$ of I, without loss of generality, we may always pick $\widetilde{I}$ as an ε-strip I_ε of I.

Then, we show that we can pick a pair of $(\delta_2, \varepsilon_2)$ where $\delta_2 < \delta_1$ and $\varepsilon_2 < \varepsilon_1$, a constant $M > 0$ and a constant $\eta > 0$ such that for every $\mathbf{s}$ with $\|\mathbf{s}\|_1 \leq \Delta - 1$,

$$\left\|\nabla F^{\widetilde{\varphi}}_{\boldsymbol{\zeta},\mathbf{s}}(\mathbf{x})\right\|_1 \leq 1 - \eta \qquad \text{and} \qquad \left\|\nabla F^{\widetilde{\varphi}}_{\mathbf{x},\mathbf{s}}(\boldsymbol{\zeta})\right\|_1 \leq M$$

for all $\boldsymbol{\zeta} \in \overline{\mathcal{B}_{\delta_2}}$ and all $\mathbf{x} \in \overline{I^k_{\varepsilon_2}}$. By real contraction, there is an $\eta' > 0$ such that $\left\|\nabla F^{\widetilde{\varphi}}_{\boldsymbol{\zeta}_0,\mathbf{s}}(\mathbf{x})\right\|_1 \leq 1 - \eta'$ for every $\mathbf{s}$ with $\|\mathbf{s}\|_1 \leq \Delta - 1$ and all $\mathbf{x} \in I^k$. Given some $\mathbf{s}$ with $\|\mathbf{s}\|_1 \leq \Delta - 1$, since $F^{\widetilde{\varphi}}_{\mathbf{s}}(\boldsymbol{\zeta}, \mathbf{x})$ is analytic on $\mathcal{B}_{\delta_1} \times I^k_{\varepsilon_1}$, by continuity we can pick some $\delta_{\mathbf{s}} < \delta_1$ and $\varepsilon_{\mathbf{s}} < \varepsilon_1$ such that $\left\|\nabla F^{\widetilde{\varphi}}_{\boldsymbol{\zeta},\mathbf{s}}(\mathbf{x})\right\|_1 \leq 1 - \frac{\eta'}{2}$ for all $\boldsymbol{\zeta} \in \overline{\mathcal{B}_{\delta_{\mathbf{s}}}}$ and all $\mathbf{x} \in \overline{I^k_{\varepsilon_{\mathbf{s}}}}$. In addition, let

$$M_{\mathbf{s}} = \sup_{\boldsymbol{\zeta} \in \overline{\mathcal{B}_{\delta_{\mathbf{s}}}}, \mathbf{x} \in \overline{I^k_{\varepsilon_{\mathbf{s}}}}} \left\|\nabla F^{\widetilde{\varphi}}_{\mathbf{x},\mathbf{s}}(\boldsymbol{\zeta})\right\|_1,$$

and we know that $M_{\mathbf{s}} < +\infty$ since $F^{\widetilde{\varphi}}$ is analytic on $\overline{\mathcal{B}_{\delta_{\mathbf{s}}}} \times \overline{I^k_{\varepsilon_{\mathbf{s}}}}$ which is closed and bounded. Finally, let

$$\eta = \frac{\eta'}{2}, \quad \delta_2 = \min_{\|\mathbf{s}\|_1 \leq \Delta - 1}\{\delta_{\mathbf{s}}\}, \quad \varepsilon_2 = \min_{\|\mathbf{s}\|_1 \leq \Delta - 1}\{\varepsilon_{\mathbf{s}}\}, \quad \text{and} \quad M = \max_{\|\mathbf{s}\|_1 \leq \Delta - 1}\{M_{\mathbf{s}}\}.$$

These choices will satisfy our requirement.

For the case that $\|\mathbf{s}\|_1 = \Delta$, we show that we can pick a pair of $(\delta_3, \varepsilon_3)$ where $\delta_3 < \delta_1$ and $\varepsilon_3 < \varepsilon_1$ such that for every $\mathbf{s}$ with $\|\mathbf{s}\|_1 = \Delta$, we have $-1 \notin F_{\mathbf{s}}(\overline{\mathcal{B}_{\delta_3}}, \widetilde{J}^k_2)$ where $\widetilde{J_2} = \widetilde{\varphi}^{-1}(\overline{I_{\varepsilon_3}})$ is a closed neighborhood of J. Since $F_{\mathbf{s}}$ is analytic, by real contraction, $-1 \notin F_{\boldsymbol{\zeta}_0,\mathbf{s}}(J^k)$ which is closed. Again by continuity we can pick some $(\delta_3, \varepsilon_3)$ that satisfy our requirement.

Since $\boldsymbol{\zeta}_0 = (\beta_0, \gamma_0, \lambda_0)$ satisfies real contraction, we have $\lambda_0 \in J$, $-\gamma_0 \notin J$ and $-1 \notin J$. Recall that $J = \widetilde{\varphi}^{-1}(I)$. Again, since $\widetilde{\varphi}^{-1}$ is analytic, by continuity we can pick some $\varepsilon \leq \min\{\varepsilon_2, \varepsilon_3\}$ such that $\lambda_0 \in \widetilde{\varphi}^{-1}(I_\varepsilon)$ (an open set), $-\gamma_0 \notin \widetilde{\varphi}^{-1}(\overline{I_\varepsilon})$ (a closed set) and $-1 \notin \widetilde{\varphi}^{-1}(\overline{I_\varepsilon})$. Moreover, we can pick some δ_4 small enough such that the disc $B_{\lambda_0,\delta_4} := \{z \in \mathbb{C} \mid |z - \lambda_0| < \delta_4\}$ is in $\widetilde{\varphi}^{-1}(I_\varepsilon)$, and the disc $B_{-\gamma_0,\delta_4} := \{z \in \mathbb{C} \mid |z - (-\gamma_0)| < \delta_4\}$ is disjoint with $\widetilde{\varphi}^{-1}(\overline{I_\varepsilon})$. Let $P = \overline{I_\varepsilon}$ and $Q = \widetilde{\varphi}^{-1}(\overline{I_\varepsilon})$. Clearly, P is convex. For every $\boldsymbol{\zeta}$ with $\|\boldsymbol{\zeta} - \boldsymbol{\zeta}_0\|_\infty < \delta$, we have $\lambda \in Q$, $-\gamma \notin Q$ and $-1 \notin Q$. In addition, we know that Q is closed and bounded since P is closed and bounded and $\widetilde{\varphi}^{-1}$ is analytic on P. Finally, let $\delta = \min\{\delta_2, \delta_3, \delta_4, \frac{\varepsilon\eta}{M}\}$. We show that for every $\mathbf{s}$ with $\|\mathbf{s}\|_1 \leq \Delta - 1$, $F^{\widetilde{\varphi}}_{\mathbf{s}}(\mathcal{B}_\delta, P^k) \subseteq P$, which implies that $F_{\mathbf{s}}(\mathcal{B}_\delta, Q^k) \subseteq Q$.

Consider some $\mathbf{x} \in P^k$. By the definition of P, there exists an $\mathbf{x}_0 \in I^k$ such that $\|\mathbf{x} - \mathbf{x}_0\|_\infty \le \varepsilon$. Also, consider some $\boldsymbol{\zeta} \in \mathcal{B}_\delta$, and we have $\|\boldsymbol{\zeta} - \boldsymbol{\zeta}_0\|_\infty < \delta$. Then, for every $\mathbf{s}$ with $\|\mathbf{s}\|_1 \le \Delta - 1$, consider $F_{\mathbf{s}}^{\widetilde{\varphi}}(\boldsymbol{\zeta}, \mathbf{x}) - F_{\mathbf{s}}^{\widetilde{\varphi}}(\boldsymbol{\zeta}_0, \mathbf{x}_0)$. We have

$$\begin{aligned}
&\left|F_{\mathbf{s}}^{\widetilde{\varphi}}(\boldsymbol{\zeta}, \mathbf{x}) - F_{\mathbf{s}}^{\widetilde{\varphi}}(\boldsymbol{\zeta}_0, \mathbf{x}_0)\right| \\
\le &\left|F_{\mathbf{s}}^{\widetilde{\varphi}}(\boldsymbol{\zeta}, \mathbf{x}) - F_{\mathbf{s}}^{\widetilde{\varphi}}(\boldsymbol{\zeta}_0, \mathbf{x})\right| + \left|F_{\mathbf{s}}^{\widetilde{\varphi}}(\boldsymbol{\zeta}_0, \mathbf{x}) - F_{\mathbf{s}}^{\widetilde{\varphi}}(\boldsymbol{\zeta}_0, \mathbf{x}_0)\right| \\
\le &\sup_{\boldsymbol{\zeta}' \in \mathcal{B}_\delta} \left\|\nabla F_{\mathbf{x},\mathbf{s}}^{\widetilde{\varphi}}(\boldsymbol{\zeta}')\right\|_1 \cdot \|\boldsymbol{\zeta} - \boldsymbol{\zeta}_0\|_\infty + \sup_{\mathbf{x}' \in P^k} \left\|\nabla F_{\boldsymbol{\zeta}_0,\mathbf{s}}^{\widetilde{\varphi}}(\mathbf{x}')\right\|_1 \cdot \|\mathbf{x} - \mathbf{x}_0\|_\infty \\
\le &M\delta + (1 - \eta) \cdot \varepsilon \le \varepsilon.
\end{aligned}$$

The second inequality above uses the fact that both $\mathcal{B}_\delta$ and P^k are convex, which ensures that the line between $\boldsymbol{\zeta}_0$ and $\boldsymbol{\zeta}$ is in $\mathcal{B}_\delta$ and the line between $\mathbf{x}_0$ and $\mathbf{x}$ is in P^k. By real contraction, we know that $F_{\mathbf{s}}^{\widetilde{\varphi}}(\boldsymbol{\zeta}_0, \mathbf{x}_0) \in I$ since $\mathbf{x}_0 \in I^k$. Thus, we have $F_{\mathbf{s}}^{\widetilde{\varphi}}(\boldsymbol{\zeta}, \mathbf{x}) \in P$. Thus, for every $\boldsymbol{\zeta}$ with $\|\boldsymbol{\zeta} - \boldsymbol{\zeta}_0\|_\infty < \delta$, we have that $\lambda \in Q$, $-\gamma \notin Q$ and $-1 \notin Q$, and

1. $F_{\boldsymbol{\zeta},\mathbf{s}}(Q^k) \subseteq Q$ for every $\mathbf{s}$ with $\|\mathbf{s}\|_1 \le \Delta - 1$ and $-1 \notin F_{\boldsymbol{\zeta},\mathbf{s}}(Q^k)$ for every $\mathbf{s}$ with $\|\mathbf{s}\|_1 = \Delta$;
2. there exists $\eta > 0$ s.t. $\left\|\nabla F_{\boldsymbol{\zeta},\mathbf{s}}^{\varphi}(\mathbf{x})\right\|_1 \le 1 - \eta$ for every $\mathbf{s}$ with $\|\mathbf{s}\|_1 \le \Delta - 1$ and all $\mathbf{x} \in P^k$.

The function $\widetilde{\varphi} : Q \to P$ is a good potential function for $\boldsymbol{\zeta}$. ◀

Combining Lemmas 15, 19, 20 and Theorem 24, we have the following result.

▶ **Theorem 25.** *Fix $\Delta \ge 3$. For every $\boldsymbol{\zeta}_0 \in \mathcal{S}_i^\Delta$ ($i \in [4]$), there exists a $\delta > 0$ such that for any $\boldsymbol{\zeta} \in \mathbb{C}^3$ where $\|\boldsymbol{\zeta} - \boldsymbol{\zeta}_0\|_\infty < \delta$, we have*

- *$Z_G^{\sigma_\Lambda}(\boldsymbol{\zeta}) \ne 0$ for every graph G of degree at most Δ and every feasible configuration σ_Λ;*
- *the Δ-bounded 2-spin system specified by $\boldsymbol{\zeta}$ exhibits correlation decay.*

Then via either Weitz's or Barvinok's algorithm, there is an FPTAS for computing $Z_G(\boldsymbol{\zeta})$.

▶ Remark 26. The choice of δ does not depend on the size of the graph, only on Δ and $\boldsymbol{\zeta}_0$. In particular, let D be a compact set in $\mathcal{S}_i^\Delta$ for some $i \in [4]$. Then there is a uniform δ such that for all $\boldsymbol{\zeta}$ in a complex neighborhood D_δ of radius δ around D, i.e., $\boldsymbol{\zeta} \in D_\delta := \{\mathbf{z} \in \mathbb{C}^3 \mid \|\mathbf{z} - \mathbf{z}_0\|_\infty < \delta, \mathbf{z}_0 \in D\}$, $Z_G(\boldsymbol{\zeta}) \ne 0$ for every graph G of degree at most Δ. In addition, in order to apply Barvinok's algorithm, by Lemma 16, we need to make sure that the zero-free regions contain $\lambda = 0$ (an easy computing point). This is true for $\mathcal{S}_1^\Delta$, $\mathcal{S}_2^\Delta$ and $\mathcal{S}_3^\Delta$. For parameters in $\mathcal{S}_4^\Delta$, we will reduce the problem to a case in $\mathcal{S}_3^\Delta$ by swapping β and γ and replacing λ by $1/\lambda$. Then, one can apply Barvinok's algorithm.

References

1 Taro Asano. Lee-Yang Theorem and the Griffiths Inequality for the Anisotropic Heisenberg Ferromagnet. *Physical Review Letters*, 24(25):1409, 1970.

2 Francisco Barahona. On the computational complexity of Ising spin glass models. *Journal of Physics A: Mathematical and General*, 15(10):3241, 1982.

3 Alexander Barvinok. *Combinatorics and Complexity of Partition Functions*, volume 9. Springer, 2016.

4 Ferenc Bencs and Péter Csikvári. Note on the zero-free region of the hard-core model. *arXiv preprint*, 2018. `arXiv:1807.08963`.

5 Andrei Bulatov and Martin Grohe. The complexity of partition functions. *Theoretical Computer Science*, 348(2-3):148–186, 2005.

6 Andrei A Bulatov. The complexity of the counting constraint satisfaction problem. *Journal of the ACM (JACM)*, 60(5):1–41, 2013.

7 Jin-Yi Cai and Xi Chen. Complexity of counting CSP with complex weights. *Journal of the ACM (JACM)*, 64(3):1–39, 2017.

8 Jin-Yi Cai, Xi Chen, and Pinyan Lu. Graph homomorphisms with complex values: A dichotomy theorem. *SIAM Journal on Computing*, 42(3):924–1029, 2013.

9 Jin-Yi Cai, Pinyan Lu, and Mingji Xia. The complexity of complex weighted Boolean #CSP. *Journal of Computer and System Sciences*, 80(1):217–236, 2014.

10 Martin Dyer, Leslie Ann Goldberg, and Mark Jerrum. The complexity of weighted Boolean #CSP. *SIAM Journal on Computing*, 38(5):1970–1986, 2009.

11 Martin Dyer and Catherine Greenhill. The complexity of counting graph homomorphisms. *Random Structures & Algorithms*, 17(3-4):260–289, 2000.

12 Martin Dyer and David Richerby. An effective dichotomy for the counting constraint satisfaction problem. *SIAM Journal on Computing*, 42(3):1245–1274, 2013.

13 Andreas Galanis, Daniel Štefankovič, and Eric Vigoda. Inapproximability of the partition function for the antiferromagnetic Ising and hard-core models. *Combinatorics, Probability and Computing*, 25(4):500–559, 2016.

14 Leslie Ann Goldberg, Martin Grohe, Mark Jerrum, and Marc Thurley. A complexity dichotomy for partition functions with mixed signs. *SIAM Journal on Computing*, 39(7):3336–3402, 2010.

15 Leslie Ann Goldberg, Mark Jerrum, and Mike Paterson. The computational complexity of two-state spin systems. *Random Structures & Algorithms*, 23(2):133–154, 2003.

16 Heng Guo, Jingcheng Liu, and Pinyan Lu. Zeros of ferromagnetic 2-spin systems. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 181–192. SIAM, 2020.

17 Heng Guo and Pinyan Lu. Uniqueness, spatial mixing, and approximation for ferromagnetic 2-spin systems. *ACM Transactions on Computation Theory (TOCT)*, 10(4):1–25, 2018.

18 Nicholas JA Harvey, Piyush Srivastava, and Jan Vondrák. Computing the independence polynomial: from the tree threshold down to the roots. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1557–1576. SIAM, 2018.

19 Mark Jerrum and Alistair Sinclair. Polynomial-time approximation algorithms for the Ising model. *SIAM Journal on computing*, 22(5):1087–1116, 1993.

20 Mark R Jerrum, Leslie G Valiant, and Vijay V Vazirani. Random Generation of Combinatorial Structures from a Uniform Distribution. *Theoretical Computer Science*, 43:169–188, 1986.

21 T. D. Lee and C. N. Yang. Statistical theory of equations of state and phase transitions. II. Lattice gas and Ising model. *Phys. Rev.*, 87:410–419, August 1952.

22 Liang Li, Pinyan Lu, and Yitong Yin. Approximate counting via correlation decay in spin systems. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 922–940. SIAM, 2012.

23 Liang Li, Pinyan Lu, and Yitong Yin. Correlation decay up to uniqueness in spin systems. In *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms*, pages 67–84. SIAM, 2013.

24 Elliott H Lieb and Alan D Sokal. A general Lee-Yang theorem for one-component and multicomponent ferromagnets. *Communications in Mathematical Physics*, 80(2):153–179, 1981.

25 Jingcheng Liu. *Approximate counting, phase transitions and geometry of polynomials*. PhD thesis, UC Berkeley, 2019.

26 Jingcheng Liu, Pinyan Lu, and Chihao Zhang. The complexity of ferromagnetic two-spin systems with external fields. *arXiv preprint*, 2014. `arXiv:1402.4346`.

27 Jingcheng Liu, Alistair Sinclair, and Piyush Srivastava. Fisher zeros and correlation decay in the Ising model. *Journal of Mathematical Physics*, 60(10):103304, 2019.

28 Jingcheng Liu, Alistair Sinclair, and Piyush Srivastava. The Ising partition function: zeros and deterministic approximation. *Journal of Statistical Physics*, 174(2):287–315, 2019.

29 Charles M Newman. Zeros of the partition function for generalized Ising systems. *Communications on Pure and Applied Mathematics*, 27(2):143–159, 1974.

30 Viresh Patel and Guus Regts. Deterministic polynomial-time approximation algorithms for partition functions and graph polynomials. *SIAM Journal on Computing*, 46(6):1893–1919, 2017.

31 Han Peters and Guus Regts. Location of zeros for the partition function of the Ising model on bounded degree graphs. *Journal of the London Mathematical Society*, 2018.

32 Han Peters and Guus Regts. On a conjecture of Sokal concerning roots of the independence polynomial. *The Michigan Mathematical Journal*, 68(1):33–55, 2019.

33 Ricardo Restrepo, Jinwoo Shin, Prasad Tetali, Eric Vigoda, and Linji Yang. Improved mixing condition on the grid for counting and sampling independent sets. *Probability Theory and Related Fields*, 156(1-2):75–99, 2013.

34 David Ruelle. Extension of the Lee-Yang circle theorem. *Physical Review Letters*, 26(6):303, 1971.

35 Alexander D Scott and Alan D Sokal. The repulsive lattice gas, the independent-set polynomial, and the Lovász local lemma. *Journal of Statistical Physics*, 118(5-6):1151–1261, 2005.

36 Shuai Shao and Yuxin Sun. Contraction: a Unified Perspective of Correlation Decay and Zero-Freeness of 2-Spin Systems. *arXiv preprint*, 2019. arXiv:1909.04244.

37 Barry Simon and Robert B Griffiths. The $(\phi^4)_2$ field theory as a classical Ising model. *Communications in Mathematical Physics*, 33(2):145–164, 1973.

38 Alistair Sinclair, Piyush Srivastava, Daniel Štefankovič, and Yitong Yin. Spatial mixing and the connective constant: optimal bounds. *Probability Theory and Related Fields*, 168(1-2):153–197, 2017.

39 Alistair Sinclair, Piyush Srivastava, and Marc Thurley. Approximation algorithms for two-state anti-ferromagnetic spin systems on bounded degree graphs. *Journal of Statistical Physics*, 155(4):666–686, 2014.

40 Allan Sly and Nike Sun. Counting in two-spin models on d-regular graphs. *The Annals of Probability*, 42(6):2383–2416, 2014.

41 Alan D Sokal. Bounds on the complex zeros of (di)chromatic polynomials and Potts-model partition functions. *Combinatorics, Probability and Computing*, 10(1):41–77, 2001.

42 Elias M Stein and Rami Shakarchi. *Complex Analysis*, volume 2. Princeton University Press, 2010.

43 Dror Weitz. Counting independent sets up to the tree threshold. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pages 140–149, 2006.

44 Jinshan Zhang, Heng Liang, and Fengshan Bai. Approximating partition functions of two-state spin systems. *arXiv preprint*, 2009. arXiv:0911.5486.

Quasi-Majority Functional Voting on Expander Graphs

Nobutaka Shimizu
The University of Tokyo, Japan
nobutaka_shimizu@mist.i.u-tokyo.ac.jp

Takeharu Shiraga
Chuo University, Tokyo, Japan
shiraga.076@g.chuo-u.ac.jp

Abstract

Consider a distributed graph where each vertex holds one of two distinct opinions. In this paper, we are interested in synchronous *voting processes* where each vertex updates its opinion according to a predefined common local updating rule. For example, each vertex adopts the majority opinion among 1) itself and two randomly picked neighbors in *best-of-two* or 2) three randomly picked neighbors in *best-of-three*. Previous works intensively studied specific rules including best-of-two and best-of-three individually.

In this paper, we generalize and extend previous works of best-of-two and best-of-three on expander graphs by proposing a new model, *quasi-majority functional voting*. This new model contains best-of-two and best-of-three as special cases. We show that, on expander graphs with sufficiently large initial bias, any quasi-majority functional voting reaches consensus within $O(\log n)$ steps with high probability. Moreover, we show that, for any initial opinion configuration, any quasi-majority functional voting on expander graphs with higher expansion (e.g., Erdős-Rényi graph $G(n,p)$ with $p = \Omega(1/\sqrt{n})$) reaches consensus within $O(\log n)$ with high probability. Furthermore, we show that the consensus time is $O(\log n/\log k)$ of best-of-$(2k+1)$ for $k = o(n/\log n)$.

2012 ACM Subject Classification Theory of computation → Random walks and Markov chains; Theory of computation → Distributed algorithms

Keywords and phrases Distributed voting, consensus problem, expander graph, Markov chain

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.97

Category Track A: Algorithms, Complexity and Games

Related Version A full version of the paper is available at `https://arxiv.org/abs/2002.07411`.

Funding *Nobutaka Shimizu*: JSPS KAKENHI Grant Number 19J12876, Japan
Takeharu Shiraga: JSPS KAKENHI Grant Number 19K20214, Japan

1 Introduction

Consider an undirected graph $G = (V, E)$ where each vertex $v \in V$ initially holds an opinion $\sigma \in \Sigma$ from a finite set Σ. In *synchronous voting process* (or simply, *voting process*), in each round, every vertex communicates with its neighbors and then all vertices simultaneously update their opinions according to a predefined protocol. The aim of the protocol is to reach a *consensus configuration*, i.e., a configuration where all vertices have the same opinion. Voting process has been extensively studied in several areas including biology, network analysis, physics and distributed computing [10, 32, 30, 22, 26, 2]. For example, in distributed computing, voting process plays an important role in the consensus problem [22, 26].

This paper is concerned with the *consensus time* of voting processes over *binary* opinions $\Sigma = \{0, 1\}$. Then voting processes have state space 2^V. A state of 2^V is called a *configuration*. The *consensus time* is the number of steps needed to reach a consensus configuration. Henceforth, we are concerned with connected and nonbipartite graphs.

1.1 Previous works of specific updating rules

In *pull voting*, in each round, every vertex adopts the opinion of a randomly selected neighbor. This is one of the most basic voting process, which has been well explored in the past [33, 27, 14, 18, 8]. In particular, the expected consensus time of this process has been extensively studied in the literature. For example, Hassin and Peleg [27] showed that the expected consensus time is $O(n^3 \log n)$ for all non-bipartite graphs and all initial opinion configurations, where n is the number of vertices. From the result of Cooper, Elsässer, Ono, and Radzik [14], it is known that on the complete graph K_n, the expected consensus time is $O(n)$ for any initial opinion configuration.

In *best-of-two* (a.k.a. *2-Choices*), each vertex v samples two random neighbors (with replacement) and, if both hold the same opinion, v adopts the opinion. Otherwise, v keeps its own opinion. Doerr, Goldberg, Minder, Sauerwald, and Scheideler [21] showed that, on the complete graph K_n, the consensus time of best-of-two is $O(\log n)$ with high probability[1] for an arbitrary initial opinion configuration. Since best-of-two is simple and is faster than pull voting on the complete graphs, this model gathers special attention in distributed computing and related area [25, 15, 16, 17, 19, 20, 37]. There is a line of works that study best-of-two on expander graphs [15, 16, 17], which we discuss later.

In *best-of-three* (a.k.a. *3-Majority*), each vertex v randomly selects three random neighbors (with replacement). Then, v updates its opinion to match the majority among the three. It follows directly from Ghaffari and Lengler [25] that, on K_n with any initial opinion configuration, the consensus time of best-of-three is $O(\log n)$ w.h.p. Kang and Rivera [28] considered the consensus time of best-of-three on graphs with large minimum degree starting from a random initial configuration. Shimizu and Shiraga [37] showed that, for any initial configurations, best-of-two and best-of-three reach consensus in $O(\log n)$ steps w.h.p. if the graph is an Erdős-Rényi graph $G(n,p)$[2] of $p = \Omega(1)$.

Best-of-k ($k \geq 1$) is a generalization of pull voting, best-of-two and best-of-three. In each round, every vertex v randomly selects k neighbors (with replacement) and then if at least $\lfloor k/2 \rfloor + 1$ of them have the same opinion, the vertex v adopts it. Note that the best-of-1 is equivalent to pull voting. Abdullah and Draief [1] studied a variant of best-of-k ($k \geq 5$ is odd) on a specific class of sparse graphs that includes n-vertex random d-regular graphs[3] $G_{n,d}$ of $d = o(\sqrt{\log n})$ with a random initial configuration. To the best of our knowledge, best-of-k has not been studied explicitly so far.

In *Majority* (a.k.a. *local majority*), each vertex v updates its opinion to match the majority opinion among the neighbors. This simple model has been extensively studied in previous works [6, 9, 24, 34, 35, 40]. For example, Majority on certain families of graphs including the Erdős-Rényi random graph [6, 40], random regular graphs [24] have been investigated. See [35] for further details.

Voting process on expander graphs

Expander graph gathers special attention in the context of Markov chains on graphs, yielding a wide range of theoretical applications. A graph G is λ-*expander* if $\max\{|\lambda_2|, |\lambda_n|\} \leq \lambda$, where $1 = \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n \geq -1$ are the eigenvalues of the transition matrix P of the

[1] In this paper "with high probability" (w.h.p.) means probability at least $1 - n^{-c}$ for a constant $c > 0$.

[2] Recall that the Erdős-Rényi random graph $G(n,p)$ is a graph on n vertices where each of possible $\binom{n}{2}$ vertex pairs forms an edge with probability p independently.

[3] An n-vertex random d-regular graph $G_{n,d}$ is a graph selected uniformly at random from the set of all labelled n-vertex d-regular graphs.

simple random walk on G. For example, an Erdős-Rényi graph $G(n,p)$ of $p \geq (1+\epsilon)\frac{\log n}{n}$ for an arbitrary constant $\epsilon > 0$ is $O(1/\sqrt{np})$-expander w.h.p. [12]. An n-vertex random d-regular graph $G_{n,d}$ of $3 \leq d \leq n/2$ is $O(1/\sqrt{d})$-expander w.h.p. [13, 39].

Cooper et al. [14] showed that the expected consensus time of pull voting is $O(n/(1-\lambda))$ on λ-expander regular graphs for any initial configuration. Compared to pull voting, the study of best-of-two on general graphs seems much harder. Most of the previous works concerning best-of-two on expander graphs put some assumptions on the initial configuration. Let A denote the set of vertices of opinion 0 and $B = V \setminus A$. Cooper, Elsässer, and Radzik [15] showed that, for any regular λ-expander graph, the consensus time is $O(\log n)$ w.h.p. if $\big||A| - |B|\big| = \Omega(\lambda n)$. This result was improved by Cooper, Elsässer, Radzik, Rivera, and Shiraga [16]. Roughly speaking, they proved that, on λ-expander graphs, the consensus time is $O(\log n)$ if $|d(A) - d(B)| = \Omega(\lambda^2 d(V))$, where $d(S) = \sum_{v \in S} \deg(v)$ denotes the volume of $S \subseteq V$. To the best of our knowledge, the worst case consensus time of best-of-k on expander graphs has not been studied.

1.2 Our model

In this paper, we propose a new class *functional voting* of voting process, which contains many known voting processes as a special case. Let $A \subseteq V$ be the set of vertices of opinion 0 and A' be the set in the next round. Let $B = V \setminus A$ and $B' = V \setminus A'$. For $v \in V$ and $S \subseteq V$, let $N(v) = \{w \in V : \{v,w\} \in E\}$ and $\deg_S(v) = |N(v) \cap S|$.

▶ **Definition 1.1** (Functional voting). *Let $f : \mathbb{R} \to \mathbb{R}$ be a function satisfying $f([0,1]) = [0,1]$ and $f(0) = 0$. A* functional voting with respect to f *is a synchronous voting process defined as*

$$\mathbf{Pr}[v \in A'] = f\left(\frac{\deg_A(v)}{\deg(v)}\right) \quad \text{if } v \in B,$$
$$\mathbf{Pr}[v \in B'] = f\left(\frac{\deg_B(v)}{\deg(v)}\right) \quad \text{if } v \in A.$$

We call the function f a betrayal function *and the function*

$$H_f(x) := x\big(1 - f(1-x)\big) + (1-x)f(x)$$

an updating function.

Since $f(0) = 0$, consensus configurations are absorbing states. The intuition behind the updating function H_f is that, letting $\alpha = |A|/n$ and $\alpha' = |A'|/n$, on a complete graph K_n (with self-loop), the functional voting with respect to f satisfies $\mathbf{E}[\alpha'] = \frac{|A|}{n}\left(1 - f\left(\frac{|B|}{n}\right)\right) + \frac{|B|}{n} f\left(\frac{|A|}{n}\right) = H_f(\alpha)$.

Functional voting contains many existing models as special cases. For example, pull voting, best-of-two, and best-of-three are functional votings with respect to x, x^2 and $3x^2 - 2x^3$, respectively. In general, best-of-k is a functional voting with respect to

$$f_k(x) = \sum_{i=\lfloor k/2 \rfloor + 1}^{k} \binom{k}{i} x^i (1-x)^{k-i}. \qquad (1)$$

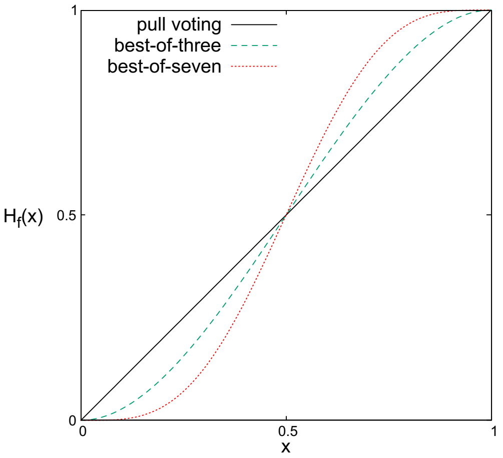


Figure 1 The updating functions $H_f(x)$ of pull voting (solid line), best-of-three (dashed line) and best-of-seven (dotted line). One can easily observe that best-of-three and best-of-seven are quasi-majority functional voting. Intuitively speaking, quasi-majority functional voting has an updating function H_f with the property so-called "the rich get richer", which coincides with Definition 1.2.

It is straightforward to check that $H_{f_k}(x) = f_k(x)$ if k is odd and $H_{f_k}(x) = f_{k+1}(x)$ if k is even. Majority is a functional voting with respect to

$$f(x) = \begin{cases} 0 & \text{if } x < \frac{1}{2}, \\ \frac{1}{2} & \text{if } x = \frac{1}{2}, \\ 1 & \text{if } x > \frac{1}{2} \end{cases} \tag{2}$$

if a vertex adopts the random opinion when it meets the tie.

Quasi-majority functional voting

In this paper, we focus on functional voting with respect to f satisfying the following property.

▶ **Definition 1.2** (Quasi-majority). *A function f is* quasi-majority *if f satisfies the following conditions.*

(i) *f is C^2 (i.e., the derivatives f' and f'' exist and they are continuous).*
(ii) $0 < f(1/2) < 1$,
(iii) *$H_f(x) < x$ whenever $x \in (0, 1/2)$.*
(iv) $H'_f(1/2) > 1$,
(v) $H'_f(0) < 1$.

A voting process is a quasi-majority functional voting *if it is a functional voting with respect to a quasi-majority function f.*

Note that $H_f(x)$ is symmetric (i.e., $H_f(1-x) = 1 - H_f(x)$) and thus the condition (iii) implies $H_f(x) > x$ for every $x \in (1/2, 1)$. Intuitively, the conditions (iii) to (v) ensure the drift towards consensus. The conditions (i) and (ii) are due to a technical reasons.

For each constant $k \geq 2$, best-of-k is quasi-majority functional voting but pull voting and Majority are not. Indeed, if H_{f_k} is the updating function of best-of-k, then $H'_{f_{2\ell}}(x) = H'_{f_{2\ell+1}}(x) = (2\ell+1)\binom{2\ell}{\ell}x^\ell(1-x)^\ell$. It is straightforward to check that this function satisfies the conditions (iii) to (v) if $\ell \neq 0$ (pull-voting). See Figure 1 for depiction of the updating functions of pull voting, best-of-three and best-of-seven.

1.3 Our result

In this paper, we study the consensus time of quasi-majority functional voting on expander graphs[4]. Let $T_{\mathrm{cons}}(A)$ denote the consensus time starting from the initial configuration $A \subseteq V$. For a graph $G = (V, E)$, let $\pi = (\pi(v))_{v\in V}$ denote the *degree distribution* defined as

$$\pi(v) = \frac{\deg(v)}{2|E|}. \tag{3}$$

Note that $\sum_{v\in V} \pi(v) = 1$ holds. We denote by $\|x\|_p := \left(\sum_{v\in V} |x_v|^p\right)^{1/p}$ the ℓ^p norm of $x \in \mathbb{R}^V$. For $\pi \in [0,1]^V$ and $A \subseteq V$, let $\pi(A) := \sum_{v\in A} \pi(v)$. Let

$$\delta(A) := \pi(A) - \pi(V \setminus A) = 2\pi(A) - 1$$

denote the *bias* between A and $V \setminus A$.

▶ **Theorem 1.3** (Main theorem). *Consider a quasi-majority functional voting with respect to f on an n-vertex λ-expander graph with degree distribution π. Then, the following holds:*

(i) *Let $C_1 > 0$ be an arbitrary constant and $\varepsilon : \mathbb{N} \to \mathbb{R}$ be an arbitrary function satisfying $\varepsilon(n) \to 0$ as $n \to \infty$. Suppose that $\lambda \le C_1 n^{-1/4}$, $\|\pi\|_2 \le C_1/\sqrt{n}$ and $\|\pi\|_3 \le \varepsilon/\sqrt{n}$. Then, for any $A \subseteq V$, $T_{\mathrm{cons}}(A) = O(\log n)$ w.h.p.*

(ii) *Let C_2 be a positive constant depending only on f. Suppose that $\lambda \le C_2$ and $\|\pi\|_2 \le C_2/\sqrt{\log n}$. Then, for any $A \subseteq V$ satisfying $|\delta(A)| \ge C_2 \max\{\lambda^2, \|\pi\|_2\sqrt{\log n}\}$, $T_{\mathrm{cons}}(A) = O(\log n)$ w.h.p.*

The following result indicates that the consensus time of Theorem 1.3(i) is optimal up to a constant factor.

▶ **Theorem 1.4** (Lower bound). *Under the same assumption of Theorem 1.3(i), $T_{\mathrm{cons}}(A) = \Omega(\log n)$ w.h.p. for some $A \subseteq V$.*

See the full version [38] for the proof of Theorem 1.4.

▶ **Theorem 1.5** (Fast consensus for $H'_f(0) = 0$). *Consider a quasi-majority functional voting with respect to f on an n-vertex λ-expander graph with degree distribution π. Let $C > 0$ be a constant depending only on f. Suppose that $H'_f(0) = 0$, $\lambda \le C$ and $\|\pi\|_2 \le C/\sqrt{\log n}$. Then, for any $A \subseteq V$ satisfying $|\delta(A)| \ge C \max\{\lambda^2, \|\pi\|_2\sqrt{\log n}\}$, it holds w.h.p. that*

$$T_{\mathrm{cons}}(A) = O\left(\log\log n + \log|\delta(A)|^{-1} + \frac{\log n}{\log \lambda^{-1}} + \frac{\log n}{\log(|\pi\|_2\sqrt{\log n})^{-1}}\right).$$

For example, for each constant $k \ge 2$, best-of-k is quasi-majority with $H'_f(0) = 0$.

▶ Remark 1.6. Roughly speaking, for $p \ge 2$, $\|\pi\|_p$ measures the imbalance of the degrees. For any graphs, $\|\pi\|_p \ge n^{-1+1/p}$ and the equality holds if and only if the graph is regular. For star graphs, we have $\|\pi\|_p \approx 1$.

Results of best-of-k

Our results above do not explore Majority since it is not quasi-majority. A plausible approach is to consider best-of-k for $k = k(n) = \omega(1)$ since each vertex is likely to choose the majority opinion if the number of neighbor sampling increases. Also, note that the betrayal function f_k

[4] Throughout the paper, we consider sufficiently large $n = |V|$.

of best-of-k given in (1) converges to that of Majority (i.e., $f_k(x) \to f(x)$ as $k \to \infty$ for each $x \in [0,1]$, where f is the betrayal function (2) of Majority). On the other hand, if $k = O(1)$, there is a tremendous gap between best-of-k and Majority: For any functional voting on the complete graph K_n, $T_{\mathrm{cons}}(A) = \Omega(\log n)$ for some $A \subseteq V$ from Theorem 1.4. Majority on K_n reaches the consensus in a single step if $|A| < |V \setminus A| - 1$. This motivates us to consider best-of-k for $k = k(n) \to \infty$ as $n \to \infty$. For simplicity, we focus on best-of-$(2k+1)$ and prove the following result (see the full version [38] for the proof).

▶ **Theorem 1.7.** *Let $k = k(n)$ be such that $k = \omega(1)$ and $k = o(n/\log n)$. Let C be an arbitrary positive constant. Consider best-of-$(2k+1)$ on an n-vertex λ-expander graph with degree distribution π such that $\lambda \le Ck^{-1/2}n^{-1/4}$, $\|\pi\|_2 \le Cn^{-1/2}$ and $\|\pi\|_3 \le Ck^{-1/6}n^{-1/2}$. Then, $T_{\mathrm{cons}}(A) = O\left(\frac{\log n}{\log k}\right)$ holds w.h.p. for any $A \subseteq V$.*

1.4 Application

Here, we apply our main theorem to specific graphs and derive some useful results.

For any $p \ge (1+\epsilon)\frac{\log n}{n}$ for an arbitrary constant $\epsilon > 0$, $G(n,p)$ is connected and $O(1/\sqrt{np})$-expander w.h.p [12, 23].

▶ **Corollary 1.8.** *Consider a best-of-k on an Erdős-Rényi graph $G(n,p)$ for an arbitrary constant $k \ge 2$. Then, $G(n,p)$ w.h.p. satisfies the following:*

(i) *Suppose that $p = \Omega(n^{-1/2})$. Then*
  - **(a)** *for any $A \subseteq V$, $T_{\mathrm{cons}}(A) = O(\log n)$ w.h.p.*
  - **(b)** *for some $A \subseteq V$, $T_{\mathrm{cons}}(A) = \Omega(\log n)$ w.h.p.*

(ii) *Suppose that $p \ge (1+\epsilon)\frac{\log n}{n}$ for an arbitrary constant $\epsilon > 0$. Then, for any $A \subseteq V$ satisfying $|\delta(A)| \ge C\max\left\{\frac{1}{np}, \sqrt{\frac{\log n}{n}}\right\}$, $T_{\mathrm{cons}}(A) = O\left(\log\log n + \log|\delta(A)|^{-1} + \frac{\log n}{\log(np)}\right)$ w.h.p., where $C > 0$ is a constant depending only on f.*

In Corollary 1.8(i), we stress that the worst-case consensus time on $G(n,p)$ was known for $p = \Omega(1)$ [37]. If $\frac{\log n}{\log(np)} = O(\log\log n)$ (or equivalently, $np = n^{\Omega(1/\log\log n)}$), Corollary 1.8(ii) implies $T_{\mathrm{cons}}(A) = O(\log\log n + \log|\delta(A)|^{-1})$ w.h.p.

▶ **Corollary 1.9.** *Let $k = k(n)$ be such that $k = \omega(1)$ and $k = O(\sqrt{n})$. Consider best-of-$(2k+1)$ on $G(n,p)$ for $p = \Omega(k/\sqrt{n})$. Then, for any $A \subseteq V$, $T_{\mathrm{cons}}(A) = O\left(\frac{\log n}{\log k}\right)$ holds w.h.p.*

From Corollary 1.9, best-of-n^ϵ on $G(n, n^{-1/2+\epsilon})$ for any constant $\epsilon \in (0, 1/2)$ reaches consensus in $O(1)$ steps. It is known that Majority on $G(n, Cn^{-1/2})$ satisfies $T_{\mathrm{cons}}(A) \le 4$ for large constant C and random $A \subseteq V$ with constant probability [6].

For $3 \le d \le n/2$, n-vertex random d-regular graph $G_{n,d}$ is connected and $O(1/\sqrt{d})$-expander w.h.p. [13, 39].

▶ **Corollary 1.10.** *Consider a best-of-k on an n-vertex random d-regular graph $G_{n,d}$ for an arbitrary constant $k \ge 2$. Then, $G_{n,d}$ w.h.p. satisfies the following:*

(i) *Suppose that $d = \Omega(n^{1/2})$ and $d \le n/2$. Then,*
  - **(a)** *for any $A \subseteq V$, $T_{\mathrm{cons}}(A) = O(\log n)$ w.h.p.*
  - **(b)** *for some $A \subseteq V$, $T_{\mathrm{cons}}(A) = \Omega(\log n)$ w.h.p.*

(ii) *Suppose that $d \ge C$ and $d \le n/2$ for a constant $C > 0$ depending only on f. Then, for any $A \subseteq V$ satisfying $|\delta(A)| \ge C\max\left\{\frac{1}{d}, \sqrt{\frac{\log n}{n}}\right\}$, it holds w.h.p. that $T_{\mathrm{cons}}(A) = O\left(\log\log n + \log|\delta(A)|^{-1} + \frac{\log n}{\log d}\right)$.*

▶ **Corollary 1.11.** *Let $k = k(n)$ be such that $k = \omega(1)$ and $k = O(\sqrt{n})$. Consider best-of-$(2k+1)$ on an n-vertex random d-regular graph $G_{n,d}$ such that $d = \Omega(k\sqrt{n})$ and $d \le n/2$. Then, for any $A \subseteq V$, $T_{\mathrm{cons}}(A) = O\left(\frac{\log n}{\log k}\right)$ holds w.h.p.*

See the full version [38] for other specific results and examples of quasi-majority functional voting.

1.5 Related work

In asynchronous voting process, in each round, a vertex is selected uniformly at random and only the selected vertex updates its opinion. Cooper and Rivera [18] introduced *linear voting model*. In this model, an opinion configuration is represented as a vector $v \in \Sigma^V$ and the vector v updates according to the rule $v \leftarrow Mv$, where M is a random matrix sampled from some probability space. This model captures a wide variety model including asynchronous push/pull voting and synchronous pull voting. Note that best-of-two and best-of-three are not included in linear voting model. Schoenebeck and Yu [36] proposed an asynchronous variant of our functional voting. The authors of [36] proved that, if the function f is symmetric (i.e., $f(1-x) = 1 - f(x)$), smooth and has "majority-like" property (i.e., $f(x) > x$ whenever $1/2 < x < 1$), then the expected consensus time is $O(n \log n)$ w.h.p. on $G(n,p)$ with $p = \Omega(1)$. This perspective has also been investigated in physics (see, e.g., [10]).

Several researchers have studied best-of-two and best-of-three on complete graphs initially involving $k \ge 2$ opinions [5, 4, 7, 25]. For example, the consensus time of best-of-three is $O(k \log n)$ if $k = O(n^{1/3}/\sqrt{\log n})$ [25]. Cooper, Radzik, Rivera, and Shiraga [17] considered best-of-two and best-of-three on regular expander graphs that hold more than two opinions.

Recently, Cruciani, Natale, and Scornavacca [20] studied best-of-two with a random initial configuration on a clustered regular graph. Shimizu and Shiraga [37] obtained phase-transition results of best-of-two and best-of-three on stochastic block models.

2 Preliminary and technical result

2.1 Formal definition

Let $G = (V, E)$ be an undirected and connected graph. Let $P \in [0,1]^{V \times V}$ be the matrix defined as

$$P(u,v) := \frac{\mathbb{1}_{\{u,v\} \in E}}{\deg(u)} \quad \forall (u,v) \in V \times V \tag{4}$$

where $\mathbb{1}_Z$ denotes the indicator of an event Z. For $v \in V$ and $S \subseteq V$, we write $P(v,S) = \sum_{s \in S} P(v,s)$.

Now, let us describe the formal definition of functional voting. For a given $A \subseteq V$, let $(X_v)_{v \in V}$ be independent binary random variables defined as

$$\begin{aligned} \mathbf{Pr}[X_v = 1] &= f\big(P(v,A)\big) \quad \text{if } v \in B, \\ \mathbf{Pr}[X_v = 0] &= f\big(P(v,B)\big) \quad \text{if } v \in A, \end{aligned} \tag{5}$$

where $B = V \setminus A$. For $A \subseteq V$ and (X_v) above, define $A' = \{v \in V : X_v = 1\}$. Note that this definition coincides with Definition 1.1 since $P(v,A) = \frac{\deg_A(v)}{\deg(v)}$. Then, a functional voting is a Markov chain $A_0, A_1, \ldots$ where $A_{t+1} = (A_t)'$.

For $A \subseteq V$, let $T_{\mathrm{cons}}(A)$ denote the consensus time of the functional voting starting from the initial configuration A. Formally, $T_{\mathrm{cons}}(A)$ is the stopping time defined as

$$T_{\mathrm{cons}}(A) := \min\{t \ge 0 : A_t \in \{\emptyset, V\}, A_0 = A\}.$$

2.2 Technical background

Consider best-of-two on a complete graph K_n (with self loop on each vertex) with a current configuration $A \subseteq V$. Let $\alpha = |A|/n$. We have $P(v, A) = \alpha$ for any $v \in V$ and $A \subseteq V$. Then, for any $A \subseteq V$, $\mathbf{E}[\alpha'] = H_f(\alpha) = 3\alpha^2 - 2\alpha^3$. Thus, in each round, $\alpha' = 3\alpha^2 - 2\alpha^3 \pm O(\sqrt{\log n/n})$ holds w.h.p. from the Hoeffding bound. Therefore, the behavior of α can be written as the iteration of applying H_f.

The most technical part is the symmetry breaking at $\alpha = 1/2$. Note that $H_f(1/2) = 1/2$ and thus, the argument above does not work in the case of $|\alpha - 1/2| = o(\sqrt{\log n/n})$. To analyze this case, the authors of [21, 11] proved the following technical lemma asserting that α w.h.p. escapes from the area in $O(\log n)$ rounds.

▶ **Lemma 2.1** (Lemma 4.5 of [11] (informal)). *For any constant C, it holds w.h.p. that $|\alpha - 1/2| \geq C\sqrt{\log n/n}$ in $O(\log n)$ rounds (the hidden constant factor depends on C) if*

(i) *For any constant h, there is a constant $C_0 > 0$ such that, if $|\alpha - 1/2| = O(\sqrt{\log n/n})$ then $\mathbf{Pr}[|\alpha' - 1/2| > h/\sqrt{n}] > C_0$.*

(ii) *If $|\alpha - 1/2| = O(\sqrt{\log n/n})$ and $|\alpha - 1/2| = \Omega(1/\sqrt{n})$, $\mathbf{Pr}[|\alpha' - 1/2| \leq (1+\epsilon)|\alpha - 1/2|] \leq \exp(-\Theta((\alpha - 1/2)^2 n))$ for some constant $\epsilon > 0$.*

Intuitively speaking, the condition (ii) means that the bias $|\alpha' - 1/2|$ is likely to be at least $(1 + \epsilon)|\alpha - 1/2|$ for some constant $\epsilon > 0$. The condition (ii) is easy to check using the Hoeffding bound. The condition (i) means that α' has a fluctuation of size $\Omega(1/\sqrt{n})$ with a constant probability. We can check condition (i) using the Central Limit Theorem (the Berry-Esseen bound). The Central Limit Theorem implies that the normalized random variable $(\alpha' - \mathbf{E}[\alpha'])/\sqrt{\mathbf{Var}[\alpha']}$ converges to the standard normal distribution as $n \to \infty$. In other words, α' has a fluctuation of size $\Theta(\sqrt{\mathbf{Var}[\alpha']})$ with constant probability. Now, to verify the condition (i), we evaluate $\mathbf{Var}[\alpha']$. On K_n, it is easy to show that $\mathbf{Var}[\alpha'] = \Theta(1/n)$, which implies the condition (i).

The authors of [16, 17] considered best-of-two on expander graphs. They focused on the behavior of $\pi(A)$ instead of α. Roughly speaking, they proved that $\mathbf{E}[\pi(A') - 1/2] \geq (1 + \epsilon)(\pi(A) - 1/2) - O(\lambda^2)$. At the heart of the proof, they showed the following result.

▶ **Lemma 2.2** (Special case of Lemma 3 of [17]). *Consider a λ-expander graph with degree distribution π. Then, for any $S \subseteq V$, $\left|\sum_{v\in V} \pi(v)P(v, S)^2 - \pi(S)^2\right| \leq \lambda^2\pi(S)\big(1 - \pi(S)\big)$.*

Then, from the Hoeffding bound, we have $\mathbf{E}[\pi(A') - 1/2] \geq (1 + \epsilon)(\pi(A) - 1/2) - O(\lambda^2 + \|\pi\|_2\sqrt{\log n}))$. Thus, if the initial bias $|\pi(A) - 1/2|$ is $\Omega(\max\{\lambda^2, \sqrt{\log n/n}\})$, we can show that the consensus time is $O(\log n)$.

Unfortunately, we can not apply the same technique to estimate $\mathbf{Var}[\pi(A')]$ on expander graphs, and due to this reason, it seems difficult to estimate the worst-case consensus time on expander graphs. Actually, any previous works put assumptions on the initial bias due to the same reason. It should be noted that Lemma 2.1 is well-known in the literature. For example, Cruciani et al. [20] used Lemma 2.1 from random initial configurations.

The technique of estimating $\mathbf{E}[\pi(A')]$ by Cooper et al. [16, 17] is specialized in best-of-two. Thus, it is not straightforward to prove the estimation of $\mathbf{E}[\pi(A')]$ for voting processes other than best-of-two.

2.3 Our technical contribution

For simplicity, in this part, we focus on a quasi-majority functional voting with respect to a *symmetric* function f (i.e., $f(1 - x) = 1 - f(x)$ for every $x \in [0, 1]$) on a λ-expander graph with degree distribution π. For example, $f(x) = 3x^2 - 2x^3$ of best-of-three is a symmetric function. Note that $f = H_f$ if f is symmetric. Similar results mentioned in this subsection holds for non-symmetric f (see Lemma 3.5 and 3.6 of the full version [38]). For a C^2 function $h : \mathbb{R} \to \mathbb{R}$, let

$$K_1(h) := \max_{x\in[0,1]} |h'(x)|\,, \quad K_2(h) := \max_{x\in[0,1]} |h''(x)|$$

be constants[5] The following technical result enables us to estimate $\mathbf{E}[\pi(A')]$ and $\mathbf{Var}[\pi(A')]$ of functional voting.

▶ **Lemma 2.3.** *Consider a functional voting with respect to a symmetric C^2 function f on a λ-expander graph with degree distribution π. Let $g(x) := f(x)(1-f(x))$. Then, for all $A \subseteq V$,*

$$\left|\mathbf{E}[\pi(A')] - H_f(\pi(A))\right| \le \frac{K_2(f)}{2}\lambda^2\pi(A)\bigl(1-\pi(A)\bigr),$$
$$\left|\mathbf{Var}[\pi(A')] - \|\pi\|_2^2 g\bigl(\pi(A)\bigr)\right| \le K_1(g)\lambda\sqrt{\pi(A)\bigl(1-\pi(A)\bigr)}\|\pi\|_3^{3/2}.$$

Note that, if f is symmetric, the corresponding functional voting satisfies that $\mathbf{Pr}[v \in A'] = f(P(v,A))$ for any $v \in V$. Thus we have

$$\mathbf{E}[\pi(A')] = \sum_{v\in V}\pi(v)f\bigl(P(v,A)\bigr), \quad \mathbf{Var}[\pi(A')] = \sum_{v\in V}\pi(v)^2 g\bigl(P(v,A)\bigr).$$

To evaluate $\mathbf{E}[\pi(A')]$ and $\mathbf{Var}[\pi(A')]$ above, we prove the following key lemma that is a generalization of Lemma 2.2 and implies Lemma 2.3.

▶ **Lemma 2.4** (Special case of Lemmas 3.2 and 3.3)**.** *Consider a λ-expander graph with degree distribution π. Then, for any $S \subseteq V$ and any C^2 function $h : \mathbb{R} \to \mathbb{R}$,*

$$\left|\sum_{v\in V}\pi(v)h\bigl(P(v,S)\bigr) - h\bigl(\pi(S)\bigr)\right| \le \frac{K_2(h)}{2}\lambda^2\pi(S)\bigl(1-\pi(S)\bigr),$$
$$\left|\sum_{v\in V}\pi(v)^2 h\bigl(P(v,S)\bigr) - \|\pi\|_2^2 h\bigl(\pi(S)\bigr)\right| \le K_1(h)\lambda\sqrt{\pi(S)\bigl(1-\pi(S)\bigr)}\|\pi\|_3^{3/2}.$$

Non-symmetric functions

For general f, we prove the following.

▶ **Lemma 2.5.** *Consider a functional voting with respect to a C^2 function f on a λ-expander graph. Let $g(x) := f(x)(1-f(x))$. Then, for all $A \subseteq V$,*

$$\left|\mathbf{E}[\pi(A')] - H_f\bigl(\pi(A)\bigr)\right| \le K_2(f)\lambda\bigl(|2\pi(A)-1| + \lambda\bigr)\pi(A)\bigl(1-\pi(A)\bigr),$$
$$\left|\mathbf{Var}[\pi(A')] - \|\pi\|_2^2 g\left(\frac{1}{2}\right)\right| \le K_1(g)\left(\frac{1}{2}\|\pi\|_2^2\,|2\pi(A)-1| + 2\|\pi\|_3^{3/2}\lambda\sqrt{\pi(A)\bigl(1-\pi(A)\bigr)}\right).$$

We refer the proof of Lemma 2.5 to the full-version [38] due to the page limitation.

[5] For example, for $f(x) = 3x^2 - 2x^3$ of best-of-three, $f''(x) = 6 - 12x$ and $K_2(f) = 6$. It should be noted that we deal with f not depending on G except for best-of-k with $k = \omega(1)$.

2.4 Proof sketch of Theorem 1.3

We present proof sketch of Theorem 1.3(i). From the assumption of Theorem 1.3(i) and Lemma 2.3, if $|\pi(A) - 1/2| = o(1)$, we have $\mathbf{Var}[\pi(A')] = \Theta(\|\pi\|_2^2 g(\pi(A))) = \Theta(\|\pi\|_2^2 g(1/2 + o(1))) = \Theta(1/n)$. Moreover, $\mathbf{E}[\pi(A')] = H_f(\pi(A)) \pm O(\pi(A)/\sqrt{n})$ holds for any $A \subseteq V$. Hence, from the Hoeffding bound, $\pi(A') = H_f(\pi(A)) + O(\sqrt{\log n/n})$ holds w.h.p. for any $A \subseteq V$.

- If $|\pi(A) - 1/2| = O(\sqrt{\log n/n})$, we use Lemma 2.1 to obtain an $O(\log n)$ round symmetry breaking. In this phase, since $|\pi(A) - 1/2| = o(1)$, $\mathbf{Var}[\pi(A') - 1/2] = \Theta(1/n)$. Then, from the Berry-Esseen bound, we can check the condition (i). To check the condition (ii), we invoke the condition $H_f'(1/2) > 1$ of the quasi-majority function. From Taylor's theorem and the assumption of Lemma 2.1(ii) ($\pi(A) - 1/2 = \Omega(1/\sqrt{n})$), $\mathbf{E}[\pi(A') - 1/2] = H_f(\pi(A)) - H_f(1/2) - O(1/\sqrt{n}) \approx (1 + \epsilon_1)(\pi(A) - 1/2)$ for some positive constant $\epsilon_1 > 0$. Note that $H_f(1/2) = 1/2$.
- If $C_1\sqrt{\log n/n} \leq |\pi(A) - 1/2| \leq C_2$ for sufficiently large constant C_1 and some constant $C_2 > 0$, we use the Hoeffding bound and then obtain $\pi(A') - 1/2 \approx (1 + \epsilon_1)(\pi(A) - 1/2) - O(\sqrt{\log n/n}) \geq (1 + (\epsilon_1/2))(\pi(A) - 1/2)$ w.h.p. Hence, $O(\log n)$ rounds suffice to yield a constant bias. (Note that this argument holds when $|\pi(A) - 1/2| \leq C_2$ due to the remainder term of Taylor's theorem.)
- If $C_3 \leq \pi(A) < 1/2$, it is straightforward to see that $\pi(A') = H_f(\pi(A)) + O(\sqrt{\log n/n}) \leq \pi(A) - \epsilon_2$ w.h.p. for some constant $\epsilon_2 > 0$. Note that we invoke the property that $H_f(x) < x$ whenever $0 < x < 1/2$.
- If $\pi(A) \leq C_3$ for sufficiently small constant C_3, we use the Markov inequality to show $\pi(A_t) = O(n^{-3})$ w.h.p. for some $t = O(\log n)$. Since $\pi(A) \geq 1/n^2$ whenever $A \neq \emptyset$, this implies that the consensus time is $O(\log n)$ w.h.p. Note that, since $H_f'(0) < 1$, we have $\mathbf{E}[\pi(A')] \leq H_f(\pi(A)) + O(\pi(A)/\sqrt{n}) \approx H_f'(0)\pi(A) + O(\pi(A)/\sqrt{n}) \leq (1 - \epsilon_3)\pi(A)$ for some constant $\epsilon_3 > 0$.

In the proof of Theorem 1.7, we modify Lemma 2.1 and apply the same argument.

3 Reversible Markov chains and Proof of Lemma 2.4

In this section, we prove Lemma 2.4 by showing Lemmas 3.2 and 3.3, which are generalizations of Lemma 2.4 in terms of *reversible Markov chain*. This enables us to evaluate $\mathbf{E}[\pi(A')]$ and $\mathbf{Var}[\pi(A')]$ for functional voting with respect to a C^2 function f (see the full version [38] for functional voting with respect to non-symmetric f).

3.1 Technical tools for reversible Markov chains

To begin with, we briefly summarize the notation of Markov chain, which we will use in this section[6]. Let V be a set of size n. A *transition matrix* P over V is a matrix $P \in [0,1]^{V \times V}$ satisfying $\sum_{v \in V} P(u,v) = 1$ for any $u \in V$. Let $\pi \in [0,1]^V$ denote the *stationary distribution* of P, i.e., a probability distribution satisfying $\pi P = \pi$. A transition matrix P is *reversible* if $\pi(u)P(u,v) = \pi(v)P(v,u)$ for any $u, v \in V$. It is easy to check that the matrix (4) is

[6] For further detailed arguments about reversible Markov chains, see e.g., [29].

a reversible transition matrix and its stationary distribution is (3). Let $\lambda_1 \geq \cdots \geq \lambda_n$ denote the eigenvalues of P. If P is reversible, it is known that $\lambda_i \in \mathbb{R}$ for all i. Let $\lambda = \max\{|\lambda_2|, |\lambda_n|\}$ be the second largest eigenvalue in absolute value[7].

For a function $h : \mathbb{R} \to \mathbb{R}$ and subsets $S, T \subseteq V$, consider the quantity $Q_h(S,T)$ defined as

$$Q_h(S,T) := \sum_{v \in S} \pi(v) h\bigl(P(v,T)\bigr). \tag{6}$$

The special case of $h(x) = x$, that is, $Q(S,T) := \sum_{v \in S} \pi(v) P(v,T)$, is well known as *edge measure* [29] or *ergodic flow* [3, 31]. Note that, for any reversible P and subsets $S, T \subseteq V$, $Q(S,T) = Q(T,S)$ holds. The following result is well known as a version of the *expander mixing lemma*.

▶ **Lemma 3.1** (See, e.g., p.163 of [29]). *Suppose P is reversible. Then, for any $S, T \subseteq V$,*

$$|Q(S,T) - \pi(S)\pi(T)| \leq \lambda \sqrt{\pi(S)\pi(T)\bigl(1 - \pi(S)\bigr)\bigl(1 - \pi(T)\bigr)}.$$

We show the following lemma which gives a useful estimation of $Q_h(S,T)$.

▶ **Lemma 3.2.** *Suppose P is reversible. Then, for any $S, T \subseteq V$ and any C^2 function $h : \mathbb{R} \to \mathbb{R}$,*

$$\Bigl| Q_h(S,T) - \pi(S) h\bigl(\pi(T)\bigr) - h'\bigl(\pi(T)\bigr)\bigl(Q(S,T) - \pi(S)\pi(T)\bigr) \Bigr| \leq \frac{K_2(h)}{2} \lambda^2 \pi(T)\bigl(1 - \pi(T)\bigr).$$

Proof of Lemma 3.2. From Taylor's theorem, it holds for any $x, y \in [0,1]$ that

$$|h(x) - h(y) - h'(y)(x-y)| \leq \frac{K_2(h)}{2}(x-y)^2.$$

Hence

$$\begin{aligned}
&\Bigl| Q_h(S,T) - \pi(S) h\bigl(\pi(T)\bigr) - h'\bigl(\pi(T)\bigr)\bigl(Q(S,T) - \pi(S)\pi(T)\bigr) \Bigr| \\
&= \left| \sum_{v \in S} \pi(v) \Bigl(h\bigl(P(v,T)\bigr) - h\bigl(\pi(T)\bigr) - h'\bigl(\pi(T)\bigr)\bigl(P(v,T) - \pi(T)\bigr) \Bigr) \right| \\
&\leq \sum_{v \in S} \pi(v) \Bigl| h\bigl(P(v,T)\bigr) - h\bigl(\pi(T)\bigr) - h'\bigl(\pi(T)\bigr)\bigl(P(v,T) - \pi(T)\bigr) \Bigr| \\
&\leq \sum_{v \in S} \pi(v) \frac{K_2(h)}{2} \bigl(P(v,T) - \pi(T)\bigr)^2 \leq \frac{K_2(h)}{2} \sum_{v \in V} \pi(v) \bigl(P(v,T) - \pi(T)\bigr)^2 \\
&\leq \frac{K_2(h)}{2} \lambda^2 \pi(T)\bigl(1 - \pi(T)\bigr).
\end{aligned}$$

The last inequality follows from Corollary A.2 of the full version [38]. ◀

Next, consider

$$R_h(S,T) := \sum_{v \in S} \pi(v)^2 h\bigl(P(v,T)\bigr) \tag{7}$$

for a function $h : \mathbb{R} \to \mathbb{R}$ and $S, T \subseteq V$. For notational convenience, for $S \subseteq V$, let $\pi_2(S) := \sum_{v \in S} \pi(v)^2$. We show the following lemma that evaluates $R_h(S,T)$.

[7] If P is *ergodic*, i.e., for any $u, v \in V$, there exists a $t > 0$ such that $P^t(u,v) > 0$ and $\mathrm{GCD}\{t > 0 : P^t(x,x) > 0\} = 1$, $1 > \lambda_2$ and $\lambda_n > -1$. For example, the transition matrix of the simple random walk on a connected and non-bipartite graph is ergodic.

▶ **Lemma 3.3.** *Suppose that P is reversible. Then, for any $S, T \subseteq V$ and any C^2 function $h : \mathbb{R} \to \mathbb{R}$,*

$$\left|R_h(S,T) - \pi_2(S)h\bigl(\pi(T)\bigr)\right| \le K_1(h)\|\pi\|_3^{3/2}\lambda\sqrt{\pi(T)\bigl(1-\pi(T)\bigr)}.$$

Proof. We first observe that

$$\bigl|h(x) - h(y)\bigr| \le K_1(h)|x-y| \tag{8}$$

holds for any $x, y \in [0,1]$ from Taylor's theorem. Hence,

$$\begin{aligned}&\left|R_h(S,T) - \pi_2(S)h\bigl(\pi(T)\bigr)\right|\\ &= \left|\sum_{v\in S}\pi(v)^2\Bigl(h\bigl(P(v,T)\bigr) - h\bigl(\pi(T)\bigr)\Bigr)\right| \le \sum_{v\in S}\pi(v)^2\left|h\bigl(P(v,T)\bigr) - h\bigl(\pi(T)\bigr)\right|\\ &\le \sum_{v\in S}\pi(v)^2K_1(h)\bigl|P(v,T)-\pi(T)\bigr| \le K_1(h)\sum_{v\in V}\pi(v)^2\bigl|P(v,T)-\pi(T)\bigr|.\end{aligned}$$

Then, applying the Cauchy-Schwarz inequality and Corollary A.2 of the full version [38],

$$\begin{aligned}\sum_{v\in V}\pi(v)^2\bigl|P(v,T)-\pi(T)\bigr| &\le \sqrt{\left(\sum_{v\in V}\pi(v)^3\right)\left(\sum_{v\in V}\pi(v)\bigl(P(v,T)-\pi(T)\bigr)^2\right)}\\ &\le \|\pi\|_3^{3/2}\lambda\sqrt{\pi(T)\bigl(1-\pi(T)\bigr)}\end{aligned}$$

and we obtain the claim. ◀

▶ Remark 3.4. The results of this paper can be extended to voting processes where the sampling probability is determined by a reversible transition matrix P. This includes voting processes on edge-weighted graphs $G = (V, E, w)$, where $w : E \to \mathbb{R}$ denotes an edge weight function. Consider the transition matrix P defined as follows: $P(u,v) = w(\{u,v\})/\sum_{x:\{u,x\}\in E} w(\{u,x\})$ for $\{u,v\} \in E$ and $P(u,v) = 0$ for $\{u,v\} \notin E$. A *weighted* functional voting with respect to f is determined by $\mathbf{Pr}[v \in A'|v \in B] = f(P(v,B))$ and $\mathbf{Pr}[v \in B'|v \in A] = f(P(v,A))$. For simplicity, in this paper, we do not explore the weighted variant and focus on the usual setting where P is the matrix (4) and its stationary distribution π is (3).

3.2 Proof of Lemma 2.4

For the first inequality, by substituting V to S of Lemma 3.2, we obtain $\Bigl|Q_h(V,T) - h\bigl(\pi(T)\bigr)\Bigr| \le \frac{K_2(h)}{2}\lambda^2\pi(T)\bigl(1-\pi(T)\bigr)$. Note that $Q(V,T) = Q(T,V) = \pi(T)$ from the reversibility of P. Similarly, we obtain the second inequality by substituting V to S of Lemma 3.3. ◀

4 Proofs of Theorems 1.3 and 1.5

Consider a quasi-majority functional voting with respect to f on an n-vertex λ-expander graph with degree distribution π. Let $A_0, A_1, \ldots,$ be the sequence given by the functional voting with initial configuration $A_0 \subseteq V$. Theorems 1.3 and 1.5 follow from the following lemma.

▶ **Lemma 4.1.** *Consider a quasi-majority functional voting with respect to f on an n-vertex λ-expander graph with degree distribution π. Let $\epsilon_h(f) := H_f'(1/2) - 1$, $\epsilon_c(f) := 1 - H_f'(0)$ and $K(f) := \max\{K_2(f), K_2(H_f)\}$ be three positive constants depending only on f. Then, the following holds:*

- **(I)** *Let $C_1 > 0$ be an arbitrary constant and $\varepsilon : \mathbb{N} \to \mathbb{R}$ be an arbitrary function satisfying $\varepsilon(n) \to 0$ as $n \to \infty$. Suppose that $\lambda \le C_1 n^{-1/4}$, $\|\pi\|_2 \le C_1/\sqrt{n}$ and $\|\pi\|_3 \le \varepsilon/\sqrt{n}$. Then, for any $A_0 \subseteq V$ such that $|\delta(A_0)| \le c_1 \log n/\sqrt{n}$ for an arbitrary constant $c_1 > 0$, $|\delta(A_t)| \ge c_1 \log n/\sqrt{n}$ within $t = O(\log n)$ steps w.h.p.*
- **(II)** *Suppose that $\lambda \le \frac{\epsilon_h(f)}{2K(f)}$. Then, for any $A_0 \subseteq V$ s.t. $\frac{2\max\{K(f),8\}}{\epsilon_h(f)} \max\{\lambda^2, \|\pi\|_2\sqrt{\log n}\} \le |\delta(A_0)| \le \frac{\epsilon_h(f)}{K(f)}$, $|\delta(A_t)| \ge \frac{\epsilon_h(f)}{K(f)}$ within $t = O(\log|\delta(A_0)|^{-1})$ steps w.h.p.*
- **(III)** *Let c_2, c_3 be two arbitrary constants satisfying $0 < c_2 < c_3 < 1/2$ and $\epsilon(f) := \min_{x\in[c_2,c_3]}\big(x - H_f(x)\big)$ be a positive constant depending f, c_2, c_3. Suppose that $\lambda \le \frac{\epsilon(f)}{2K(f)}$ and $\|\pi\|_2 \le \frac{\epsilon(f)}{4\sqrt{\log n}}$. Then, for any $A_0 \subseteq V$ satisfying $c_2 \le \pi(A_0) \le c_3$, $\pi(A_t) \le c_2$ within constant steps w.h.p.*
- **(IV)** *Suppose that $\lambda \le \frac{\epsilon_c(f)}{2K(f)}$ and $\|\pi\|_2 \le \frac{\epsilon_c(f)^2}{32K(f)\sqrt{\log n}}$. Then, for any $A_0 \subseteq V$ satisfying $\pi(A_0) \le \frac{\epsilon_c(f)}{8K(f)}$, $\pi(A_t) = 0$ within $t = O(\log n)$ steps w.h.p.*
- **(V)** *Suppose that $H_f'(0) = 0$, $\lambda \le \frac{1}{10K(f)}$ and $\|\pi\|_2 \le \frac{1}{64K(f)\sqrt{\log n}}$. Then, for any $A_0 \subseteq V$ satisfying $\pi(A_0) \le \frac{1}{7K(f)}$, it holds w.h.p. that $\pi(A_t) = 0$ within*

$$t = O\left(\log\log n + \frac{\log n}{\log \lambda^{-1}} + \frac{\log n}{\log(\|\pi\|_2\sqrt{\log n})^{-1}}\right) \textit{ steps.}$$

Proof of Theorem 1.3(ii). Since $\|\pi\|_2 \ge 1/\sqrt{n}$, we have $|\delta(A_0)| = \Omega(\sqrt{\log n/n})$. This implies that Phase (II) takes at most $O(\log n)$. Thus, we obtain the claim since we can merge Phases (II) to (IV) by taking appropriate constants c_2, c_3 in Phase (III). ◀

Proof of Theorem 1.3(i). Under the assumption of Theorem 1.3(i), for any positive constant C, a positive constant C' exists such that $C(\lambda^2 + \|\pi\|_2\sqrt{\log n}) \le C'\frac{\log n}{\sqrt{n}}$. Thus, we can combine Phase (I) and Theorem 1.3(ii), and we obtain the claim. ◀

Proof of Theorem 1.5. Combining Phases (II), (III) and (V), we obtain the claim. ◀

In the rest of this section, we show Phases (I) to (V) of Lemma 4.1. For notational convenience, let

$$\alpha := \pi(A),\ \alpha' := \pi(A'),\ \alpha_t := \pi(A_t), \delta := \delta(A) = 2\alpha - 1,\ \delta' := \delta(A'),\ \delta_t := \delta(A_t).$$

4.1 Phase (I): $0 \le |\delta| \le c_1 \log n/\sqrt{n}$

We use the following lemma to show Lemma 4.1(I).

▶ **Lemma 4.2** (Lemma 4.5 of [11]). *Consider a Markov chain $(X_t)_{t=1}^{\infty}$ with finite state space Ω and a function $\Psi : \Omega \to \{0, \dots, n\}$. Let C_3 be arbitrary constant and $m = C_3\sqrt{n}\log n$. Suppose that Ω, Ψ and m satisfies the following conditions:*

- **(i)** *For any positive constant h, there exists a positive constant $C_1 < 1$ such that*

$$\mathbf{Pr}\left[\Psi(X_{t+1}) < h\sqrt{n} \,\middle|\, \Psi(X_t) \le m\right] < C_1.$$

(ii) *Three positive constants γ, C_2 and h exist such that, for any $x \in \Omega$ satisfying $h\sqrt{n} \leq \Psi(x) < m$,*

$$\mathbf{Pr}\left[\Psi(X_{t+1}) < (1+\gamma)\Psi(X_t) \mid X_t = x\right] < \exp\left(-C_2\frac{\Psi(x)^2}{n}\right).$$

Then, $\Psi(X_t) \geq m$ holds w.h.p. for some $t = O(\log n)$.

Let us first prove the following lemma concerning the growth rate of $|\delta|$, which we will use in the proofs of (I) and (II) of Lemma 4.1.

▶ **Lemma 4.3.** *Consider a quasi-majority functional voting with respect to f on an n-vertex λ-expander graph with degree distribution π. Let $\epsilon_h(f) := H_f'(1/2) - 1$ and $K(f) := \max\{K_2(f), K_2(H_f)\}$ be positive constants depending only on f. Suppose that $\lambda \leq \frac{\epsilon_h(f)}{2K(f)}$. Then, for any $A \subseteq V$ satisfying $\frac{2K(f)}{\epsilon_h(f)}\lambda^2 \leq |\delta| \leq \frac{\epsilon_h(f)}{K(f)}$,*

$$\mathbf{Pr}\left[|\delta'| \leq \left(1 + \frac{\epsilon_h(f)}{8}\right)|\delta|\right] \leq 2\exp\left(-\frac{\epsilon_h(f)^2\delta^2}{128\|\pi\|_2^2}\right).$$

Proof. Combining Lemma 2.5 and Taylor's theorem, we have

$$\begin{aligned}\left|\mathbf{E}[\delta'] - H_f'\left(\frac{1}{2}\right)\delta\right| &= 2\left|\mathbf{E}[\alpha'] - \frac{1}{2} - H_f'\left(\frac{1}{2}\right)\left(\alpha - \frac{1}{2}\right)\right| \\ &= 2\left|\mathbf{E}\left[\alpha'\right] - H_f\left(\alpha\right) + H_f\left(\alpha\right) - H_f\left(\frac{1}{2}\right) - H_f'\left(\frac{1}{2}\right)\left(\alpha - \frac{1}{2}\right)\right| \\ &\leq 2K_2(f)\lambda\left(|\delta| + \lambda\right)\alpha(1-\alpha) + K_2(H_f)\left(\alpha - \frac{1}{2}\right)^2 \\ &\leq \left(\frac{K(f)}{2}\lambda + \frac{K(f)}{4}|\delta|\right)|\delta| + \frac{K(f)}{2}\lambda^2 \end{aligned} \tag{9}$$

Note that $H_f(1/2) = 1/2$ from the definition. From assumptions of $\lambda \leq \frac{\epsilon_h(f)}{2K(f)}$, $|\delta| \leq \frac{\epsilon_h(f)}{K(f)}$ and $\lambda^2 \leq \frac{\epsilon_h(f)}{2K(f)}|\delta|$, we have $\left|H_f'\left(\frac{1}{2}\right)\delta\right| - |\mathbf{E}[\delta']| \leq \left|H_f'\left(\frac{1}{2}\right)\delta - \mathbf{E}[\delta']\right| \leq \frac{3}{4}\epsilon_h(f)|\delta|$. Hence, it holds that

$$\left|\mathbf{E}[\delta']\right| \geq \left|H_f'\left(\frac{1}{2}\right)\delta\right| - \frac{3}{4}\epsilon_h(f)|\delta| = (1+\epsilon_h(f))|\delta| - \frac{3}{4}\epsilon_h(f)|\delta| = \left(1 + \frac{\epsilon_h(f)}{4}\right)|\delta|.$$

We observe that, for any $\kappa > 0$,

$$\mathbf{Pr}\left[|\delta'| \leq \left|\mathbf{E}[\delta']\right| - \kappa\right] \leq 2\exp\left(-\frac{\kappa^2}{2\|\pi\|_2^2}\right) \tag{10}$$

from Corollary A.4 of the full version [38]. Note that $\delta' = \sum_{v\in V}\pi(v)(2X_v - 1)$ for independent indicator random variables $(X_v)_{v\in V}$ (see (5) for the definition of X_v). Thus,

$$\begin{aligned}\mathbf{Pr}\left[|\delta'| \leq \left(1 + \frac{\epsilon_h(f)}{8}\right)|\delta|\right] &= \mathbf{Pr}\left[|\delta'| \leq \left(1 + \frac{\epsilon_h(f)}{4}\right)|\delta| - \frac{\epsilon_h(f)}{8}|\delta|\right] \\ &\leq \mathbf{Pr}\left[|\delta'| \leq \left|\mathbf{E}[\delta']\right| - \frac{\epsilon_h(f)}{8}|\delta|\right] \leq 2\exp\left(-\frac{\epsilon_h(f)^2\delta^2}{128\|\pi\|_2^2}\right)\end{aligned}$$

and we obtain the claim. ◀

Proof of Lemma 4.1(I). We check the conditions (i) and (ii) of Lemma 4.2 with letting $\Psi(A) = \lfloor n|\delta(A)|\rfloor$ and $m = c_1\sqrt{n}\log n$.

Condition (i). First, we show the following claim that evaluates $\mathbf{Var}[\delta']$.

▷ Claim 4.4. Under the same assumption as Lemma 4.1(I),

$$\frac{\epsilon_{\mathrm{var}}(f)}{n} \le \mathbf{Var}[\delta'] \le \frac{5C_1^2}{n}$$

holds, where $\epsilon_{\mathrm{var}}(f) := f(1/2)(1-f(1/2))$ is a positive constant depending only on f.

Proof of the claim. From Lemma 2.5 and assumptions, we have

$$\left|\frac{\mathbf{Var}[\delta']}{4} - \|\pi\|_2^2 g\left(\frac{1}{2}\right)\right| = \left|\mathbf{Var}[\alpha'] - \|\pi\|_2^2 g\left(\frac{1}{2}\right)\right| \le K_1(g)\left(\|\pi\|_2^2\frac{|\delta|}{2} + \|\pi\|_3^{3/2}\lambda\right)$$
$$\le \frac{K_1(g)}{n}\left(C_1^2 c_1\frac{\log n}{\sqrt{n}} + C_1\epsilon^{3/2}\right) = \frac{1}{n}\cdot o(1).$$

Note that $\mathbf{Var}[\delta'] = \mathbf{Var}[2\alpha' - 1] = 4\,\mathbf{Var}[\alpha']$. Since $\|\pi\|_2^2 \ge 1/n$, we have

$$\frac{\epsilon_{\mathrm{var}}(f)}{n} \le \frac{4\epsilon_{\mathrm{var}}(f) - o(1)}{n} \le \mathbf{Var}[\delta'] \le \frac{4C_1^2 + o(1)}{n} \le \frac{5C_1^2}{n}. \qquad \triangleleft$$

From Corollary A.6 of the full version [38] with letting $Y_v = \pi(v)(2X_v - 1)$, we have

$$\mathbf{Pr}\left[|\delta'| \le x\sqrt{\frac{\epsilon_{\mathrm{var}}(f)}{n}}\right] \le \mathbf{Pr}\left[|\delta'| \le x\sqrt{\mathbf{Var}[\delta']}\right] \le \Phi(x) + \frac{5.6\|\pi\|_3^3}{\mathbf{Var}[\delta']^{3/2}}$$
$$\le \Phi(x) + 5.6\frac{\epsilon^3}{n^{3/2}}\cdot\frac{n^{3/2}}{\epsilon_{\mathrm{var}}(f)^{3/2}} = \Phi(x) + o(1) \tag{11}$$

for any $x \in \mathbb{R}$, where $\Phi(x) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x} \mathrm{e}^{-y^2/2}\mathrm{d}y$. Thus, for any constant $h > 0$, there exists some constant $C > 0$ such that $\mathbf{Pr}[\Psi(A') < h\sqrt{n} \mid \Psi(A) \le m] < C$, which verifies the condition (i).

Condition (ii). Set $h = \frac{2K(f)}{\epsilon_h(f)}C_1^2$ and assume $h\sqrt{n} \le \Psi(A) < m$. Then

$$\frac{2K(f)}{\epsilon_h(f)}\lambda^2 n \le \frac{2K(f)}{\epsilon_h(f)}C_1^2\sqrt{n} = h\sqrt{n} \le \Psi(A) \le |\delta|n = o(n).$$

Thus, we can apply Lemma 4.3 and positive constants γ, C exist such that, for any $h\sqrt{n} \le \Psi(A) \le c_1\sqrt{n}\log n$, $\mathbf{Pr}[\Psi(A') < (1+\gamma)\Psi(A)] < \exp\left(-C\frac{\Psi(A)^2}{n}\right)$. Note that $\|\pi\|_2^2 = \Theta(1/n)$ from the assumption. Thus the condition (ii) holds and we can apply Lemma 4.2. ◀

4.2 Phase (II): $\frac{2\max\{K(f),8\}}{\epsilon_h(f)}\max\{\lambda^2, \|\pi\|_2\sqrt{\log n}\} \le |\delta| \le \frac{\epsilon_h(f)}{K(f)}$

Proof of Lemma 4.1(II). Since $|\delta| \ge \frac{16}{\epsilon_h(f)}\|\pi\|_2\sqrt{\log n}$ from assumptions, applying Lemma 4.3 yields $\mathbf{Pr}\left[|\delta'| \le \left(1 + \frac{\epsilon_h(f)}{8}\right)|\delta|\right] \le \frac{2}{n^2}$. Thus, it holds with probability larger than $(1 - 2/n^2)^t$ that $|\delta_t| \ge \left(1 + \frac{\epsilon_h(f)}{8}\right)^t |\delta_0|$ and we obtain the claim by substituting $t = O(\log|\delta_0|^{-1})$. ◀

4.3 Phase (III): $0 < c_2 \le \alpha \le c_3 < 1/2$

Proof of Lemma 4.1(III). We first observe that, for any $\kappa > 0$,

$$\mathbf{Pr}\left[|\alpha' - \mathbf{E}[\alpha']| \ge \kappa\|\pi\|_2\sqrt{\log n}\right] \le 2n^{-2\kappa} \tag{12}$$

from the Hoeffding theorem. Note that $\alpha' = \sum_{v\in V}\pi(v)X_v$ for independent indicator random variables $(X_v)_{v\in V}$. Hence, applying Lemma 2.5 yields

$$|\alpha' - H_f(\alpha)| \le |\alpha' - \mathbf{E}[\alpha']| + |\mathbf{E}[\alpha'] - H_f(\alpha)| \le \|\pi\|_2\sqrt{\log n} + \frac{K_2(f)}{4}(|\delta| + \lambda)\lambda \tag{13}$$

with probability larger than $1 - 2/n^2$. Then, for any $\alpha \in [c_2, c_3]$, it holds with probability larger than $1 - 2/n^2$ that

$$\alpha' \le H_f(\alpha) + \frac{K(f)}{2}\lambda + \|\pi\|_2\sqrt{\log n} \le \alpha - \epsilon(f) + \frac{\epsilon(f)}{4} + \frac{\epsilon(f)}{4} \le \alpha - \frac{\epsilon(f)}{2}.$$

Thus, for $\alpha_0 \in [c_2, c_3]$, $\alpha_t \le c_2$ within $t = 2(c_3 - c_2)/\epsilon(f) = O(1)$ steps w.h.p. ◀

4.4 Phase (IV): $0 \le \alpha \le \frac{\epsilon_c(f)}{8K(f)}$

We show the following lemma which is useful for proving (IV) and (V) of Lemma 4.1.

▶ **Lemma 4.5.** *Let $\epsilon \in (0, 1]$ be an arbitrary constant. Consider functional voting on an n-vertex connected graph with degree distribution π. Suppose that, for some $\alpha_* \in [0, 1]$ and $K \in [0, 1 - \epsilon]$, $\mathbf{E}[\alpha'] \le K\alpha$ holds for any $A \subseteq V$ satisfying $\alpha \le \alpha_*$ and $\|\pi\|_2 \le \frac{\epsilon\alpha_*}{2\sqrt{\log n}}$. Then, for any $A_0 \subseteq V$ satisfying $\alpha_0 \le \alpha_*$, $\alpha_t = 0$ w.h.p. within $O\left(\frac{\log n}{\log K^{-1}}\right)$ steps.*

Proof. For any $\alpha \le \alpha_*$, from (12) and assumptions of $\mathbf{E}[\alpha'] \le \alpha$ and $\|\pi\|_2 \le \frac{\epsilon\alpha_*}{2\sqrt{\log n}}$, it holds with probability larger than $1 - 2/n^4$ that

$$\alpha' \le \mathbf{E}[\alpha'] + 2\|\pi\|_2\sqrt{\log n} \le K\alpha + \epsilon\alpha_* \le (1 - \epsilon)\alpha_* + \epsilon\alpha_* = \alpha_*.$$

Thus, for any $\alpha_0 \le \alpha_*$, we have

$$\begin{aligned}\mathbf{E}[\alpha_t] &= \sum_{x \le a_*}\mathbf{E}\left[\alpha_t | \alpha_{t-1} = x\right]\mathbf{Pr}\left[\alpha_{t-1} = x\right] + \sum_{x > a_*}\mathbf{E}\left[\alpha_t | \alpha_{t-1} = x\right]\mathbf{Pr}\left[\alpha_{t-1} = x\right] \\ &\le \sum_{x \le a_*} Kx\,\mathbf{Pr}\left[\alpha_{t-1} = x\right] + \mathbf{Pr}\left[\alpha_{t-1} > a_*\right] \le K\,\mathbf{E}[\alpha_{t-1}] + \frac{2t}{n^4} \\ &\le \cdots \le K^t\alpha_0 + \frac{2t^2}{n^4} \le K^t + \frac{2t^2}{n^4}.\end{aligned}$$

This implies that, $\mathbf{E}[\alpha_t] = O(n^{-3})$ within $t = O\left(\frac{\log n}{\log K^{-1}}\right)$ steps. Let $\pi_{\min} := \min_{v\in V}\pi(v) \ge 1/(2|E|) \ge 1/n^2$. We obtain the claim from the Markov inequality, which yields $\mathbf{Pr}[\alpha_t = 0] = 1 - \mathbf{Pr}[\alpha_t \ge \pi_{\min}] \ge 1 - \frac{\mathbf{E}[\alpha_t]}{\pi_{\min}} = 1 - O(1/n)$. ◀

Proof of Lemma 4.1 of (IV). Combining Lemma 2.5 and Taylor's theorem,

$$\begin{aligned}\left|\mathbf{E}[\alpha'] - H_f'(0)\alpha\right| &= \left|\mathbf{E}[\alpha'] - H_f(\alpha) + H_f(\alpha) - H_f(0) - H_f'(0)(\alpha - 0)\right| \\ &\le K_2(f)\lambda\left(|\delta| + \lambda\right)\alpha(1 - \alpha) + \frac{K_2(H_f)}{2}\alpha^2 \\ &\le 2K(f)\lambda\alpha + \frac{K(f)}{2}\alpha^2. \end{aligned}\tag{14}$$

Hence, for any $\alpha \le \frac{\epsilon_c(f)}{8K(f)}$, we have $\mathbf{E}[\alpha'] \le \left(H'_f(0) + 2K(f)\lambda + \frac{K(f)}{2}\alpha\right)\alpha \le \left(1 - \frac{\epsilon_c(f)}{2}\right)\alpha$. Letting $\epsilon = \epsilon_c(f)/2$, $K = 1 - \epsilon_c(f)/2$ and $\alpha_* = \frac{\epsilon_c(f)}{8K(f)}$, from the assumption, $\|\pi\|_2 \le \frac{\epsilon_c(f)^2}{32K(f)\sqrt{\log n}} = \frac{\epsilon\alpha_*}{2\sqrt{\log n}}$. Thus, we can apply Lemma 4.5 and we obtain the claim. ◀

4.5 Phase (V): $H'_f(0) = 0$ and $0 \le \alpha \le \frac{1}{7K(f)}$

Proof of Lemma 4.1(V). In this case, from (14),

$$\mathbf{E}[\alpha'] \le 2K(f)\lambda\alpha + \frac{K(f)}{2}\alpha^2. \tag{15}$$

We consider the following two cases.

Case 1. $\max\left\{\lambda, \sqrt{\frac{\|\pi\|_2\sqrt{\log n}}{K(f)}}\right\} \le \alpha \le \frac{1}{7K(f)}$: In this case, combining (12) and (15), it holds with probability larger than $1 - 2/n^2$ that

$$\alpha' \le \left(\frac{2K(f)\lambda}{\alpha} + \frac{K(f)}{2} + \frac{\|\pi\|_2\sqrt{\log n}}{\alpha^2}\right)\alpha^2 \le \frac{7K(f)}{2}\alpha^2.$$

Applying this inequality iteratively, for any $\alpha_0 \le 7K(f)^{-1}$,

$$\alpha_t \le \frac{7K(f)}{2}\alpha_{t-1}^2 \le \cdots \le \frac{2}{7K(f)}\left(\frac{7K(f)}{2}\alpha_0\right)^{2^t} \le \frac{2}{7K(f)2^{2^t}}.$$

holds with probability larger than $(1 - 2/n^2)^t$. This implies that, within $t = O(\log\log n)$ steps, $\alpha_t \le \max\left\{\lambda, \sqrt{\frac{\|\pi\|_2\sqrt{\log n}}{K(f)}}\right\}$ w.h.p. Note that $\max\left\{\lambda, \sqrt{\frac{\|\pi\|_2\sqrt{\log n}}{K(f)}}\right\} \ge \sqrt{\frac{\|\pi\|_2\sqrt{\log n}}{K(f)}} \ge \sqrt{\frac{\sqrt{\log n/n}}{K(f)}}$ since $\|\pi\|_2^2 \ge 1/n$.

Case 2. $\alpha \le \max\left\{\lambda, \sqrt{\frac{\|\pi\|_2\sqrt{\log n}}{K(f)}}\right\}$: Set $\alpha_* = \max\left\{\lambda, \sqrt{\frac{\|\pi\|_2\sqrt{\log n}}{K(f)}}\right\} \ge \sqrt{\frac{\|\pi\|_2\sqrt{\log n}}{K(f)}}$, $K = \frac{5K(f)}{2}\lambda + \frac{1}{2}\sqrt{K(f)\|\pi\|_2\sqrt{\log n}}$ and $\epsilon = 1/4$. Then, from $\lambda \le \frac{1}{10K(f)}$ and $\|\pi\|_2 \le \frac{1}{64K(f)\sqrt{\log n}}$, we have $K \le 1 - \epsilon$,

$$\|\pi\|_2 = (\sqrt{\|\pi\|_2})^2 \le \frac{\sqrt{\|\pi\|_2}}{8\sqrt{K(f)\sqrt{\log n}}} = \sqrt{\frac{\|\pi\|_2\sqrt{\log n}}{K(f)}}\frac{\epsilon}{2\sqrt{\log n}} \le \frac{\epsilon\alpha_*}{2\sqrt{\log n}},$$

$$\mathbf{E}[\alpha'] \le \left(2K(f)\lambda + \frac{K(f)}{2}\alpha\right)\alpha \le \left(2K(f)\lambda + \frac{K(f)}{2}\lambda + \frac{1}{2}\sqrt{K(f)\|\pi\|_2\sqrt{\log n}}\right)\alpha = K\alpha.$$

Thus, applying Lemma 4.5, we obtain the claim. ◀

5 Conclusion

In this paper we propose functional voting as a generalization of several known voting processes. We show that the consensus time is $O(\log n)$ for any quasi-majority functional voting on $O(n^{-1/2})$-expander graphs with balanced degree distributions. This result extends previous works concerning voting processes on expander graphs. Possible future direction of this work includes

1. Does $O(\log n)$ worst-case consensus time holds for quasi-majority functional voting on graphs with less expansion (i.e., $\lambda = \omega(n^{-1/2})$)?
2. Is there some relationship between best-of-k and Majority?

References

1 M. A. Abdullah and M. Draief. Global majority consensus by local majority polling on graphs of a given degree sequence. *Discrete Applied Mathematics*, 1(10):1–10, 2015.

2 Y. Afek, N. Alon, O. Barad, E. Hornstein, N. Barkai, and Z. Bar-Joseph. A biological solution to a fundamental distributed computing problem. *Science*, 331(6014):183–185, 2011.

3 D. Aldous and J. Fill. Reversible Markov chains and random walks on graphs. URL: `https://www.stat.berkeley.edu/users/aldous/RWG/book.html`.

4 L. Becchetti, A. Clementi, E. Natale, F. Pasquale, R. Silvestri, and L. Trevisan. Simple dynamics for plurality consensus. *Distributed Computing*, 30(4):293–306, 2017.

5 L. Becchetti, A. Clementi, E. Natale, F. Pasquale, and L. Trevisan. Stabilizing consensus with many opinions. *In Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 620–635, 2016.

6 I. Benjamini, S.-O. Chan, R. O'Donnell, O. Tamuzc, and L.-Y. Tand. Convergence, unanimity and disagreement in majority dynamics on unimodular graphs and random graphs. *Stochastic Processes and their Applications*, 126(9):2719–2733, 2016.

7 P. Berenbrink, A. Clementi, R. Elsässer, P. Kling, F. Mallmann-Trenn, and E. Natale. Ignore or comply? On breaking symmetry in consensus. *In Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, pages 335–344, 2017.

8 P. Berenbrink, G. Giakkoupis, Anne-Marie Kermarrec, and F. Mallmann-Trenn. Bounds on the voter model in dynamic networks. *In Proceedings of the 43rd International Colloquium on Automata, Languages, and Programming (ICALP)*, 2016.

9 E. Berger. Dynamic monopolies of constant size. *Journal of Combinatorial Theory Series B*, 83(2):191–200, 2001.

10 R. Pastor-Satorras C. Castellano, M. A. Muñoz. The non-linear q-voter model. *Physical Review E*, 80, 2009.

11 A. Clementi, M. Ghaffari, L. Gualà, E. Natale, F. Pasquale, and G. Scornavacca. A tight analysis of the parallel undecided-state dynamics with two colors. *In Proceedings of the 43rd International Symposium on Mathematical Foundations of Computer Science (MFCS)*, 117(28):1–15, 2018.

12 A. Coja-Oghlan. On the laplacian eigenvalues of $G_{n,p}$. *Combinatorics, Probability and Computing*, 16(6):923–946, 2007.

13 Nicholas Cook, Larry Goldstein, and Tobias Johnson. Size biased couplings and the spectral gap for random regular graphs. *The Annals of Probability*, 46(1):72–125, 2018.

14 C. Cooper, R. Elsässer, H. Ono, and T. Radzik. Coalescing random walks and voting on connected graphs. *SIAM Journal on Discrete Mathematics*, 27(4):1748–1758, 2013.

15 C. Cooper, R. Elsässer, and T. Radzik. The power of two choices in distributed voting. *In Proceedings of the 41st International Colloquium on Automata, Languages, and Programming (ICALP)*, 2:435–446, 2014.

16 C. Cooper, R. Elsässer, T. Radzik, N. Rivera, and T. Shiraga. Fast consensus for voting on general expander graphs. *In Proceedings of the 29th International Symposium on Distributed Computing (DISC)*, pages 248–262, 2015.

17 C. Cooper, T. Radzik, N. Rivera, and T. Shiraga. Fast plurality consensus in regular expanders. *In Proceedings of the 31st International Symposium on Distributed Computing (DISC)*, 91(13):1–16, 2017.

18 C. Cooper and N. Rivera. The linear voting model. *In Proceedings of the 43rd International Colloquium on Automata, Languages, and Programming (ICALP)*, 55(144):1–12, 2016.

19 E. Cruciani, E. Natale, A. Nusser, and G. Scornavacca. Phase transition of the 2-choices dynamics on core-periphery networks. *In Proceedings of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 777–785, 2018.

20 E. Cruciani, E. Natale, and G. Scornavacca. Distributed community detection via metastability of the 2-choices dynamics. *In Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI)*, pages 6046–6053, 2019.

21 B. Doerr, L. A. Goldberg, L. Minder, T. Sauerwald, and C. Scheideler. Stabilizing consensus with the power of two choices. *In Proceedings of the 23rd Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 149–158, 2011.

22 M. Fischer, N. Lynch, and M. Merritt. Easy impossibility proofs for distributed consensus problems. *Distributed Computing*, 1(1):26–39, 1986.

23 A. Frieze and M. Karońsky. *Introduction to random graphs.* Campridge University Press, 2016.

24 B. Gärtner and A. N. Zehmakan. Majority model on random regular graphs. *In Proceedings of the 13th Latin American Symposium on Theoretical Informatics (LATIN)*, pages 572–583, 2018.

25 M. Ghaffari and J. Lengler. Nearly-tight analysis for 2-choice and 3-majority consensus dynamics. *In Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, pages 305–313, 2018.

26 S. Gilbert and D. Kowalski. Distributed agreement with optimal communication complexity. *In Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 965–977, 2010.

27 Y. Hassin and D. Peleg. Distributed probabilistic polling and applications to proportionate agreement. *Information and Computation*, 171(2):248–268, 2001.

28 N. Kang and R. Rivera. Best-of-three voting on dense graphs. *In Proceedings of the 31st ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 115–121, 2019.

29 D. A. Levin and Y. Peres. *Markov chain and mixing times: second edition.* The American Mathematical Society, 2017.

30 T. M. Liggett. *Interacting particle systems.* Springer-Verlag, 1985.

31 R. Montenegro and P. Tetali. *Mathematical aspects of mixing times in Markov chains.* NOW Publishers, 2006.

32 E. Mossel, J. Neeman, and O. Tamuz. Majority dynamics and aggregation of information in social networks. *Autonomous Agents and Multiagent Systems*, 28(3):408–429, 2014.

33 T. Nakata, H. Imahayashi, and M. Yamashita. Probabilistic local majority voting for the agreement problem on finite graph. *In Proceedings of the 5th Annual International Computing and Combinatorics Conference (COCOON)*, pages 330–338, 1999.

34 D. Peleg. Size bounds for dynamic monopolies. *Discrete Applied Mathematics*, 86(2–3):263–273, 1998.

35 D. Peleg. Local majorities, coalitions and monopolies in graphs: a review. *Theoretical Computer Science*, 282(2):231–257, 2002.

36 G. Schoenebeck and F. Yu. Consensus of interacting particle systems on Erdős-Rényi graphs. *In Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1945–1964, 2018.

37 N. Shimizu and T. Shiraga. Phase transitions of best-of-two and best-of-three on stochastic block models. *In Proceedings of the 33rd International Symposium on Distributed Computing (DISC)*, pages 32:1–32:17, 2019.

38 N. Shimizu and T. Shiraga. Quasi-majority functional voting on expander graphs. *arXiv*, 2020. `arXiv:2002.07411`.

39 K. Tikhomirov and P. Youssef. The spectral gap of dense random regular graphs. *The Annals of Probability*, 47(1):362–419, 2019.

40 A. N. Zehmakan. Opinion forming in Erdős-Rényi random graph and expanders. *In Proceedings of the 29th International Symposium on Algorithms and Computation (ISAAC)*, pages 4:1–4:13, 2018.

Property Testing of LP-Type Problems

Rogers Epstein
Massachusetts Institute of Technology, Cambridge, MA, USA
rogersep@mit.edu

Sandeep Silwal
Massachusetts Institute of Technology, Cambridge, MA, USA
silwal@mit.edu

Abstract

Given query access to a set of constraints S, we wish to quickly check if some objective function φ subject to these constraints is at most a given value k. We approach this problem using the framework of property testing where our goal is to distinguish the case $\varphi(S) \leq k$ from the case that at least an ϵ fraction of the constraints in S need to be removed for $\varphi(S) \leq k$ to hold. We restrict our attention to the case where (S, φ) are LP-Type problems which is a rich family of combinatorial optimization problems with an inherent geometric structure. By utilizing a simple sampling procedure which has been used previously to study these problems, we are able to create property testers for any LP-Type problem whose query complexities are independent of the number of constraints. To the best of our knowledge, this is the first work that connects the area of LP-Type problems and property testing in a systematic way. Among our results are property testers for a variety of LP-Type problems that are new and also problems that have been studied previously such as a tight upper bound on the query complexity of testing clusterability with one cluster considered by Alon, Dar, Parnas, and Ron (FOCS 2000). We also supply a corresponding tight lower bound for this problem and other LP-Type problems using geometric constructions.

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

Keywords and phrases property pesting, LP-Type problems, random sampling

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.98

Category Track A: Algorithms, Complexity and Games

Funding *Sandeep Silwal*: Research supported by the NSF Graduate Research Fellowship under Grant No. 1122374.

Acknowledgements We would like to thank Ronitt Rubinfeld, Piotr Indyk, Bertie Ancona, and Rikhav Shah for helpful feedback.

1 Introduction

Many problems in combinatorial optimization can be represented as a pair (S, φ) where S is a set of constraints and φ is a function of the constraints that we would like to minimize. This class includes many problems that are NP-hard, even for the decision version of some problems where we would like to determine if $\varphi(S)$ is at most some constant k. For instance, let S be constraints that say two nodes in a graph are connected by an edge (so S can be thought of as a set of edges) and φ be the chromatic number of a graph with those edges. Then it is NP-complete to determine if $\varphi(S) \leq 3$.

In this work, we consider a relaxation of the above hard class of problems by using the framework of *property testing*. Specifically, given a value parameter k and distance parameter ϵ, we wish to determine if $\varphi(S) \leq k$ or if (S, φ) is ϵ-far from $\varphi(S) \leq k$, where ϵ-far means that at least $\epsilon|S|$ many of the constraints of S need to be removed for $\varphi(S) \leq k$ to hold. We assume we have query access to the constraints and knowledge of φ and our goal is to perform property testing while minimizing the number of constraints of S we access.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 98; pp. 98:1–98:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Even under the property testing setting, this question is too broad. Therefore, we focus our attention to *LP-Type Problems*, formally described in Definition 1. Informally, these problems have an underlying geometrical structure, much like that of linear programs, which can be used to create efficient testing algorithms.

Our main result is a general algorithm that is able to perform property testing for *any* LP-Type problem with $O(\delta/\epsilon)$ queries where δ is the dimension of the LP-Type problem, formally defined in Section 2. In many cases, this bound is tight such as testing clusterability using one cluster considered in [1]. To the best of our knowledge, this is the first work that connects the area of LP-Type problems and property testing in a systematic way. The class of these problems is quite general and includes problems which have been previously studied individually in property testing, such as testing clusterability of points in [1], and newer testing problems, such as determining if a set of linear constraints is feasible or "far" from feasible. We also give a matching lower bound for many of these problems using geometric constructions. For a comprehensive overview of our contributions, see Section 2.2.

1.1 Related Work

Many problems in property testing can be modeled as a set of constraints and some optimization function over these constraints. These include well studied graph problems such as bipartiteness, expansion, k-colorability, and many other problems [14, 15, 16, 23]. This line of work was initiated by Goldreich and Ron and there are many results in the area of graph property testing. For more information about graph property testing, see [13] and the references within. Overall, these testing problems differ from our setting where our queries are essentially access to random constraints.

This model where queries are accesses to constraints have also been studied in the case where we wish to test properties of a metric space and queries are access to points (see [19, 1, 21]). There are instances of these problems that are also examples of LP-Type problems that we consider. For more information, see Section 2.2.

LP-Type problems have a rich literature and there have been many previous work on them, including general algorithms to solve LP-Type problems [25, 6, 26, 18]. The algorithms for these problems have runtimes that are generally linear in the number of constraints, but exponential in the dimension of the LP-Type problem (see Definition 3). This is in contrast to our testing algorithms that have no dependence on the number of constraints.

Furthermore, many properties of LP-Type problems have been generalized to a larger class of problems called violator spaces [11, 4]. We do not explicitly consider them here since these problems do not yield any additional interesting property testing applications but our results carry over to this setting in a straightforward manner.

There are previous works on geometric property testing, for example, testing convexity of point sets [7, 20], testing disjointness of geometric bodies [9], and testing properties of two dimensional images [22, 24, 2, 3]. While many of these work share the paradigm of "sample few objects and test" that is common among many property testers, including ours, the problems considered in these works are very different than the problems we focus on (see 2.2).

There is also existing work on property testing for constraint satisfaction problems (CSPs) where given an instance of a CSP, one is given query access to an assignment of the variables and the task is to determine if the assignment is "close" or "far" from satisfying the instance [5]. This is different than our setting where we wish to check if $\varphi(S) \leq k$ where φ is a function of the constraints in S.

The paper of Czumaj and Sohler cited in [8] is closest in spirit to our work, as it also introduces a framework for property testing for a general class of problems which they refer to as *abstract combinatorial problems* (ACPs). This class of problems are similar to LP-type problems as they both include a set of constraints and consider a restricted set of subsets called a basis (see Section 2 for a formal definition of LP-Type problems). Furthermore, [8] exploit the combinatorial structure of ACPs to develop testing algorithms, just as we do for LP-Type problems. However, there are some key differences between ACPs and LP-Type problems, such as the fact that LP-Type problems are equipped with a function ϕ that is optimized over a set of constraints. LP-Type problems also have a clean correspondence between dimension and the number of "violators" for a particular basis as formulated in Lemma 11 and Corollary 13, which lets us automatically translate between the dimension and the property testing query complexity.

In terms of concrete property testing problems, [8] also consider the smallest enclosing ball problem (see Section 2.2). We achieve a better query complexity bound than [8] for this problem, but it is important to note that the results of [8] also hold for the general problem of clustering with multiple balls while we only consider one ball. However, the rest of the problems considered in [8] do not overlap with the problems we present in this paper and furthermore, the problems we study are inherently geometric in nature. Lastly, while ACPs are the invention of the authors in [8], there exists rich literature for LP-Type problems outside of the domain of property testing.

Lastly, we note that using a few samples to determine information about your underlying data has also been studied in the PAC-learning literature. In that setting, the goal is to learn a classifier of your dataset and recently, the question of optimal sample complexity for PAC learning has been settled, see for instance [17]. The key differences between the PAC learning setting and our work is that our "datapoints" do not necessarily come with a label. For instance, the PAC learning framework would be valid in the example where we have labeled points in $\mathbb{R}^d$ and we wish to learn a separator from a family class such as balls. However, this framework would not apply if we wished to compute the smallest enclosing ball of the points. We do consider one instance where a PAC learning framework would apply and that is problem of property testing if a set of labeled points are linearly separable or far from it (for more information, see Section 2.2). In this example, a valid approach would be to use the PAC learning framework and learn a linear classifier. Instead of pursuing this route, we are able to cast this problem as an example of a LP-Type problem and achieve the optimal sample complexity using the theory we develop.

1.2 Organization

In Section 2, we formally define the class of LP-Type problems. In Section 2.2 we outline our contributions. In Section 3, we present our algorithms and prove their correctness and in Section 4, we apply our algorithm to specific LP-Type problems. Finally, we complement some of these problems with lower bounds in Section 5.

2 Preliminaries

2.1 Notation and Definitions

We define LP-Type problems as well as some related concepts. These definitions are standard in the literature for LP-Type problems but we reproduce them below for the sake of completeness. For more information, see [26, 18, 12].

▶ **Definition 1** (LP-Type Problem). *Let S be a finite set and φ be a function that maps subsets of S to some value. We say (S, φ) is a LP-Type problem if φ satisfies the following two properties:*

- *Monotonicity: if $A \subseteq B \subseteq S$ then $\varphi(A) \leq \varphi(B)$*
- *Locality: For all $A \subseteq B \subseteq S$ and elements $x \in S$, if $\varphi(A) = \varphi(B) = \varphi(A \cup \{x\})$, then $\varphi(A) = \varphi(B \cup \{x\})$.*

In general, one should think of S as a set of constraints, and φ as some objective function we wish to minimize (or equivalently, maximize) over these constraints. The canonical example of a LP-Type problem is a linear program where S is a set of linear constraints and φ is a linear functional.

Just as linear programs can be associated with a dimension, which is the number of variables, LP-Type problems in general also have a natural definition of dimension which influences the runtime of many algorithms for LP-Type problems as well as our algorithm for property testing. First, we must define the notion of a basis.

▶ **Definition 2** (Basis of LP-Type problems). *Given an LP-Type problem, a* basis *$B \subseteq S$ is a set such that for all proper subsets $B' \subset B$, we have $\varphi(B') < \varphi(B)$.*

Given the above definitions, we can now define the dimension of a LP-Type problem.

▶ **Definition 3** (Dimension of LP-Type problem). *The* dimension *δ of an LP-Type problem is the largest possible size of a basis $B \subseteq S$. This is sometimes also called the* combinatorial dimension*.*

There are many examples of well-studied LP-Type problems and in many of these cases explicit bounds, if not exact values, are known regarding their dimensions. For more information, see our contributions in Section 2.2. In general, the dimension of the problem tends to grow with the "difficulty" of solving it and for property testing, our query complexity bound is also a function of the dimension. We now formally define property testing for LP-Type problems.

▶ **Definition 4** (Property Testing of LP-Type problems). *Given an LP-Type problem (S, φ), a parameter k, a distance parameter ϵ, and query access to the constraints in S, we wish to distinguish the following two cases:*

- *Output accept with probability at least $2/3$ if $\varphi(S) \leq k$ (Completeness Case)*
- *Output reject with probability at least $2/3$ if at least $\epsilon|S|$ constraints need to be removed from S for $\varphi(S) \leq k$ to hold (Soundness Case).*

▶ Remark 5. We say that S is ϵ-far if it falls in the soundness case.

2.2 Our Contributions

The main contribution of this paper is a comprehensive algorithm for property testing of LP-Type problems with query complexity $O(\delta/\epsilon)$ where δ is the dimension of the LP-Type problem. Note that this bound is independent of the number of constraints which is $|S|$. Our algorithm is simple and proceeds by first sampling a small set of random constraints in S, constructing a partial solution, and "testing" this partial solution against few other randomly chosen constraints. The analysis that we reject in the ϵ-far case (soundness) is straightforward. However, the main technical challenge lies in showing that our algorithm accepts in the completeness case. To do so, we use a "sampling" lemma which roughly says that for a randomly chosen subset R of S of a particular size (depending on the dimension δ) and x a randomly chosen element of $S \setminus R$, we have $\varphi(R) = \varphi(R \cup \{x\})$. Using this result, we show that we are likely to accept in the completeness case. For the full detailed analysis, see Section 3. All of our algorithms have two-sided error.

We highlight the power of our approach by considering the query complexity bounds that we get for a few selected problems. In many cases, we are also able to get matching lower bounds. More specifically, we obtain the following results as an application of our framework:

1. We consider the problem of determining if a set of linear inequalities in d variables is feasible (there exists a satisfying assignment) or if at least ϵ-fraction of the constraints need to be removed for the set of constraints to be feasible. While this problem does not exactly fall under the LP-Type definition (since there is no optimization function), we modify our general algorithm slightly to given an algorithm with query complexity is $O(d/\epsilon)$. We also modify our approach for this problem to give a *tolerant* tester for linear program feasibility. This passes programs which are ϵ/c-close to feasible for some fixed constant $c > 1$, and rejects those which are ϵ-far from feasible. This tester also has query complexity $O(d/\epsilon)$
2. We study the problem of determining if a set of points in d dimensions labeled $\{+1, -1\}$ is linearly separable or if at least ϵ-fraction of the points need to be relabeled or removed for the points to be linearly separable. Using result 1 above, we directly get a query complexity bound of $O(d/\epsilon)$. We also give a matching lower bound for this problem which implies a lower bound for result 1.
3. We obtain a result for property testing of many classical LP-Type problems. In particular, we consider the following problems:
   - *Smallest enclosing ball*: Accept if a set of points in $\mathbb{R}^d$ can be covered by a ball of radius r and reject if at least ϵ-fraction of the points need to be removed to be able to be covered by a ball of radius r. This problem has been previously studied in [1]. We improve upon the upper bound obtained in this paper in the case of two-sided error by getting a tight query complexity of $O(d/\epsilon)$ queries (also see point 4 below). Note that the algorithm in [1] has one sided error but has slightly worse query complexity.
   - *Smallest intersecting ball*: Accept if a set of closed convex bodies in $\mathbb{R}^d$ can all be intersected by a ball of radius r and reject if at least ϵ-fraction of the convex bodies need to be removed to be able to be intersected by a ball of radius r.
   - *Smallest volume annulus*: Accept if a set of points in $\mathbb{R}^d$ can be enclosed in an annulus of volume V and reject if at least ϵ-fraction of the points need to be removed to be encloseable by an annulus of volume V.

   In all these cases, it is known that the dimension of the LP-Type problem is linearly related to the dimension of the points in S, so we get an upper bound of $O(d/\epsilon)$ queries.
4. We get a matching lower bound of $\Omega(d/\epsilon)$ queries for the smallest enclosing ball problem and the smallest intersecting ball problem. This provides a lower bound for the radius cost of clustering considered by Alon et al. in [1] in the case of 1 cluster.

▶ Remark 6. Note that there are also many examples of LP-Type problems where the constraints in S describe points in dimension d but the dimension of the LP-Type problem is not a linear function of d. For example, if $\varphi(S)$ is the smallest ellipsoid that encloses the set of points in S which are in $\mathbb{R}^d$, then (S, φ) has dimension $O(d^2)$ as a LP-Type problem [12]. We did not explicitly highlight these problems but our approach also gives an upper bound on the query complexity for the property testing versions of these problems.

3 General Algorithm for Property Testing of LP-Type problems

We now present our general algorithm, LP-Type Tester, for property testing of LP-Type problems as defined in Definition 4. Given a LP-Type problem (S, φ), Our algorithm first samples a subset R of $O(\delta/\epsilon)$ constraints from S where δ is the dimension of the LP-Type problem. It then calculates the value of φ on the sampled subset. After this step, an additional $O(1/\epsilon)$ constraints are sampled randomly from S. If $\varphi(R \cup \{x\})$ differs from $\varphi(R)$ where x is any of the additional random constraints, then our algorithm outputs reject. Otherwise, the algorithm outputs accept. We present our approach in Algorithm 1 along with our main theorem, Theorem 7 which proves the correctness of Algorithm 1.

Algorithm 1 LP-Type Tester.

```
Input  : $\delta, \epsilon, k$, query access to constraints in $S$
Output : accept or reject
$r \leftarrow \lceil 10\delta/\epsilon \rceil$
$R \leftarrow$ random sample of size $r$ of constraints from $S$.
if $\varphi(R) > k$ then
    Output reject and abort.
for $2/\epsilon$ rounds do
    $x \leftarrow$ uniformly random constraint of $S \setminus R$
    if $\varphi(R \cup \{x\}) \neq \varphi(R)$ then
        Output reject and abort.
Output accept.
```

▶ **Theorem 7** (Correctness of LP-Type Tester). *Given an LP-Type problem (S, φ) of dimension δ and parameters k and ϵ, the following statements hold with probability at least $2/3$:*

- ***Completeness case:*** *LP-Type Tester outputs accept $\varphi(S) \leq k$.*
- ***Soundness Case:*** *LP-Type Tester outputs reject if at least $\epsilon|S|$ constraints need to be removed from S for $\varphi(S) \leq k$ to hold.*

▶ Remark 8. Note that the query complexity of Algorithm 1 is $O(\delta/\epsilon)$ which is independent of $|S|$, the number of constraints.

3.1 Overview of the proof

To prove the correctness of LP-Type Tester, we analyze the completeness case and the soundness case separately. For the soundness case, we show that with sufficiently large probability, either $\varphi(R) > k$ or LP-Type Tester outputs reject during the second sampling phase where we sample an additional $O(1/\epsilon)$ constraints. To show this, we use the locality property of LP-Type problems (see Definition 1) to show that there must be "many" x such that $\varphi(R \cup \{x\}) \neq \varphi(R)$. To analyze the completeness case, we use the *Sampling Lemma*, Lemma 11, to show that there are "few" x such that $\varphi(R \cup \{x\}) \neq \varphi(R)$ so that Algorithm 1 outputs accept with sufficiently large probability.

Before we present the proof of Theorem 7, we present the Sampling Lemma as described above. This lemma has previously been used to study LP-Type problems. For completeness, we present a proof. For more information, see [26, 18, 11, 10]. Before we present the lemma, we introduce two new definitions.

▶ **Definition 9** (Violators and Extreme Elements). *For a subset $R \subseteq S$, define the violators and extreme elements of R as the following:*

- *Define the* ***violators*** *of R as the set $V(R) = \{s \in S \backslash R \mid \varphi(R \cup \{s\}) \neq \varphi(R)\}$.*
- *Define the* ***extreme elements*** *of R as the set $X(R) = \{s \in R \mid \varphi(R) \neq \varphi(R \backslash \{s\}\}$.*

▶ Remark 10. Note that s is a violator of R if and only if s is an extreme element of in $R \cup \{s\}$.

We now present the Sampling Lemma.

▶ **Lemma 11** (Sampling Lemma). *Let $v_r = \mathbb{E}[|V(R)|]$ and $x_r = \mathbb{E}[|X(R)|]$ where both expectations are taken over the random subsets R of S which have size r. Suppose $|S| = n$. Then for $0 \le r \le n$, we have*

$$\frac{v_r}{n-r} = \frac{x_{r+1}}{r+1}.$$

Proof. Let $\mathbf{1}\{\cdot\}$ denote an indicator variable. Note that

$$\begin{aligned}\binom{n}{r} v_r &= \sum_{R \in \binom{S}{r}} \sum_{s \in S \setminus R} \mathbf{1}\{s \text{ is a violator of } R\} = \sum_{R \in \binom{S}{r}} \sum_{s \in S \setminus R} \mathbf{1}\{s \text{ is extreme for } R \cup \{s\}\} \\ &= \sum_{Q \in \binom{S}{r+1}} \sum_{s \in Q} \mathbf{1}\{s \text{ is extreme for } Q\} = \binom{n}{r+1} x_{r+1}.\end{aligned}$$

The proof follows from the following calculation.

$$\frac{\binom{n}{r+1}}{\binom{n}{r}} = \frac{r!(n-r)!}{(r+1)!(n-r-1)!} = \frac{n-r}{r+1}. \qquad ◀$$

▶ Remark 12. Note that (S, φ) does not need to be a LP-Type problem for the Sampling Lemma to hold true.

If (S, φ) is a LP-Type problem, there is a direct relationship between the expected number of violators and the dimension of (S, φ) as defined in 3. The following corollary also appears in many forms in literature (for instance [26, 18, 11, 4]) but we present its proof for completeness.

▶ **Corollary 13.** *Let (S, φ) be a LP-Type problem of dimension δ and let $|S| = n$. If $R \subseteq S$ is subset of size r chosen uniformly at random, then $v_r = \mathbb{E}[|V(R)|]$ satisfies*

$$v_r \le \frac{\delta(n-r)}{r+1}.$$

Proof. We show that for any set $R \subseteq S$, we have $|X(R)| \le \delta$. Then the corollary follows from Lemma 11. Let R' be the smallest subset of R such that $\varphi(R') = \varphi(R)$. We first claim that $V(R') = V(R)$. It is clear that $V(R) \subseteq V(R')$ by monotonicity (see Definition 1). For the other inclusion, consider $x \in V(R')$. If $x \notin V(R)$, we have $\varphi(R \cup \{x\}) = \varphi(R) = \varphi(R')$ so by locality, we have $\varphi(R') = \varphi(R' \cup \{x\})$ which contradicts the fact that $x \in V(R')$. Therefore, our claim holds true.

We now claim that R' is a basis as defined in Definition 2. Suppose for the sake of contradiction that R' is not a basis. Then there exists a $F \subset R'$ such that $\varphi(F) = \varphi(R')$. We now claim that $V(R') = V(F)$. It is clear that $V(R') \subseteq V(F)$. To show the other inclusion, let $x \in V(F)$. Then if x was not a violator of R', then $\varphi(R' \cup \{x\}) = \varphi(R') = \varphi(F)$ which would imply that $\varphi(F \cup \{x\}) = \varphi(F)$ by the locality property in Definition 1 which is false by definition. Hence, $V(F) = V(R') = V(R)$ which contradicts the minimality of R'. Therefore, R' is a basis.

Finally, we claim that if $x \in X(R)$ then $x \in R'$. This must be true because otherwise, we have $R' \subseteq R \setminus \{x\} \subseteq R$ which results in a contradiction by monotonicity. Finally, since $X(R) \subseteq R'$ and R' is a basis, it follows that $|X(R)| \le \delta$, as desired. ◀

▶ Remark 14. Corollary 13 holds for a larger class of problems than LP-Type problems called *violator spaces* ([11, 4]. However, we omitted this extra layer of abstraction since there are no additional natural property testing consequences from considering violator spaces over LP-Type problems.

Proof of Theorem 7. We first prove the soundness case. Consider the set R that was randomly sampled in step 2 of LP-Type Tester. Assume that $\varphi(R) \le k$ since this can only decrease the probability that our algorithm outputs reject. Now we claim that there must be at least $\epsilon|S|$ choices of x in step 6 of LP-Type Tester that results in $\varphi(R \cup \{x\}) > \varphi(R)$ (so that we correctly output reject). To show this, note that if $\varphi(R) = \varphi(R \cup \{x\}) = \varphi(R \cup \{y\})$ for $x \neq y$ then by locality, it follows that $\varphi(R) = \varphi(R \cup \{x, y\})$. Therefore, if there are less than $\epsilon|S|$ choices of x in step 6 of LP-Type Tester for some R such that $\varphi(R \cup \{x\}) > \varphi(R)$, then we would have $\varphi(R \cup R') = \varphi(R) \le k$ where $|R \cup R'| \ge (1-\epsilon)|S|$ which would contradict our assumption that at least $\epsilon|S|$ constraints need to be removed from S for $\varphi(S) \le k$ to hold true. Therefore, the probability our algorithm does not output reject in any of the $2/\epsilon$ rounds is at most

$$(1-\epsilon)^{2/\epsilon} \le e^{-2} < \frac{1}{3} \tag{1}$$

which means that we output reject with probability at least 2/3, as desired.

We now prove Theorem 7 for the completeness case. Let $v_r = \mathbb{E}[|V(R)|]$. Since $r = |R| = 10\delta/\epsilon$, Corollary 13 gives us

$$v_r \le \frac{\delta(|S| - r)}{r+1} \le \frac{\epsilon|S|}{10}.$$

Therefore in the completeness case, the probability that a randomly chosen x satisfies $\varphi(R \cup \{x\}) \neq \varphi(R)$ is at most $\epsilon/10$. Since we choose $2/\epsilon$ random constraints, the probability we don't find such a x is at least

$$(1-\epsilon/10)^{2/\epsilon} \ge 1 - \frac{2}{10} > \frac{2}{3}. \tag{2}$$

Therefore, LP-Type Tester outputs accept with probability at least 2/3, as desired. ◀

4 Property Testing Applications of LP-Type Tester

We now give applications of the framework we build in Section 3. We first consider the problem of testing feasibility of a set of linear inequalities. As a direct consequence, we can test if a set of labeled points can be linearly sepearable (either by linear hyperplanes or by functions that have a finite basis). These two applications will not be an immediate corollary of Theorem 7 since there is no objective function that we want to optimize, but our results follow from Theorem 7 with some slight modificatons.

We then consider direct applications of Algorithm 1 to some cannonical LP-Type problems such as the smallest enclosing ball. Theorem 7 gives direct upper bounds for property testing for these problems.

4.1 Testing Feasibility of a System of Linear Equations

We first begin by considering testing feasibility of a set of linear inequalities. Recall that in this problem, we have n linear constraints in $\mathbb{R}^d$ (such as $x_1 + \cdots + x_d \le 1$) and we want to distinguish the following two cases with probability at least 2/3:

- The system of linear inequalities can all be mutually satisfied, i.e., the system is feasible (Completeness Case)
- At least $\epsilon|S|$ many of the constraints need to be removed (or flipped) for the system to be feasible (Soundness Case).

This is not exactly a LP-Type problem since we do not have an optimization function φ. We note that if φ was an indicator function for a subset of constraints being feasible then φ would break the locality condition in Definition 1. One way to see this is consider the case when A is the single constraint that $y \geq 0$, B is the set of two constraints $0 \leq y$ and $y \leq 1$, and x is the additional constraint that $y \geq 2$. We can see that A, B, and $A \cup \{x\}$ are all feasible, but $B \cup \{x\}$ is not, contradicting locality. However, we perform a slight modification of Algorithm 1 to create a new algorithm for this problem.

Our algorithm for this testing problem, Linear Feasibility Tester, uses the fact that if we pick any arbitrary $x \in \mathbb{R}^d$, then x will violate "many" of the linear constraints in S in the completeness case. In the soundness case, we use ideas from LP-Type Tester and show that if we introduce an arbitrary linear optimization function (thus turning our problem into an instance of linear programming), then a solution that optimizes a small subset of the constraints will not violate "too many" of the other constraints. We present our algorithm below along with Theorem 15 that proves its correctness.

Algorithm 2 Linear Feasibility Tester.

```
Input   : $d, \epsilon$, query access to constraints of $S$
Output : Accept or Reject
$r \leftarrow \lceil 10d/\epsilon \rceil$
$R \leftarrow$ random sample of size $r$ of constraints from $S$
Create the linear program $L$: $\max x_1$ subject to the constraints in $R$
$x \leftarrow$ solution of $L$
if $L$ is not feasible then
    Reject and abort
for $2/\epsilon$ rounds do
    $y \leftarrow$ uniformly random constraint of $S$
    if $x$ does not satisfy $y$ then
        Output reject and abort.
Output accept.
```

▶ **Theorem 15** (Correctness of Linear Feasibility Tester). *Given a set S of linear inequalities in $\mathbb{R}^d$, the following statements hold with probability at least $2/3$:*

- ***Completeness case:*** *Linear Feasibility Tester outputs accept if there exists $x \in \mathbb{R}^d$ that satisfies all of the constraints in S.*
- ***Soundness Case:*** *Linear Feasibility Tester outputs reject if at least $\epsilon|S|$ constraints need to be removed from S for S to be feasible.*

▶ Remark 16. Note that the query complexity of Algorithm 1 is $O(d/\epsilon)$ which is independent of $|S|$, the number of constraints. Furthermore, the runtime is polynomial in d/ϵ since we are solving a linear programs in d variables and $O(d/\epsilon)$ constraints.

Proof. The proof of the soundness case follows similarly to Theorem 7 using the fact that for any x, there are at least $\epsilon|S|$ constraints in x such that x violates these constraints. Then the probability that Linear Feasibility Tester outputs reject in this case can be calculated to be at least $2/3$ using the same bound as Eq. (1) in the proof of Theorem 7.

For the completeness case, we note that if we introduce the optimization function $\varphi(S) = \max x_1$ subject to the constraints in S, then (S, φ) is an LP-Type problem of dimension d (assuming that the constraints are non degenerate which can be assumed by perturbing the constraints and then taking the limit of the perturbation to 0. For more details, see [6, 25]). Now let x be the solution to the linear program that we solved in Step 4 of Linear Feasibility Tester. Using Corollary 13, we know that if $|R| = 10\lceil d/\epsilon \rceil$, then the number of constraints v_r in S that satisfy $\varphi(R \cup \{y\}) \neq \varphi(R)$ is at most $v_r \leq (d|S|)/(10d/\epsilon) = \epsilon|S|/10$ in expectation. Knowing that x not satisfying y implies that y is a violator of R, the probability that x does not satisfy a randomly chosen y is at most $\epsilon/10$. Thus, using the exact calculation as in Eq.(2) of Theorem 7, we have that Linear Feasibility Tester outputs accept in the completeness case with probability at least 2/3, as desired. ◀

In Section 6 we give a *tolerant* tester for testing linear feasibility. A tolerant tester accepts instances that are ϵ-close and rejects instances that are $c\epsilon$-far for some constant $c > 1$.

4.2 Testing if Labeled Points can be Linearly Separated

As a direct consequence of the Theorem 15, we can test if a set of points in d dimensions labeled $\{+1, -1\}$ can be linearly separated. More formally, we have the following corollary.

▶ **Corollary 17.** *Given a set S of points in $\mathbb{R}^d$ with labels in $\{+1, -1\}$, the following statements hold with probability at least $2/3$:*

- ***Completeness case:*** *Linear Feasibility Tester outputs accept if there exists a hyperplane that separates the two sets of labeled points.*
- ***Soundness Case:*** *Linear Feasibility Tester outputs reject if at least $\epsilon|S|$ points need to be removed (or relabeled) for S to be linearly sepearable.*

Proof. The proof follows directly from the fact that we can write a linear inequality that represents a separating hyperplane. For example, if $p \in S$ is labeled 1, we want to find x such that $p^T x \geq 1$ and if p is labeled -1, we want to find x such that $p^T x \leq -1$. ◀

We consider generalizations of this problem where we wish to separate labelled points by arbitrary functions, rather than just linear hyperplanes. In Section 7 we address the issue of separating using arbitrary functions when we know the basis of the functions, and the case of multiple labels.

4.3 Upper Bounds for Canonical LP-Type Problems

We now give direct applications of LP-Type Tester to some canonical LP-Type problems. The correctness of these applications follows directly from Theorem 7. Our list is not exhaustive and we only consider some of the more well known LP-Type problems. In all of the following problems, Theorem 7 tells us that the following statements hold with probability at least 2/3:

- LP-Type Tester outputs accept if $\varphi(S) \leq k$ (Completeness Case)
- LP-Type Tester outputs reject if at least $\epsilon|S|$ constraints need to be removed from S for $\varphi(S) \leq k$ to hold (Soundness Case).

Our results are the following:

- *Smallest enclosing ball*: In this problem, $\varphi(S)$ is the radius of the smallest enclosing ball of a set of points S in $\mathbb{R}^d$. It is known that the dimension of this LP-Type problem is $d + 1$ (see [12]) so we can test if $\varphi(S) \leq k$ with query complexity $O(d/\epsilon)$ queries.

- *Smallest intersecting ball*: In this problem, $\varphi(S)$ is the smallest radius ball that intersects a set of closed convex bodies S in $\mathbb{R}^d$. The dimension of this LP-Type problem is $O(d)$ ([12]) so we can test if $\varphi(S) \leq k$ with query complexity $O(d/\epsilon)$ queries.
- *Smallest volume annulus*: In this problem, $\varphi(S)$ is the volume of the smallest annulus that contains a set of points S in $\mathbb{R}^d$. Again, the dimension of this LP-Type problem is $O(d)$ ([12]) so we can test if $\varphi(S) \leq k$ with query complexity $O(d/\epsilon)$.

5 Lower Bounds

In this section, we give matching lower bounds for all the testing problems that we considered in Section 4.

5.1 Lower Bound for Testing Feasibility of Linear Constraints

Since linear separability is a special case of feasibility of linear constraints, we can lower bound the necessary query complexity of the latter by providing one for the former. In particular, we aim to show that $\Omega(d/\epsilon)$ queries are needed to determine if a set of points in d dimensions is linearly separable. By the reduction of linear separability to feasibility of linear constraints, this implies that $\Omega(d/\epsilon)$ constraint queries are needed to test feasibility of a system of linear constraints, which matches our upper bound.

Our overall approach is to first introduce a set of $O(d)$ points in $\mathbb{R}^d$ that have the property that if we do not look at a large enough collection of these points, they can be separated by a hyperplane even with arbitrary labels. However, there will exist a labeling of all of the points such that "many" of the points will have to be removed or relabeled for this labeling to be separated. The existence of these points is given in Lemma 18 (and is inspired by the moment curve).

Then, repeating these points with carefully chosen multiplicities allows us to construct our set S of points. Then a coupon collector argument gives us our desired lower bound on the query complexity. This argument is formalized in the proof of Theorem 19.

▶ **Lemma 18.** *There exists a set S of $3d+1$ points in $\mathbb{R}^d$ that satisfy the following conditions:*

1. *There exists a labeling of the points of S such that at least d points have to be relabeled for the points to be linearly separable.*
2. *Any subset of points of S of size $d+1$ with arbitrary labels in $\{-1, 1\}$ is linearly separable.*

Proof. We construct our set S as follows. Let x_i be the point $(i^1, \cdots, i^d) \in \mathbb{R}^d$ for $1 \leq i \leq 3d+1$ (note that this set of points is referred to as the moment curve). We prove the first claim using a standard relationship between the moment curve and polynomials. Assign the point x_i to the label $(-1)^i$. Let k be the number of relabeled points such that S is linearly separable. Then there exists $w \in \mathbb{R}^d$ and $w_0 \in \mathbb{R}$ such that $\text{Sign}(x_i^T w + w_0)$ matches the label of every point $x_i \in S$. In other words, there exists a polynomial $P(x) = \sum_{j=0}^{d} c_j x^j$ such that $\text{Sign}(P(i))$ matches the label of x_i. Now note that if there are two consecutive indices i and $i+1$ that have different labels, then P must have a root in the interval $(i, i+1)$. Originally, there are $3d$ such alternating intervals. Now note that the relabeling of any point can decrease the total number of such alternating intervals by at most 2. Hence after k relabelings, there must be at least $3d - 2k$ alternating intervals. However, since P is a d degree polynomial, it must have at most d roots which means $3d - 2k \leq d$ and therefore, $k \geq d$, as desired.

We now prove the second claim. Let $x_{a_1}, \cdots, x_{a_{d+1}}$ be a subset of $d+1$ points of S. Without loss of generality, suppose that $a_1 < \cdots < a_{d+1}$. We now show that for every labelings of these $d+1$ points, there exists a polynomial of degree d such that the sign of $P(a_i)$ matches the label of x_{a_i}. Towards this goal, pick t elements $b_1, \cdots, b_t$ of the set $\{a_2, \cdots, a_{d+1}\}$ where $t \le d$. Consider the $t+1$ intervals

$$[a_1, b_1), [b_1, b_2), \cdots, [b_{t-1}, b_t), [b_t, a_{d+1}+1).$$

We can then find a polynomial of degree d such that

- the sign of P is constant on $I \cap \{a_1, a_2, \cdots, a_{d+1}\}$ where I is any of the $t+1$ intervals above,
- the sign of P alternates between consecutive intervals.

This is possible since we are only specifying the value of P on $d+1$ locations. Now the total number of labelings described by all possible choices of P is given by $2\sum_{t=0}^{d}\binom{d}{t} = 2^{d+1}$ where the factor of 2 comes from specifying the sign of P on the first interval. Note that 2^{d+1} is exactly the total number of different ways to label $d+1$ points, which proves the second claim. ◀

With Lemma 18 on hand, we can prove our desired lower bound on the query complexity.

▶ **Theorem 19.** *Any algorithm that tests if a set S of labeled points in d dimensions can be linearly separated requires $\Omega(d/\epsilon)$ queries.*

Proof. Let $|S| = n$. We create two families of n points in $\mathbb{R}^d$ with a specific labeling such that any S from one family can be linearly separated while any S from the other family is ϵ-far from being linearly separable. First, consider the set of $3d+1$ points supplied by Lemma 18 and the labeling from part 1 of the lemma. The first family $\mathcal{F}_1$ consists of picking a subset of $d+1$ of these points (with the labeling above), repeating d of these points $n\epsilon/d$ times, and repeating the remaining point $(1-\epsilon)n$ times. The second family $\mathcal{F}_2$ (again with the same labeling) consists of picking all of the $3d+1$ points from Lemma 18, repeating some $3d$ of these points with multiplicity $n\epsilon/(3d)$, and repeating the last point with multiplicity $(1-\epsilon)n$.

By Lemma 18, we know that if S is from $\mathcal{F}_1$ then S is linearly separable while if S is from $\mathcal{F}_2$, then S is at least $\epsilon/(3d) \cdot d = O(\epsilon)$-far from separable. Any algorithm that queries points randomly must discover at least $d+1$ *unique* points out of the points that were repeated $n\epsilon/d$ time from any S in $\mathcal{F}_2$ to discover that this S is $O(\epsilon)$-far from separable (otherwise, the points look separable). Call points that are identical "groups". Now given a random point from S, the probability of hitting any one group is $\epsilon/(3d)$. Therefore by coupon collector, the expected number of queries required to hit at least $d+1$ of these $3d$ groups is at least

$$\frac{1}{\epsilon}\left(\frac{3d}{3d} + \frac{3d}{3d-1} + \cdots + \frac{3d}{3d-d}\right) = \frac{3d}{\epsilon}(H_{3d} - H_{2d-1}) = \Theta\left(\frac{d}{\epsilon}\right). \quad ◀$$

As a corollary, we have the following lower bound as well. This is due to the reduction from linear separability to linear program feasibility from the proof of Corollary 17.

▶ **Theorem 20.** *Any algorithm that tests if n linear inequalities in d dimensions are feasible requires $\Omega(d/\epsilon)$ queries.*

We now give matching query complexity lower bounds for the LP-Type problems that we considered in Section 4.

5.2 Lower bound for Testing Smallest Enclosing Ball

We first give a lower bound for property testing the radius of the smallest enclosing ball of a set of points. Our approach is to first construct a set of points in $\mathbb{R}^j$, for any j, whose smallest enclosing ball can be calculated exactly. This set of points will have the property that a small enough subset of the points will have a significantly smaller enclosing ball. Therefore, if an algorithm does not query enough points, it will incorrectly believe that this set of points can be covered by a ball of small radius. Our construction for this case will be a regular simplex and explained below. First we prove an auxiliary lemma.

▶ **Lemma 21.** *The radius of the circumcircle of a unit simplex in $\mathbb{R}^j$ is $\sqrt{j}/(\sqrt{2(j+1)})$.*

Proof. Note that we can embed a regular j-simplex in $\mathbb{R}^{j+1}$ using the coordinates $\{e_i\}_{i=1}^{j+1}$ where e_i is the all zero vector with a single 1 in the ith coordinate. This simplex has edge length $\sqrt{2}$ so we can scale appropriately to find the circumcircle of a unit simplex. Now the centroid of this simplex is easily seen to be located at $(1/(j+1), \cdots, 1/(j+1))$ which means that the circumcircle has radius

$$\sqrt{\left(1-\frac{1}{j+1}\right)^2+\frac{j}{(j+1)^2}}=\sqrt{\frac{j}{j+1}}.$$

Now scaling by $1/\sqrt{2}$ gives us the desired value. ◀

▶ **Theorem 22.** *Any algorithm that tests if a set of n points in $\mathbb{R}^d$ can be enclosed by a ball of radius k, where k is given, requires $\Omega(d/\epsilon)$ queries.*

Proof. Let k be fixed. We construct two families of points in $\mathbb{R}^{O(d)}$ such that any S from one family can be enclosed by a ball of radius k while any S from the second family is ϵ-far from being enclosed by a ball of radius k. Before constructing these families, we first pick ℓ such that the regular simplex of side length ℓ in $\mathbb{R}^{d+1}$ has circumradius k.

Now to create the first family $\mathcal{F}_1$, we first pick any $d+1$ points of the regular simplex with side length ℓ in $\mathbb{R}^{3d+1}$. Then we repeat one of these points with multiplicity $(1-\epsilon)n$ and we repeat the other d points with multiplicity $n\epsilon/d$ each. To create the second family $\mathcal{F}_2$, we pick a point of the regular simplex with side length ℓ in $\mathbb{R}^{3d+1}$, repeat it with multiplicity $(1-\epsilon)n$, and repeat the other $3d$ points with multiplicity $n\epsilon/(3d)$. Finally, let S be a set of n points from $\mathcal{F}_2$. From Lemma 21, we can check that the circumradius of a regular unit simplex is an increasing function of the dimension and that any subset of the vertices of a regular simplex is a regular simplex itself. Therefore, the smallest radius of the points in S is much larger than k and S is $O(\epsilon)$-far from being encloseable by a ball of radius k. However, similar to the argument in Theorem 19, any algorithm that rejects S must have discovered at least $d+1$ distinct "groups" of repeated points. By the same coupon collector argument as in the proof of Theorem 19, we have that this task takes at least $\Omega(d/\epsilon)$ queries in expectation. ◀

As a simple application of Theorem 22, we get the following lower bounds as well.

▶ **Corollary 23.** *Any algorithm for testing the smallest intersecting ball for n convex bodies in $\mathbb{R}^d$ requires $\Omega(d/\epsilon)$ queries.*

Proof. The proof follows from the fact that a set of singleton points is also a set of convex bodies. In this case, the smallest intersecting ball is equivalent to the smallest ball that encloses these points. Therefore, the same lower bound as in Theorem 22 holds. ◀

6 Tolerant Tester for Testing Feasibility of Linear Constraints

We generalize our argument in Section 4.1 by giving a *tolerant* tester for testing feasibility of a system of linear constraints. In the tolerant version, we output accept if there only "few" constraints need to be removed for a set of linear inequalities to be feasible. More formally, we wish to distinguish the following two cases with probability at least 2/3:

- At most $c\epsilon|S|$ many inequalities in S need to be removed for S (or flipped) for S to be feasible, i.e., S is $c\epsilon$-close to being feasible for some fixed positive $c < 1$ (Completeness Case).
- At least $\epsilon|S|$ many of the constraints need to be removed (or flipped) for the system to be feasible (Soundness Case).

Our approach is a slightly modified version of Linear Feasibility Tester, Algorithm 2, that we presented in Section 4.1. The challenge here is the completeness case where we must accept if we only have a "few" bad constraints. To accomplish this, we carefully select a solution to a small linear program that we run. For more details, see Algorithm 3. Our main theorem in this section, Theorem 24 shows that we can perform *tolerant* testing using the same query complexity we used for the non tolerant tester in Section 4.1, namely $O(d/\epsilon)$. However, as we will explain below, the *running time* of Algorithm 3, Tolerant Linear Feasibility Tester, is exponential in the running time of Algorithm 2. Our algorithm, Tolerant Linear Feasibility Tester, is presented below.

Algorithm 3 Tolerant Linear Feasibility Tester.

```
   Input   : $d, \epsilon$, query access to constraints of LP
   Output : Accept or Reject
1  $r \leftarrow \lceil 10d/\epsilon \rceil$
2  $R \leftarrow$ random sample of size $r$ of constraints from $S$
3  $x \leftarrow$ solution of the largest subset $R'$ of $R$ such that the linear program $L$: $\max x_1$
     subject to the constraints in $R'$ is feasible
4  if No $L$ is not feasible then
5      Reject and abort
6  for $2/\epsilon$ rounds do
7      $y \leftarrow$ uniformly random constraint of $S$
8      if $x$ does not satisfy $y$ then
9          Output reject and abort.
10 Output accept.
```

Unlike Linear Feasibility Tester where we run a linear program, we solve a slightly different program given in step 3 of Tolerant Linear Feasibility Tester. The step determines the largest feasible subset of these constraints. Note that this step is clearly exponential in the number of constraints (which is $O(d/\epsilon)$). Therefore, the overall *runtime* of Tolerant Linear Feasibility Tester will be exponential in the runtime of Linear Feasibility Tester. The correctness of Tolerant Linear Feasibility Tester is proven in Theorem 24.

▶ **Theorem 24** (Correctness of Tolerant Linear Feasibility Tester)**.** *Given a set S of linear inequalities in $\mathbb{R}^d$, there exists a constant $c < 1$ such that the following statements hold with probability at least $2/3$:*

- ***Completeness case:*** *Tolerant Linear Feasibility Tester outputs accept if there exists $x \in \mathbb{R}^d$ that satisfies $(1-c\epsilon)|S|$ of the constraints in S.*
- ***Soundness Case:*** *Tolerant Linear Feasibility Tester outputs reject if at least $\epsilon|S|$ constraints need to be removed from S for S to be feasible.*

▶ Remark 25. Note that the query complexity of Algorithm 3 is $O(d/\epsilon)$ which is independent of $|S|$, the number of constraints.

Proof. Note that the proof of the soundness case is identical to the proof of the soundness case in Theorem 15 since for any x we find in step 3 of Tolerant Linear Feasibility Tester, there exists at least $\epsilon|S|$ choices of y in step 7 such that x does not satisfy the constraint y. Then a similar calculation as in Eq. (1) implies that we reject with probability at least $2/3$.

We now focus on the completeness case where we know there is a subset of $(1-c\epsilon)|S|$ constraints that are feasible. We call this the *good* set, and the rest, the *bad* set. Consider the sample R from step 2 of Tolerant Linear Feasibility Tester. The expected number of constraints from the good set in R is $(1-c\epsilon)r$. This means at most $c\epsilon r$ constraints in R come from the bad set in expectation. Hence with probability at least $9/10$, we know that the number of constraints from the bad set is at most $10c\epsilon r$ by Markov's inequality, which means the number of constraints coming from the good set is at least $(1-10c\epsilon)r$. We condition on this event. Now note that one valid subset R' to use in step 3 of Tolerant Linear Feasibility Tester is to take all the constraints coming from the good set only. This results in $|R'| \geq (1-10c\epsilon)$. Since we are maximizing $|R'|$, this means that at most $10c\epsilon r$ of the constraints coming from the good set that are in R will not be included in R'. Thus, x satisfies at least $(1-20c\epsilon)r$ constraints in the good set with probability at least $9/10$. Now we proceed similarly as the proof of Theorem 15. By Corollary 13, the probability that x violates any other constraint in the good set is at most

$$\frac{d(n'-r+1)}{n'(r-d)} \leq \frac{dn'}{10dn'/\epsilon} = \frac{\epsilon}{10}$$

where n' is the size of the good set. Furthermore, x can possibly violate any constraint in the bad set which means that the probability x violates any other constraint is at most $\epsilon/10 + c\epsilon < \epsilon/6$ for sufficiently small c, i.e. $c < 1/15$. Then, the probability that we find such a constraint in $2/\epsilon$ rounds is at most

$$1 - \left(1 - \frac{\epsilon}{6}\right)^{2/\epsilon} \leq 1 - \left(1 - \frac{1}{3}\right) = \frac{1}{3}.$$

Therefore, we accept with probability at least $2/3$, as desired. Note that we can take any $c < 1/15$ in the statement of the Theorem for instance. ◀

7 Separating Points with Arbitrary Functions and Multiple Labels

7.1 Separating labeled points using arbitrary functions

We can generalize our result from Section 4.2 by separating labeled points using arbitrary *functions*: given a family of functions $\mathcal{F}$, we can ask if there is a $f \in \mathcal{F}$ such that $f(p) > 0$ for all points with a particular label and $f(p) < 0$ for all the points with the other label.

We now translate this problem to a setting with linear inequalities. Our approach is standard in machine learning and is known as feature maps. If the family $\mathcal{F}$ has a finite basis $f_1, \cdots, f_k$, meaning that every $f \in \mathcal{F}$ is a linear combination of $f_1, \cdots, f_k$, then we can create a system of linear inequalities as follows. For each point $p \in S$, we can make a new constraint which is $(f_1(p), \cdots, f_k(p))x \geq 1$ (note there that x is a column vector of variables) if p has one particular label or ≤ -1 if p has another label. Then this system of linear constraints is feasible iff there are scalars $a_1, \cdots, a_k$ such that $\sum_i a_i f_i(p) \geq 0$ for all p with one label and $\sum_i a_i f_i(p) \leq 0$ for all p with the other label. Then our separating function is precisely $f = \sum_i a_i f_i$. Note that in this formulation, we have k variables. Thus, the query complexity is $O(k/\epsilon)$.

As an example, we consider the case that $\mathcal{F}$ is the family of polynomials in d variables with degree $\leq t$. The basis of this family is all the possible terms of the form $x_1^{t_1} \cdots x_d^{t_d}$ where the t_i are non-negative and add to at most t. By a standard balls and bins argument, the number of these terms is $\binom{t+d}{d}$. For constant t, this is $O(d^t)$, which means that our system of linear constraints has $O(d^t)$ variables. Thus, the query complexity is $O(d^t/\epsilon)$.

7.2 Separating Points with Multiple Labels

Suppose that in Section 4.2, instead of assigning each point one of 2 labels, we instead chose to assign it one of $\ell \geq 2$ labels. One common interpretation of separability for this setup is to check if each of the $\binom{\ell}{2}$ pairs of label sets are separable. We modify our notion of ϵ-far to reflect this.

▶ **Definition 26.** *S is ϵ-far from linearly separable if at least $\epsilon|S|$ many labels in S have to be changed for S to be separable.*

If such a data set is ϵ-far from separable, then some subset with consisting of two labels must be $\epsilon/\binom{\ell}{2}$-far from separable. As such, we can consider an algorithm that runs Algorithm Linear Feasibility Tester on each pair of labels with $\epsilon' = \epsilon/\binom{\ell}{2}$ and outputs accept if all these tests output accept. We need to reduce the error probability for each pair such that the overall error probability of outputting the incorrect answer (acquired by a union bound) is still at most 1/3. This can be done by using a stronger version of the original algorithm where we run it $O(\log \ell)$ times and taking the majority answer. By a standard Chernoff bound argument, the probability this process gives the wrong answer is at most say $1/\ell^3$. Thus, we can distinguish separability in this case by running this stronger version over all pairs of distinct labels, resulting in $O(\ell^2 \log \ell)$ instances of Linear Feasibility Tester, using $\epsilon' = \epsilon/\binom{\ell}{2}$. So, the total query complexity will be $O(d\ell^4 \log \ell/\epsilon)$.

Additionally, the completeness case has error at most $\binom{\ell}{2} 1/\ell^3 = o(1)$ by a Union Bound argument. Clearly, the soundness case has error at most $1/\ell^3$, since there is one pair of distinct labels which is ϵ'-far from separable.

References

1 N. Alon, S. Dar, M. Parnas, and D. Ron. Testing of clustering. In *Proceedings 41st Annual Symposium on Foundations of Computer Science*, pages 240–250, November 2000. doi:10.1109/SFCS.2000.892111.

2 Piotr Berman, Meiram Murzabulatov, and Sofya Raskhodnikova. The power and limitations of uniform samples in testing properties of figures. *Algorithmica*, 81(3):1247–1266, March 2019. doi:10.1007/s00453-018-0467-9.

3 Piotr Berman, Meiram Murzabulatov, and Sofya Raskhodnikova. Testing convexity of figures under the uniform distribution. *Random Structures & Algorithms*, 54(3):413–443, 2019. doi:10.1002/rsa.20797.

4 Yves Brise and Bernd Gärtner. Clarksons algorithm for violator spaces. *CoRR*, abs/0906.4706, 2009. arXiv:0906.4706.

5 H. Chen, M. Valeriote, and Y. Yoshida. Testing assignments to constraint satisfaction problems. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 525–534, October 2016. doi:10.1109/FOCS.2016.63.

6 Kenneth L. Clarkson. Las vegas algorithms for linear and integer programming when the dimension is small. *J. ACM*, 42(2):488–499, March 1995. doi:10.1145/201019.201036.

7 Artur Czumaj and Christian Sohler. Property testing with geometric queries. In Friedhelm Meyer auf der Heide, editor, *Algorithms — ESA 2001*, pages 266–277, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

8 Artur Czumaj and Christian Sohler. Abstract combinatorial programs and efficient property testers. *SIAM J. Comput.*, 34(3):580–615, March 2005. doi:10.1137/S009753970444199X.

9 Artur Czumaj, Christian Sohler, and Martin Ziegler. Property testing in computational geometry. In Mike S. Paterson, editor, *Algorithms - ESA 2000*, pages 155–166, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

10 B. Gärtner and E. Welzl. A simple sampling lemma: Analysis and applications in geometric optimization. *Discrete & Computational Geometry*, 25(4):569–590, April 2001. doi:10.1007/s00454-001-0006-2.

11 Bernd Gärtner, Jiří Matoušek, Leo Rüst, and Petr Škovroň. Violator spaces: Structure and algorithms. In Yossi Azar and Thomas Erlebach, editors, *Algorithms – ESA 2006*, pages 387–398, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

12 Bernd Gärtner and Emo Welzl. Linear programming — randomization and abstract frameworks. In Claude Puech and Rüdiger Reischuk, editors, *STACS 96*, pages 667–687, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

13 Oded Goldreich. *Introduction to Property Testing*. Cambridge University Press, 2017. doi:10.1017/9781108135252.

14 Oded Goldreich, Shari Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *J. ACM*, 45(4):653–750, July 1998. doi:10.1145/285055.285060.

15 Oded Goldreich and Dana Ron. A sublinear bipartiteness tester for bounded degree graphs. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, pages 289–298, New York, NY, USA, 1998. ACM. doi:10.1145/276698.276767.

16 Oded Goldreich and Dana Ron. Property testing in bounded degree graphs. *Algorithmica*, 32(2):302–343, 2002. doi:10.1007/s00453-001-0078-7.

17 Steve Hanneke. The optimal sample complexity of pac learning. *J. Mach. Learn. Res.*, 17(1):1319–1333, January 2016.

18 J. Matoušek, M. Sharir, and E. Welzl. A subexponential bound for linear programming. *Algorithmica*, 16(4):498–516, October 1996. doi:10.1007/BF01940877.

19 Krzysztof Onak. Testing properties of sets of points in metric spaces. In *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part I: Tack A: Algorithms, Automata, Complexity, and Games*, pages 515–526, 2008. doi:10.1007/978-3-540-70575-8_42.

20 Luis Rademacher and Santosh Vempala. Testing geometric convexity. In Kamal Lodaya and Meena Mahajan, editors, *FSTTCS 2004: Foundations of Software Technology and Theoretical Computer Science*, pages 469–480, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

21 Sofya Raskhodnikova. Approximate testing of visual properties. In Sanjeev Arora, Klaus Jansen, José D. P. Rolim, and Amit Sahai, editors, *Approximation, Randomization, and Combinatorial Optimization.. Algorithms and Techniques*, pages 370–381, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

22 Sofya Raskhodnikova. Approximate testing of visual properties. In Sanjeev Arora, Klaus Jansen, José D. P. Rolim, and Amit Sahai, editors, *Approximation, Randomization, and Combinatorial Optimization.. Algorithms and Techniques*, pages 370–381, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

23 Dana Ron. Algorithmic and analysis techniques in property testing. *Foundations and TrendsÂ® in Theoretical Computer Science*, 5(2):73–205, 2010. `doi:10.1561/0400000029`.

24 Dana Ron and Gilad Tsur. Testing properties of sparse images. *ACM Trans. Algorithms*, 10(4), August 2014. `doi:10.1145/2635806`.

25 Raimund Seidel. Small-dimensional linear programming and convex hulls made easy. *Discrete & Computational Geometry*, 6(3):423–434, September 1991. `doi:10.1007/BF02574699`.

26 Micha Sharir and Emo Welzl. A combinatorial bound for linear programming and related problems. In Alain Finkel and Matthias Jantzen, editors, *STACS 92*, pages 567–579, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.

Lower Bounds for Dynamic Distributed Task Allocation

Hsin-Hao Su
Boston College, MA, USA

Nicole Wein
Massachusetts Institute of Technology, Cambridge, MA, USA

Abstract

We study the problem of distributed task allocation in multi-agent systems. Suppose there is a collection of agents, a collection of tasks, and a *demand vector*, which specifies the number of agents required to perform each task. The goal of the agents is to cooperatively allocate themselves to the tasks to satisfy the demand vector. We study the *dynamic* version of the problem where the demand vector changes over time. Here, the goal is to minimize the *switching cost*, which is the number of agents that change tasks in response to a change in the demand vector. The switching cost is an important metric since changing tasks may incur significant overhead.
We study a mathematical formalization of the above problem introduced by Su, Su, Dornhaus, and Lynch [20], which can be reformulated as a question of finding a low distortion embedding from symmetric difference to Hamming distance. In this model it is trivial to prove that the switching cost is at least 2. We present the first non-trivial lower bounds for the switching cost, by giving lower bounds of 3 and 4 for different ranges of the parameters.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases distributed task allocation, combinatorics, lower bounds, multi-agent systems, low-distortion embedding, dynamic algorithms, biological distributed algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.99

Category Track A: Algorithms, Complexity and Games

Funding *Nicole Wein*: supported by an NSF Graduate Fellowship and NSF Grant CCF-1514339.

Acknowledgements We would like to thank Yufei Zhao for a discussion.

1 Introduction

Task allocation in multi-agent systems is a fundamental problem in distributed computing. Given a collection of tasks, a collection of task-performing agents, and a *demand vector* which specifies the number of agents required to perform each task, the agents must collectively allocate themselves to the tasks to satisfy the demand vector. This problem has been studied in a wide variety of settings. For example, agents may be identical or have differing abilities, agents may or may not be permitted to communicate with each other, agents may have limited memory or computational power, agents may be faulty, and agents may or may not have full information about the demand vector. See Georgiou and Shvartsman's book [7] for a survey of the distributed task allocation literature. See also the more recent line of work by Dornhaus, Lynch and others on algorithms for task allocation in ant colonies [4, 20, 5, 17].

We consider the setting where the demand vector *changes dynamically* over time and agents must redistribute themselves among the tasks accordingly. We aim to minimize the *switching cost*, which is the number of agents that change tasks in response to a change in the demand vector. The switching cost is an important metric since changing tasks may incur significant overhead. Dynamic task allocation has been extensively studied in practical, heuristic, and experimental domains. For example, in swarm robotics, there is much

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 99; pp. 99:1–99:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

experimental work on heuristics for dynamic task allocation (see e.g. [10, 19, 13, 14, 11, 12]). Additionally, in insect biology it has been empirically observed that demands for tasks in ant colonies change over time based on environmental factors such as climate, season, food availability, and predation pressure [15]. Accordingly, there is a large body of biological work on developing hypotheses about how insects collectively perform task allocation in response to a changing environment (see surveys [1, 18]).

Despite the rich experimental literature, to the best of our knowledge there are only two works on dynamic distributed task allocation from a theoretical algorithmic perspective. Su, Su, Dornhaus, and Lynch [20] present and analyze gossip-based algorithms for dynamic task allocation in ant colonies. Radeva, Dornhaus, Lynch, Nagpal, and Su [17] analyze dynamic task allocation in ant colonies when the ants behave randomly and have limited information about the demand vector.

1.1 Problem Statement

We study the formalization of dynamic distributed task allocation introduced by Su, Su, Dornhaus, and Lynch [20].

Objective. Our goal is to minimize the *switching cost*, which is the number of agents that change tasks in response to a change in the demand vector.

Properties of agents.

1. the agents have complete information about the changing demand vector
2. the agents are heterogeneous
3. the agents cannot communicate
4. the agents are memoryless

The first two properties specify *capabilities* of the agents while the third and fourth properties specify *restrictions* on the agents. Although the exclusion of communication and memory may appear overly restrictive, our setting captures well-studied models of both collective insect behavior and swarm robotics, as outlined in Section 1.1.3.

From a mathematical perspective, our model captures the *combinatorial* aspects of dynamic distributed task allocation. In particular, as we show in Section 2, the problem can be reformulated as finding a *low distortion embedding* from symmetric difference to Hamming distance.

1.1.1 Formal statement

Formally, the problem is defined as follows. There are three positive integer parameters: n is the number of agents, k is the number of tasks, and D is the target *maximum switching cost*, which we define later. The goal is to define a set of n deterministic functions $f_1^{n,k}, f_2^{n,k}, \dots, f_n^{n,k}$, one for each agent, with the following properties.

- **Input:** For each agent a, the function $f_a^{n,k}$ takes as input a *demand vector* $\vec{v} = \{v_1, v_2, \dots, v_k\}$ where each v_i is a non-negative integer and $\sum_i v_i = n$. Each v_i is the number of agents required for task i, and the total number of agents required for tasks is exactly the total number of agents.
- **Output:** For each agent a, the function $f_a^{n,k}$ outputs some $i \in [k]$. The output of $f_a^{n,k}(\vec{v})$ is the task that agent a is assigned when the demand vector is $\vec{v}$.

- **Demand satisfied:** For all demand vectors $\vec{v}$ and all tasks i, we require that the number of agents a for which $f_a^{n,k}(\vec{v}) = i$ is exactly v_i. That is, the allocation of agents to tasks defined by the set of functions $f_1^{n,k}, f_2^{n,k}, \dots, f_n^{n,k}$ exactly satisfies the demand vector.
- **Switching cost satisfied:** The *switching cost* of a pair $(\vec{v}, \vec{v'})$ of demand vectors is defined as the number of agents a for which $f_a^{n,k}(\vec{v}) \neq f_a^{n,k}(\vec{v'})$; that is, the number of agents that switch tasks if the demand vector changes from $\vec{v}$ to $\vec{v'}$ (or from $\vec{v'}$ to $\vec{v}$). We say that a pair of demand vectors $\vec{v}$, $\vec{v'}$ are *adjacent* if $|\vec{v} - \vec{v'}|_1 = 2$; that is, if we can get from $\vec{v}$ to $\vec{v'}$ by moving exactly one unit of demand from one task to another. The *maximum switching cost* of a set of functions $f_1^{n,k}, f_2^{n,k}, \dots, f_n^{n,k}$ is defined as the maximum switching cost over all pairs of adjacent demand vectors; that is, the maximum number of agents that switch tasks in response to the movement of a single unit of demand from one task to another. We require that the maximum switching cost of $f_1^{n,k}, f_2^{n,k}, \dots, f_n^{n,k}$ is at most D.

▶ **Question.** *Given n and k, what is the minimum possible maximum switching cost D over all sets of functions $f_1^{n,k}, \dots, f_n^{n,k}$?*

1.1.2 Remarks

▶ Remark 1. The problem statement only considers the switching cost of pairs of *adjacent* demand vectors. We observe that this also implies a bound on the switching cost of non-adjacent vectors: if every pair of adjacent demand vectors has switching cost at most D, then every pair of demand vectors with ℓ_1 distance d has switching cost at most $D(d/2)$.

▶ Remark 2. The problem statement is consistent with the properties of the agents listed above. In particular, the agents have complete information about the changing demand vector because for each agent, the function $f_a^{n,k}$ takes as input the current demand vector. The agents are heterogeneous because each agent a has a separate function $f_a^{n,k}$. The agents have no communication or memory because the *only* input to each function $f_a^{n,k}$ is the current demand vector.

▶ Remark 3. Forbidding communication among agents is crucial in the formulation of the problem, as otherwise the problem would be trivial. In particular, it would always be possible to achieve maximum switching cost 1: when the current demand vector changes to an adjacent demand vector, the agents simply reach consensus about which single agent will move.

1.1.3 Applications

1.1.3.1 Collective insect behavior

There are a number of hypotheses that attempt to explain the mechanism behind task allocation in ant colonies (see the survey [1]). One such hypothesis is the *response threshold model*, in which ants decide which task to perform based on individual preferences and environmental factors. Specifically, the model postulates that there is an environmental stimulus associated with each task, and each individual ant has an internal threshold for each task, whereby if the stimulus exceeds the threshold, then the ant performs that task. The response threshold model was introduced in the 70s and has been studied extensively since (for comprehensive background on this model see the survey [1] and the introduction of [6]).

Our setting captures the essence of the response threshold model since agents are permitted to behave based on individual preferences (property 2: agents are heterogeneous) and environmental factors (property 1: agents have complete information about the demand vector). We study whether models like the response threshold model can achieve low switching costs.

Inspired by collective insect behavior, researchers have also studied the response threshold model in the context of swarm robotics [2, 9, 22]. Our setting also relates more generally to swarm robotics:

1.1.3.2 Swarm robotics

There is a body of work in swarm robotics specifically concerned with property 3 of our setting: eliminating the need for communication (e.g. [21, 3, 8, 16]). In practice, communication among agents may be unfeasible or costly. In particular, it may be unfeasible to build a fast and reliable network infrastructure capable of dealing with delays and failures, especially in a remote location.

Regarding property 4 of our setting (the agents are memoryless), it may be desirable for robots in a swarm to not rely on memory. For example, if a robot fails and its memory is lost, we may wish to be able to introduce a new robot into the system to replace it.

Concretely, dynamic task allocation in swarm robotics may be applicable to disaster containment [16, 23], agricultural foraging, mining, drone package delivery, and environmental monitoring [19].

1.2 Past Work

Our problem was previously studied only by Su, Su, Dornhaus, and Lynch [20], who presented two upper bounds and a lower bound.

The first upper bound is a very simple set of functions $f_1^{n,k}, \ldots, f_n^{n,k}$ with maximum switching cost $k-1$. Each agent has a unique ID in $[n]$ and the tasks are numbered from 1 to k. The functions $f_1^{n,k}, \ldots, f_n^{n,k}$ are defined so that for all demand vectors, the agents populate the tasks in order from 1 to k in order of increasing agent ID. That is, for each agent a, $f_a^{n,k}$ is defined as the task j such that $\sum_{i=0}^{j-1} d_i < \mathrm{ID}(a)$ and $\sum_{i=0}^{j} d_i \geq \mathrm{ID}(a)$. Starting with any demand vector, if one unit of demand is moved from task i to task j, the switching cost is at most $|i-j|$ because at most one agent from each task numbered between i and j (including i but not including j) shifts to a new task. Thus, the maximum switching cost is $k-1$.

The lower bound of Su et al. is also very simple. It shows that there does not exist a set of functions $f_1^{n,k}, \ldots, f_n^{n,k}$ with maximum switching cost 1 for $n \geq 2$ and $k \geq 3$. Suppose for contradiction that there exists a set of functions $f_1^{n,k}, \ldots, f_n^{n,k}$ with maximum switching cost 1 for $n = 2$ and $k = 3$ (the argument can be easily generalized to higher n and k).

Suppose the current demand vector is $[1, 1, 0]$, that is, one agent is required for each of tasks 1 and 2 while no agent is required for task 3. Suppose agents a and b are assigned to tasks 1 and 2, respectively, which we denote $[a, b, \emptyset]$. Now suppose the demand vector changes from $[1, 1, 0]$ to the adjacent demand vector $[1, 0, 1]$. Since the maximum switching cost is 1, only one agent moves, so agent b moves to task 3, so we have $[a, \emptyset, b]$. Now suppose the demand vector changes from $[1, 0, 1]$ to the adjacent demand vector $[0, 1, 1]$. Again, since the maximum switching cost is 1, agent a moves from task 1 to task 2 resulting in $[\emptyset, a, b]$. Now suppose the demand vector changes from $[0, 1, 1]$ to the adjacent demand vector $[1, 1, 0]$, which was the initial demand vector. Since the maximum switching cost is 1, agent b moves from task 3 to task 1 resulting in $[b, a, \emptyset]$.

The problem statement requires that the allocation of agents depends *only* on the current demand vector, so the allocation of agents for any given demand vector must be the same regardless of the history of changes to the demand vector. However, we have shown that the allocation of agents for $[1,1,0]$ was initially $[a,b,\emptyset]$ and is now $[b,a,\emptyset]$, a contradiction. Thus, the maximum switching cost is at least 2.

The second upper bound of Su et al. states that there exists a set of functions $f_1^{n,k},\ldots,f_n^{n,k}$ with maximum switching cost 2 if $n \leq 6$ and $k = 4$. They prove this result by exhaustively listing all 84 demand vectors along with the allocation of agents for each vector.

1.3 Our results

We initiate the study of non-trivial lower bounds for the switching cost. In particular, with the current results it is completely plausible that the maximum switching cost can always be upper bounded by 2, regardless of the number of tasks and agents. Our results show that this is not true and provide further evidence that the maximum switching cost grows with the number of tasks.

One might expect that the limitations on n and k in the second upper bound of Su et al. is due to the fact the space of demand vectors grows exponentially with n and k so their method of proof by exhaustive listing becomes unfeasible. However, our first result is that the second upper bound of Su et al. is actually *tight* with respect to k. In particular, we show that achieving maximum switching cost 2 is impossible even for $k = 5$ (for *any* $n > 2$).

▶ **Theorem 4.** *For $n \geq 3$, $k \geq 5$, every set of functions $f_1^{n,k},\ldots,f_n^{n,k}$ has maximum switching cost at least 3.*

We then consider the next natural question: *For what values of n and k is it possible to achieve maximum switching cost 3?* Our second result is that maximum switching cost 3 is not always possible:

▶ **Theorem 5.** *There exist n and k such that every set of functions $f_1^{n,k},\ldots,f_n^{n,k}$ has maximum switching cost at least 4.*

The value of k for Theorem 5 is an extremely large constant derived from hypergraph Ramsey numbers. Specifically, there exists a constant c so that Theorem 5 holds for $n \geq 5$ and $k \geq t_{n-1}(cn)$ where the tower function $t_j(x)$ is defined by $t_1(x) = x$ and $t_{i+1}(x) = 2^{t_i(x)}$.

We remark that while our focus on small constant values of the switching cost may appear restrictive, functions with maximum switching cost 3 already have a highly non-trivial combinatorial structure.

1.4 Our techniques

We introduce two novel techniques, each tailored to a different parameter regime. One parameter regime is when $n \ll k$ and the demand for each task is either 0 or 1. This regime seems to be the most natural for the goal of proving the highest possible lower bounds on the switching cost.

1.4.1 The $n \ll k$ regime

We develop a proof framework for the $n \ll k$ regime and use it to prove Theorem 4 for $n = 3$, $k = 5$, and more importantly, to prove Theorem 5. We begin by supposing for contradiction that there exists a set of functions $f_1^{n,k},\ldots,f_n^{n,k}$ with switching cost 2 and 3, respectively, and then reason about the structure of these functions. The main challenge in proving Theorem 5 as compared to Theorem 4 is that functions with switching cost 3 can have a much more involved combinatorial structure than functions with switching cost 2. In principle, our proof framework could also apply to higher switching costs, but at present it is unclear how exactly to implement it for this setting.

The first step in our proofs is to reformulate the problem as that of finding a low distortion embedding from symmetric difference to Hamming distance, which we describe in Section 2. This provides a cleaner way to reason about the problem in the $n \ll k$ parameter regime. Our proofs are written in the language of the problem reformulation, but here we will briefly describe our proof framework in the language of the original problem statement.

The simple upper bound of $k-1$ described in Section 1.2 can be viewed as each agent having a "preference" for certain tasks. The main idea of our lower bound is to show that for *any* set of functions $f_1^{n,k}, \dots, f_n^{n,k}$ with low switching cost, many agents must have a "preference" for certain tasks. More formally, we introduce the idea of a task being *frozen* to an agent. A task t is frozen to agent a if for every demand vector in a particular large set of demand vectors, agent a is assigned to task t. Our framework has three steps:

- In step 1, we show roughly that in total, many tasks are frozen to some agent.
- In step 2, we show roughly that for many agents a, only few tasks are frozen to a.
- In step 3, we use a counting argument to derive a contradiction: we count a particular subset of frozen task/agent pairs in two different ways using steps 1 and 2, respectively.

The proof of Theorem 4 for $n = 3$ and $k = 5$ serves as a simple illustrative example of our proof framework, while the proof of Theorem 5 is more involved. In particular, in step 1 of the proof of Theorem 5, we derive *multiple* possible structures of frozen task/agent pairs. Then, we use Ramsey theory to show that there exists a collection of tasks that all obey only *one* of the possible structures. This allows us to reason about each of the possible structures independently in steps 2 and 3.

1.4.2 The remaining parameter regime

In the remaining parameter regime, we complete the proof of Theorem 4. In the previous parameter regime, we only addressed the $n = 3$, $k = 5$ case, and now we need to consider all larger values of n and k. Extending to larger k is trivial (we prove this formally in Section 4). However, it is not at all clear how to extend a lower bound to larger values of n. In particular, our proof framework from the $n \ll k$ regime immediately breaks down as n grows.

The main challenge of handling large n is that having an abundance of agents can actually allow *more* pairs of adjacent demand vectors to have switching cost 2, so it becomes more difficult to find a pair with switching cost greater than 2. To see this, consider the following example.

Consider the subset S_i of demand vectors in which a particular task i has an unconstrained amount of demand and each remaining task has demand at most $n/(k-1)$. We claim that there exists a set of functions $f_1^{n,k}, \dots, f_n^{n,k}$ so that every pair of adjacent demand vectors from S_i has switching cost 2. Divide the agents into $k-1$ groups of $n/(k-1)$ agents each, and associate each task except i to such a group of agents. We define the functions $f_1^{n,k}, \dots, f_n^{n,k}$ so that given any demand vector in S_i, the set of agents assigned to each task except i is simply a subset of the group of agents associated with that task (say, the subset of such agents with smallest ID). This is a valid assignment since the demand of each task except i is at most the size of the group of agents associated with that task. The remaining agents are assigned to task i. Then, given a pair $(\vec{v}, \vec{v'})$ of adjacent demand vectors in S_i, whose demands differ only for tasks s and t, their switching cost is 2 because the only agents assigned to different tasks between $\vec{v}$ and $\vec{v'}$ are: one agent from each of the groups associated with tasks s and t, respectively.

Because it is possible for many pairs of adjacent demand vectors to have switching cost 2, finding a pair of adjacent demand vectors with larger switching cost requires reasoning about a very precise set of demand vectors. To do this, we use roughly the following strategy. We identifying a task that serves the role of i in the above example and then successively move demand out of task i until task i is empty and can thus no longer fill this role. At this point, we argue that we have reached a pair of adjacent demand vectors with switching cost more than 2.

2 Problem reformulation

2.1 Notation

Let A and B be multisets. The intersection of A and B denoted $A \cap B$ is the maximal multiset of elements that appear in both A and B. For example, $\{a, a, b, b\} \cap \{a, b, b, c\} = \{a, b, b\}$. The symmetric difference between A and B, denoted $A \oplus B$, is the multiset of elements in either A or B but not in their intersection. For example, $\{a, a, b, b\} \oplus \{a, b, b, c\} = \{a, c\}$ since we are left with a after removing $\{a, b, b\}$ from $\{a, a, b, b\}$ and we are left with c after removing $\{a, b, b\}$ from a, b, b, c.

A permutation of a multiset A is simply a permutation of the elements of the multiset. For example, one permutation of $\{a, a, b\}$ is aba. We treat permutation as strings and perform string operations on them. For strings X and Y (which may be permutations), let $d(X, Y)$ denote the *Hamming distance* between X and Y. For example, $d(aba, bca) = 2$.

2.2 Problem statement

Given positive integers n, k, and D, the goal is to find a function $\pi_{n,k}$ with the following properties.

- Let $\mathcal{S}_{n,k}$ be the set of all size n multisets of $[k]$. The function $\pi_{n,k}$ takes as input a set $S \in \mathcal{S}_{n,k}$ and outputs a permutation of S.
- We say that a pair $S, S' \in \mathcal{S}_{n,k}$ has *distortion* D' with respect to $\pi_{n,k}$ if $|S \oplus S'| = 2$ and $d(\pi_{n,k}(S), \pi_{n,k}(S')) = D'$. In other words, a pair of multisets has distortion D' if they have the *smallest* possible symmetric distance but *large* Hamming distance (at least D'). We say that $\pi_{n,k}$ has *maximum distortion* D' if the maximum distortion over all pairs $S, S' \in \mathcal{S}_{n,k}$ with $|S \oplus S'| = 2$ is D'. We require that the function $\pi_{n,k}$ has maximum distortion at most D.

We are interested in the question of for which values of the parameters n, k, and D, there exists $\pi_{n,k}$ that satisfies the above properties. In particular, we aim to minimize the maximum distortion:

▶ **Question.** *Given n and k, what is the minimum possible maximum distortion over all functions $\pi_{n,k}$?*

In other words, the question is whether there exists a function $\pi_{n,k}$ such that *every* pair $S, S' \in \mathcal{S}_{n,k}$ has distortion at least D. Our theorems are lower bounds, so we show that for every function $\pi_{n,k}$ there *exists* a pair $S, S' \in \mathcal{S}_{n,k}$ with distortion at least D.

2.3 Equivalence to original problem statement

We claim that the new problem statement from Section 2.2 is equivalent to the original problem statement from Section 1.1.

▷ Claim 6. Given parameters n and k (the same for both problem statements) there exists a function $\pi_{n,k}$ with maximum distortion D if and only if there exists a set of functions $f_1^{n,k}, \ldots, f_n^{n,k}$ with maximum switching cost D.

We describe the correspondence between the two problem statements:

- **Demand vector.** $\mathcal{S}_{n,k}$ is the set of all possible demand vectors since a demand vector is simply a size n multiset of the k tasks. For example, the multiset $S = \{1, 1, 3\}$ is equivalent to the demand vector $\vec{v} = [2, 0, 1]$; both notations indicate that task 1 requires two units of demand, task 2 requires no demand, and task 3 requires one unit of demand.
- **Allocation of agents to tasks.** If $\vec{v}$ is the demand vector representing the multiset $S \in \mathcal{S}_{n,k}$, a permutation $\pi_{n,k}(S)$ is an allocation $f_1^{n,k}(\vec{v}), \ldots, f_n^{n,k}(\vec{v})$ of agents to tasks so that $\pi_{n,k}(S)[i] = f_i^{n,k}(\vec{v})$; that is, agent i performs the task that is the i^{th} element in the permutation $\pi_{n,k}(S)$. For example, $\pi_{3,3}(\{1, 1, 3\}) = 131$ is equivalent to the following: $f_1^{3,3}([2, 0, 1]) = 1$, $f_2^{3,3}([2, 0, 1]) = 3$, and $f_3^{3,3}([2, 0, 1]) = 1$; both notations indicate that agents 1 and 3 both performs task 1, while agent 2 performs task 2.
- **Switching cost.** If $\vec{v}, \vec{v'}$ are the demand vectors representing the multisets $S, S' \in \mathcal{S}_{n,k}$ respectively, the value $d(\pi_{n,k}(S), \pi_{n,k}(S'))$ is the switching cost because from the previous bullet point, $\pi_{n,k}(S)[i] \neq \pi_{n,k}(S')[i]$ if and only if $f_a^{n,k}(\vec{v}) \neq f_a^{n,k}(\vec{v'})$.
- **Adjacent demand vectors.** The set of all pairs $S, S' \in \mathcal{S}_{n,k}$ such that $|S \oplus S'| = 2$ is the set of all pairs of adjacent demand vectors. This is because $|S \oplus S'| = 2$ means that starting from S, one can reach S' by changing exactly one element in S from some $i \in [k]$ to some $j \in [k]$. Equivalently, starting from the demand vector represented by S and moving one unit of demand from task i to task j results in the demand vector represented by S'.
- **Maximum switching cost.** If $f_1^{n,k}, \ldots, f_n^{n,k}$ is the set of functions representing $\pi_{n,k}$, then $\pi_{n,k}$ has maximum distortion D if and only if $f_1^{n,k}, \ldots, f_n^{n,k}$ has maximum switching cost D. This is because $S, S' \in \mathcal{S}_{n,k}$ has distortion D if and only if $|S \oplus S'| = 2$ and $d(\pi_{n,k}(S), \pi_{n,k}(S')) = D$ which is equivalent to saying that the demand vectors $\vec{v}$ and $\vec{v'}$ that represent S and S' are adjacent and have switching cost D.

2.4 Restatement of results

We restate Theorems 4 and 5 in the language of the problem restatement.

▶ **Theorem 7** (Restatement of Theorem 4). *Let $n \geq 3$ and $k \geq 5$. Every function $\pi_{n,k}$ has maximum distortion at least 3.*

▶ **Theorem 8** (Restatement of Theorem 5). *There exist n and k so that every function $\pi_{n,k}$ has maximum distortion at least 4.*

2.5 Example instance

To build intuition about the problem restatement, we provide a concrete example of a small instance of the problem. Suppose $n = 3$ and $k = 2$. For notational clarity, instead of denoting $[k] = \{0, 1\}$ we denote $[k] = \{a, b\}$. Then $\mathcal{S}_{3,2}$ is the set of all size 3 multisets of $\{a, b\}$; that is, $\mathcal{S}_{3,2} = \{\{a, a, a\}, \{a, a, b\}, \{a, b, b\}, \{b, b, b\}\}$. $\pi_{3,2}$ is a function that maps each element of $\mathcal{S}_{3,2}$ to a permutation of itself. For example, $\pi_{3,2}$ could be defined as follows:

$$\pi_{3,2}(\{a,a,a\}) = aaa, \quad \pi_{3,2}(\{a,a,b\}) = aba \quad \pi_{3,2}(\{a,b,b\}) = bab, \quad \pi_{3,2}(\{b,b,b\}) = bbb.$$

We are concerned with all pairs $S, S' \in \mathcal{S}_{3,2}$ such that $|S \oplus S'| = 2$ (since the maximum distortion of $\pi_{3,2}$ is defined in terms of only these pairs). In this example, the only such pairs are as follows:

$$\{a,a,a\} \oplus \{a,a,b\} = 2, \quad \{a,a,b\} \oplus \{a,b,b\} = 2, \quad \{a,b,b\} \oplus \{b,b,b\} = 2.$$

For each such pair, we consider $d(\pi_{3,2}(S), \pi_{3,2}(S'))$:

$$d(aaa, aba) = 1, \quad d(aba, bab) = 3, \quad d(bab, bbb) = 1.$$

This particular choice of $\pi_{3,2}$ has maximum distortion 3 (since the largest value in the above row is 3), however we could have chosen $\pi_{3,2}$ with maximum distortion 1 (for example if $\pi_{3,2}(\{a,b,b\}) = bba$ instead of bab).

3 The $n \ll k$ regime

In this section we will prove Theorem 7 for $n = 3$, $k = 5$, and Theorem 8. The proofs are written in the language of the problem reformulation from Section 2. For these proofs it will suffice to consider only the elements of $\mathcal{S}_{n,k}$ that are *subsets* of $[k]$, rather than multisets. This corresponds to the set of demand vectors where each task has demand either 0 or 1. *For the rest of this section we consider only subsets of* $[k]$*, rather than multisets.*

We call each element of $[k]$ a *character* (e.g. in the above example instance, a and b are characters).

3.1 Proof framework

As described in Section 1.4, we develop a three-step proof framework for the $n \ll k$ regime. Suppose we are trying to prove that every function $\pi_{n,k}$ has maximum distortion at least D for a particular n and k. We begin by supposing for contradiction that there exists $\pi_{n,k}$ with maximum distortion less than D. That is, we suppose that every pair $S, S' \in \mathcal{S}_{n,k}$ with $|S \oplus S'| = 2$ has $d(\pi_{n,k}(S), \pi_{n,k}(S')) < D$. Under the assumption that such a $\pi_{n,k}$ exists, steps 1 and 2 of the framework show that $\pi_{n,k}$ must obey a particular structure. For the remainder of this section, we drop the subscript of π since n and k are fixed.

▶ **Notation.** *For any set $R \subseteq [k]$, let $\mathcal{U}_R$ be the set of all sets $S \subseteq [k]$ such that $R \subset S$ and $|S| = |R| + 1$.*

Step 1: Structure of size $n - 1$ sets

We begin by fixing a size $n - 1$ set $R \subseteq [k]$. Now, consider $\mathcal{U}_R$ (defined above). We note that all pairs $S, S' \in \mathcal{U}_R$ are by definition such that $|S \oplus S'| = 2$. Because we initially supposed that π has maximum distortion less than D, we know that for all pairs $S, S' \in \mathcal{U}_R$, we have $d(\pi(S), \pi(S')) < D$.

Then we prove a structural lemma which roughly says that many characters $r \in R$ have a "preference" to be in a particular position in the permutations $\pi(S)$ for $S \in \mathcal{U}_R$. We say that R *i-freezes* the character r if $\pi(S)[i] = r$ for many $S \in \mathcal{U}_R$. Our structural lemma roughly says that for many characters $r \in R$, there exists an index $i \in [n]$ such that R i-freezes r. In other words, for many $S \in \mathcal{U}_R$, the $\pi(S)$s agree on the position of many characters in the permutation.

Step 2: Structure of size $n-2$ sets

We begin by fixing a size $n-2$ set $Q \subseteq [k]$. Now, consider $\mathcal{U}_Q$. We note that each $R \in \mathcal{U}_Q$ obeys the structural lemma from step 1; that is, for many characters $r \in R$, there exists an index $i \in [n]$ such that R i-freezes r.

We prove a structural lemma which roughly says that the sets $P \in \mathcal{U}_Q$ are for the most part *consistent* about which characters they freeze to which index of the permutation. More specifically, for many characters $q \in Q$, for all pairs $P, P' \in \mathcal{U}_Q$, if R i-freezes r and R' j-freezes r, then $i = j$.

Step 3: Counting argument

In step 3, we use a counting argument to derive a contradiction. For the proof of Theorem 7, a simple argument suffices. The idea is that step 1 shows that many characters are frozen overall while step 2 shows that each character can only be frozen to a single index. Then, the pigeonhole principle implies that more than one character is frozen to a single index, which helps to derive a contradiction.

For the proof of Theorem 8, it no longer suffices to just show that more than one character is frozen to a single index. Instead, we require a more sophisticated counting argument and a careful choice of what quantity to count. We end up counting the number of pairs (Q, a) such that $R \in \mathcal{U}_Q$, where $Q \subset [k]$ is a size $n-2$ set and $a \in [n] \setminus Q$. To reach a contradiction, we count this quantity in two different ways, using steps 1 and 2 respectively.

Having reached a contradiction, we conclude that π has maximum distortion at least D.

3.2 Proof of Theorem 7 for $n = 3$, $k = 5$

In this section, we prove Theorem 7 for $n = 3$, $k = 5$, which serves as a simple illustrative example of our proof framework from Section 3.1.

▶ **Theorem 9** (Special case of Theorem 7). *Every function $\pi_{3,5}$ has maximum distortion at least 3.*

Proof. Suppose by way of contradiction that there is a function $\pi_{3,5}$ with maximum distortion at most 2. For the remainder of this section we omit the subscript of π since $n = 3$, $k = 5$ are fixed. For clarity of notation, we let $\{a, b, c, d, e\}$ be the characters in $[k]$ for $k = 5$. Thus, we are considering the set of all $\binom{5}{3} = 10$ size 3 subsets of $\{a, b, c, d, e\}$. (Recall that we are only concerned with subsets, not multisets.)

Step 1: Structure of size $n-1$ sets. We begin by fixing a set $\{x, y\} \subseteq \{a, b, c, d, e\}$ of size $n-1 = 2$. Recall that $\mathcal{U}_{\{x,y\}}$ is the set of all size 3 sets S such that $\{x, y\} \subseteq S \subseteq \{a, b, c, d, e\}$. For example, $\mathcal{U}_{\{a,b\}} = \{\{a, b, c\}, \{a, b, d\}, \{a, b, e\}\}$. We note that by definition all pairs $S, S' \in \mathcal{U}_{\{x,y\}}$ have $|S \oplus S'| = 2$. Thus, to find a pair with distortion 3 and thereby obtain a contradiction, it suffices to find a pair $S, S' \in \mathcal{U}_{\{x,y\}}$ with Hamming distance $d(\pi(S), \pi(S')) = 3$. Since $n = 3$, this means we are looking for permutations $\pi(S), \pi(S')$ that disagree about the position of *all* elements.

The following lemma says that π places one of x or y at the *same* position for *all* $\pi(S)$ with $S \in \mathcal{U}_{\{x,y\}}$. For ease of notation, we give this phenomenon a name:

▶ **Definition 10** (freeze). *We say that a pair $\{x, y\} \subseteq \{a, b, c, d, e\}$ i-*freezes *a character $p \in \{x, y\}$ if for all $S \in \mathcal{U}_{\{x,y\}}$, we have $\pi(S)[i] = p$. We simply say that $\{x, y\}$* freezes *p if i is unspecified. Equivalently, we say that a character p is i-*frozen *(or just* frozen*) by a pair.*

▶ **Lemma 11.** *For every $\{x, y\} \subseteq \{a, b, c, d, e\}$, there exists i so that $\{x, y\}$ i-freezes either x or y.*

For example, one way that the pair $\{a, b\}$ could satisfy Lemma 11 is if the permutations $\pi(\{a, b, c\})$, $\pi(\{a, b, d\})$, and $\pi(\{a, b, e\})$ *all* place the character a in the 0^{th} position. In this case, we would say that the pair $\{a, b\}$ 0-freezes a.

Proof of Lemma 11. Without loss of generality, consider $\{x, y\} = \{a, b\}$. In this case, $\mathcal{U}_{\{x,y\}} = \mathcal{U}_{\{a,b\}} = \{\{a, b, c\}, \{a, b, d\}, \{a, b, e\}\}$. Thus, we are trying to show that $\{a, b, c\}$, $\{a, b, d\}$, and $\{a, b, e\}$ *all* agree on the position of either a or b.

Suppose without loss of generality that $\pi(\{a, b, c\}) = abc$. We first note that $\pi(\{a, b, c\})$ and $\pi(\{a, b, d\})$ must agree on the position of either a or b because otherwise we would have $d(\pi(\{a, b, c\}), \pi(\{a, b, d\})) = 3$ which would mean that $\pi(\{a, b, c\})$ and $\pi(\{a, b, d\})$ would have distortion 3, and we would have proved Theorem 9. Without loss of generality, suppose $\pi(\{a, b, c\})$ and $\pi(\{a, b, d\})$ agree on the position of a; that is, $\pi(\{a, b, d\})$ is either abd or adb.

By the same reasoning, $\pi(\{a, b, c\})$ and $\pi(\{a, b, e\})$ agree on the position of either a or b, and $\pi(\{a, b, d\})$ and $\pi(\{a, b, e\})$ agree on the position of either a or b. If $\pi(\{a, b, e\})$ agrees with *either* $\pi(\{a, b, c\})$ or $\pi(\{a, b, d\})$ on the position of a, then it agrees with *both* (in which case we are done) since $\pi(\{a, b, c\})$ and $\pi(\{a, b, d\})$ agree on the position of a, by the previous paragraph. Thus, the only option is that $\pi(\{a, b, e\})$ agrees with *both* $\pi(\{a, b, c\})$ and $\pi(\{a, b, d\})$ on the position of b. This completes the proof. ◀

Step 2: Structure of size $n - 2$ sets. Since $n - 2 = 1$, we begin by fixing a single element $x \in \{a, b, c, d, e\}$. In the following lemma we prove that x cannot be frozen to two different indices.

▶ **Lemma 12.** *If a pair $\{x, y\} \subseteq \{a, b, c, d, e\}$ i-freezes x and a pair $\{x, z\} \subseteq \{a, b, c, d, e\}$ j-freezes x then $i = j$.*

Proof. Since $\{x, y\}$ i-freezes x, then in particular, $\pi(\{x, y, z\})[i] = x$. Since $\{x, z\}$ j-freezes x, then in particular, $\pi(\{x, y, z\})[j] = x$. A single character cannot be in multiple positions of the permutation $\pi(\{x, y, z\})$ so $i = j$. ◀

Step 3: Counting argument. Lemma 11 implies that for each character $x \in \{a, b, c, d, e\}$ except for at most one, *some* pair $\{x, y\}$ freezes x. That is, at least 4 characters are frozen by some pair. However $n = 3$ so by the pigeonhole principle, two characters $x, y \in \{a, b, c, d, e\}$ are frozen to the same index i.

Fix x, y, and i, and suppose x and y are each i-frozen. By Lemma 11, the pair $\{x, y\}$ freezes either x or y. Without loss of generality, say $\{x, y\}$ freezes x. By Lemma 12, since x is i-frozen by some pair, *all* pairs that freeze x must i-freeze x. Thus, the pair $\{x, y\}$ i-freezes x.

Let $\{y, z\} \subseteq \{a, b, c, d, e\}$ be a pair that i-freezes y. Thus we have $\pi(\{x, y, z\})[i] = y$. However, since $\{x, y\}$ i-freezes x, we also have $\pi(\{x, y, z\})[i] = x$. This is a contradiction since $\pi(\{x, y, z\})[i]$ cannot take on two different values. ◀

We defer the proof of Theorem 8, which is the remainder of Section 3, to the full version.

4 The remaining parameter regime

▶ **Theorem 13** (restatement of Theorem 4). *For $n \geq 3$, $k \geq 5$, every set of functions $f_1^{n,k}, \ldots, f_n^{n,k}$ has maximum switching cost at least 3.*

▶ Remark. We note that the proof framework from Section 3 immediately breaks down if we try to apply it to Theorem 13 for all n, k. For example, when $n > k$, there are no size n subsets of $[k]$ so we must instead consider size n *multisets* of $[k]$. Even if we have the same setting of parameters as Theorem 7 but we are considering multisets, in step 1 of the proof framework Lemma 11 is no longer true. That is, it is not true that for all size 2 multisets $\{x, y\}$ of $[k]$, we have that $\{x, y\}$ i-freezes either x or y for some i. In particular, suppose $\{x, y\} = \{a, a\}$. Then if is possible that $\pi(\{a, a, b\}) = aab$, $\pi(\{a, a, c\}) = aca$, and $\pi(\{a, a, d\}) = daa$, in which case a is not frozen to any index. Since the proof framework from Section 3 no longer applies, we develop entirely new techniques in this section. (However we do use this proof framework to prove Theorem 8.)

For the rest of this section we will use the language of the original problem statement rather than that of the problem reformulation.

4.1 Preliminaries

To prove the Theorem 13, we need to show that Theorem 9 extends to larger k and n. As noted in Section 1.4.2, extending to larger n is challenging, while extending to larger k is trivial, as shown in the following lemma.

▶ **Lemma 14.** *Fix n and k. If there exists a set of functions $f_1^{n,k}, \ldots, f_n^{n,k}$ with maximum switching cost D, then for all $k' < k$, there exists a set of functions $g_1^{n,k'}, \ldots, g_n^{n,k'}$ with maximum switching cost D.*

Proof. For each demand vector $\vec{v}$ with n agents and k tasks such that only the first k' entries of $\vec{v}$ are non-zero, let $\vec{v'}$ be the length k' vector consisting of only the first k' entries of $\vec{v}$. We note that the set of all such vectors $\vec{v'}$ is the set of all demand vectors for n agents and k' tasks. Set each $g_i^{n,k'}(\vec{v'}) = f_i^{n,k}(\vec{v})$. Then the switching cost for any adjacent pair $(\vec{v_1'}, \vec{v_2'})$ with respect to $g_1^{n,k'}, \ldots, g_n^{n,k'}$ is equal to the switching cost of the corresponding adjacent pair $(\vec{v_1}, \vec{v_2})$ with respect to $f_1^{n,k}, \ldots, f_n^{n,k}$. Thus, the maximum switching cost of $g_1^{n,k'}, \ldots, g_n^{n,k'}$ is equal to the maximum switching cost of $f_1^{n,k}, \ldots, f_n^{n,k}$. ◀

▶ **Notation.** *We say that an ordered pair of adjacent demand vectors $(\vec{v_1}, \vec{v_2})$ is (s, t)-adjacent if starting with $\vec{v_1}$ and moving exactly one unit of demand from task s to task t results in $\vec{v_2}$. We say that an agent a is (i, j)-*mobile *with respect to an ordered pair of adjacent demand vectors $(\vec{v_1}, \vec{v_2})$ if $f_a^{n,k}(\vec{v_1}) = i$, $f_a^{n,k}(\vec{v_2}) = j$, and $i \neq j$.*

We note that if $(\vec{v_1}, \vec{v_2})$ is (s, t)-adjacent and has switching cost 2, then for some task i, some agent a must be (s, i)-mobile and another agent b must be (i, t)-mobile. We say that i is the intermediate *task with respect to $(\vec{v_1}, \vec{v_2})$.*

4.2 Proof overview

We begin by supposing for contradiction that there exists a set of functions $f_1^{n,k}, \ldots, f_n^{n,k}$ with maximum switching cost 2, and then we prove a series of structural lemmas about such functions.

As previously mentioned, the main challenge of proving Lemma 13 is handling large n. To illustrate this challenge, we repeat the example from Section 1.4.2. This example shows that having large n can allow more pairs of adjacent demand vectors to have switching cost 2, making it more difficult to find a pair with switching cost greater than 2.

Consider the subset S_i of demand vectors in which a particular task i has an unconstrained amount of demand and each remaining task has demand at most $n/(k-1)$. We claim that there exists a set of functions $f_1^{n,k}, \ldots, f_n^{n,k}$ so that every pair of adjacent demand vectors from S_i has switching cost 2. Divide the agents into $k-1$ groups of $n/(k-1)$ agents each, and associate each task except i to such a group of agents. We define the functions $f_1^{n,k}, \ldots, f_n^{n,k}$ so that given any demand vector in S_i, the set of agents assigned to each task except i is simply a subset of the group of agents associated with that task (say, the subset of such agents with smallest ID). This is a valid assignment since the demand of each task except i is at most the size of the group of agents associated with that task. The remaining agents are assigned to task i. Then, given a pair $(\vec{v}, \vec{v'})$ of adjacent demand vectors in S_i, whose demands differ only for tasks s and t, their switching cost is 2 because the only agents assigned to different tasks between $\vec{v}$ and $\vec{v'}$ are: one agent from each of the groups associated with tasks s and t, respectively.

To overcome the challenge illustrated by the above example, our general method is to identify a task that serves the role of task i and then successively move demand out of task i until task i is empty, and thus can no longer serve its original role. We note that in the above example, the task i serves as the intermediate task for all pairs of adjacent demand vectors from S_i. Thus, we will choose i to be an intermediate task.

In particular, we show that there is a demand vector $\vec{v}$ so that we can identify tasks i and t with the following important property: if we start with $\vec{v}$ and move a unit of demand to task t from *any* other task except i, the switching cost is 2 and the intermediate task is i.

Furthermore, we prove that if we start with demand vector $\vec{v}$ and move a unit of demand from task i to task t resulting in demand vector $\vec{v_1}$, then t and i have the important property from the previous paragraph with respect to $\vec{v_1}$. Applying this argument inductively, we show that no matter how many units of demand we successively move from i to t, i and t still satisfy the important property with respect to the current demand vector.

We move demand from i to t until task i is empty. Then, the final contradiction comes from the fact that if we now move a unit of demand from any non-i task to t, then the important property implies that the switching cost is 2 and the intermediate task is i; however, i is empty and an empty task cannot serve as an intermediate task.

We defer the proof of Theorem 13, which is the remainder of Section 4 to the full version.

References

1 Samuel N Beshers and Jennifer H Fewell. Models of division of labor in social insects. *Annual review of entomology*, 46(1):413–440, 2001.

2 Eduardo Castello, Tomoyuki Yamamoto, Yutaka Nakamura, and Hiroshi Ishiguro. Task allocation for a robotic swarm based on an adaptive response threshold model. In *2013 13th International Conference on Control, Automation and Systems (ICCAS 2013)*, pages 259–266. IEEE, 2013.

3 Jianing Chen. *Cooperation in Swarms of Robots without Communication*. PhD thesis, University of Sheffield, 2015.

4 Alejandro Cornejo, Anna Dornhaus, Nancy Lynch, and Radhika Nagpal. Task allocation in ant colonies. In *International Symposium on Distributed Computing*, pages 46–60. Springer, 2014.

5 Anna Dornhaus, Nancy Lynch, Frederik Mallmann-Trenn, Dominik Pajak, and Tsvetomira Radeva. Self-stabilizing task allocation in spite of noise. *arXiv preprint*, 2018. `arXiv:1805.03691`.

6 Ana Duarte, Ido Pen, Laurent Keller, and Franz J Weissing. Evolution of self-organized division of labor in a response threshold model. *Behavioral ecology and sociobiology*, 66(6):947–957, 2012.

7 Chryssis Georgiou and Alexander A Shvartsman. Cooperative task-oriented computing: Algorithms and complexity. *Synthesis Lectures on Distributed Computing Theory*, 2(2):1–167, 2011.

8 Serge Kernbach, Dagmar Häbe, Olga Kernbach, Ronald Thenius, Gerald Radspieler, Toshifumi Kimura, and Thomas Schmickl. Adaptive collective decision-making in limited robot swarms without communication. *The International Journal of Robotics Research*, 32(1):35–55, 2013.

9 Min-Hyuk Kim, Hyeoncheol Baik, and Seokcheon Lee. Response threshold model based uav search planning and task allocation. *Journal of Intelligent & Robotic Systems*, 75(3-4):625–640, 2014.

10 Michael JB Krieger, Jean-Bernard Billeter, and Laurent Keller. Ant-like task allocation and recruitment in cooperative robots. *Nature*, 406(6799):992, 2000.

11 Kristina Lerman, Chris Jones, Aram Galstyan, and Maja J Matarić. Analysis of dynamic task allocation in multi-robot systems. *The International Journal of Robotics Research*, 25(3):225–241, 2006.

12 Kathryn Sarah Macarthur, Ruben Stranders, Sarvapali D Ramchurn, and Nicholas R Jennings. A distributed anytime algorithm for dynamic task allocation in multi-agent systems. In *AAAI*, pages 701–706, 2011.

13 James McLurkin and Daniel Yamins. Dynamic task assignment in robot swarms. In *Robotics: Science and Systems*, volume 8. Citeseer, 2005.

14 James Dwight McLurkin. *Stupid robot tricks: A behavior-based distributed algorithm library for programming swarms of robots.* PhD thesis, Massachusetts Institute of Technology, 2004.

15 George F Oster and Edward O Wilson. *Caste and ecology in the social insects.* Princeton University Press, 1979.

16 Jacques Penders, Lyuba Alboul, Ulf Witkowski, Amir Naghsh, Joan Saez-Pons, Stefan Herbrechtsmeier, and Mohamed El-Habbal. A robot swarm assisting a human fire-fighter. *Advanced Robotics*, 25(1-2):93–117, 2011.

17 Tsvetomira Radeva, Anna Dornhaus, Nancy Lynch, Radhika Nagpal, and Hsin-Hao Su. Costs of task allocation with local feedback: Effects of colony size and extra workers in social insects and other multi-agent systems. *PLoS computational biology*, 13(12):e1005904, 2017.

18 Gene E Robinson. Regulation of division of labor in insect societies. *Annual review of entomology*, 37(1):637–665, 1992.

19 Erol Şahin. Swarm robotics: From sources of inspiration to domains of application. In *International workshop on swarm robotics*, pages 10–20. Springer, 2004.

20 Hsin-Hao Su, Lili Su, Anna Dornhaus, and Nancy Lynch. Ant-inspired dynamic task allocation via gossiping. In *International Symposium on Stabilization, Safety, and Security of Distributed Systems*, pages 157–171. Springer, 2017.

21 Zijian Wang and Mac Schwager. Multi-robot manipulation with no communication using only local measurements. In *CDC*, pages 380–385, 2015.

22 Yongming Yang, Changjiu Zhou, and Yantao Tian. Swarm robots task allocation based on response threshold model. In *2009 4th International Conference on Autonomous Robots and Agents*, pages 171–176. IEEE, 2009.

23 Emaad Mohamed H Zahugi, Mohamed M Shanta, and TV Prasad. Oil spill cleaning up using swarm of robots. In *Advances in Computing and Information Technology*, pages 215–224. Springer, 2013.

On the Degree of Boolean Functions as Polynomials over $\mathbb{Z}_m$

Xiaoming Sun
Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
sunxiaoming@ict.ac.cn

Yuan Sun
Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
sunyuan2016@ict.ac.cn

Jiaheng Wang
School of Electronics Engineering and Computer Science, Peking University, Beijing, China
pw384@hotmail.com

Kewen Wu
School of Electronics Engineering and Computer Science, Peking University, Beijing, China
shlw_kevin@hotmail.com

Zhiyu Xia
Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
xiazhiyu@ict.ac.cn

Yufan Zheng
Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
yufan.zheng@pku.edu.cn

Abstract

Polynomial representations of Boolean functions over various rings such as $\mathbb{Z}$ and $\mathbb{Z}_m$ have been studied since Minsky and Papert (1969). From then on, they have been employed in a large variety of areas including communication complexity, circuit complexity, learning theory, coding theory and so on. For any integer $m \geq 2$, each Boolean function has a unique multilinear polynomial representation over ring $\mathbb{Z}_m$. The degree of such polynomial is called *modulo-m degree*, denoted as $\deg_m(\cdot)$.

In this paper, we investigate the lower bound of modulo-m degree of Boolean functions. When $m = p^k$ ($k \geq 1$) for some prime p, we give a tight lower bound $\deg_m(f) \geq k(p-1)$ for any non-degenerate function $f : \{0,1\}^n \to \{0,1\}$, provided that n is sufficient large. When m contains two different prime factors p and q, we give a nearly optimal lower bound for any symmetric function $f : \{0,1\}^n \to \{0,1\}$ that $\deg_m(f) \geq \frac{n}{2+\frac{1}{p-1}+\frac{1}{q-1}}$.

2012 ACM Subject Classification Theory of computation → Computational complexity and cryptography

Keywords and phrases Boolean function, polynomial, modular degree, Ramsey theory

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.100

Category Track A: Algorithms, Complexity and Games

Funding This work was supported in part by the National Natural Science Foundation of China Grants No. 61832003, 61761136014, the 973 Program of China Grant No. 2016YFB1000201, the Strategic Priority Research Program of Chinese Academy of Sciences Grant No. XDB28000000, and K. C. Wong Education Foundation.

Acknowledgements We thank Noga Alon and Xiaoxu Guo for pointing us to relevant references [1, 9]. We also appreciate anonymous reviewers' careful feedback and constructive advice.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 100; pp. 100:1–100:19
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Given a Boolean function $f : \{0,1\}^n \to \{0,1\}$, the *degree* (resp., *modulo-m degree*), denoted as $\deg(f)$ (resp., $\deg_m(f)$), is the degree of the unique[1] multilinear polynomial representation of f over $\mathbb{R}$ (resp., $\mathbb{Z}_m$). These complexity measures and related notions have been studied extensively since the work of Minsky and Papert [23]. The polynomial representation of a Boolean function has found numerous applications in the study of query complexity (see e.g. [5]), communication complexity [4, 28, 31, 30, 29, 24, 10], learning theory [17, 20, 16, 25], explicit combinatorial constructions [13, 14, 11, 7], circuit lower bounds [33, 27, 1, 12] and coding theory [36, 37, 15, 21], etc.

In this paper, we focus on modulo-m degree of Boolean functions. Throughout, all Boolean functions are assumed to be *non-degenerate*[2], if not specifically mentioned. One of the complexity theoretic motivations of studying $\deg_m(f)$ is to understand the power of modular counting. For example, the famous Razborov–Smolensky polynomial method [27, 33] reduces the task of proving size lower bounds for $\mathsf{AC}^0[p]$ circuits to proving a lower bound of approximate modulo-p degree of the target Boolean function. However, this approach mainly works when p is a prime.[3] Another example, in which m can be composite, is that a $(1/2 + o(1))$-inapproximability of a Boolean function f by degree-$O(1)$ polynomials over $\mathbb{Z}_m$ implies that f cannot be computed by $\mathsf{MAJ}_{O(1)} \circ \mathsf{MOD}_m \circ \mathsf{AND}_{O(1)}$ circuits [1]. In general, it has been proved important to understand the computational power of polynomials over $\mathbb{Z}_m$ for general m.

Towards the complexity measure $\deg_m(f)$ itself, the case when m is a prime has been studied a lot in previous works. For example, one natural question is whether $\deg_m(f)$ is polynomially related to $\deg(f)$ for general m, as other complexity measures like decision tree complexity $\mathrm{D}(f)$ do? The answer is NO according to the parity function $\mathsf{PARITY}(x) := \bigoplus_{i=1}^n x_i$. That is, $\deg_2(\mathsf{PARITY}) = 1$ but $\deg(\mathsf{PARITY}) = n$. Though this function works as a counterexample for the relationship between $\deg_2(f)$ and $\deg(f)$, it is still inspiring because its modulo-3 degree is large. By writing PARITY as $\frac{1}{2} - \frac{1}{2}\prod_{i=1}^n(1 - 2x_i)$ and taking modulo 3, one can get $\deg_3(\mathsf{PARITY}) = n$. Actually, Gopalan et al. [12] give the following relationship between the polynomial degrees modulo two different primes p and q:

$$\deg_q(f) \geq \frac{n}{\lceil \log_2 p \rceil \deg_p(f) p^{2\deg_p(f)}}.$$

Daunting at the first glance, the inequality implies an essential fact that, as long as $\deg_p(f) = o(\log n)$, a lower bound of $\Omega(n^{1-o(1)})$ for $\deg_q(f)$ follows. Moreover, if m has at least two different prime factors p and q, then $\deg_m(f) \geq \max\left\{\deg_p(f), \deg_q(f)\right\} = \Omega(\log n)$.

Having negated the possibility for the case of prime m, it is natural to study the case of composite number. The systematic study of this case was initiated by Barrington et al. [3]. Alas, whether $\deg_m(f)$ is polynomially related to $\deg(f)$ is still a widely open problem. Though the case m being a prime power is proved to be not true in Gopalan's thesis [10], we are unable to find better separation between $\deg_m(f)$ and $\deg(f)$, for $m = pq$ with p and q being two distinct primes, than the quadratic one given by Li and Sun [19]. This leads to the following conjecture:

[1] The existence and uniqueness are guaranteed by the Möbius inversion, see e.g. [12].

[2] A Boolean function is called non-degenerate if it depends on all its n variables.

[3] It is a folklore that $\mathsf{AC}^0[m] = \mathsf{AC}^0[\mathrm{rad}(m)]$, where $\mathrm{rad}(m)$ is the square-free part of m. Therefore in fact we are able to handle $\mathsf{AC}^0[q]$ circuits for any prime power q.

▶ **Conjecture 1.1.** *Let f be a Boolean function. If m has at least two distinct prime factors, then*

$$\deg(f) = O(\mathrm{poly}(\deg_m(f))).$$

Towards this conjecture, the first step is to deal with *symmetric* Boolean functions. Lee et al. [18] proves that $2\deg_p(f)\deg_q(f) > n$ for any distinct primes p, q and non-trivial symmetric Boolean function $f : \{0,1\}^n \to \{0,1\}$, implying the correctness of Conjecture 1.1 in symmetric cases. Li and Sun [19] improved their bound to $p\deg_p(f) + q\deg_q(f) > n$, which implies $\deg_{pq}(f) > \frac{n}{p+q}$. This is far from being tight; actually, as we will present later, the denominator $p + q$ can be reduced to 3.5.

On the tight lower bound of $\deg(f)$, Nisan and Szegedy [26] give the bound $\deg(f) \geq \log_2 n - O(\log\log n)$ as long as f is non-degenerate. Very recently, this bound is improved to $\deg(f) \geq \log_2 n - O(1)$ by [6, 35], which is tight up to the additive $O(1)$-term by the *address function*. Gathen and Roche [34] show that $\deg(f) \geq \deg_{p(n)}(f) \geq p(n) - 1$ for any non-trivial *symmetric* Boolean function, where $p(n)$ is the largest prime below $n + 2$. (Notice that the module degree gives a lower bound on the degree.) Using the currently best result on prime gaps [2], this gives an $n - O(n^{0.525})$ lower bound. On the other side, Gathen and Roche give a polynomial family with $\deg(f) = n - 3$, and they propose Conjecture 1.2 below with a probabilistic heuristic argument:

▶ **Conjecture 1.2.** *For any non-trivial symmetric Boolean function $f : \{0,1\}^n \to \{0,1\}$,*

$$\deg(f) \geq n - O(1).$$

Our Results. In this work, we prove the following four theorems, giving better lower bounds for $\deg_m(f)$. As we have already mentioned, the gap between $\deg(f)$ and $\deg_{p^k}(f)$ can be arbitrarily large. Nevertheless, we claim that $\deg_{p^k}(f)$ cannot be too small either. This begins with symmetric functions:

▶ **Theorem 1.3.** *For any prime p, positive integer k, and non-trivial symmetric function $f : \{0,1\}^n \to \{0,1\}$,*

$$\deg_{p^k}(f) \geq (p-1)\cdot k$$

when $n \geq (k-1)\varphi(p^\mu) + p^\mu - 1 \in O(p^2k^2)$ where $\mu = \lceil \log_p((p-1)k - 1)\rceil$. The bound $(p-1)\cdot k$ is tight.

The proof of Theorem 1.3 is centered around Mahler expansion [22], which has been deemed useful in several fields of study, from analytic functions to combinatorics. Wilson [36] studied Mahler coefficients and related degree to period of symmetric functions. However, by introducing some more insights, we are able to give a stronger analysis to settle this case once for all. To be a bit more concrete, our argument (i) introduces the base-p period to replace normal period, and then (ii) spans every symmetric functions into two fashions, by MODs or binomials, and then (iii) introduces Mahler coefficient matrix and determines its kernel.

In addition, Theorem 1.3 can be extended to non-degenerate Boolean functions. We achieve this by showing that one can embed an $\omega(1)$-size non-trivial symmetric Boolean function into any non-degenerate functions by applying Erdős–Rado Theorem from Ramsey theory.[4] This leads to the same tight bound, provided that the input size is sufficiently large.

[4] We note that a similar embedding argument has appeared before in [1].

▶ **Theorem 1.4.** *For any prime p, positive integer k, and non-degenerate function $f : \{0,1\}^n \to \{0,1\}$ with sufficiently large n,*

$$\deg_{p^k}(f) \geq (p-1) \cdot k.$$

The bound $(p-1) \cdot k$ is tight.

Now turn to the case of non-prime-power composite m. The following theorem provides a lower bound on $\deg_m(f)$.

▶ **Theorem 1.5.** *For any composite number m with at least two different prime factors p, q and any non-trivial symmetric Boolean function $f : \{0,1\}^n \to \{0,1\}$,*

$$\deg_m(f) \geq \frac{1}{2 + \frac{1}{p-1} + \frac{1}{q-1}} \cdot n.$$

Note that this bound approaches $n/2$ when p and q become larger and larger. It improves the $n/(p+q)$ bound in [19]. To prove this theorem, we show a stronger version of Theorem 1.3 for $k = 1$, which requires a more elaborate analysis. Then we utilizes Periodicity Lemma [9] to obtain the desired lower bound.

On the other hand, the next theorem shows that the bound in Theorem 1.5 cannot be larger than $(1 + o(1))n/2$:

▶ **Theorem 1.6.** *Let m be a square-free composite number. There exists a symmetric Boolean function $f : \{0,1\}^n \to \{0,1\}$ with arbitrarily large n, such that $\deg_m(f) \leq n/2 + o_m(n)$.*[5]

Organization. In Section 2, we give necessary definitions and concepts. Then we give the proofs of Theorem 1.3 and Theorem 1.4 respectively in Section 3.1 and Section 3.2. In Section 4.2 we prove Theorem 1.5, and in Section 4.3 we prove Theorem 1.6. Finally, we conclude the paper in Section 5.

2 Preliminaries

We denote $\{1, 2, \ldots, n\}$ as $[n]$ throughout this paper. $\varphi(\cdot)$ denotes Euler's totient function. Notation $\log^{\circ k}(n)$ is defined as $\underbrace{\log\log\cdots\log}_{k} n$, and $\log^*(n)$ is for the iterated logarithm, that is, $\min\{k : \log^{\circ k}(n) \leq 1\}$.

2.1 Basics of Boolean Functions

An n-bit *Boolean function* $f(x)$ is a mapping from $\{0,1\}^n$ to $\{0,1\}$. Sometimes we write $\boldsymbol{x}$ to indicate the n-dimensional 0-1 vector corresponding to string $x \in \{0,1\}^n$. The following operation will be frequently used: Suppose $x \in \{0,1\}^n$ is a (input) string, and $S \subseteq [n]$ is a set of indices. Denote the string obtained by flipping all bits in x whose indices are in S as $x^{\oplus S}$. As a common practice, $x^{\oplus\{i\}}$ is abbreviated as $x^{\oplus i}$.

[5] The subscript "m" in the $o(\cdot)$ notation means that the hidden factor depends on m.

Here we list some subclasses of Boolean functions, which we will frequently deal with later:

- A Boolean function is called *non-trivial* if it is not a constant.
- A Boolean function is called *non-degenerate* if its value depends on all input bits. In other words, there does not exist such t that, for every $x \in \{0,1\}^n$ the equality $f(x) = f(x^{\oplus t})$ holds. Such bit, if exists, is also known as *dumb* bit.
- A Boolean function is called *symmetric*, if $f(x) = f(y)$ for any x, y satisfying $|x| = |y|$. Here $|x|$ denotes the Hamming weight of x, i.e., number of 1's.

There exists a unique polynomial representing f over $\mathbb{Z}_m$ or $\mathbb{Z}$. More formally:

▶ **Fact 2.1.** *For any Boolean function $f : \{0,1\}^n \to \{0,1\}$, the unique polynomial*

$$\sum_{a\in\{0,1\}^n} f(a) \prod_{i=1}^{n} ((2a_i - 1)x_i + 1 - a_i) =: \sum_{S\subseteq[n]} c_S \prod_{i\in S} x_i$$

represents f over $\mathbb{Z}$. On top of this, the polynomial $\sum_{S\subseteq[n]}(c_S \bmod m)\prod_{i\in S} x_i$ represents f over $\mathbb{Z}_m$.

▶ **Definition 2.2.** *The degree (resp., modulo-m degree) of a Boolean function f, denoted by* $\deg(f)$ *(resp.,* $\deg_m(f)$*), is the degree of the polynomial representing f over $\mathbb{Z}$ (resp., $\mathbb{Z}_m$).*

This measure has some simple but useful properties. The following fact is a consequence of the Chinese Remainder Theorem; see [19, Fact 5].

▶ **Fact 2.3.** *Suppose $f : \{0,1\}^n \to \{0,1\}$ is a Boolean function, and m, m' are coprime. Then* $\deg_{m'm}(f) = \max\{\deg_m(f), \deg_{m'}(f)\}$.

With some input bits fixed, the degree of a Boolean function may decrease. This can be easily derived by substituting those variables with their values. More formally, we define the *restriction* of Boolean functions and restate this fact below.

▶ **Definition 2.4** (Restriction). *Suppose $f : \{0,1\}^n \to \{0,1\}$ is a Boolean function, $S \subseteq [n]$ is a set of indices, and there is a mapping $\sigma : [n]\backslash S \to \{0,1\}$. For every $i \notin S$, fix the i-th bit in the input of f to be $\sigma(i)$ to obtain a new Boolean function with input size $|S|$. We call it the restriction of f over σ, denoted as $f|_\sigma$.*

▶ **Fact 2.5.** *Suppose $f : \{0,1\}^n \to \{0,1\}$ is a Boolean function. For any integer $m \geq 2$ and restriction $f|_\sigma$, we have* $\deg_m(f) \geq \deg_m(f|_\sigma)$.

A common complexity measure, the *sensitivity*, will be used in Section 3.2. Simon gave a lower bound on this measure [32].

▶ **Definition 2.6** (Sensitivity). *Given a Boolean function $f : \{0,1\}^n \to \{0,1\}$ and an input x, we say bit i is sensitive if $f(x) \neq f(x^{\oplus i})$. The sensitivity of f on input x is $s(f,x) := |\{i : i \in [n], f(x) \neq f(x^{\oplus i})\}|$. The sensitivity of f is then defined as $s(f) := \max_x s(f,x)$.*

▶ **Theorem 2.7** ([32]). *For any non-degenerate Boolean function $f : \{0,1\}^n \to \{0,1\}$, we have*

$$s(f) \geq \frac{1}{2}\log n - \frac{1}{2}\log\log n + \frac{1}{2}.$$

2.2 Periodicity and Mahler Expansion

We consider symmetric Boolean functions in this section. For a symmetric function $f : \{0,1\}^n \to \{0,1\}$, clearly there exists a unique $F : \{0,\dots,n\} \to \{0,1\}$, called the *univariate version of* f, such that $f(x) = F(|x|)$ for every x. We call f (and F) *ℓ-periodic*, if $F(a) = F(b)$ for any $0 \le a, b \le n$ satisfying $\ell \,|\, a - b$. For example, f is trivially ℓ-periodic for any $\ell > n$. We are also interested in integer power period length. Hence we introduce the following definition.

▶ **Definition 2.8** (Base-m period). *Assume $f : \{0,1\}^n \to \{0,1\}$ is a symmetric Boolean function. The base-m period is the minimum ℓ such that ℓ is a power of m, and f is ℓ-periodic. Denote it as $\pi_m(f)$.*

Here are some concrete examples for a clearer illustration.

- The not-all-equal NAE function is defined as $\mathsf{NAE}_n(x_1,\dots,x_n) := \mathbb{I}[\exists i,j \text{ s.t. } x_i \ne x_j]$. Then $\pi_3(\mathsf{NAE}_3) = 3$ while $\pi_3(\mathsf{NAE}_4) = 9$. That is, $\pi_m(f)$ may be larger than n.
- If f is a trivial function, then $\pi_3(f) = 1$.

One may write F as a univariate polynomial over $\mathbb{Q}$, but it will not always induce a polynomial when we move to work on $\mathbb{Z}_m$, like what f does. Fortunately, the following representation, also known as *Mahler expansion* [36], serves the purpose similar to polynomial representation.

▶ **Theorem 2.9** (Mahler expansion). *Assume that $f : \{0,1\}^n \to \{0,1\}$ is a symmetric Boolean function, and F is the corresponding univariate version. Let $d := \max\{n, m-1\}$. Then there exists a unique sequence $\alpha_0, \alpha_1, \cdots, \alpha_d \in \mathbb{Z}_m$ such that*

$$\sum_{j=0}^{d} \alpha_j \binom{t}{j} = \begin{cases} F(t), & 0 \le t \le n; \\ 0, & n < m-1 \text{ and } n < t < m. \end{cases}$$

We call $\sum_{j=0}^{d} \alpha_j \binom{t}{j}$ the Mahler expansion of F over $\mathbb{Z}_m$, and α_j the j-th Mahler coefficient.

There are some connections between polynomial degree and Mahler expansion. Over ring $\mathbb{Z}_m$, let $d^* := \max\{\ell : \alpha_\ell \not\equiv 0 \pmod m, \ell \le n\}$. If we only take 0-th to d^*-th terms in the Mahler expansion to get $\hat{F}(t) = \sum_{j=0}^{d^*} \alpha_j \binom{t}{j}$, then $\hat{F}(|x|) = F(|x|)$ for all $x \in \{0,1\}^n$, which implies

▶ **Fact 2.10.** $\deg_m(f) = \max\{\ell : \alpha_\ell \not\equiv 0 \pmod m, \ell \le n\}$.

▶ Remark. The fact above does not hold if we take away the condition $\ell \le n$. The next example shows that on $\mathbb{Z}_m$, the existence of high-order non-zero Mahler coefficient does not imply high degree, if the input length is too short.

▶ Example 2.11. Let $n = 2$ and $f(x) = -x_0x_1 + x_0 + x_1 = x_0 \vee x_1$. On $\mathbb{Z}_5$, one can verify its Mahler expansion is

$$f(x) = \binom{|x|}{1} + 4\binom{|x|}{2} + 2\binom{|x|}{4}.$$

But $\deg_5(f) = 2$.

This phenomenon does not come from nowhere; intuitively speaking, in the Mahler expansion over $\mathbb{Z}_m$, one may need to imitate Lagrange-style interpolation for $|x| > n$, and hence introduce some high-order terms. (Although $|x|$ can never be above n, we need to utilize MOD functions later in this paper, which requires $F(t)$ to be zero for $n < t < m$.)

Wilson [36] showed the following result about the degree and Mahler expansion of symmetric Boolean functions, given the base-p period.

▶ **Theorem 2.12** ([36, Lemma 1]). *Let p be a prime, and t, k be positive integers. Assume $f : \{0,1\}^n \to \{0,1\}$ is a symmetric Boolean function, and $\{\alpha_\ell\}$ are its Mahler coefficients over $\mathbb{Z}_{p^k}$. If f is p^t-periodic, then*

$$\deg_{p^k}(f) \le (k-1)\cdot\varphi(p^t) + p^t - 1.$$

In addition, for any positive integer j and $\ell \ge j\cdot\varphi(p^t) + p^t$, we have $\alpha_\ell \equiv 0 \pmod{p^j}$.

2.3 MOD and Its Mahler Expansion over $\mathbb{Z}_{p^k}$

We first look into the Mahler expansion of weight modular functions. This special case is illuminating in our later proofs. The MOD function is defined as

$$\mathsf{MOD}_n^{c,m}(x) := \mathbb{I}\big[|x| \equiv c \pmod{m}\big] \in \{0,1\},$$

where $n \ge m-1$ denotes the length of input x, and $\mathbb{I}[\cdot]$ is the indicator function. The following theorem gives the degree of MOD_n^{0,p^t}.

▶ **Theorem 2.13** ([36, Theorem 10]). *Let p be a prime, and t, k be positive integers. Denote $d := (k-1)\cdot\varphi(p^t) + p^t - 1$. Then for any $n \ge d$, we have*

$$\deg_{p^k}(\mathsf{MOD}_n^{0,p^t}) = d.$$

In fact, we can achieve a more general result by further analysis. Fix n, p, t and k. Notate the Mahler coefficient of MOD_n^{a,p^t} over $\mathbb{Z}_{p^k}$ as $\alpha_\ell^{(a,p^t)}$ i.e., $\mathsf{MOD}_n^{a,p^t}(x) = \sum_{j=0}^{d} \alpha_j^{(a,p^t)}\binom{|x|}{j}$. Moreover, MOD_n^{a,p^t} can also be represented with $\alpha_\ell^{(0,p^t)}$ as

$$\mathsf{MOD}_n^{a,p^t}(x) = \sum_{j=0}^{d} \alpha_j^{(0,p^t)}\binom{|x|-a}{j}.$$

Then expand each $\binom{|x|-a}{j}$ by Vandermonde convolution to get

$$\alpha_\ell^{(a,p^t)} = \sum_{i=0}^{d-\ell}\binom{-a}{i}\alpha_{i+\ell}^{(0,p^t)}. \tag{1}$$

Specially, by setting $\ell = d$, we get $\alpha_d^{(a,p^t)} = \alpha_d^{(0,p^t)}$. This equation generalizes the theorem to all remainders.

▶ **Corollary 2.14.** *Let p be a prime, and t and k be positive integers. Denote $d := (k-1)\cdot\varphi(p^t) + p^t - 1$. For any $n \ge d$ and $0 \le a < p^t$, we have*

$$\deg_{p^k}(\mathsf{MOD}_n^{a,p^t}) = d.$$

3 Lower Bound of $\deg_{p^k}(f)$

By identifying the degree of MOD_n^{i,p^t} over $\mathbb{Z}_{p^k}$, we show that the degree of all p^t-periodic functions is constantly small since they can be spanned by $\{\mathsf{MOD}_n^{j,p^t}\}_{j=0}^{p^t-1}$. In Section 3.1, we prove that the degree of any p^t-periodic (but not p^{t-1}-periodic) function will not decrease too much from $(k-1)\cdot\varphi(p^t) + p^t - 1$ during the spanning, despite the cancellation of the high-order coefficients. By a Ramsey-type argument in Section 3.2, we further extend our lower bound to all non-degenerate Boolean function with sufficiently many input bits.

3.1 Symmetric Functions – Proof of Theorem 1.3

We begin with the periodicity of symmetric Boolean functions with low degree. In our proof, the following Lucas's Theorem is important.

▶ **Theorem 3.1** (Lucas). *Let $n, m \in \mathbb{N}$, and p be a prime. Assume in base p, n and m can be represented as $n = (n_v n_{v-1} \cdots n_0)_p$ and $m = (m_v m_{v-1} \cdots m_0)_p$ (the number with fewer digits are padded with 0). Then*

$$\binom{n}{m} \equiv \binom{n_v}{m_v}\binom{n_{v-1}}{m_{v-1}} \cdots \binom{n_0}{m_0} \pmod{p}.$$

The next lemma indicates the periodicity of symmetric Boolean functions with low degree.

▶ **Lemma 3.2.** *Let $f : \{0,1\}^n \to \{0,1\}$ be a symmetric Boolean function. For prime p and positive integers t and k, if $\deg_{p^k}(f) \le p^t - 1$, then f is p^t-periodic.*

Proof. Denote $d := \deg_{p^k}(f)$, and suppose α_ℓ are the Mahler coefficients of f over ring $\mathbb{Z}_{p^k}$, i.e., $f(x) = \sum_{j=0}^{d} \alpha_j \binom{|x|}{j} \bmod p^k$. According to Lucas's Theorem, if $a \equiv b \pmod{p^t}$, then for any $0 \le j \le p^t - 1$, we have $\binom{a}{j} \equiv \binom{a_v}{j_v} \cdots \binom{a_t}{j_t}\binom{a_{t-1}}{j_{t-1}} \cdots \binom{a_0}{j_0} \pmod{p}$. Here, a_i (resp. b_i and j_i) is the representation of a (resp. b and j) in the base p. Note that $j_i = 0$ for any $i \ge t$. Hence $\binom{a}{j} \equiv \binom{a_{t-1}}{j_{t-1}} \cdots \binom{a_0}{j_0} \pmod{p}$. For the same reason, $\binom{b}{j} \equiv \binom{b_{t-1}}{j_{t-1}} \cdots \binom{b_0}{j_0} \pmod{p}$. In addition, a and b's last t digits are the same as $a \equiv b \pmod{p^t}$. Thus $\sum_{j=0}^{d} \alpha_j \binom{a}{j} \bmod p = \sum_{j=0}^{d} \alpha_j \binom{b}{j} \bmod p$. By the definition that $f(x) \in \{0,1\}$, we get $\sum_{j=0}^{d} \alpha_j \binom{a}{j} \bmod p^k = \sum_{j=0}^{d} \alpha_j \binom{b}{j} \bmod p^k$. ◀

Next, provided $\pi_p(f)$, we give some lower bounds on the degree by the following two lemmas, conditioned on that n is large enough. Together with the lemma above we lead to a contradiction, and our theorem follows eventually. Before continuing, notice again that the value $f(x)$ is related only to the Hamming weight of x, and thus $f(x) = \sum_{0 \le i \le n : F(i)=1} \mathsf{MOD}_n^{i,n+1}(x)$. Specially, if $f(x)$ is t-periodic, then we can write $f(x)$ as $f(x) = \sum_{0 \le i \le t-1 : F(i)=1} \mathsf{MOD}_n^{i,t}(x)$ and apply Corollary 2.14.

▶ **Lemma 3.3.** *Assume p is a prime, and $f : \{0,1\}^n \to \{0,1\}$ is a symmetric Boolean function of period p. If for some positive integer k, it holds that $n \ge (p-1) \cdot k$, then*

$$\deg_{p^k}(f) = (p-1) \cdot k.$$

Proof. Expand $f(x)$ to get $\sum_{0 \le i \le p-1 : F(i)=1} \mathsf{MOD}_n^{i,p}(x)$. According to Corollary 2.14, the degree of each term in the summation is $\deg_{p^k}\left(\mathsf{MOD}_n^{i,p}\right) = (p-1) \cdot k =: d$. On the other hand, Section 2.3 says the coefficient of degree d term is identical in the polynomial of every MOD. As $f(x)$ is non-trivial, the number of terms in the summation, denoted as N, is neither 0 nor p. Therefore, the degree d term in $f(x)$ has coefficient $N \cdot \alpha_d^{(0,p)} \not\equiv 0 \pmod{p}$, implying $\deg_{p^k}(f) = d$ as desired. ◀

▶ **Lemma 3.4.** *Assume p is a prime, and $f : \{0,1\}^n \to \{0,1\}$ is a symmetric Boolean function with $\pi_p(f) = p^t$. Here $t \ge 1$ and $n \ge (k-1) \cdot \varphi(p^t) + p^t - 1$. Then*

$$\deg_{p^k}(f) \ge (k-2) \cdot \varphi(p^t) + p^t.$$

Proof. Consider over the ring $\mathbb{Z}_{p^k}$. Let $d := (k-1) \cdot \varphi(p^t) + p^t - 1$. Provided p^t, we abbreviate $\alpha_\ell^{(j,p^t)}$, the ℓ-th Mahler coefficients of MOD_n^{j,p^t}, as $\alpha_\ell^{(j)}$ for convenience. According to Corollary 2.14, we have $\deg_{p^k}\left(\mathsf{MOD}_n^{j,p^t}\right) = d$.

For all $0 \le i \le p^t - 1$, Theorem 2.12 implies the fact that $\alpha_\ell^{(i)}$ can be divided by p^{k-2} when $\ell \ge (k-2)\cdot\varphi(p^t) + p^t = d - \varphi(p^t) + 1$. Therefore, we divide every such coefficient by p^{k-2}, and then take the remainder modulo p to get $\tilde{\alpha}_\ell^{(i)} := \left(\alpha_\ell^{(i)}/p^{k-2}\right) \bmod p$, where $d - \varphi(p^t) + 1 \le \ell \le d$. All these $\tilde{\alpha}_\ell^{(i)}$ forms the matrix

$$\boldsymbol{S} := \begin{bmatrix} \tilde{\alpha}_d^{(0)} & \cdots & \tilde{\alpha}_d^{(p^t-1)} \\ \vdots & \ddots & \vdots \\ \tilde{\alpha}_{d-\varphi(p^t)+1}^{(0)} & \cdots & \tilde{\alpha}_{d-\varphi(p^t)+1}^{(p^t-1)} \end{bmatrix} \in \mathbb{F}_p^{\varphi(p^t)\cdot p^t}.$$

Take its first $\varphi(p^t)$ columns to get a square matrix

$$\boldsymbol{S}' := \begin{bmatrix} \tilde{\alpha}_d^{(0)} & \cdots & \tilde{\alpha}_d^{(\varphi(p^t)-1)} \\ \vdots & \ddots & \vdots \\ \tilde{\alpha}_{d-\varphi(p^t)+1}^{(0)} & \cdots & \tilde{\alpha}_{d-\varphi(p^t)+1}^{(\varphi(p^t)-1)} \end{bmatrix}.$$

When $d - \varphi(p^t) + 1 \le \ell \le d$, divide both sides of Equation (1) by p^{k-2} and take the remainder modulo p to get

$$\tilde{\alpha}_\ell^{(a)} = \sum_{j=0}^{d-\ell} \binom{-a}{j} \tilde{\alpha}_{j+\ell}^{(0)},$$

which leads to the following decomposition of matrix $\boldsymbol{S}'$:

$$\boldsymbol{S}' = \begin{bmatrix} \tilde{\alpha}_d^{(0)} & & \\ \vdots & \ddots & \\ \tilde{\alpha}_{d-\varphi(p^t)+1}^{(0)} & \cdots & \tilde{\alpha}_d^{(0)} \end{bmatrix} \cdot \begin{bmatrix} \binom{0}{0} & \cdots & \binom{1-\varphi(p^t)}{0} \\ \vdots & \ddots & \vdots \\ \binom{0}{\varphi(p^t)-1} & \cdots & \binom{1-\varphi(p^t)}{\varphi(p^t)-1} \end{bmatrix} =: \boldsymbol{T}\cdot\boldsymbol{C}.$$

As $\tilde{\alpha}_d^{(0)} \ne 0$, the first matrix has determinant $\det(\boldsymbol{T}) \ne 0$. The latter one, consisting of binomial coefficients, is also invertible. We will prove this fact later. Eventually, $\operatorname{rank}(\boldsymbol{S}') = \varphi(p^t)$, so the kernel of $\boldsymbol{S}$ has dimension $\dim\ker\boldsymbol{S} = (p^t) - \varphi(p^t) = p^{t-1}$.

On one hand, because $f(x)$ is p^t-periodic, we can expand it by $\{\mathsf{MOD}_n^{j,p^t}\}_{j=0}^{p^t-1}$ with coefficients w_j.

$$f(x) = \sum_{j=0}^{p^t-1} w_j \mathsf{MOD}_n^{j,p^t}(x) = \sum_{j=0}^{p^t-1}\left(w_j \sum_{\ell=0}^{d} \alpha_\ell^{(j)}\binom{|x|}{\ell}\right) = \sum_{\ell=0}^{d}\left(\left(\sum_{j=0}^{p^t-1} w_j\alpha_\ell^{(j)}\right)\binom{|x|}{\ell}\right). \quad (2)$$

On the other hand, $\mathsf{MOD}_n^{i,p^{t-1}}$ can also be spanned by $\{\mathsf{MOD}_n^{j,p^t}\}$. In other words, assume $w_j^{(i)} := \mathbb{I}[i \equiv j \pmod{p^{t-1}}]$, then

$$\mathsf{MOD}_n^{i,p^{t-1}} = \sum_{j=0}^{p^t-1} w_j^{(i)}\mathsf{MOD}_n^{j,p^t} = \sum_{\ell=0}^{d}\left(\left(\sum_{j=0}^{p^t-1} w_j^{(i)}\alpha_\ell^{(j)}\right)\binom{|x|}{\ell}\right). \quad (3)$$

We claim $\deg_{p^k}(\mathsf{MOD}_n^{i,p^{t-1}}) \le d - \varphi(p^t)$, and because $n \ge d$, the coefficients of highest $\varphi(p^t)$ terms in its Mahler expansion (i.e., from degree $d - \varphi(p^t) + 1$ to degree d) are all zero. This is due to Corollary 2.14, where we have

$$\begin{aligned}\deg_{p^k}(\mathsf{MOD}_n^{i,p^{t-1}}) &= (k-1)\varphi(p^{t-1}) + p^{t-1} - 1\\ &= p^{t-2}\left((k-1)(p-1)+p\right) - 1\\ &\le p^{t-2}\left((k-1)(p^2-p)+p\right) - 1\\ &= d - \varphi(p^t).\end{aligned}$$

Further, if we set column vector $\boldsymbol{w}^{(i)} = \left(w_0^{(i)}, ..., w_{p^t-1}^{(i)}\right)^\top$ where $0 \le i \le p^{t-1} - 1$, then the above fact, together with Equation (3), indicates the following equation:

$$\boldsymbol{S}\boldsymbol{w}^{(i)} = \left[\sum_{j=0}^{p^t-1} w_j^{(i)} \tilde{\alpha}_d^{(j)}, \sum_{j=0}^{p^t-1} w_j^{(i)} \tilde{\alpha}_{d-1}^{(j)}, \cdots, \sum_{j=0}^{p^t-1} w_j^{(i)} \tilde{\alpha}_{d-\varphi(p^t)+1}^{(j)}\right]^\top = \mathbf{0}.$$

One can verify that $\left\{\boldsymbol{w}^{(0)}, \cdots, \boldsymbol{w}^{(p^{t-1}-1)}\right\}$ is linear independent, and therefore they form a base of the kernel of $\boldsymbol{S}$ i.e., $\ker \boldsymbol{S} = \operatorname{span}\left\{\boldsymbol{w}^{(0)}, \cdots, \boldsymbol{w}^{(p^{t-1}-1)}\right\}$.

However, $f(x)$ is not p^{t-1}-periodic because $\pi_p(f) = p^t$. This means $f(x)$ cannot be written as the sum of some $\mathsf{MOD}_n^{i,p^{t-1}}$. Thus $\boldsymbol{w} := (w_0, ..., w_{p^t-1})^\top \notin \operatorname{span}\left\{\boldsymbol{w}^{(0)}, \cdots, \boldsymbol{w}^{(p^{t-1}-1)}\right\}$ i.e., $\boldsymbol{S}\boldsymbol{w} \neq \mathbf{0}$. This leads to the existence of $D \in [d - \varphi(p^t) + 1, d]$ such that $\sum_{j=0}^{p^t-1} w_j^{(i)} \tilde{\alpha}_D^{(j)} \neq 0$. Namely, the degree D term in Equation (2) exists as desired. ◀

To finish the proof of this lemma, we show that the matrix $\boldsymbol{C}$ is invertible, due to the following proposition.

▶ **Proposition 3.5.** $\det(\boldsymbol{C}) = \pm 1$.

Proof. In fact,

$$\boldsymbol{C} = \operatorname{diag}\{1, -1, 1, -1, ...(-1)^{m-1}\} \cdot \begin{bmatrix} 1 & 1 & \binom{1}{1} & \cdots & \binom{m-2}{m-2} \\ 0 & \binom{1}{0} & \binom{2}{1} & \cdots & \binom{m-1}{m-2} \\ 0 & \binom{2}{0} & \binom{3}{1} & \cdots & \binom{m}{m-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \binom{m-1}{0} & \binom{m}{1} & \cdots & \binom{2m-3}{m-2} \end{bmatrix}$$

where $m := \varphi(p^t)$. Denote the second matrix as $\boldsymbol{C}'$. Take Row m of $\boldsymbol{C}'$ and subtract Row $m-1$ from it. Then take Row $m-1$ and subtract Row $m-2$ from it. $\cdots$ Take Row 3 and subtract Row 2 from it. Then C' has been transformed to

$$\begin{bmatrix} 1 & 1 & \binom{1}{1} & \cdots & \binom{m-2}{m-2} \\ 0 & \binom{1}{0} & \binom{2}{1} & \cdots & \binom{m-1}{m-2} \\ 0 & 0 & \binom{2}{0} & \cdots & \binom{m-1}{m-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \binom{m-1}{0} & \cdots & \binom{2m-4}{m-3} \end{bmatrix}.$$

Keep repeating this step, and $\boldsymbol{C}'$ will be transformed into an upper triangular matrix, whose diagonal elements are all 1. ◀

Now we are ready to prove Theorem 1.3.

Proof of Theorem 1.3. Note that Lemma 3.3 provides tight instances, so below we are going to prove the inequality.

Assume towards contradiction that there exists $f : \{0,1\}^n \to \{0,1\}$ satisfying $\deg_{p^k}(f) < (p-1)\cdot k$, and $n \geq (k-1)\cdot\varphi(p^\mu) + p^\mu - 1$ where $\mu = \lceil \log_p((p-1)k-1)\rceil$. Lemma 3.2 tells us that f is p^μ-periodic. The non-triviality of f indicates that there exists $1 \leq t \leq \mu$ such that $\pi_p(f) = p^t$.

If $t = 1$, then according to Lemma 3.3 we have $\deg_{p^k}(f) = (p-1)\cdot k$. If $t \geq 2$, then Lemma 3.4 indicates

$$\begin{aligned}\deg_{p^k}(f) &\geq (k-2)\cdot\varphi(p^t) + p^t\\ &\geq (k-2)\cdot\varphi(p^2) + p^2\\ &= k\cdot(p-1)^2 + 2p - p^2 + (p-1)\cdot k\\ &\geq (p-1)^2 + 2p - p^2 + (p-1)\cdot k\\ &> (p-1)\cdot k.\end{aligned}$$

Both cases lead to contradiction. ◀

3.2 Non-degenerate Functions – Proof of Theorem 1.4

For the general non-degenerate case, our key idea is to embed a symmetric Boolean function into it, and then apply Theorem 1.3. The following lemma is crucial.

▶ **Lemma 3.6.** *There exists a monotone increasing function $r(n) = \omega(1)$ such that the following holds. Let $f : \{0,1\}^n \to \{0,1\}$ be a non-degenerate Boolean function. Then there exists a set of indices $S \subseteq [n]$ with $|S| \geq r(n)$, and a mapping $\sigma : [n]\backslash S \to \{0,1\}$ such that $f|_\sigma$ is a non-trivial symmetric Boolean function.*

Generally speaking, with this lemma in hand, every $h(n)$ lower bound on complexity measure of symmetric functions that is monotone decreasing w.r.t. restriction (e.g., Fact 2.5) can yield an $h(r(n))$ lower bound on that of all non-degenerate functions. In the setting of modulo-p^k degree, the bound $h(n)$ is a constant function (when n is large than some threshold), so we can get the same bound, except that the threshold for n blows up. However, as indicated by our proof, the function $r(n)$ grows extraordinary slow (approximately the square root of iterated logarithm of n).

First, let us see how to utilize this lemma in proving Theorem 1.4.

Proof of Theorem 1.4. As long as Lemma 3.6 holds, by Fact 2.5 and Theorem 1.3, if $n \geq r^{-1}((k-1)\cdot\varphi(p^\mu) + p^\mu - 1)$, then $\deg_{p^k}(f) \geq (p-1)\cdot k$, deriving the desired lower bound. In addition, any symmetric function with period p (and input long enough) still serves as an instance with $\deg_{p^k}(f) = (p-1)\cdot k$. ◀

In the rest part of this section we prove Lemma 3.6.

For convenience, we introduce the following notation. If $x_i = 1$ for every $i \in S \subseteq [n]$, then define $\mathrm{DOWN}(S,x,k) := \left\{x^{\oplus T} \mid T \subseteq S, |T| = k\right\}$. Intuitively speaking, it is the set of strings obtained by flipping k bits, whose indices are in S, of x from 1 to 0.

According to Theorem 2.7, there exists $\tilde{x}$ such that $s(f,\tilde{x}) = \Omega(\log n)$. Without loss of generality we assume the set $\{i \in [n] : \tilde{x}_i = 1, f(\tilde{x}) \neq f(\tilde{x}^{\oplus i})\}$, defined as S_0, is of cardinality $\Omega(\log n)$. Recursively define S_t to be the largest set satisfying the following two conditions:

- $S_t \subseteq S_{t-1}$.
- The value $f(y)$ are identical for any $y \in \mathrm{DOWN}(S_t, \tilde{x}, t)$.

We then make the following claim, and prove it.

▷ Claim 3.7.

$$|S_t| = \Omega\left(\log^{\circ(t-1)}(|S_{t-1}|)\right).$$

Our proof relies on Erdős–Rado Theorem [8] from Ramsey theory on hypergraphs.

▶ **Definition 3.8** (k-Uniform Hypergraph Ramsey Number)**.** *Suppose V is a set of vertices, and all size-k subsets of V forms $\mathcal{F}_k(V)$. If $E \subseteq \mathcal{F}_k(V)$, then we call (V, E) as a k-uniform hypergraph of order $|V|$. Naturally, we call $(V, \mathcal{F}_k(V))$ a complete k-uniform hypergraph.*

If the following property holds for complete k-uniform hypergraph of order M but not $M-1$, then $r_k(s,t) := M$ is called the k-uniform hypergraph Ramsey number: color every k-hyperedge red or blue arbitrarily, then there must exist a complete red hyper-subgraph of order s, or a complete blue hyper-subgraph of order t.

▶ **Theorem 3.9** ([8])**.** *$r_2(s,t) \le \binom{s+t-2}{t-1}$ and*

$$r_k(s,t) \le 2^{\binom{r_{k-1}(s-1,t-1)}{k-1}}, \quad (k > 2).$$

This theorem implies the following fact: if we color k-uniform hypergraph of order n with two colors, then there exists a monochromatic clique of size $\Omega(\log^{\circ k}(n))$.

Proof of Claim 3.7. Construct the following complete t-uniform hypergraph. The vertex set is S_{t-1}. For any $x \in \mathrm{DOWN}(S_{t-1}, \tilde{x}, t)$, we color the hyperedge $\{i : x_i \neq \tilde{x}_i\}$ with red if $f(x) = 1$, or blue otherwise. Take the largest monochromatic clique and suppose its vertex set is S. According to Theorem 3.9, $|S| = \Omega(\log^{\circ k}(n))$. Furthermore, the monochromaticity implies the value $f(z)$ is identical for any $z \in \mathrm{DOWN}(S, \tilde{x}, t)$. Hence S satisfies the desired conditions where our claim follows immediately. ◁

With the help of the claim and notations above, we can complete our proof of Lemma 3.6.

Proof of Lemma 3.6. By invoking Claim 3.7 iteratively,

$$|S_t| \ge Z \cdot \log^{\circ((t-1)t/2+1)}(n),$$

where Z is a positive constant irrelevant to n. Take $t' = t'(n) := \lfloor \sqrt{\log^*(n) - 2} \rfloor$. Then $(t'-1)t'/2 + 1 < \log^*(n)$ and $|S_{t'}| = \omega(1)$. In addition, define

$$r = r(n) := \min\left\{Z \cdot \log^{\circ(t'(t'-1)/2+1)}(n), t'(n)\right\} = \omega(1).$$

Then we have $r(n) \le |S_{r(n)}|$. This is because

- if $t'(n) \le Z \cdot \log^{\circ(t'(t'-1)/2+1)}(n)$, then we have $r(n) = t'(n) \le Z \cdot \log^{\circ(r(r-1)/2+1)}(n) \le |S_{r(n)}|$;
- if $t'(n) > Z \cdot \log^{\circ(t'(t'-1)/2+1)}(n)$, then $t'(n) > r(n)$, so $r(n) = Z \cdot \log^{\circ(t'(t'-1)/2+1)}(n) \le Z \cdot \log^{\circ(r(r-1)/2+1)}(n) \le |S_{r(n)}|$.

Now take a size $r(n)$ subset T of $S_{r(n)}$ arbitrarily. Define the mapping $\sigma : [n] \backslash T \to \{0,1\}$ such that $\sigma(i) = \tilde{x}_i$. Restrict f over σ to obtain a new function $g := f|_\sigma$. We will prove g is symmetric and non-trivial, and then the lemma follows immediately.

Symmetric. Assume $x, y \in \{0,1\}^{r(n)}$ such that $|x| = |y|$. Define x' (resp. y') to be the string of size n obtained from x (resp. y) and σ. Recall the definition of $S_{r(n)}$. That is, for any $i \in S_{r(n)}$, it holds that $\tilde{x}_i = 1$. Therefore,

$$x', y' \in \mathrm{DOWN}(S_{r(n)}, \tilde{x}, |x|) \subseteq \mathrm{DOWN}(S_{|x|}, \tilde{x}, |x|).$$

By definition of DOWN, we have $f(x') = f(y')$ i.e., $g(x) = g(y)$.

Non-trivial. Assume $z, w \in \{0,1\}^{r(n)}$ such that $|z| = 0$ and $|w| = 1$. We define z' and w' similarly to x' and y'. Suppose $z' = w' \oplus e_i$. As $i \in T \subseteq S_0$, the i-th bit is sensitive. Therefore, $f(z') \neq f(w')$ i.e., $g(z) \neq g(w)$. ◀

4 Lower Bounds of $\deg_{pq}(f)$ for Symmetric Functions

We first go further with the analysis of Mahler coefficients of MOD functions, then prove Theorem 1.5 in Section 4.2, and give an instance in Section 4.3, showing one can never improve the constant factor 1/2.

4.1 More Analyses of MOD and Its Mahler Coefficients

Let p be a prime and t be a positive integer. Consider over the ring $\mathbb{Z}_p$. Recall the notation $\alpha_\ell^{(a,p^t)}$: it is the ℓ-th Mahler coefficient of MOD_n^{a,p^t}. Since $\deg_p(\mathsf{MOD}_n^{a,p^t}) = p^t - 1$, the following $p^t \times p^t$ matrix collects all the coefficients of MOD_n^{a,p^t}:

$$\boldsymbol{A}_{p^t} := \begin{bmatrix} \alpha_0^{(0,p^t)} & \cdots & \alpha_0^{(p^t-1,p^t)} \\ \vdots & \ddots & \vdots \\ \alpha_{p^t-1}^{(0,p^t)} & \cdots & \alpha_{p^t-1}^{(p^t-1,p^t)} \end{bmatrix}.$$

In fact,

$$(\boldsymbol{A}_{p^t})_{i,j} = \alpha_i^{(j,p^t)} = \binom{p^t-1-j}{p^t-1-i}.$$

This is because

$$\sum_{i=0}^{p^t-1} \binom{p^t-1-j}{p^t-1-i}\binom{|x|}{i} = \binom{p^t-1-j+|x|}{p^t-1} = \begin{cases} 1 & \text{if } |x| \equiv j \pmod{p^t}, \\ 0 & \text{otherwise}, \end{cases} \tag{4}$$

by Vandermonde's convolution.

The matrix $\boldsymbol{A}_{p^t}$ has many elegant properties. For example, the following one shows the relationship between $\boldsymbol{A}_{p^t}$ and $\boldsymbol{A}_p$. We use $\otimes$ to denote matrix tensor product.

▶ **Proposition 4.1.** *On the ring $\mathbb{Z}_p$,*

$$\boldsymbol{A}_{p^t} = \boldsymbol{A}_p^{\otimes t} := \underbrace{\boldsymbol{A}_p \otimes \boldsymbol{A}_p \otimes \cdots \otimes \boldsymbol{A}_p}_{t}.$$

Proof. Let i_ℓ and j_ℓ be the representation of i and j in base p. Then by Lucas's Theorem,

$$\binom{p^t-1-j}{p^t-1-i} \equiv \binom{\sum_{\ell=0}^{t-1}(p-1-j_\ell)\cdot p^\ell}{\sum_{\ell=0}^{t-1}(p-1-i_\ell)\cdot p^\ell} \equiv \prod_{\ell=0}^{t-1}\binom{p-1-j_\ell}{p-1-i_\ell} \equiv \prod_{\ell=0}^{t-1}(\boldsymbol{A}_p)_{i_\ell,j_\ell} \pmod{p}. ◀$$

Below we give another observation, which assists with our proof of Theorem 1.5.

▶ **Lemma 4.2.** *Suppose p is a prime, and $n < p - 1$ is a positive integer. Then for any $v \in \{0,1\}^p$ satisfying $v_i \neq v_j$ for some $0 \leq i < j \leq n$, there exists $\lfloor n/2 \rfloor + 1 \leq \ell \leq n$ such that $(\boldsymbol{A}_p \boldsymbol{v})_\ell \neq 0$.*

Our proof of Lemma 4.2 utilizes the following proposition on another binomial coefficient matrix.

▶ **Proposition 4.3.** *For any prime p, integers j, k with $j + k < p$ and distinct $a_0, \dots, a_k \in \mathbb{F}_p$ satisfying $a_0, \dots, a_k \geq j$, the matrix*

$$\begin{bmatrix} \binom{a_0}{j} & \binom{a_1}{j} & \cdots & \binom{a_k}{j} \\ \binom{a_0}{j+1} & \binom{a_1}{j+1} & \cdots & \binom{a_k}{j+1} \\ \vdots & \vdots & \ddots & \vdots \\ \binom{a_0}{j+k} & \binom{a_1}{j+k} & \cdots & \binom{a_k}{j+k} \end{bmatrix}$$

is invertible over $\mathbb{F}_p$.

Proof. One can verify that

$$\operatorname{diag}\left\{ \frac{(j+0)^{\underline{0}}}{\binom{a_0}{j}}, \dots, \frac{(j+k)^{\underline{k}}}{\binom{a_k}{j}} \right\} \cdot S \cdot \begin{bmatrix} \binom{a_0}{j} & \cdots & \binom{a_k}{j} \\ \vdots & \ddots & \vdots \\ \binom{a_0}{j+k} & \cdots & \binom{a_k}{j+k} \end{bmatrix} = \begin{bmatrix} (a_0 - j)^0 & \cdots & (a_k - j)^0 \\ \vdots & \ddots & \vdots \\ (a_0 - j)^k & \cdots & (a_k - j)^k \end{bmatrix},$$

where S is the second Stirling number matrix, i.e., $S_{ij} = \left\{ {i \atop j} \right\}$, and the notation $x^{\underline{y}}$ stands for the falling factorial power $x(x-1)\cdots(x-y+1)$. The Vandermonde matrix on the R.H.S. is also invertible because $a_0, \dots, a_k$ are distinct. ◀

Proof of Lemma 4.2. Assume towards contradiction that there exists some v satisfying the condition, but $(\boldsymbol{A}_p v)_\ell = 0$ for all $\lfloor n/2 \rfloor + 1 \leq \ell \leq n$. In other words, if we take row $\lfloor n/2 \rfloor + 1$ to n and column 0 to n from $\boldsymbol{A}_p$ to get another $\lceil n/2 \rceil \times (n+1)$ matrix $\boldsymbol{B}$ i.e.,

$$\boldsymbol{B} := \begin{bmatrix} \binom{p-1-0}{p-1-(\lfloor n/2 \rfloor + 1)} & \binom{p-1-1}{p-1-(\lfloor n/2 \rfloor + 1)} & \cdots & \binom{p-1-n}{p-1-(\lfloor n/2 \rfloor + 1)} \\ \vdots & \vdots & \ddots & \vdots \\ \binom{p-1-0}{p-1-n} & \binom{p-1-1}{p-1-n} & \cdots & \binom{p-1-n}{p-1-n} \end{bmatrix},$$

then $\boldsymbol{B}\boldsymbol{v}' = \boldsymbol{0}$ where $\boldsymbol{v}' = \{v_0, ..., v_n\}^T$. (This is because $(\boldsymbol{A}_p)_{i,j} = \binom{p-1-j}{p-1-i} = 0$ for all $\lfloor n/2 \rfloor + 1 \leq i \leq n$ and $n + 1 \leq j \leq p - 1$.)

Next, for any $t \in [(\lfloor n/2 \rfloor + 1), n]$, the sum of row t is $\binom{p-1-0}{p-1-t} + \binom{p-1-1}{p-1-t} + \cdots + \binom{p-1-n}{p-1-t} = \binom{p}{p-t} \equiv 0 \pmod{p}$. Therefore $\boldsymbol{B}\boldsymbol{1} = \boldsymbol{0}$, so we can assume the number of 1's in $\boldsymbol{v}'$ is no more that $\lceil n/2 \rceil$, without loss of generality. (Otherwise, subtract $\boldsymbol{v}'$ from $\boldsymbol{1}$.) This means that we can take $s \leq \lceil n/2 \rceil$ column vectors of $\boldsymbol{B}$, the summation of which is $\boldsymbol{0}$, and furthermore, the last s dimensions of these vectors form a singular matrix with form $\boldsymbol{B}'_{i,j} = \binom{a_j}{p-1-n+s-1+i}$. However, by flipping it upside down and applying Proposition 4.3, this matrix is invertible. ◀

4.2 Proof of Theorem 1.5

Our proof requires the following two lemmas. The first one is often referred to as Periodicity Lemma. It says any function with coprime periods is constant, if the domain is large enough.

▶ **Lemma 4.4** (Periodicity Lemma, [9]). *Let g be an a-periodic and b-periodic function on domain $\{0,1,\dots,n\}$ with $\gcd(a,b)=1$ and $n \geq a+b-2$. Then g is a constant function.*

The next one can be regarded as a stronger version of Theorem 1.3 with $k=1$.

▶ **Lemma 4.5.** *Assume p is a prime. For any non-trivial symmetric $f:\{0,1\}^n \to \{0,1\}$,*

$$\deg_p(f) \geq \min\left\{\frac{n}{2}, \left(1-\frac{1}{p}\right)\pi_p(f)\right\}.$$

Note that the base-p period appear explicitly in the lower bound. This allows us to apply Lemma 4.4. We prove Theorem 1.5 first.

Proof of Theorem 1.5. If $\max\{\deg_p(f), \deg_q(f)\} \geq \frac{n}{2}$, the theorem then follows naturally. Otherwise, according to Fact 2.3 and Lemma 4.5, we have

$$\deg_m(f) \geq \deg_{pq}(f) = \max\{\deg_p(f), \deg_q(f)\} \geq \max\left\{\left(1-\frac{1}{p}\right)\pi_p(f), \left(1-\frac{1}{q}\right)\pi_q(f)\right\}. \tag{5}$$

On the other hand, the non-triviality of $f(x)$ implies $\pi_p(f)+\pi_q(f) > n+2$ owing to Lemma 4.4. The last term in Inequality (5) is $> \frac{n+2}{2+1/(p-1)+1/(q-1)} > \frac{n}{2+1/(p-1)+1/(q-1)}$, and hence the theorem is also true. ◀

It remains to show why Lemma 4.5 is true.

Proof of Lemma 4.5. Consider over the ring $\mathbb{Z}_p$. Suppose $\pi_p(f)=p^t$. We write $f(x)$ as we did in Equation (2), and let α_ℓ be the ℓ-th Mahler coefficient of $f(x)$. Then $\sum_{j=0}^{p^t-1} w_j \alpha_\ell^{(j,p^t)} = \alpha_\ell$, or

$$\boldsymbol{\alpha} = \boldsymbol{A}_{p^t}\boldsymbol{w} \tag{6}$$

if we set $\boldsymbol{w} := (w_0, ..., w_{p^t-1})^\top$ and $\boldsymbol{\alpha} := (\alpha_0, ..., \alpha_{p^t-1})^\top$.

Divide $\boldsymbol{w}$ and $\boldsymbol{\alpha}$ into blocks of length p^{t-1} as $\boldsymbol{w} = (\boldsymbol{w}^{\langle 0\rangle}, \dots, \boldsymbol{w}^{\langle p-1\rangle})^\top$ and $\boldsymbol{\alpha} = (\boldsymbol{\alpha}^{\langle 0\rangle}, \dots, \boldsymbol{\alpha}^{\langle 0\rangle})^\top$ where $\boldsymbol{w}^{\langle i\rangle} \in \{0,1\}^{p^{t-1}}, \boldsymbol{\alpha}^{\langle i\rangle} \in \mathbb{F}_p^{p^{t-1}}$. By Proposition 4.1, we have

$$\boldsymbol{\alpha}^{\langle i\rangle} = \boldsymbol{A}_{p^{t-1}} \sum_{j=0}^{p-1}\left((\boldsymbol{A}_p)_{ij}\boldsymbol{w}^{\langle j\rangle}\right). \tag{7}$$

Consider two cases. One deals with the case $\pi_p(f)=p^t<n$, where we show $\deg_p(f) > \frac{p-1}{p}\cdot\pi_p(f)$; another deals with $\pi_p(f) \geq n$, where we can obtain $\deg_p(f) \geq n/2$.

Case I ($\boldsymbol{p^t < n}$). First, assume $\boldsymbol{\alpha}^{\langle p-1\rangle} = \boldsymbol{0}$. Note that $\boldsymbol{A}_{p^{t-1}}$ is full-rank according to Proposition 4.3, which allows Equation (7) to be transformed into

$$\boldsymbol{A}_{p^{t-1}}^{-1}\boldsymbol{\alpha}^{\langle p-1\rangle} = \sum_{j=0}^{p-1}\left((\boldsymbol{A}_p)_{p-1,j}\boldsymbol{w}^{\langle j\rangle}\right).$$

Since $(\boldsymbol{A}_p)_{p-1,j}=1$, we have $\sum_{j=0}^{p-1}\boldsymbol{w}^{\langle j\rangle}=\boldsymbol{0}$. This implies $\boldsymbol{w}^{\langle 0\rangle} = \dots = \boldsymbol{w}^{\langle p-1\rangle}$ as $\boldsymbol{w}^{\langle i\rangle} \in \{0,1\}^{p^{t-1}}$, and further, $f(x)$ becomes p^{t-1}-periodic, conflicting with $\pi_p(f)=p^t$. Eventually, we have $\boldsymbol{\alpha}^{\langle p-1\rangle} \neq \boldsymbol{0}$. Because $n > p^t$, the highest non-zero Mahler coefficient indicates the degree of f (see the remark below Fact 2.10), and then $\deg_p(f) > (p-1)p^{t-1} = \frac{p-1}{p}\cdot\pi_p(f)$.

Case II ($p^t \geq n$). By Equation (6),

$$\left(\boldsymbol{I}_p \otimes (\boldsymbol{A}_{p^{t-1}})^{-1}\right) \boldsymbol{A}_{p^t} \boldsymbol{w} = \left(\boldsymbol{I}_p \otimes (\boldsymbol{A}_{p^{t-1}})^{-1}\right) \boldsymbol{\alpha}. \tag{8}$$

The R.H.S. of (8) is just

$$\left(\boldsymbol{I}_p \otimes (\boldsymbol{A}_{p^{t-1}})^{-1}\right) \boldsymbol{\alpha} = \begin{bmatrix} (\boldsymbol{A}_{p^{t-1}})^{-1}\boldsymbol{\alpha}^{\langle 0\rangle} \\ \vdots \\ (\boldsymbol{A}_{p^{t-1}})^{-1}\boldsymbol{\alpha}^{\langle p-1\rangle} \end{bmatrix} =: \begin{bmatrix} \boldsymbol{\beta}^{\langle 0\rangle} \\ \vdots \\ \boldsymbol{\beta}^{\langle p-1\rangle} \end{bmatrix}.$$

The L.H.S. of (8) can be written as

$$\begin{aligned} \left(\boldsymbol{I}_p \otimes (\boldsymbol{A}_{p^{t-1}})^{-1}\right) \boldsymbol{A}_{p^t} \boldsymbol{w} &= \left(\boldsymbol{I}_p \otimes (\boldsymbol{A}_{p^{t-1}})^{-1}\right) (\boldsymbol{A}_p \otimes \boldsymbol{A}_{p^{t-1}})\boldsymbol{w} \\ &= (\boldsymbol{I}_p\boldsymbol{A}_p) \otimes ((\boldsymbol{A}_{p^{t-1}})^{-1}\boldsymbol{A}_{p^{t-1}})\boldsymbol{w} \\ &= (\boldsymbol{A}_p \otimes \boldsymbol{I}_{p^{t-1}})\boldsymbol{w}. \end{aligned}$$

Therefore,

$$(\boldsymbol{A}_p \otimes \boldsymbol{I}_{p^{t-1}})\boldsymbol{w} = \left(\boldsymbol{\beta}^{\langle 0\rangle}, \cdots, \boldsymbol{\beta}^{\langle p-1\rangle}\right)^\top. \tag{9}$$

For $0 \leq j < p^{t-1}$ we define

$$\tilde{\boldsymbol{\beta}}^{\langle j\rangle} := \left(\boldsymbol{\beta}_j^{\langle 0\rangle}, \cdots, \boldsymbol{\beta}_j^{\langle p-1\rangle}\right)^\top \text{ and } \tilde{\boldsymbol{w}}^{\langle j\rangle} := \left(\boldsymbol{w}_j^{\langle 0\rangle}, \cdots, \boldsymbol{w}_j^{\langle p-1\rangle}\right)^\top,$$

Intuitively, vectors with tildes here contain entries taken from the original vector with stride p^{t-1}. Then Equation (9) implies

$$\boldsymbol{A}_p\tilde{\boldsymbol{w}}^{\langle j\rangle} = \tilde{\boldsymbol{\beta}}^{\langle j\rangle}.$$

Let $n' = \lfloor (n+1)/p^{t-1}\rfloor - 1$, $n'' = \lceil (n+1)/p^{t-1}\rceil - 1$, and $m' = n \bmod p^{t-1}$. Consider the following two subcases:

Subcase II-1. Suppose there exists $\ell \leq m'$ and $i, j \in [0, n'']$ such that $\tilde{\boldsymbol{w}}_i^{\langle \ell\rangle} \neq \tilde{\boldsymbol{w}}_j^{\langle \ell\rangle}$. According to Lemma 4.2, there exists $i' \in [\lfloor n''/2\rfloor + 1, n'']$ satisfying $0 \neq \left(\boldsymbol{A}_p\tilde{\boldsymbol{w}}^{\langle j\rangle}\right)_{i'} = \tilde{\boldsymbol{\beta}}_{i'}^{\langle \ell\rangle} = \boldsymbol{\beta}_\ell^{\langle i'\rangle}$. Because $\boldsymbol{A}_{p^{t-1}}$ is invertible, we have $\boldsymbol{\alpha}^{\langle i'\rangle} = \boldsymbol{A}_{p^{t-1}}\boldsymbol{\beta}^{\langle i'\rangle} \neq \boldsymbol{0}$. What's more,

- if $i' < n''$ and recall Fact 2.10, we have $\deg_p(f) \geq (\lfloor n''/2\rfloor + 1) \cdot p^{t-1} \geq n/2$;
- if $i' = n''$, we select the minimum ℓ such that $\boldsymbol{\beta}_\ell^{\langle i'\rangle} \neq 0$. Due to the fact $(\boldsymbol{A}_{p^{t-1}})_{\ell,j} = \binom{p^{t-1}-1-j}{p^{t-1}-1-\ell} = 0$ when $j > \ell$, it follows that

$$\boldsymbol{\alpha}_\ell^{\langle i'\rangle} = \sum_{j=0}^{\ell} (\boldsymbol{A}_{p^{t-1}})_{\ell,j}\boldsymbol{\beta}_j^{\langle i'\rangle} = (\boldsymbol{A}_{p^{t-1}})_{\ell,\ell}\boldsymbol{\beta}_\ell^{\langle i'\rangle} \neq 0. \tag{10}$$

Eventually $\deg_p(f) \geq n'' \cdot p^{t-1} \geq n/2$.

Subcase II-2. Otherwise, there exists a minimum $\ell \in [m'+1, p^{t-1}-1]$ and $i, j \in [0, n']$ such that $\tilde{\boldsymbol{w}}_i^{\langle \ell\rangle} \neq \tilde{\boldsymbol{w}}_j^{\langle \ell\rangle}$ as $f(x)$ is non-trivial. The same argument shows that there exists $i' \in [\lfloor n'/2\rfloor + 1, n']$ satisfying $\boldsymbol{\beta}_\ell^{\langle i'\rangle} \neq 0$. In addition, the condition $\tilde{\boldsymbol{w}}_0^{\langle \ell'\rangle} = \cdots = \tilde{\boldsymbol{w}}_{n'}^{\langle \ell'\rangle}$ for all $\ell' < \ell$ implies

$$\boldsymbol{\beta}_{\ell'}^{\langle i'\rangle} = \tilde{\boldsymbol{\beta}}_{i'}^{\langle \ell'\rangle} = \sum_{j=0}^{p-1} (\boldsymbol{A}_p)_{i',j}\tilde{\boldsymbol{w}}_j^{\langle \ell'\rangle} = \boldsymbol{w}_0^{\langle \ell'\rangle} \cdot \sum_{j=0}^{i'} \binom{p-1-j}{p-1-i'} = \boldsymbol{w}_0^{\langle \ell'\rangle} \cdot \binom{p}{p-i'} = 0.$$

Hence, by imitating (10) we can obtain $\boldsymbol{\alpha}_\ell^{\langle i'\rangle} \neq 0$, which leads to $\deg_p(f) \geq \lfloor n'/2\rfloor \cdot p^{t-1} + m' + 1 \geq n/2$. ◀

4.3 Proof of Theorem 1.6

We will later apply the following lemma about Diophantine approximation, which is an immediate corollary of Kronecker's Theorem.

▶ **Lemma 4.6.** *Suppose real numbers $a_1, \dots, a_k$ satisfy that $1, a_1, \dots, a_k$ are linearly independent over $\mathbb{Q}$. Then, for any $\varepsilon > 0$, there exist infinitely many positive integers ℓ such that $\ell a_i \bmod 1 \in (1-\varepsilon, 1)$ for each $i = 1, \dots, k$.*

Now we prove Theorem 1.6.

Proof of Theorem 1.6. Write $m = p_1 p_2 \cdots p_k$ for p_i being primes. Choose a prime q different from all p_i. Fix an arbitrary $\varepsilon > 0$. Let $a_i = \log q / \log p_i$ for $i = 1, \dots, k$. Then $1, a_1, \dots, a_k$ are linearly independent over $\mathbb{Q}$, otherwise a nontrivial linear relation can be exponentiated to contradict the unique factorization theorem over $\mathbb{Z}^+$. Applying Lemma 4.6 we get infinitely many ℓ that satisfy the condition $\ell \cdot \log q / \log p_i \bmod 1 \in (1-\varepsilon, 1)$, which implies $p_i^{r_i}/q^\ell \in (1, p_i^\varepsilon)$ where $r_i = \lceil \ell \log q / \log p_i \rceil$.

Now, choose a sufficiently large ℓ, let $n = 2q^\ell$ and define $f : \{0,1\}^n \to \{0,1\}$ by $f(x) = \mathbb{I}[|x| = q^\ell]$. Then f is $p_i^{r_i}$-periodic since $p_i^{r_i} > q^\ell$. Therefore $\deg_{p_i}(f) \leq p_i^{r_i} - 1$ by Theorem 2.12. Thus,

$$\deg_m(f) \leq \max_{1 \leq i \leq k} \{p_i^{r_i}\} \leq \frac{n}{2} \max_{1 \leq i \leq k} \{p_i^\varepsilon\}.$$

The theorem follows by letting $\varepsilon \to 0$. ◀

5 Conclusion

In a nutshell, we explore and exploit the matrices consisting of Mahler coefficients of the MOD function, serving as a significant extension of Wilson's arguments. This approach fully characterizes the modulo degree of Boolean functions when the base is prime or prime power, and provides good lower bounds for the composite case with the help of periodicity lemma. In addition, we also show a practical way to generalize properties of symmetric functions to non-degenerate ones by a Ramsey-type argument.

Nevertheless, there is still ample room for further discussion. First and foremost, we conjecture that the constant factor in Theorem 1.5 can be improved to 1/2 in correspondence with Theorem 1.6. Moreover, an anonymous reviewer also raises a good question with regard to Theorem 1.4: Could the extraordinary large prerequisite $n \geq \mathrm{tower}(\mathrm{poly}(p,k))$ (which is implicit in the proof) be improved to something like $n \geq \exp(\mathrm{poly}(p,k))$? We also wonder if it is possible to embed other kinds of functions to derive similar results. Above all, both Conjecture 1.1 and Conjecture 1.2 remain open.

References

1 Noga Alon and Richard Beigel. Lower bounds for approximations by low degree polynomials over $\mathbb{Z}_m$. In *Proceedings of the 16th Annual IEEE Conference on Computational Complexity, Chicago, Illinois, USA, June 18-21, 2001*, pages 184–187. IEEE Computer Society, 2001. doi:10.1109/CCC.2001.933885.

2 Roger C Baker, Glyn Harman, and János Pintz. The difference between consecutive primes, ii. *Proceedings of the London Mathematical Society*, 83(3):532–562, 2001.

3 David A. Mix Barrington, Richard Beigel, and Steven Rudich. Representing boolean functions as polynomials modulo composite numbers. *Computational Complexity*, 4:367–382, 1994. doi:10.1007/BF01263424.

4 Harry Buhrman and Ronald de Wolf. Communication complexity lower bounds by polynomials. In *Proceedings of the 16th Annual IEEE Conference on Computational Complexity, Chicago, Illinois, USA, June 18-21, 2001*, pages 120–130. IEEE Computer Society, 2001. doi:10.1109/CCC.2001.933879.

5 Harry Buhrman and Ronald de Wolf. Complexity measures and decision tree complexity: a survey. *Theor. Comput. Sci.*, 288(1):21–43, 2002. doi:10.1016/S0304-3975(01)00144-X.

6 John Chiarelli, Pooya Hatami, and Michael E. Saks. An asymptotically tight bound on the number of relevant variables in a bounded degree boolean function. *Combinatorica*, 40(1):237–244, 2020. doi:10.1007/s00493-019-4136-7.

7 Klim Efremenko. 3-query locally decodable codes of subexponential length. *SIAM J. Comput.*, 41(6):1694–1703, 2012. doi:10.1137/090772721.

8 P. Erdős and R. Rado. Combinatorial theorems on classifications of subsets of a given set. *Proceedings of the London Mathematical Society*, s3-2(1):417–439, 1952.

9 Nathan J Fine and Herbert S Wilf. Uniqueness theorems for periodic functions. *Proceedings of the American Mathematical Society*, 16(1):109–114, 1965.

10 Parikshit Gopalan. *Computing with polynomials over composites.* PhD thesis, Georgia Institute of Technology, 2006.

11 Parikshit Gopalan. Constructing ramsey graphs from boolean function representations. *Combinatorica*, 34(2):173–206, 2014. doi:10.1007/s00493-014-2367-1.

12 Parikshit Gopalan, Shachar Lovett, and Amir Shpilka. On the complexity of boolean functions in different characteristics. In *Proceedings of the 24th Annual IEEE Conference on Computational Complexity, CCC 2009, Paris, France, 15-18 July 2009*, pages 173–183. IEEE Computer Society, 2009. doi:10.1109/CCC.2009.14.

13 Vince Grolmusz. Superpolynomial size set-systems with restricted intersections mod 6 and explicit ramsey graphs. *Combinatorica*, 20(1):71–86, 2000. doi:10.1007/s004930070032.

14 Vince Grolmusz. Constructing set systems with prescribed intersection sizes. *J. Algorithms*, 44(2):321–337, 2002. doi:10.1016/S0196-6774(02)00204-3.

15 D. J. Katz. p-adic valuation of weights in abelian codes over $\mathbb{Z}_{p^d}$. *IEEE Trans. Inf. Theory*, 51(1):281–305, 2005. doi:10.1109/TIT.2004.839495.

16 Adam R. Klivans and Rocco A. Servedio. Learning DNF in time $2^{\tilde{\mathrm{o}}(\mathrm{n}^{1/3})}$. *J. Comput. Syst. Sci.*, 68(2):303–318, 2004. doi:10.1016/j.jcss.2003.07.007.

17 Eyal Kushilevitz and Yishay Mansour. Learning decision trees using the fourier spectrum. *SIAM J. Comput.*, 22(6):1331–1348, 1993. doi:10.1137/0222080.

18 Chia-Jung Lee, Satyanarayana V. Lokam, Shi-Chun Tsai, and Ming-Chuan Yang. Restrictions of nondegenerate boolean functions and degree lower bounds over different rings. In *IEEE International Symposium on Information Theory, ISIT 2015, Hong Kong, China, June 14-19, 2015*, pages 501–505. IEEE, 2015. doi:10.1109/ISIT.2015.7282505.

19 Qian Li and Xiaoming Sun. On the modulo degree complexity of boolean functions. *Theor. Comput. Sci.*, 818:32–40, 2020. doi:10.1016/j.tcs.2018.04.049.

20 Nathan Linial, Yishay Mansour, and Noam Nisan. Constant depth circuits, fourier transform, and learnability. *J. ACM*, 40(3):607–620, 1993. doi:10.1145/174130.174138.

21 Xiaoyu Liu. An equivalence of ward's bound and its application. *Des. Codes Cryptogr.*, 58(1):1–9, 2011. doi:10.1007/s10623-010-9380-1.

22 Kurt Mahler. An interpolation series for continuous functions of a p-adic variable. *J. reine angew. Math*, 199:23–34, 1958.

23 Marvin Minsky and Seymour Papert. An introduction to computational geometry. *Cambridge tiass., HIT*, 1969.

24 Ashley Montanaro and Tobias Osborne. On the communication complexity of XOR functions. *CoRR*, abs/0909.3392, 2009. arXiv:0909.3392.

25 Elchanan Mossel, Ryan O'Donnell, and Rocco A. Servedio. Learning juntas. In Lawrence L. Larmore and Michel X. Goemans, editors, *Proceedings of the 35th Annual ACM Symposium on Theory of Computing, June 9-11, 2003, San Diego, CA, USA*, pages 206–212. ACM, 2003. doi:10.1145/780542.780574.

26 Noam Nisan and Mario Szegedy. On the degree of boolean functions as real polynomials. *Computational Complexity*, 4:301–313, 1994. doi:10.1007/BF01263419.

27 Alexander A Razborov. Lower bounds for the size of circuits of bounded depth with basis $\{\wedge, \oplus\}$. *Math. notes of the Academy of Sciences of the USSR*, 41(4):333–338, 1987.

28 Alexander A Razborov. Quantum communication complexity of symmetric predicates. *Izvestiya: Mathematics*, 67(1):145, 2003.

29 Alexander A. Sherstov. Communication lower bounds using dual polynomials. *Bulletin of the EATCS*, 95:59–93, 2008.

30 Alexander A. Sherstov. Strong direct product theorems for quantum communication and query complexity. *SIAM J. Comput.*, 41(5):1122–1165, 2012. doi:10.1137/110842661.

31 Yaoyun Shi and Yufan Zhu. Quantum communication complexity of block-composed functions. *Quantum Information & Computation*, 9(5):444–460, 2009. URL: http://www.rintonpress.com/xxqic9/qic-9-56/0444-0460.pdf.

32 Hans Ulrich Simon. A tight $\Omega(\log\log n)$-bound on the time for parallel RAM's to compute nondegenerated boolean functions. *Information and Control*, 55(1-3):102–106, 1982. doi:10.1016/S0019-9958(82)90477-6.

33 Roman Smolensky. Algebraic methods in the theory of lower bounds for boolean circuit complexity. In Alfred V. Aho, editor, *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 77–82. ACM, 1987. doi:10.1145/28395.28404.

34 Joachim von zur Gathen and James R. Roche. Polynomials with two values. *Combinatorica*, 17(3):345–362, 1997. doi:10.1007/BF01215917.

35 Jake Wellens. A tighter bound on the number of relevant variables in a bounded degree boolean function. *CoRR*, abs/1903.08214, 2019. arXiv:1903.08214.

36 Richard M. Wilson. A lemma on polynomials modulo p^m and applications to coding theory. *Discrete Mathematics*, 306(23):3154–3165, 2006. doi:10.1016/j.disc.2004.10.030.

37 Bahattin Yildiz. Weights modulo p^e of linear codes over rings. *Des. Codes Cryptogr.*, 43(2-3):147–165, 2007. doi:10.1007/s10623-007-9076-3.

On Quasipolynomial Multicut-Mimicking Networks and Kernelization of Multiway Cut Problems

Magnus Wahlström
Department of Computer Science, Royal Holloway, University of London, UK
Magnus.Wahlstrom@rhul.ac.uk

Abstract

We show the existence of an exact mimicking network of $k^{O(\log k)}$ edges for minimum multicuts over a set of terminals in an undirected graph, where k is the total capacity of the terminals. Furthermore, if Small Set Expansion has an approximation algorithm with a ratio slightly better than $\Theta(\log n)$, then a mimicking network of quasipolynomial size can be computed in polynomial time. As a consequence of the latter, several problems would have quasipolynomial kernels, including Edge Multiway Cut, Group Feedback Edge Set for an arbitrary group, 0-Extension for integer-weighted metrics, and Edge Multicut parameterized by the solution and the number of cut requests. The result works via a combination of the matroid-based irrelevant edge approach used in the kernel for s-Multiway Cut with a recursive decomposition and sparsification of the graph along sparse cuts. The main technical contribution is a matroid-based marking procedure that we can show will mark all non-irrelevant edges, assuming that the graph is sufficiently densely connected. The only part of the result that is not currently constructive and polynomial-time computable is the detection of such sparse cuts.

This is the first progress on the kernelization of Multiway Cut problems since the kernel for s-Multiway Cut for constant value of s (Kratsch and Wahlström, FOCS 2012).

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Multiway Cut, Kernelization, Small Set Expansion, Mimicking Networks

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.101

Category Track A: Algorithms, Complexity and Games

Related Version A full version of the paper is available at https://arxiv.org/abs/2002.08825.

1 Introduction

Graph separation questions are home to some of the most intriguing open questions in theoretical computer science. In approximation algorithms, the well-known *unique games conjecture* (UGC) has been central to the area for close to two decades, and is closely related to graph separation problems. Even more directly, the *small set expansion hypothesis*, proposed by Raghavendra and Steurer [31], roughly states that it is NP-hard to approximate the Small Set Expansion problem (SSE) up to a constant factor, where SSE is the problem of finding a small-sized set in a graph with minimum expansion. (More precise statements are given in Section 2.2.) Despite significant research, the best result available in polynomial time is an $O(\log n)$-approximation due to Räcke [30].

Another interesting notion from parameterized complexity is *kernelization*. Informally, a kernelization algorithm is a procedure that takes an input of a parameterized, usually NP-hard problem and *reduces* it to an equivalent instance of size bounded in the parameter, e.g., by discarding irrelevant parts of the input or transforming some part of the input into a smaller object with equivalent behaviour. For example, the seminal Nemhauser-Trotter theorem on the half-integrality of Vertex Cover [27] implies that an instance of Vertex Cover can be reduced to have at most $2k$ vertices, where k is the bound given on the

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 101; pp. 101:1–101:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

solution size. On the flip side, Fortnow and Santhanam [10] and Bodlaender et al. [3] gave a framework to exclude the existence of a kernel of any polynomial size, under a standard complexity-theoretic conjecture. An extensive collection of upper and lower bounds for kernelization exists (see, e.g., the recent book of Fomin et al. [9]), but a handful of central "hard questions" remain unanswered. One of the most notorious is Multiway Cut.

Let $G = (V, E)$ be a graph and $T \subseteq V$ a set of terminals in G. An *(edge) multiway cut* for T in G is a set of edges $X \subseteq E$ such that no two terminals are connected in $G - X$, and Multiway Cut is the problem of finding a multiway cut of at most k edges, given a parameter k. The problem is FPT [24] and NP-hard for $|T| \geq 3$ [6]. Using methods from matroid theory, Kratsch and Wahlström [17] were able to show that if $|T| \leq s$, then Multiway Cut has a kernel with $O(k^{s+1})$ vertices, hence the problem has a polynomial kernel for every constant s. However, if $|T|$ is unbounded, the only known size bound for a kernel is $2^{O(k)}$, following from the FPT algorithm [24], and the question of whether Multiway Cut has a polynomial kernel in the general case is completely open.

We show a connection between kernelization of Multiway Cut-type problems and approximation algorithms for Small Set Expansion. Specifically, we show the existence of a kind of *mimicking network* for the problem, of size quasipolynomial in k; and if SSE has approximation algorithms slightly better than current state of the art, then it can be computed in polynomial time and Multiway Cut has a quasipolynomial kernel.

1.1 Mimicking networks and multiway cut sparsifiers

Although kernelization is most commonly described in terms of polynomial-time preprocessing as above, there is also a clear connection with *succinct information representation*. For example, consider a graph $G = (V, E)$ with a set of k terminals $T \subseteq V$. The pair (G, T) is referred to as a *terminal network*. A *mimicking network* for (G, T) is a graph $G' = (V', E')$ with $T \subseteq V'$ such that for any sets $A, B \subseteq T$, the min-cut between A and B in G and G' have the same value. A mimicking network of size bounded in k always exists, but the size of G' can be significant. The best known general upper bound is double-exponential in k [12, 15], and there is an exponential lower bound [20]. Better bounds are known for special graph classes, but even for planar graphs the best possible general bound has $2^{\Theta(k)}$ vertices [20, 14] (see also recent improvements by Krauthgamer and Rika [19]).

A related notion is *cut sparsifiers*, which solve the same task up to some approximation factor $q \geq 1$ [26, 21], typically $q = \omega(1)$ in the general case. We focus on mimicking networks; see Krauthgamer and Rika [19] for an overview of cut sparsifiers.

However, if we include the capacity of the set of terminals in the bound (and if edges have integer capacity), then significantly stronger results are possible. Chuzhoy [4] showed that if the total capacity of T is $\text{cap}_G(T) = \sum_{t \in T} d(t) = k$, then there exists an $O(1)$-approximate cut sparsifier of size $O(k^3)$. Kratsch and Wahlström [17] sharpened this to an exact mimicking network with $O(k^3)$ edges, which furthermore can be computed in randomized polynomial time. This is particularly remarkable given that the network has to replicate the exact cut-value for exponentially many pairs (A, B). The network can be constructed via contractions on G.[1] This built on an earlier result that used linear representations of matroids to encode the sizes of all (A, B)-min cuts into an object using $\tilde{O}(k^3)$ bits of space [18], although this earlier version did not produce an explicit graph, i.e., not a mimicking network.

[1] The results of [17] are phrased in terms of vertex cuts, but the above follows easily from [17].

These results had significant consequences for kernelization. The succinct representation in [18] was used to produce a (randomized) polynomial kernel for the Odd Cycle Transversal problem, thereby solving a notorious open problem in parameterized complexity [18]; and the mimicking network of [17] brought further (randomized) polynomial kernels for a range of problems, in particular including Almost 2-SAT, i.e., the problem of satisfying all but at most k clauses of a given 2-CNF formula.

Similar methods are relevant for the question of separating a set of terminals into more than two parts. Let (G, T) be a terminal network, and let $\mathcal{T} = T_1 \cup \ldots \cup T_s$ be a partition of T. A *multiway cut for* $\mathcal{T}$ is a set of edges $X \subseteq E(G)$ such that $G - X$ contains no path between any pair of terminals $t \in T_i$ and $t' \in T_j$ for $t, t' \notin X$ and $i \neq j$. Let us define a *multicut-mimicking network* for (G, T) as a terminal network (G', T) where $T \subseteq V(G')$ and for every partition $\mathcal{T} = T_1 \cup \ldots \cup T_s$ of T, the size of a minimum multiway cut for $\mathcal{T}$ is identical in G and G'. (The term *multicut-mimicking*, as opposed to *multiway cut-mimicking*, is justified; see Section 2.1.) The minimum size of a multicut-mimicking network, in terms of $k = \mathrm{cap}_G(T)$, appears to lie at the core of the difficulty of the question of a polynomial kernelization of Multiway Cut. The kernel for s-Multiway Cut mentioned above builds on the computation of a mimicking network of size $O(k^{s+1})$ for partitions of T into at most s parts [17]. The kernel for s-Multiway Cut then essentially follows from considering the partition $\mathcal{T} = \{t_1\} \cup \ldots \cup \{t_s\}$ of a set T of $|T| = s$ terminals (along with known reduction rules bounding $\mathrm{cap}_G(T)$). We are not aware of any non-trivial lower bounds on the size of a multicut-mimicking network in terms of k; it seems completely consistent with known bounds that every terminal network (G, T) would have a multicut-mimicking network of size $\mathrm{poly}(k)$, even for partitions into an unbounded number of sets.

In this paper, we show that any terminal network (G, T) with $\mathrm{cap}_G(T) = k$ admits a multicut-mimicking network (G', T) where $|V(G')| = k^{O(\log k)}$; and furthermore, such a network could be computed in randomized polynomial time, given a polynomial number of queries to a sufficiently good approximation algorithm for a graph separation problem similar to Small Set Expansion. We also see a tradeoff between the quality of the approximation algorithm and the size of (G', T). In particular, if Small Set Expansion has an approximation algorithm with a ratio of $\alpha(n, k) = \log^{1-\varepsilon} n \cdot \log^{O(1)} k$ for some $\varepsilon > 0$, where k is the number of edges cut in the optimal solution, then (G', T) can be computed efficiently, with $|V(G')|$ being quasipolynomial in k. Such an algorithm goes beyond the bounds of what is currently known – namely, a ratio of $O(\log n)$ due to Räcke [30], improved for certain regimes by Bansal et al. [2] – but does not appear to be excluded by any established hardness conjecture. We also consider the existence result very interesting in its own right, and invite further study of capacity-based bounds for multicut-mimicking networks; in particular, whether a $\mathrm{poly}(k)$-sized multicut-mimicking network always exists. The results strongly suggest the existence of a quasipolynomial kernel for Edge Multiway Cut.

Flow sparsifiers. Finally, similarly to cut sparsifiers, there is a notion of a *flow sparsifier* of a terminal network (G, T). Here the goal is to approximately preserve the minimum congestion for any multicommodity flow on (G, T). Chuzhoy [4] showed flow sparsifiers with quality $O(1)$ and with $k^{O(\log \log k)}$ vertices, where k is the total terminal capacity; for further results on achievable bounds for flow sparsifiers, see [1, 7]. However, the notion is incomparable to multicut-mimicking networks, because even an exact flow sparsifier would be subject to the corresponding multicommodity flow-multicut approximation gap, which is $\Theta(\log k)$ in the worst case [11].

Further related work. The general approach of decomposing a graph along sparse cuts is well established; cf. Räcke [29] and follow-up work. For further applications of matroid tools to kernelization, see Hols and Kratsch [13], Kratsch [16], and Reidl and Wahlström [32].

1.2 Our results

More formally, we have the following.

▶ **Theorem 1.** *Let A be an approximation algorithm for* Small Set Expansion *with an approximation ratio of $\alpha(n,k) = O(\log^{1-\varepsilon} n \log^d k)$, where $\varepsilon > 0$, $d = O(1)$, and k is the number of edges cut in the optimal solution. Let (G,T) be a terminal network with $\mathrm{cap}_G(T) = k$. Then there is a set $Z \subseteq E(G)$ with $|Z| = k^{O(\alpha(n,k)\log k)}$ such that for every partition $\mathcal{T} = T_1 \cup \ldots \cup T_s$ of T, there is a minimum multiway cut X for $\mathcal{T}$ such that $X \subseteq Z$. Furthermore, Z can be computed in randomized polynomial time using calls to A.*

The precise requirement for the approximation algorithm is slightly relaxed from the above. We refer to the precise algorithm we need as a *sublogarithmic terminal expansion tester*; see Definition 5. Simplifying the statement a bit gives us the following.

▶ **Corollary 2.** *Let (G,T) be a terminal network with $\mathrm{cap}_G(T) = k$. The following holds.*

1. *There is a multicut-mimicking network for (G,T) with $k^{O(\log k)}$ edges.*
2. *If there is a sublogarithmic terminal expansion tester – in particular, if* Small Set Expansion *has an approximation ratio as in Theorem 1 – then a multicut-mimicking network of size quasipolynomial in k can be computed in randomized polynomial time.*

This would give us the following selection of conditional breakthrough results in kernelization. We refer to previous kernelization work [17, 32] for the necessary definitions.

▶ **Corollary 3.** *If there is a sublogarithmic terminal expansion tester, then the following problems have randomized quasipolynomial kernels.*

1. Edge Multiway Cut *parameterized by solution size.*
2. Edge Multicut *parameterized by the solution size and the number of cut requests.*
3. Group Feedback Edge Set *parameterized by solution size, for any group.*
4. Subset Feedback Edge Set *with undeletable edges, parameterized by solution size.*
5. *0-*Extension *for integer-weighted graphs, parameterized by solution cost.*

Preliminaries. A *parameterized problem* is a decision problem where inputs are given as pairs $I = (X,k)$, where k is the *parameter*. A *polynomial kernelization* is a polynomial-time procedure that maps an instance (X,k) to an instance (X',k') where (X,k) is positive if and only if (X',k') is positive, and $|X'|, k' \leq g(k)$ for some function $g(k)$ referred to as the *size* of the kernel. A problem has a *polynomial kernel* if it has a kernel where $g(k) = k^{O(1)}$. We extend this to discuss *quasipolynomial kernels*, which is the case that $g(k) = k^{\log^{O(1)} k}$.

We use standard terminology from graph theory and parameterized complexity; see, e.g., [5, 9] for references.

2 Terminal separation notions

For a graph $G = (V,E)$ and sets $A, B \subseteq V$, we let $E_G(A,B) = \{uv \in E \mid u \in A, v \in B\}$. As shorthand for $S \subseteq V$ we also write $E(S) = E(S,S)$, $\partial_G(S) = E_G(S, V \setminus S)$, and $\delta_G(S) = |\partial_G(S)|$. The *total capacity* of a set of vertices S in a graph G is $\mathrm{cap}_G(S) := \sum_{v \in S} d(v)$. In all cases, we may omit the index G if understood from context.

2.1 Multicut-mimicking networks

Let $G = (V, E)$ be a graph and $T \subseteq V$ a set of terminals with $\text{cap}_G(T) = k$. An *(edge) multiway cut* for T in G is a set of edges $X \subseteq E$ such that no two vertices in T are connected in $G - X$. More generally, let $\mathcal{T} = \{T_1, \ldots, T_r\}$ be a partition of T. Then an *(edge) multiway cut for $\mathcal{T}$ in G* is a set of edges $X \subseteq E$ such that in $G - X$ every connected component contains terminals from at most one part of $\mathcal{T}$. Hence a multiway cut for (G, T) is equivalent to a multiway cut for $(G, \{\{t\} \mid t \in T\})$. Furthermore, let $R \subseteq \binom{T}{2}$ be a set of pairs over T, referred to as *cut requests*. A *multicut for R in G* is a set of edges $X \subseteq E$ such that every connected component in $G - X$ contains at most one member of every pair $\{u, v\} \in R$. A *minimum multicut for R in G* is a multicut for R in G of minimum cardinality. Similarly, a *minimum multiway cut for $\mathcal{T}$ in G* is a multiway cut for $\mathcal{T}$ in G of minimum cardinality.

We define a *multicut-mimicking network* for T in G as a graph $G' = (V', E')$ such that $T \subseteq V'$ and such that for every set of cut requests $R \subseteq \binom{T}{2}$, the size of a minimum multicut for R is equal in G and in G'. We observe that this is equivalent to preserving the sizes of minimum multiway cuts over all partitions of T.

▶ **Proposition 4** (★[2]). *A graph G' with $T \subseteq V(G')$ is a multicut-mimicking network for T in G if and only if, for every partition $\mathcal{T}$ of T, the size of a minimum multiway cut for $\mathcal{T}$ is equal in G and in G'.*

As a slightly sharper notion, a *multicut-covering set* for (G, T) is a set $Z \subseteq E(G)$ such that for every set of cut requests $R \subseteq \binom{T}{2}$, there is a minimum multicut X for R in G such that $X \subseteq Z$. Note that a multicut-covering set Z is essentially equivalent to a multicut-mimicking network formed by contraction (contracting all edges of $E(G) \setminus Z$). Our main result in this paper is the existence of a multicut-covering set of size quasipolynomial in $k = \text{cap}(T)$ in any undirected graph G. Furthermore, such a set can be computed in polynomial time, subject to the existence of certain approximation algorithms that we will make precise later in this section. The term is a generalization of a *cut-covering set*, used in previous work [17].

2.2 Graph separation algorithms

The central technical approximation assumption needed in this paper is the following. For a graph G with a set of terminals T, define the *T-capacity of S in G* as $\text{cap}_T(S) = \text{cap}_G(T \cap S) + \delta_G(S)$. Then we define the following notion.

▶ **Definition 5** (Sublogarithmic terminal expansion tester). *Let (G, T) be a terminal network with* $\text{cap}_G(T) = k$. *A* terminal polynomial expansion tester *(with approximation ratio α) is a (possibly randomized) algorithm that, given as input (G, T) and an integer $c \in \mathbb{N}$, with $c = \Omega(\log k)$, does one of the following.*

1. *Either returns a set $S \subset V$ such that $N_G[S] \neq V(G)$ and $\text{cap}_T(S) < |S|^{1/c}$,*
2. *or guarantees that for every set S with $\emptyset \subset (S \cap T) \subset T$ and $|S| \leq |V(G)|/2$ we have $\text{cap}_T(S) \geq |S|^{1/c}/\alpha$.*

A sublogarithmic terminal expansion tester *is a terminal polynomial expansion tester with an approximation ratio $\alpha = O(\log^{1-\varepsilon} n \log^{O(1)} k)$ for some $\varepsilon > 0$. We say that (G, T) is (α, c)-*dense *if case 2 above applies, i.e., for every set S with $S \cap T \notin \{\emptyset, T\}$ and $|S| \leq |V(G)|/2$ we have $\text{cap}_T(S) \geq |S|^{1/c}/\alpha$.*

[2] Proofs marked with ★ are found in the full version of the paper

The conditions can be relaxed somewhat. It is sufficient if the algorithm works with parameters $c = \Omega(\alpha \log k)$. It is also possible to put a lower bound on the size of sets S for which the guarantee needs to apply. However, these relaxed assumptions do not seem to make a difference for any algorithms we are aware of for the problem.

We note that such an algorithm would follow from a slightly improved approximation algorithm for Small Set Expansion. Let $G = (V, E)$ be a graph and $S \subseteq V$ a set of vertices. The *edge expansion* of S is $\Phi(S) := \frac{\delta(S)}{|S|}$. For a real number $\rho \in (0, 1/2]$, one also defines the *small set expansion* $\Phi_\rho(G) := \min_{S \subseteq V, |S| \leq \rho n} \Phi(S)$. In particular, for a value $s \in [n/2]$, $\Phi_{s/n}(G)$ denotes the worst (i.e., minimum) expansion among subsets of G of size at most s. A sufficiently good approximation algorithm for Small Set Expansion implies a sublogarithmic terminal expansion tester, as follows.

▶ **Lemma 6 (★).** *Assume that* Small Set Expansion *has a bicriteria approximation algorithm that on input (G, ρ) returns a set S with $|S| \leq \beta \rho n$ and $\Phi(S) \leq \alpha \cdot \Phi_\rho$, for some $\alpha, \beta \geq 1$. If $\alpha\beta = O(\log^{1-\varepsilon} n \log^{O(1)}(n \cdot \Phi_\rho))$, for some $\varepsilon > 0$, then there is a sublogarithmic terminal expansion tester with ratio $\Theta(\alpha\beta)$ (with $n \cdot \Phi_\rho$ replaced by k).*

Existing approximation algorithms do not meet this threshold; the best known results are an $O(\log n)$-approximation due to Räcke [30] and a bicriteria algorithm of Bansal et al. [2] which achieves a ratio of $O(\sqrt{\log n \log(1/\rho)})$. Unfortunately, the latter improvement is insufficient to make the analysis in the next section work. However, it seems clear that no existing hardness conjecture could possibly rule out the existence of such an algorithm. Furthermore, testing for (α, c)-denseness when $c = \Omega(\alpha \log k)$ corresponds to looking for significantly worse expanding sets than the regime usually focused on in the approximation literature. Hence we proceed with conditional results in the rest of the paper.

3 Multicut-covering sets

We now present the main result of the paper, namely the existence of quasipolynomial multicut-mimicking networks for terminal networks (G, T), and the conditional efficient computability of such objects given a sublogarithmic terminal expansion tester.

At a high level, the process works through recursive decomposition of the graph G across very sparse cuts, treating each piece $G[S]$ of the recursion as a new instance of multicut-covering set computation, where the edges of $\partial(S)$ are considered as additional terminals. The process repeatedly finds a single edge $e \in E(G)$ with a guarantee that for every set of cut requests $R \subseteq \binom{T}{2}$ there is a minimum multicut X for R in G such that $e \notin X$. We may then contract the edge e and repeat the process. Thus the end product is a multicut-mimicking network, and the edges that survive until the end of the process form a multicut-covering set.

In somewhat more detail, the process uses a novel variant of the representative sets approach, which was previously used in the kernel for s-Multiway Cut [17]. Refer to an edge e as *essential for* R, for some $R \subseteq \binom{T}{2}$, if every minimum multicut for R in G contains e, and *essential for* (G, T) if it is essential for R for some $R \subseteq \binom{T}{2}$. We use a representative sets approach to return a set of at most k^c edges which is guaranteed to contain every essential edge, *if* (G, T) is already (α, c)-dense, for an appropriate value $c = \Theta(\alpha \log k)$. On the other hand, if (G, T) is not (α, c)-dense, then (by careful choice of parameters) we can identify a cut through G which is sufficiently sparse that we can reduce the size of one side of this cut via a recursive call. This gives a tradeoff between the size of the resulting multicut-covering set and the denseness-guarantee we may assume through the approximation algorithm. The threshold for feasibility for this tradeoff is precisely the existence of a sublogarithmic terminal expansion tester.

3.1 Recursive replacement

We now present the recursive decomposition step in detail. Let (G,T) be a terminal network with $\mathrm{cap}_G(T)=k$. For a set $S \subseteq V$, we define the graph $G_S = G[N_G[S]] - E(S)$, i.e., G_S equals the graph $G[S]$ with the edges of $\partial(S)$ added back in. We also denote $T(S) = (T \cap S) \cup N_G(S)$ as the *terminals of* S. Under these definitions, the T*-capacity of* S *in* G has two equivalent definitions as $\mathrm{cap}_T(S) = \mathrm{cap}_{G_S}(T(S)) = \mathrm{cap}_G(T \cap S) + \delta_G(S)$. The *recursive instance at* S consists of the terminal network $(G_S, T(S))$. This is the basis of our recursive replacement procedure. Indeed, we show the following. Note that we consider $E(G_S) \subseteq E(G)$ in the following.

▶ **Lemma 7** (★). *Let $(G_S, T(S))$ be the recursive instance at S for some $S \subseteq V(G)$. Let Z_S be a multicut-covering set for $(G_S, T(S))$ and let $e \in E(G_S) \setminus Z_S$. Then e is not essential for (G,T).*

Let us also briefly note the formal correctness of contracting a non-essential edge.

▶ **Proposition 8** (★). *Let $e \in E(G)$ be a non-essential edge. Then for every $X \subseteq E(G)$ with $e \notin X$, and every partition $\mathcal{T}$ of T, X is a multiway cut for $\mathcal{T}$ in G if and only if it is a multiway cut for $\mathcal{T}$ in G/e. Furthermore, G/e is a multicut-mimicking network for (G,T), and any multicut-covering set $Z \subseteq E(G/e)$ for $(G/e,T)$ is also multicut-covering for (G,T).*

The process now works as follows. Recall that (G,T) is (α,c)-dense if $\mathrm{cap}_T(S) \geq |S|^{1/c}/\alpha$ for every set S with $S \cap T \neq \emptyset$ and $|S| \leq |V|/2$. The main technical result is a marking process that marks all essential edges for (G,T) on the condition that (G,T) is (α,c)-dense, and which marks at most k^c edges in total. In such a case, we are clearly allowed to select and contract any unmarked edge of G. Now, assume that (G,T) is not (α,c)-dense. Then by definition there exists a set $S \subset V$ such that $\mathrm{cap}_T(S) < |S|^{1/c}/\alpha$. If we can detect a set S such that $\mathrm{cap}_T(S) < |S|^{1/c}$, then we can recursively compute a multicut-covering set Z_S for $(G_S, T(S))$, consisting of at most $\mathrm{cap}_T(S)^c < |S|$ edges. By the above, we may again select any single edge $e \in E(G_S) \setminus Z_S$ and contract e in G. In either case, we replace G by a strictly smaller graph until $|E(G)| \leq k^c$, at which point we are done.

The two ingredients in the above are thus the marking process for (α,c)-dense graphs, which we present next, and the ability to distinguish the two cases, which is precisely the assumption of the existence of a sublogarithmic terminal expansion tester.

3.2 The (α,c)-dense case

Let us now focus on the marking procedure. Let a terminal network (G,T) with $\mathrm{cap}_G(T)=k$ and an integer c be given, and assume that $c = \Omega(\alpha \log k)$ for some α. We show a process that marks a set of at most k^c edges that contains every essential edge, assuming that (G,T) is (α,c)-dense. (A more precise bound on the relationship between c and α is given later, but the constant factors involved are not important to our main result.)

We will prove the following result. The proof takes up the rest of the subsection.

▶ **Lemma 9.** *Assume that (G,T) is (α,c)-dense where $c = \Omega(\alpha \log k)$. A multicut-covering set $Z \subseteq E(G)$ of size less than k^c can be computed in randomized polynomial time.*

The basis is the following. If (G,T) is (α,c)-dense then for every partition $\mathcal{T}$ of T, every minimum multiway cut X for $\mathcal{T}$, and every connected component H of $G-X$ except possibly the largest one, it holds that $\mathrm{cap}_T(V(H)) \geq |V(H)|^{1/c}/\alpha$. We also have

$$\sum_{H \in G-X} \mathrm{cap}_T(V(H)) = \mathrm{cap}_G(T) + 2|X| < 3k,$$

where the sum ranges over connected components H. This implies restrictions on the possible sizes of components of $G - X$, which will help in the marking process (as we shall see). Essentially, if too many components are too large, then the above sum will exceed $3k$ and we can conclude non-optimality of the corresponding multiway cut.

Finally, let us eliminate a silly edge case to assume $c \leq k$.

▶ **Lemma 10** (★). *If $c > k$ then a multicut-covering set of at most k^c edges can be marked deterministically.*

3.2.1 Matroid constructions

Before we show the marking procedure, we need some additional preliminaries. We refer to Oxley [28] and Marx [25] for further relevant background on matroids.

A *matroid* is a pair $M = (E, \mathcal{I})$ where $\mathcal{I} \subseteq 2^E$ is the *independent sets* of M, subject to the following axioms.

1. $\emptyset \in \mathcal{I}$;
2. if $B \in \mathcal{I}$ and $A \subseteq B$ then $A \in \mathcal{I}$; and
3. if $A, B \in \mathcal{I}$ with $|B| > |A|$ then there exists an element $x \in B \setminus A$ such that $A + x \in \mathcal{I}$.

A *basis* of M is a maximum independent set of M; the *rank* of M is the size of a basis.

Let A be a matrix, and let E label the columns of A. The *column matroid* of A is the matroid $M = (E, \mathcal{I})$ where $S \in \mathcal{I}$ for $S \subseteq E$ if and only if the columns indexed by S are linearly independent. A matrix A *represents* a matroid M if M is isomorphic to the column matroid of A. We refer to A as a *linear representation* of M.

We need three classes of matroids to build from. First, for a set E, the *uniform matroid* over E of rank r is the matroid $U(E, r) := (E, \{S \subseteq E \mid |S| \leq r\})$. Uniform matroids are representable over any sufficiently large field.

The second class is a *truncated graphic matroid.* Given a graph $G = (E, V)$, the *graphic matroid* of G is the matroid $M(G) = (E, \mathcal{I})$ where a set $F \subseteq E$ is independent if and only if F is the edge set of a forest in G. Graphic matroids can be deterministically represented over all fields. The *r-truncation* of a matroid $M = (E, \mathcal{I})$ for some $r \in \mathbb{N}$ is the matroid $M' = (E, \mathcal{I}')$ where $S \in \mathcal{I}'$ if and only if $S \in \mathcal{I}$ and $|S| \leq r$. Given a linear representation of M, over some field $\mathbb{F}$, a truncation of M can be computed in randomized polynomial time, possibly by moving to an extension field of $\mathbb{F}$ [25]. There are also methods for doing this deterministically [22], but the basic randomized form will suffice for us.

The final class is more involved. Let $D = (V, A)$ be a directed graph and $S \subseteq V$ a set of source vertices. A set $T \subseteq V$ is *linked to S in D* if there are $|T|$ pairwise vertex-disjoint paths starting in S and ending in T. Let $U \subseteq V$. Then $M(D, S, U) = (U, \{T \subseteq U \mid T \text{ is linked to } S \text{ in } D\})$ defines a matroid over U, referred to as a *gammoid.* Note that by Menger's theorem, a set T is dependent in M if and only if there is an (S, T)-vertex cut in D of cardinality less than $|T|$ (where the cut is allowed to overlap S and T). Like uniform matroids, gammoids are representable over any sufficiently large field, and a representation can be computed in randomized polynomial time [28, 25]. We will work over a variant of gammoids we refer to as *edge-cut gammoids*, which are defined as gammoids, except in terms of edge cuts instead of vertex cuts. Informally, for a graph $G = (V, E)$ and a set of source vertices $S \subseteq V$, the edge-cut gammoid of (G, S) is a matroid on a ground set of edges, where a set F of edges is independent if and only if it can be linked to S via pairwise edge-disjoint paths. However, we also need to introduce the "edge version" of *sink-only copies* of vertices, as used in previous work [17]. That is, we introduce a second set $E' = \{e' \mid e \in E\}$ containing copies of edges $e \in E$ which can only be used as the endpoints of linkages, not as initial or intermediate edges.

More formally, for a graph $G = (V, E)$ and a set of source vertices $S \subseteq E$ we perform the following transformation.

1. Subdivide every edge $e \in E$ by a new vertex z_e.
2. Let $p = \text{cap}_G(S)$. Inflate every vertex $v \in V$ into a twin class of $p + 1$ vertices (but do not inflate vertices z_e introduced in the previous step).
3. Replace every edge uv in the resulting graph by the two directed edges (u, v), (v, u), creating a directed graph D_G.
4. For every edge $e = uv \in E$, introduce a further new vertex z'_e, and create directed edges (u_i, z'_e) and (v_j, z'_e) for every copy u_i, v_j in D_G of the vertices u, v of G.

Slightly abusing notation, we let E refer to the vertices z_e in D_G, and we let E' refer to the vertices z'_e in D_G. The *edge-cut gammoid of* (G, S) is the gammoid $(D_G, \partial(S), E \cup E')$. Let us observe the resulting notion of independence.

▶ **Proposition 11.** *Let $G = (V, E)$ and $S \subseteq V$ be given. Let $M = (E \cup E', \mathcal{I})$ be the edge-cut gammoid of (G, S). Let $X \subseteq E \cup E'$ be given, and let $F = (X \cap E) \cup \{e \mid e' \in F \cap E'\}$. Then X is independent in M if and only if there exists a set $\mathcal{P}$ of $|X|$ paths linking X to S, where paths are pairwise edge-disjoint except that if $\{e, e'\} \subseteq X$ for some edge e, then two distinct paths in $\mathcal{P}$ end in e.*

We let $U(E, p)$ denote the uniform matroid of rank p on ground set $E(G)$, $M_G(p)$ the p-truncated graphic matroid of G, and $M(T)$ the edge-cut gammoid of (G, T).

If $M_1 = (E_1, \mathcal{I}_1)$ and $M_2 = (E_2, \mathcal{I}_2)$ are two matroids with $E_1 \cap E_2 = \emptyset$, then their *disjoint union* is the matroid $M_1 \uplus M_2 = (E_1 \cup E_2, \{I_1 \cup I_2 \mid I_1 \in \mathcal{I}_1, I_2 \in \mathcal{I}_2\})$. If M_1 and M_2 are represented by matrices A_1 and A_2 over the same field, then $M_1 \uplus M_2$ is represented by the matrix $A = \begin{pmatrix} A_1 & 0 \\ 0 & A_2 \end{pmatrix}$. We will define matroids M as the disjoint union over several copies of the base matroids $M(T)$, $M_G(p)$ and $U(E, p)$ defined above. In such a case, we refer to the individual base matroids making up M as the *layers* of M.

Representative sets

Our main technical tool is the representative sets lemma, due to Lovász [23] and Marx [25]. This result has been important in FPT algorithms [25, 8] and has been central to the previous kernelization algorithms for cut problems, including variants of Multiway Cut [17]. We also introduce some further notions.

▶ **Definition 12.** *Let $M = (E, \mathcal{I})$ be a matroid and $X, Y \in \mathcal{I}$. We say that Y* extends *X* in *M if $r(X \cup Y) = |X| + |Y|$, or equivalently, if $X \cap Y = \emptyset$ and $X \cup Y \in \mathcal{I}$. Furthermore, let $c = O(1)$ be a constant and let $\mathcal{Y} \subseteq \binom{E}{c}$. We say that a set $\hat{\mathcal{Y}} \subseteq \mathcal{Y}$* represents *$\mathcal{Y}$* in *$M$ if the following holds: For every $X \in \mathcal{I}$ for which there exists some $Y \in \mathcal{Y}$ such that Y extends X in M, then there exists some $Y' \in \hat{\mathcal{Y}}$ such that Y' extends X in M.*

▶ **Lemma 13** (representative sets lemma [23, 25])**.** *Let $M = (E, \mathcal{I})$ be a linear matroid represented by a matrix A of rank $r + s$, and let $\mathcal{Y} \subseteq \binom{E}{s}$ be a collection of independent sets of M, where $s = O(1)$. In time polynomial in the size of A and the size of $\mathcal{Y}$, we can compute a set $\hat{\mathcal{Y}} \subseteq \mathcal{Y}$ of size at most $\binom{r+s}{s}$ which represents $\mathcal{Y}$ in M.*

We will use the following *product form* of the representative sets lemma, with stronger specialized bounds. Assume that the rank of M is $r = r_1 + \ldots + r_c$, where r_i is the rank of layer i of M. Then Lemma 13 gives a bound on $|\hat{\mathcal{Y}}|$ as $\Theta((r_1 + \ldots + r_c)^c)$, but the following bound is significantly better when the layers of M have different rank.

▶ **Lemma 14** ([17]). *Let $M = (E, \mathcal{I})$ be a linear matroid, given as the disjoint union of c matroids $M_i = (E_i, \mathcal{I}_i)$, where M_i has rank r_i. Let $\mathcal{Y} \subseteq \binom{E}{c}$ be such that every set $Y \in \mathcal{Y}$ contains precisely one member in each layer M_i of M. Then the representative set $\hat{\mathcal{Y}} \subseteq \mathcal{Y}$ computed by the representative sets lemma will have $|\hat{\mathcal{Y}}| \leq \prod_{i=1}^{c} r_i$.*

3.2.2 The marking process

We are now ready to present the marking process.

Let $r = c - 2$. We define a process that marks edges of G in r passes, where each pass is a call to the representative sets lemma with a different matroid construction. Specifically, for each $i \in [r]$, define the following. The matroid M_i is the disjoint union of i copies of the edge-cut gammoid $M(T)$, one copy of $M_G(k^{r-i})$, and one copy of $U(E, k)$, where for $i = r$ we simply skip the copy of $M_G(k^0)$. We refer to the first i layers in M_i as the *gammoid layers* and the remaining as the *additional layers.* Note that a linear representation of M_i over some common field $\mathbb{F}$ can be computed in randomized polynomial time, since every layer of M_i can be represented over any sufficiently large field.

For each edge $e \in E$, let $t_i(e)$ be the set that contains a copy of z'_e in every gammoid layer, and a copy of e in every additional layer. Let $E_i = \{t_i(e) \mid e \in E\}$. For each pass $i \in [r]$, we compute a representative set $\hat{E}_i \subseteq E_i$ in the matroid M_i, and let $Z_i \subseteq E$ be the set of edges represented in $\hat{E}_i$. Let $Z = Z_1 \cup \ldots \cup Z_r$. We consider an edge $e \in E$ *marked* if $e \in Z$. We finish the description by observing the bound on the number of marked edges.

▶ **Lemma 15.** *The total number of marked edges is at most $rk^{r+1} < k^c$.*

Proof. By the product form of the representative sets lemma, $|Z_i| \leq k^{r+1}$ for every $i \in [r]$. ◀

Our main correctness condition for the marking is as follows. Consider a partition $\mathcal{T}$ of T and a corresponding minimum multiway cut $X \subseteq E$. Note that $|X| \leq k$ since $E(T, V)$ is a multiway cut for every partition. Say that X is *p-way plus q* if the p largest connected components of $G - X$ together cover all but q of the vertices. Say that X is *covered* if all edges essential for $\mathcal{T}$ are marked. We then have the following.

▶ **Lemma 16** (★). *If X is p-way plus k^{r-p} for $p \in [r]$, then X is covered in pass p above.*

Proof sketch. We define a set F_e such that $t_p(f)$ extends F_e if and only if $f = e$. By the existence of the set F_e, we then have a guarantee that $e \in Z_p$. To construct F_e, we follow [17] in the first p layers by letting F_e contain $X \cup \partial(T_i)$, where T_i is the set of vertices of T contained in the ith largest layer, thereby "blocking out" any edge contained in the p largest components from extending F_e. We use the two additional layers to block out edges containing in small components, respectively edges of $X - e$. Then by construction $t_p(f)$ fails to extend F_e for every edge $f \neq e$. Furthermore, as as in [17], we show that $t_p(e)$ extends F_e: if $t_p(e)$ fails to extend F_e in a gammoid layer, then this yields a "pushed solution" X_2 which is a minimum multiway cut for $\mathcal{T}$ with $e \notin X_2$, contradicting that e is essential for $\mathcal{T}$. In the additional layers, the argument for why e extends F_e is trivial. ◀

3.2.3 Correctness

We now argue that if (G, T) is (α, c)-dense for $c = \Omega(\alpha \log k)$ then every partition of T has a minimum multiway cut that is p-way plus k^{r-p} for some $p \in [r]$. For this, assume for a contradiction that for some partition $\mathcal{T}$ of T the minimum multiway cut X of $\mathcal{T}$ is not covered in any of the above passes. We will derive that $|X| > k$, contradicting that X is minimum. Assume that $G - X$ has p components, and let $n_1 \geq \ldots \geq n_p$ be the number of vertices in each component, sorted by size. The converse to Lemma 16 is the following.

▶ **Corollary 17.** *If X is not covered, then for every $i \in [r]$ it holds that $\sum_{j=i+1}^{p} n_j > k^{r-i}$.*

For $i \in [r]$, let us write $n_{\geq i} = \sum_{j=i}^{p} n_j$. Hence for each $i \in [r]$, $n_{\geq i+1} > k^{r-i}$.

Now, refer as previously to the vertex sets of the connected components of $G - X$ in order as $V_1, \ldots, V_p$, where $|V_i| = n_i$, $i \in [p]$. By the density assumption, for every $i \geq 2$, $\mathrm{cap}_T(V_i) \geq n_i^{1/c}/\alpha$. On the other hand, as previously noted, if X is minimum we have

$$\sum_{i=1}^{p} \mathrm{cap}_T(V_i) = \sum_{i=1}^{p} (\mathrm{cap}_G(T \cap V_i) + \delta(V_i)) = k + 2|X| \leq 3k. \tag{1}$$

It now remains to estimate the value of the following system:

$$\begin{array}{rl} \min & \sum_{i=2}^{p} n_i^{1/c}/\alpha \\ \text{s.t.} & \sum_{j=i+1}^{p} n_j > k^{r-i} \quad \forall i \in [r] \\ & \sum_{i=1}^{p} n_i = n \\ & n_1 \geq \ldots \geq n_p \geq 0 \end{array} \tag{2}$$

If we can determine that this value is greater than $3k$, then we will have derived a contradiction, showing that the cut X is covered. This is somewhat intricate, but not very difficult.

▶ **Lemma 18 (★).** *There is a $c = \Theta(\alpha \log k)$ such that the following holds: If (G, T) is (α, c)-dense, and if X is a multiway cut for some partition $\mathcal{T}$ of T such that X is not covered, then $|X| > k$.*

Proof sketch. Through concavity, one can show that the worst-case component sizes (i.e., the distribution n_i for which the system above achieves its minimum value) is when $n_i = k^{r-i+1}(1 - o(1))$ for every $i \geq 2$. The value of the system then becomes a geometric sum with ratio $k^{1/c} = 2^{\Theta(1/\alpha)}$, hence the total contribution is $\Theta((1/\alpha)k/(k^{1/c} - 1))$. Computing the asymptotics of the contributing factor $k^{1/c} - 1$ shows that it defeats $1/\alpha$, and establishes the result. ◀

3.3 Completing the result

By the above, every terminal network (G, T) that is (α, c)-dense for some $c = \Theta(\alpha \log k)$ has a multicut-covering set of at most k^c edges, which can be computed in randomized polynomial time. We extend the result to any (G, T), using a sublogarithmic terminal expansion tester.

▶ **Theorem 19** (Theorem 1 restated). *Let A be a sublogarithmic terminal expansion tester with ratio $\alpha(n, k)$. Let (G, T) be a terminal network with $\mathrm{cap}_G(T) = k$. There is a multicut-covering set $Z \subseteq E(G)$ with $|Z| \leq k^{O(\alpha(n,k) \log k)}$, which furthermore can be computed in randomized polynomial time using calls to A.*

Proof. Set $c = \Theta(\alpha \log k)$ as in Lemma 18. If $|E(G)| < k^c$ then return $Z = E(G)$, otherwise call A on (G, T, c). If (G, T) is (α, c)-dense, then Lemma 9 applies and we are done. Otherwise, let $S \subseteq V(G)$ be the set returned by A, and let $k_S = \mathrm{cap}_T(S)$. Let $(G_S, T(S))$ be the recursive instance at S, and note that $|V(G_S)| = |N_G[S]| < |V|$ and $|S| > k_S^c$ by definition of A. We may now proceed by induction on $|V|$ and assume that we can compute a multicut-covering set $Z_S \subseteq E(G_S)$ of size $|Z_S| < k_S^c$. To eliminate a corner case, if there is a vertex $v \in V(G_S)$ with $v \notin T(S)$ and $d_{G_S}(v) \leq 2$, then delete v if v is a leaf, otherwise contract one edge incident with v. Note that since $v \notin T(S)$ we have $d_G(v) = d_{G_S}(v)$ and $v \notin T$, hence these reduction rules are clearly correct. If this rule does not apply, there must be some edge $e \in E(G_S) \setminus Z_S$, and by construction e corresponds directly to an edge in G. Hence by

Prop. 8 we may contract e in G and repeat. This yields a graph G' with $|V(G')| < |V|$, hence by induction we can create a multicut-covering set Z for G', which is also a multicut-covering set of G by Prop. 8. Hence we can compute a multicut-covering set Z with $|Z| < k^c$. ◀

We observe the following consequences.

▶ **Corollary 20.** *Let (G,T) be a terminal network with $\text{cap}_G(T) = k$. The following holds.*

1. *There is a multicut-mimicking network for (G,T) with $k^{O(\log k)}$ edges.*
2. *If there is a sublogarithmic terminal expansion tester – in particular, if* Small Set Expansion *has an approximation ratio as in Theorem 19 – then a multicut-mimicking network of size quasipolynomial in k can be computed in randomized polynomial time.*

Proof. The first is immediate using $\alpha(n,k) = 1$. For the second, all that remains is to clean up the value $|Z|$. For this, let $\alpha(n,k) \leq \log^{1-\varepsilon} n \log^d k$ and $c = b\alpha \log k$, for some bounded values b, d, and first assume that $|Z| \geq |V(G)| = n$. Then

$$\begin{aligned} n \leq |Z| < k^{b\alpha \log k} &\Rightarrow \\ \log n < b\alpha \log^2 k &\Rightarrow \\ \log n < b \log^{1-\varepsilon} n \log^{d+2} k &\Rightarrow \\ \log^{\varepsilon} n < b \log^{d+2} k &\Rightarrow \\ \log n < (b \log^{d+2} k)^{1/\varepsilon}&, \end{aligned}$$

hence $|Z| \leq k^{\log^{O(1)} k}$, as promised. Otherwise, we contract all edges not present in Z and compute a new multicut-covering set Z' for the new system (G', T). Eventually, this process halts, and at this point we will have a multicut-covering set Z with $|Z| \leq k^{\log^{O(1)} k}$ for some graph G'' created by contractions from G, and by Prop. 8 this set Z is also a multicut-covering set for (G,T). ◀

3.4 Kernelization extensions and consequences

As noted, we get the following consequences.

▶ **Corollary 21 (★).** *If there is a sublogarithmic terminal expansion tester, then the following problems have randomized quasipolynomial kernels.*

1. Edge Multiway Cut *parameterized by solution size.*
2. Edge Multicut *parameterized by the solution size and the number of cut requests.*
3. Group Feedback Edge Set *parameterized by solution size, for any group.*
4. Subset Feedback Edge Set *with undeletable edges, parameterized by solution size.*
5. 0-Extension *for integer-weighted graphs, parameterized by solution cost.*

Finally, as in [32], the latter result extends to "0-Extension sparsifiers" which hold independent of the choice of metric. Let us briefly recall some details. An instance of 0-Extension can be defined as a terminal network (G,T), a metric $\mu\colon D \times D \to \mathbb{R}^+$ for some label set D, and a partial labelling $\tau\colon T \to D$. The goal is to find $\lambda\colon V(G) \to D$ extending τ, to minimize the cost $\sum_{uv \in E(G)} \mu(\lambda(u), \lambda(v))$. We note that the "kernel" in the previous result can be constructed without needing access to μ or τ, i.e., it is valid for every metric μ and every partial labelling τ.

▶ **Theorem 22 (★).** *Let $G = (V,E)$ be an undirected, unweighted graph and $T \subseteq V$ a set of terminals, $|T| = r$. For any integer $p \in \mathbb{N}$, let $k = p + r$; there exists a set $Z \subseteq E$ with $|Z| = k^{O(\log k)}$ such that the following holds. For any metric $\mu\colon D \times D \to \mathbb{R}^+$ and any labelling $\tau\colon T \to D$, if there exists a labelling $\lambda\colon V \to D$ extending τ where $\lambda(u) \neq \lambda(v)$ for at most p edges $uv \in E$, then there exists such a labelling λ, of minimum cost among all such labellings, such that $\lambda(u) = \lambda(v)$ for every edge $uv \in E \setminus Z$.*

4 Discussion

We defined the notion of a *multicut-mimicking network*, and showed that every terminal network (G, T) with $k = \text{cap}_G(T)$ admits one of size $k^{O(\log k)}$, which furthermore may be computable in randomized polynomial time, subject to the precise approximation guarantees available for a restricted variant of Small Set Expansion. The mimicking network can be constructed via contractions on G, i.e., it simply consists of a set of edges which form a *multicut-covering set.* As a consequence of such a result, a range of parameterized problems, starting from Edge Multiway Cut, would have quasipolynomial kernels. Unfortunately, the approximation guarantee required for this latter result appears to go just below the range of available guarantees from the literature.

A natural question is whether an appropriate approximation algorithm can be constructed. We note that an approximation ratio of polylog(k) for Small Set Expansion is sufficient, where $k = \delta(S)$ is the number of edges cut in the optimal solution S. We are not aware of approximation ratios in term of this parameter having been investigated. Also note that it is sufficient if the approximation algorithm has a running time quasipolynomial in k (but polynomial in n).

Another question is whether the existence of a polynomial-sized multicut-mimicking network can be established. Can such a result be excluded, even for the apparently more demanding situation of sparsifiers for 0-Extension instances (as in Theorem 22)?

We also have not investigated the vertex-deletion versions of these problems, which seem likely to bring significant additional difficulty (if such a generalization is possible).

In either case, the existence of a quasipolynomial multicut-covering set appears to rule out any possibility of a lower bound against the kernelizability of Edge Multiway Cut for any size better than quasipolynomial, given the nature of the lower bound results against kernelization. We hope, therefore (but dare not explicitly conjecture) that Edge Multiway Cut and related problems have quasipolynomial (randomized) kernels or better, unconditionally.

References

1 Alexandr Andoni, Anupam Gupta, and Robert Krauthgamer. Towards $1+\varepsilon$-approximate flow sparsifiers. In *SODA*, pages 279–293. SIAM, 2014.

2 Nikhil Bansal, Uriel Feige, Robert Krauthgamer, Konstantin Makarychev, Viswanath Nagarajan, Joseph Naor, and Roy Schwartz. Min-max graph partitioning and small set expansion. *SIAM J. Comput.*, 43(2):872–904, 2014.

3 Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Danny Hermelin. On problems without polynomial kernels. *J. Comput. Syst. Sci.*, 75(8):423–434, 2009. doi:10.1016/j.jcss.2009.04.001.

4 Julia Chuzhoy. On vertex sparsifiers with Steiner nodes. In *STOC*, pages 673–688. ACM, 2012.

5 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms.* Springer, 2015.

6 Elias Dahlhaus, David S. Johnson, Christos H. Papadimitriou, Paul D. Seymour, and Mihalis Yannakakis. The complexity of multiterminal cuts. *SIAM J. Comput.*, 23(4):864–894, 1994.

7 Matthias Englert, Anupam Gupta, Robert Krauthgamer, Harald Räcke, Inbal Talgam-Cohen, and Kunal Talwar. Vertex sparsifiers: New results from old techniques. *SIAM J. Comput.*, 43(4):1239–1262, 2014.

8 Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, and Saket Saurabh. Efficient computation of representative families with applications in parameterized and exact algorithms. *J. ACM*, 63(4):29:1–29:60, 2016. doi:10.1145/2886094.

9 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: Theory of Parameterized Preprocessing.* Cambridge University Press, 2019. doi:10.1017/9781107415157.

10 Lance Fortnow and Rahul Santhanam. Infeasibility of instance compression and succinct PCPs for NP. *J. Comput. Syst. Sci.*, 77(1):91–106, 2011. doi:10.1016/j.jcss.2010.06.007.

11 Naveen Garg, Vijay V. Vazirani, and Mihalis Yannakakis. Approximate max-flow min-(multi)cut theorems and their applications. *SIAM J. Comput.*, 25(2):235–251, 1996. doi:10.1137/S0097539793243016.

12 Torben Hagerup, Jyrki Katajainen, Naomi Nishimura, and Prabhakar Ragde. Characterizing multiterminal flow networks and computing flows in networks of small treewidth. *J. Comput. Syst. Sci.*, 57(3):366–375, 1998.

13 Eva-Maria C. Hols and Stefan Kratsch. A randomized polynomial kernel for subset feedback vertex set. *Theory Comput. Syst.*, 62(1):63–92, 2018. doi:10.1007/s00224-017-9805-6.

14 Nikolai Karpov, Marcin Pilipczuk, and Anna Zych-Pawlewicz. An exponential lower bound for cut sparsifiers in planar graphs. *Algorithmica*, 81(10):4029–4042, 2019.

15 Arindam Khan and Prasad Raghavendra. On mimicking networks representing minimum terminal cuts. *Inf. Process. Lett.*, 114(7):365–371, 2014.

16 Stefan Kratsch. A randomized polynomial kernelization for vertex cover with a smaller parameter. *SIAM J. Discrete Math.*, 32(3):1806–1839, 2018. doi:10.1137/16M1104585.

17 Stefan Kratsch and Magnus Wahlström. Representative sets and irrelevant vertices: New tools for kernelization. In *FOCS*, pages 450–459, 2012. doi:10.1109/FOCS.2012.46.

18 Stefan Kratsch and Magnus Wahlström. Compression via matroids: A randomized polynomial kernel for odd cycle transversal. *ACM Trans. Algorithms*, 10(4):20:1–20:15, 2014.

19 Robert Krauthgamer and Havana Inbal Rika. Refined vertex sparsifiers of planar graphs. *SIAM J. Discrete Math.*, 34(1):101–129, 2020.

20 Robert Krauthgamer and Inbal Rika. Mimicking networks and succinct representations of terminal cuts. In *SODA*, pages 1789–1799. SIAM, 2013.

21 Frank Thomson Leighton and Ankur Moitra. Extensions and limits to vertex sparsification. In *STOC*, pages 47–56. ACM, 2010.

22 Daniel Lokshtanov, Pranabendu Misra, Fahad Panolan, and Saket Saurabh. Deterministic truncation of linear matroids. *ACM Trans. Algorithms*, 14(2):14:1–14:20, 2018.

23 László Lovász. Flats in matroids and geometric graphs. In *Proc. Sixth British Combinatorial Conf.*, Combinatorial Surveys, pages 45–86, 1977.

24 Dániel Marx. Parameterized graph separation problems. *Theor. Comput. Sci.*, 351(3):394–406, 2006. doi:10.1016/j.tcs.2005.10.007.

25 Dániel Marx. A parameterized view on matroid optimization problems. *Theor. Comput. Sci.*, 410(44):4471–4479, 2009. doi:10.1016/j.tcs.2009.07.027.

26 Ankur Moitra. Approximation algorithms for multicommodity-type problems with guarantees independent of the graph size. In *FOCS*, pages 3–12. IEEE Computer Society, 2009.

27 G. Nemhauser and L. Trotter. Vertex packing: structural properties and algorithms. *Mathematical Programming*, 8:232–248, 1975. doi:10.1007/BF01580444.

28 James Oxley. *Matroid Theory.* Oxford University Press, 2011.

29 Harald Räcke. Minimizing congestion in general networks. In *FOCS*, pages 43–52. IEEE Computer Society, 2002. doi:10.1109/SFCS.2002.1181881.

30 Harald Räcke. Optimal hierarchical decompositions for congestion minimization in networks. In *STOC*, pages 255–264. ACM, 2008.

31 Prasad Raghavendra and David Steurer. Graph expansion and the unique games conjecture. In *STOC*, pages 755–764. ACM, 2010.

32 Felix Reidl and Magnus Wahlström. Parameterized algorithms for zero extension and metric labelling problems. In *ICALP*, volume 107 of *LIPIcs*, pages 94:1–94:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018.

Hardness of Equations over Finite Solvable Groups Under the Exponential Time Hypothesis

Armin Weiß
Universität Stuttgart, Institut für Formale Methoden der Informatik (FMI), Germany
armin.weiss@fmi.uni-stuttgart.de

Abstract

Goldmann and Russell (2002) initiated the study of the complexity of the equation satisfiability problem in finite groups by showing that it is in P for nilpotent groups while it is NP-complete for non-solvable groups. Since then, several results have appeared showing that the problem can be solved in polynomial time in certain solvable groups of Fitting length two. In this work, we present the first lower bounds for the equation satisfiability problem in finite solvable groups: under the assumption of the exponential time hypothesis, we show that it cannot be in P for any group of Fitting length at least four and for certain groups of Fitting length three. Moreover, the same hardness result applies to the equation identity problem.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness

Keywords and phrases equations in groups, solvable groups, exponential time hypothesis

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.102

Category Track A: Algorithms, Complexity and Games

Related Version Preprint available at `https://arxiv.org/abs/2002.10145`.

Funding *Armin Weiß*: Funded by DFG project DI 435/7-1.

Acknowledgements I am grateful to Moses Ganardi for bringing my attention both to the AND-weakness conjecture and to the exponential time hypothesis. I am also thankful to David A. Mix Barrington for an interesting email exchange concerning the AND-weakness conjecture and the idea to include steps of the lower central series in Proposition 8 to get a more refined upper bound. Furthermore, I am indebted to Caroline Mattes and Jan Philipp Wächter for many helpful discussions. Finally, I want to thank the anonymous referees for their valuable comments.

1 Introduction

The study of equations over algebraic structures has a long history in mathematics. Some of the first explicit decidability results in group theory are due to Makanin [33], who showed that equations over free groups are decidable. Subsequently several other decidability and undecidability results as well as complexity results on equations over infinite groups emerged (see [11, 14, 32, 37] for a random selection). For a fixed group G, the equation satisfiability problem EQN-SAT is as follows: given an expression $\alpha \in (G \cup \mathcal{X} \cup \mathcal{X}^{-1})^*$ where $\mathcal{X}$ is some set of variables, the question is whether there exists some assignment $\sigma : \mathcal{X} \to G$ such that $\sigma(\alpha) = 1$ (here σ is extended to expressions in the natural way – $\mathcal{X}^{-1}$ is a disjoint copy of $\mathcal{X}$ representing the inverses of $\mathcal{X}$). Likewise EQN-ID is the problem, given an expression, decide whether it evaluates to 1 under *all* assignments.

Henceforth, all groups we consider are finite. In this case, equation satisfiability and related questions are clearly decidable by an exhaustive search. Still the complexity is an interesting topic of research: its study has been initiated by Goldmann and Russell [15], who showed that satisfiability of systems of equations can be decided in P if and only if the group is abelian (assuming P ≠ NP) – otherwise, the problem is NP-complete. They also obtained

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 102; pp. 102:1–102:19
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

some results for single equations: EQN-SAT is NP-complete for non-solvable groups, while for nilpotent groups it is in P. This left the case of solvable but non-nilpotent groups open. Indeed, Burris and Lawrence raised the question whether EQN-ID$(G) \in$ P for all finite solvable groups G [9, Problem 1]. Moreover, Horváth [18] conjectured a positive answer.

Contribution. In this work we give a negative answer to this question assuming the exponential time hypothesis by showing the following result:

▶ **Corollary A.** *Let G be finite solvable group and assume that either*

- *the Fitting length of G is at least four, or*
- *the Fitting length of G is three and there is no Fitting-length-two normal subgroup whose index is a power of two.*

Then EQN-SAT(G) and EQN-ID(G) are not in P *under the exponential time hypothesis.*

To the best of our knowledge, this constitutes the first hardness results for EQN-SAT(G) and EQN-ID(G) if G is solvable.[1] The Fitting length of a group G is the minimal d such that there is a sequence $1 = G_0 \trianglelefteq \cdots \trianglelefteq G_d = G$ with all quotients G_{i+1}/G_i nilpotent.

Moreover, we show that if S is a semigroup with a group divisor (i.e., a group which is a quotient of a subsemigroup of S) meeting the requirements of Corollary A, EQN-SAT(S) (here the input consists of two expressions) is also not in P under the exponential time hypothesis. Finally, using the same ideas as for our main result, we derive an upper bound of $2^{\mathcal{O}(n^{1/(d-1)})}$ for the length of the shortest G-program (definition see below) for the n-input AND function in a finite solvable group of Fitting length $d \geq 2$. Notice that a corresponding $2^{n^{\Omega(1)}}$ lower bound would imply that EQN-SAT(G) and EQN-ID(G) can be solved in quasipolynomial time for finite solvable groups G.

General approach. The complexity of EQN-SAT is closely related to the complexity of the satisfiability problem for G-programs (denoted by ProgramSAT – for a definition see Section 3). Indeed, [5] gives a reduction from EQN-SAT to ProgramSAT (be aware that, while the problems EQN-SAT and ProgramSAT are well-defined for finitely generated infinite groups, in general, such a reduction exists only in the case of finite groups). Moreover, also ProgramSAT is in P for nilpotent groups and NP-complete for non-solvable groups [6].

In order to show hardness of these problems, one usually reduces some NP-complete problem like 3SAT or C-Coloring to them. Typically, this requires to encode big logical conjunctions into the group G. Therefore, the complexity of these problems is linked to the length of the shortest G-program for the AND function. Indeed, [5, Theorem 4] shows that, if the AND function can be computed by a P-uniform family of G-programs of polynomial length, then ProgramSAT$(G \wr C_k)$ for $k \geq 4$ is NP-complete (here C_k denotes the cyclic group of order k; P-uniform means that the n-input G-program can be computed in time polynomial in n). Thus, if there exists a solvable group with efficiently computable polynomial length G-programs for the AND function, then there is a solvable group with an NP-complete ProgramSAT problem.

[1] Recently (a preprint appeared only days after the submission of this paper), in [24] Idziak, Kawałek, and Krzaczkowski succeeded to show that EQN-SAT(S_4) is not in P under the exponential time hypothesis (S_4 denotes the symmetric group over four elements). Moreover, they proved similar results as in this work for the case of algebras from congruence modular varieties. This complements our main result Corollary A. Indeed, a joint paper proving a quasipolynomial lower bound on EQN-SAT and EQN-ID for *all* finite groups of Fitting length three is under preparation.

It is well-known that G-programs describe the circuit complexity class CC^0 [34] with the depth of the circuit relating to the Fitting length of the group. One can make a depth size trade-off for the AND function using a divide-and-conquer approach: Assume there is a circuit of depth two and size 2^n for the n-input AND (which is the case by [3]). Since the n-input AND can be decomposed as $\sqrt{n}$-input AND of $\sqrt{n}$ many $\sqrt{n}$-input ANDs, we obtain a CC^0 circuit of depth 4 and size roughly $2^{\sqrt{n}}$.

This observation plays a crucial role for our results: it allows us to reduce an m-edge C-Coloring instance to an equation of size roughly $2^{\sqrt{m}}$. We compare this to the exponential time hypothesis (ETH), which conjectures that n-variable 3SAT cannot be solved in time $2^{o(n)}$. ETH implies that C-Coloring cannot be solved in time $2^{o(m)}$, which gives us a quasipolynomial lower bound on EQN-SAT and EQN-ID. Notice that in the literature there are several other quasipolynomial lower bounds building on the exponential time hypothesis – see [1, 7, 8] for some examples.

Outline. In Section 2, we fix our notation and state some basic results on inducible and atomically universally definable subgroups. Some of these observations are well-known, while others, to the best of our knowledge, have not been stated explicitly. Section 3 gives a little excursion to the complexity of the AND-function in terms of G-programs over finite solvable groups deriving an upper bound $2^{\mathcal{O}(n^{1/(d-1)})}$ if $d \geq 2$ is the Fitting length of G.

Section 4 and Section 5 are the main part of our paper: we reduce the C-Coloring problem to EQN-SAT and EQN-ID. For the reduction, we need some special requirements on the group G. In Section 5 we show that actually the requirements of Corollary A are enough using the concept of inducible and atomically universally definable subgroups. Finally, in Corollary 22 we examine consequences to EQN-SAT in semigroups.

Related work on equations. Since the work of Goldman and Russell [15] and Barrington et al. [5], a long list of literature has appeared investigating EQN-ID and EQN-SAT in groups and other algebraic structures. In [9] it is shown that EQN-ID is in P for nilpotent groups as well as for dihedral groups D_k where k is odd. Horváth resp. Horváth and Szabó [19, 22] extended these results by showing the following among other results: EQN-SAT(G) is in P for $G = C_n \rtimes B$ with B abelian, $n = p^k$ or $n = 2p^k$ for some prime p and EQN-ID is in P for semidirect products $G = C_{n_1} \rtimes (C_{n_2} \rtimes \cdots \rtimes (C_{n_k} \rtimes (A \rtimes B)))$ with A, B abelian (be aware that such a group is two-step solvable). Furthermore, in [12] it is proved that EQN-SAT$(G) \in \mathsf{P}$ for so-called semi-pattern groups. Finally, in [13] Földvári and Horváth established that EQN-SAT is in P for the semidirect product of a p-group and an abelian group and that EQN-ID is in P for the semidirect product of a nilpotent group with an abelian group. Notice that all these groups have in common that their Fitting length is at most two.

In [20, 21] the EQN-SAT and EQN-ID problems for generalized terms are introduced. Here a generalized term means an expression which may also use commutators or even more complicated terms inside the input expression. Using commutators is a more succinct representation, which allows for showing that EQN-SAT is NP-complete and EQN-ID is coNP-complete in the alternating group A_4 [21]. In [31] this result is extended by showing that, with commutators and the generalized term $w(x, y_1, y_2, y_3) = x^8[x, y_1, y_2, y_3]$, EQN-SAT is NP-complete and EQN-ID is coNP-complete for all non-nilpotent groups.

There is also extensive literature on equations in other algebraic structures – for instance, [2, 5, 26, 27, 28, 29, 38, 39, 40] in semigroups. We only mention two of them explicitly: [27] showed that identity checking (EQN-ID without constants in the input) in semigroups is coNP complete. Moreover, among other results, [2] reduces the identity checking problem in the direct product of maximal subgroups to identity checking in some semigroup.

2 Preliminaries

The set of words over some alphabet Σ is denoted by Σ^*. The length of a word $w \in \Sigma^*$ is denoted by $|w|$. We denote the interval of integers $\{ i, \dots, j \}$ by $[i \mathinner{..} j]$.

Complexity. We use standard notation from complexity theory. In several cases we use the notion of AC^0 many-one reductions (denoted by $\leq_{\mathrm{m}}^{\mathsf{AC}^0}$) meaning that the reducing function can be computed in AC^0 (i.e., by a polynomial-size, constant-depth Boolean circuit). The reader unfamiliar with this terminology may think about logspace or polynomial time reductions. Also be aware that in order to obtain AC^0 many-one reductions in most cases we need the presence of a letter representing the group identity for padding reasons.

Exponential time hypothesis. The exponential time hypothesis (ETH) is the conjecture that there is some $\delta > 0$ such that every algorithm for 3SAT needs time $\Omega(2^{\delta n})$ in the worst case where n is the number of variables of the given 3SAT instance. By the sparsification lemma [25, Thm. 1] this is equivalent to the existence of some $\epsilon > 0$ such that every algorithm for 3SAT needs time $\Omega(2^{\epsilon(m+n)})$ in the worst case where m is the number of clauses of the given 3SAT instance (see also [10, Thm. 14.4]). In particular, under ETH there is no algorithm for 3SAT running in time $2^{o(n+m)}$.

***C*-Coloring.** A C-coloring for $C \in \mathbb{N}$ of a graph $\Gamma = (V, E)$ is a map $\chi : V \to [1 \mathinner{..} C]$. A coloring χ is called *valid* if $\chi(u) \neq \chi(v)$ whenever $\{ u, v \} \in E$. The problem C-Coloring is as follows: given an undirected graph $\Gamma = (V, E)$, the question is whether there is a valid C-coloring of Γ. The C-Coloring problem is one of the classical NP-complete problems for $C \geq 3$. Moreover, by [10, Thm. 14.6], 3-Coloring cannot be solved in time $2^{o(|V|+|E|)}$ unless ETH fails. Since 3-Coloring can be reduced to C-Coloring for fixed $C \geq 3$ by introducing only a linear number of additional edges and a constant number of vertices, it follows for every $C \geq 3$ that also C-Coloring cannot be solved in time $2^{o(|V|+|E|)}$ unless ETH fails.

Commutators and Fitting series. Throughout, we only consider finite groups G. We use notation similar to [36]. We write $[x, y] = x^{-1}y^{-1}xy$ for the commutator and $x^y = y^{-1}xy$ for the conjugation. Moreover, we write $[x_1, \dots, x_n] = [[x_1, \dots, x_{n-1}], x_n]$ for $n \geq 3$.

As usual for subsets $X, Y \subseteq G$, we write $\langle X \rangle$ for the subgroup generated by X and we define $[X, Y] = \langle\, [x, y] \mid x \in X, y \in Y \,\rangle$ and $[X_1, \dots, X_k] = [[X_1, \dots, X_{k-1}], X_k]$ for $X_1, \dots, X_k \subseteq G$. In contrast, we write $[X, Y]_{\mathrm{set}} = \{ [x, y] \mid x \in X, y \in Y \}$ (thus, $[X, Y] = \langle [X, Y]_{\mathrm{set}} \rangle$) and $[X_1, \dots, X_k]_{\mathrm{set}} = [[X_1, \dots, X_{k-1}]_{\mathrm{set}}, X_k]_{\mathrm{set}}$.

Finally, we denote the set $\{ g^x \mid x \in X \}$ with g^X (be aware that here we differ from [36]) and define $X^Y = \{ x^y \mid x \in X, y \in Y \}$.

▶ **Lemma 1.** *If $X_i^G = X_i \subseteq G$ for $i = 1, \dots, k$, then*

$$[\langle X_1 \rangle, \dots, \langle X_k \rangle] = \langle [X_1, \dots, X_k]_{\mathrm{set}} \rangle .$$

Proof. By [36, 5.1.7], we have $[\langle X \rangle, \langle Y \rangle] = [X, Y]^{\langle X \rangle \langle Y \rangle}$ for arbitrary $X, Y \subseteq G$. Thus, if $X = X^G$ and $Y = Y^G$, we have $[\langle X \rangle, \langle Y \rangle] = [X, Y]$. We use this to show the lemma by induction:

$$\begin{aligned}
[\langle X_1 \rangle, \dots, \langle X_k \rangle] &= \big[[\langle X_1 \rangle, \dots, \langle X_{k-1} \rangle], \langle X_k \rangle\big] \\
&= \big[\langle [X_1, \dots, X_{k-1}]_{\mathrm{set}} \rangle, \langle X_k \rangle\big] && \text{(by induction)} \\
&= \big[[X_1, \dots, X_{k-1}]_{\mathrm{set}}, X_k\big] && \text{(by [36, 5.1.7])} \\
&= \langle [X_1, \dots, X_k]_{\mathrm{set}} \rangle
\end{aligned}$$

◀

For $x, y \in G$, we write $[x, {}_k\, y] = [x, \underbrace{y, \dots, y}_{k \text{ times}}]$ and likewise for $X, Y \subseteq G$, we write $[X, {}_k\, Y] = [X, \underbrace{Y, \dots, Y}_{k \text{ times}}]$ and $[{}_k\, Y] = [\underbrace{Y, \dots, Y}_{k \text{ times}}]$ and analogously $[X, {}_k\, Y]_{\text{set}}$ and $[{}_k\, Y]_{\text{set}}$.

Since G is finite, there is some $M = M(G) \in \mathbb{N}$ such that $[X, {}_M\, Y] = [X, {}_i\, Y]$ for all $i \geq M$ and all $X, Y \subseteq G$ with $X^G = X$ and $Y^G = Y$ (notice that $[X, {}_i\, Y] \leq [X, {}_j\, Y]$ for $j \leq i$ due to the normality of $[X, Y]$). It is clear that $M = |G|$ is large enough, but typically much smaller values suffice.

▶ **Lemma 2.** *For all $X, Y \subseteq G$ with $X^G = X$ we have $[X, {}_M\, Y] = [[X, G], {}_M\, Y]$.*

Proof. We have $[X, G] \leq \langle X \rangle$ because $[x, g] = x^{-1}x^g \in X$. Thus, the inclusion right to left follows. The other inclusion is because $[X, {}_M\, Y] = [X, {}_{M+1}\, Y] \leq [X, G, {}_M Y] = [[X, G], {}_M\, Y]$. ◀

The k-th term of the lower central series is $\gamma_k G = [G, {}_k\, G]$. The *nilpotent residual* of G is defined as $\gamma_\infty G = \gamma_M G$ where M is as above (i.e., $\gamma_\infty G = \gamma_i G$ for every $i \geq M$). Recall that a finite group G is nilpotent if and only if $\gamma_\infty G = 1$.

The *Fitting* subgroup $\mathrm{Fit}(G)$ is the union of all nilpotent normal subgroups. Let G be a finite solvable group. It is well-known that $\mathrm{Fit}(G)$ itself is a nilpotent normal subgroup (see e.g. [23, Satz 4.2]). The *upper Fitting series*

$$1 = \mathcal{U}_0 G \lhd \mathcal{U}_1 G \lhd \cdots \lhd \mathcal{U}_k G = G$$

is defined by $\mathcal{U}_{i+1}G / \mathcal{U}_i G = \mathrm{Fit}(G/\mathcal{U}_i G)$. The *lower Fitting series*

$$1 = \mathcal{L}_d G \lhd \cdots \lhd \mathcal{L}_1 G \lhd \mathcal{L}_0 G = G$$

is defined by $\mathcal{L}_{i+1} G = \gamma_\infty(\mathcal{L}_i G)$. We have $d = k$ (see e.g. [23, Satz 4.6]) and this number is called the *Fitting length* $\mathrm{FitLen}(G)$ (sometimes also referred to as *nilpotent length*). The following fact can be derived by a straightforward induction from the characterization of $\mathrm{Fit}(G)$ as largest nilpotent normal subgroup (for a proof see e.g. [41]):

▶ **Lemma 3.** *Let $H \trianglelefteq G$ be a normal subgroup. Then for all i, we have $\mathcal{U}_i H = \mathcal{U}_i G \cap H$. In particular,*

(i) *if $\mathrm{FitLen}(H) = i$, then $H \leq \mathcal{U}_i G$,*
(ii) *if $g \in \mathcal{U}_i G \setminus \mathcal{U}_{i-1} G$, then $\mathrm{FitLen}(\langle g^G \rangle) = i$.*

Equations in groups. An *expression* (also called a *polynomial* in [39, 22, 31]) over a group G is a word α over the alphabet $G \cup \mathcal{X} \cup \mathcal{X}^{-1}$ where $\mathcal{X}$ is a set of variables. Here $\mathcal{X}^{-1}$ denotes a formal set of inverses of the variables. Since we are dealing with finite groups only, a variable $X^{-1} \in \mathcal{X}^{-1}$ for $X \in \mathcal{X}$ can be considered as an abbreviation for $X^{|G|-1}$. Sometimes we write $\alpha(X_1, \dots, X_n)$ for an expression α to indicate that the variables occurring in α are from the set $\{ X_1, \dots, X_n \}$. Moreover, if $\beta_1, \dots, \beta_n$ are other expressions, we write $\alpha(\beta_1, \dots, \beta_n)$ for the expression obtained by substituting each occurrence of a variable X_i by the expression β_i.

An assignment for an expression α is a mapping $\sigma : \mathcal{X} \to G$ – here σ is canonically extended by $\sigma(X^{-1}) = \sigma(X)^{-1}$ and $\sigma(g) = g$ for $g \in G$. An assignment σ is *satisfying* if $\sigma(\alpha) = 1$ in G. The problems EQN-SAT(G) and EQN-ID(G) are as follows: for both of them the input is an expression α. For EQN-SAT(G) the question is whether there *exists* a satisfying assignment, for EQN-ID(G) the question is whether *all* assignments are satisfying.

Notice that in the literature EQN-SAT is also denoted by POL-SAT [39, 22] or Eq [31], while EQN-ID is also referred to as POL-EQ (e.g. in [39, 22, 28]) or Id [31].

If $\mathcal{X} = \mathcal{Y} \cup \mathcal{Z}$ with $\mathcal{Y} \cap \mathcal{Z} = \emptyset$ and we are given assignments $\sigma_1 : \mathcal{Y} \to G$ and $\sigma_2 : \mathcal{Z} \to G$, we obtain a new assignment $\sigma_1 \cup \sigma_2$ defined by $(\sigma_1 \cup \sigma_2)(X) = \sigma_1(X)$ if $X \in \mathcal{Y}$ and $(\sigma_1 \cup \sigma_2)(X) = \sigma_2(X)$ if $X \in \mathcal{Z}$. We write $[X \mapsto g]$ for the assignment $\{ X \} \to G$ mapping X to g.

Inducible subgroups. According to [15], we call a subset $S \subseteq G$ *inducible* if there is some expression $\alpha \in (G \cup \mathcal{X} \cup \mathcal{X}^{-1})^*$ such that $S = \{ \sigma(\alpha) \mid \sigma \colon \mathcal{X} \to G \}$. In this case we say that α *induces* S. Notice that in a finite group every verbal subgroup is inducible. (A subgroup is called *verbal* if it is generated by a set of the form $\{ \sigma(\alpha) \mid \sigma \colon \mathcal{X} \to G, \alpha \in \mathcal{A} \}$ where $\mathcal{A} \subseteq (\mathcal{X} \cup \mathcal{X}^{-1})^*$ is a *finite* set of expressions without constants.) This shows the first three points of the following lemma (for $\gamma_1 G$, see also [15, Lemma 5]):

▶ **Lemma 4.** *Let G be a finite group. Then*

(i) *for every $k \in \mathbb{N}$, the subgroup generated by all k-th powers is inducible,*

(ii) *every element $\gamma_k G$ of the lower central series is inducible,*

(iii) *every element $\mathcal{L}_k G$ of the lower Fitting series is inducible,*

(iv) *if $K \leq H \leq G$ and K is inducible in H and H inducible in G, then K is also inducible in G,*

(v) *if $H \leq G$ with $H = [G, H]$, then H is inducible.*

The fourth point follows simply by "plugging in" an expression for H inside an expression for K. The last point follows from the proof of [31, Lemma 9].

The notion of inducible subgroup turns out to be very useful for proving lower bounds on the complexity. Indeed, the following facts are straightforward:

▶ **Lemma 5** ([15, Lemma 8], [20, Lemma 9, 10]). *Let $H \leq G$ be an inducible subgroup. Then*

- *EQN-SAT(H) $\leq_{\mathrm{m}}^{\mathsf{AC}^0}$ EQN-SAT(G), and*
- *EQN-ID(H) $\leq_{\mathrm{m}}^{\mathsf{AC}^0}$ EQN-ID(G).*
- *If, moreover, H is normal in G, then EQN-SAT(G/H) $\leq_{\mathrm{m}}^{\mathsf{AC}^0}$ EQN-SAT(G).*

Let us briefly sketch the ideas to see this lemma: Fix an expression β inducing H. For first and second reduction, replace every occurring variable of a given equation by a copy of β with disjoint variables. The third reduction simply appends β to an input equation.

Atomically universally definable subgroups. The situation for reducing EQN-ID(G/H) to EQN-ID(G) is slightly more complicated. For this we need a new definition: We call a subset $S \subseteq G$ *atomically universally definable* if there is some expression $\alpha \in (G \cup \mathcal{X} \cup \mathcal{X}^{-1})^*$ where $\mathcal{X} = \{ X \} \cup \{ Y_1, Y_2, \ldots \}$ such that

$$S = \{ g \in G \mid (\sigma \cup [X \mapsto g])(\alpha) = 1 \text{ for all } \sigma \colon \{ Y_1, Y_2, \ldots \} \to G \} .$$

In this case we say that α *atomically universally defines* S. (Notice that *universally definable* usually is defined analogously but instead of a single equation α one allows a Boolean formula of equations.) It is clear that the center of a group is atomically universally definable by the expression $[X, Y]$. This generalizes as follows:

▶ **Lemma 6.** *Let G be a finite group.*

- *The Fitting group* $\mathrm{Fit}(G)$ *is atomically universally definable.*
- *If $N \leq H \leq G$ and N is normal in G and H/N is atomically universally definable in G/N and N is atomically universally definable in G, then H is atomically universally definable in G.*
- *All terms $\mathcal{U}_i G$ of the upper Fitting series are atomically universally definable.*
- *If $H \leq G$ is inducible, then the centralizer $C_G(H) = \{ g \in G \mid gh = hg \text{ for all } h \in H \}$ is atomically universally definable.*

Proof. By Lemma 3, the normal subgroup $\langle g^G \rangle$ generated by $g \in G$ is nilpotent if and only if $g \in \mathrm{Fit}(G)$. Therefore, $g \in \mathrm{Fit}(G)$ if and only if $\left[_M \langle g^G \rangle\right] = 1$ (M as in Section 2 large enough), which, by Lemma 1, is the case if and only if $\left[_M\, g^G\right]_{\mathrm{set}} = 1$. Hence, the expression $[X^{Y_1}, \dots, X^{Y_M}]$ atomically universally defines $\mathrm{Fit}(G)$.

Now, suppose that $\beta \in (G \cup \mathcal{X}_\beta \cup \mathcal{X}_\beta^{-1})^*$ with $\mathcal{X}_\beta = \{ X, Y_1, \dots, Y_k \}$ atomically universally defines H/N in G/N and that $\alpha \in (G \cup \mathcal{X}_\alpha \cup \mathcal{X}_\alpha^{-1})^*$ with $\mathcal{X}_\alpha = \{ Z, Y_{k+1}, \dots, Y_m \}$ atomically universally defines N in G. Thus, $g \in H$ if and only if $\beta(g, Y_1, \dots, Y_k) \in N$ for all $Y_1, \dots, Y_k \in G$ and $h \in N$ if and only if $\alpha(h, Y_{k+1}, \dots, Y_m) = 1$ for all $Y_{k+1}, \dots, Y_m \in G$. Hence, $\alpha(\beta(g, Y_1, \dots, Y_k), Y_{k+1}, \dots, Y_m) = 1$ for all $Y_1, \dots, Y_m \in G$ if and only if $g \in H$ and so H is atomically universally definable.

The third point follows by induction from the first and second point. The fourth point is essentially due to [20, Lemma 10]: if β is an expression inducing H, then $[X, \beta]$ atomically universally defines $C_G(H)$. ◀

▶ **Lemma 7.** *Let $H \trianglelefteq G$ be an atomically universally definable normal subgroup. Then*

$$\text{EQN-ID}(G/H) \leq_{\mathrm{m}}^{\mathsf{AC}^0} \text{EQN-ID}(G).$$

Proof. Denote $Q = G/H$. Let $\beta \in (G \cup \mathcal{X}_\beta \cup \mathcal{X}_\beta^{-1})^*$ with $\mathcal{X}_\beta = \{ Z, Y_1, \dots, Y_k \}$ atomically universally define H and let $\alpha \in (Q \cup \mathcal{X} \cup \mathcal{X}^{-1})^*$ be an instance for EQN-ID(Q) (with $\mathcal{X} \cap \mathcal{X}_\beta = \emptyset$). Let $\tilde{\alpha}$ denote the expression obtained from α by replacing every constant of Q by an arbitrary preimage in G. Then $\sigma(\alpha) = 1$ in Q for all assignments $\sigma : \mathcal{X} \to Q$ if and only if $\tilde{\sigma}(\tilde{\alpha}) \in H$ for all assignments $\tilde{\sigma} : \mathcal{X} \to G$. By the choice of β, the latter is the case if and only if $\hat{\sigma}(\beta(\tilde{\alpha}, Y_1, \dots, Y_k)) = 1$ for all assignments $\hat{\sigma} : \mathcal{X} \cup \{ Y_1, \dots, Y_k \} \to G$. ◀

3 G-programs and AND-weakness

Let G be a finite group. An n-input G-program of length ℓ with variables (input bits) from $\{ B_1, \dots, B_n \}$ is a sequence

$$P = \langle B_{i_1}, a_1, b_1 \rangle \langle B_{i_2}, a_2, b_2 \rangle \cdots \langle B_{i_\ell}, a_\ell, b_\ell \rangle \in (\{ B_1, \dots, B_n \} \times G \times G)^*.$$

For a mapping $\sigma : \{ B_1, \dots, B_n \} \to \{0, 1\}$ (called an assignment) we define $\sigma(P) \in G$ as the group element $c_1 c_2 \cdots c_\ell$, where $c_j = a_j$ if $B_{i_j} = 0$ and $c_j = b_j$ if $B_{i_j} = 1$ for all $1 \leq j \leq \ell$. We say that an n-input G-program P *computes* a function $f : \{0, 1\}^n \to \{0, 1\}$ if P is over the variables $B_1, \dots, B_n$ and there is some $S \subseteq G$ such that $\sigma(P) \in S$ if and only if $f(\sigma) = 1$.

ProgramSAT is the following problem: given a G-program P with variables $B_1, \dots, B_n$, decide whether there is an assignment $\sigma : \{ B_1, \dots, B_n \} \to G$ such that $\sigma(P) = 1$.

The AND-weakness conjecture. In [6], Barrington, Straubing and Thérien conjectured that, if G is finite and solvable, every G-program computing the n-input AND requires length exponential in n. This is called the AND-*weakness conjecture*.

Unfortunately, the term "exponential" seems to be a source of a possible misunderstanding: while often it means $2^{\Omega(n)}$, in other occasions it is used for $2^{n^{\Omega(1)}}$. Indeed, in [15, 5], the conjecture is restated as its *strong version*: "every G-program over a solvable group G for the n-input AND requires length $2^{\Omega(n)}$." However, already in the earlier paper [4], it is remarked that the n-input AND can be computed by depth-k CC^0 circuits of size $2^{\mathcal{O}(n^{1/(k-1)})}$ for every $k \geq 2$ (a CC^0 circuit is a circuit consisting only of MOD_m gates for some $m \in \mathbb{N}$) – thus, disproving the strong version of the AND-weakness conjecture. For a recent discussion about the topic also referencing the cases where the conjecture actually is proved, we refer to [30].

In this section we provide a more detailed upper bound on the length of G-programs for the AND function in terms of the Fitting length of G. We can view our upper bound as a refined version of the $2^{\mathcal{O}(n^{1/(k-1)})}$ upper bound for depth-k CC^0 circuits. This is because, by [34, Theorem 2.8], for every depth-k CC^0 circuit family there is a fixed group G of Fitting length k (indeed, of derived length k) such that the n-input circuit can be transformed into a G-program of length polynomial in n.

▶ **Proposition 8.** *Let G be a finite solvable group and consider a strictly ascending series $1 = H_0 \lhd H_1 \lhd \cdots \lhd H_m = G$ of normal subgroups where $H_i = \gamma_{k_i}(H_{i+1})$ with $k_i \in \mathbb{N} \cup \{ \infty \}$ for $i \in [1 \mathinner{..} m-1]$ and $k_0 = \infty$. Denote $c = |\{ i \in [1 \mathinner{..} m-1] \mid k_i = \infty \}|$ and $C = \prod_{k_i < \infty}(k_i + 1)$.*

Then the n-input AND *function can be computed by a G-program of length $\mathcal{O}(2^{Dn^{1/c}})$ where $D = \frac{c}{C^{1/c}}$. More precisely, for every $n \in \mathbb{N}$ there is some $1 \neq g \in G$ and a G-program Q_n of length $\mathcal{O}(2^{Dn^{1/c}})$ such that*

$$\sigma(Q_n) = \begin{cases} g & \textit{if } \sigma(B_1) = \cdots = \sigma(B_n) = 1, \\ 1 & \textit{otherwise.} \end{cases}$$

Clearly we have $c \leq d - 1$ if d is the Fitting length of G. The lower Fitting series is the special example of such a series where $H_i = \mathcal{L}_{d-i}G$ and $k_i = \infty$ for all $i \in \{ 0, \ldots, d \}$. Thus, we get the following corollary:

▶ **Corollary 9.** *Let G be a finite solvable group of Fitting length $d \geq 2$. Then the n-input* AND *function can be computed by a G-program of length $2^{\mathcal{O}(n^{1/(d-1)})}$.*

▶ **Example 10.** The symmetric group on four elements S_4 has Fitting length 3 with $S_4 \geq A_4 \geq C_2 \times C_2 \geq 1$ being both the upper and lower Fitting series. Therefore, we obtain a length-$\mathcal{O}(2^{2\sqrt{n}})$ program for the n-input AND by Proposition 8. In particular, the strong version of the AND-weakness conjecture does not hold for the group S_4. Note that according to [6], S_4 is the smallest group for which the $2^{\Omega(n)}$ lower bound from [6] does not apply.

On the other hand, consider the group $G = (C_3 \times C_3) \rtimes D_4$ where D_4 (the dihedral group of order eight) acts faithfully on $C_3 \times C_3$[2]. It has Fitting length two. Moreover, its derived subgroup $G' = (C_3 \times C_3) \rtimes C_2$ still has Fitting length two. Hence, we have a series $H_3 = G$, $H_2 = G' = \gamma_1 G$, $H_1 = \gamma_\infty G' = C_3 \times C_3$, and $H_0 = 1$. Therefore, we get an upper bound of $\mathcal{O}(2^{n/2})$ for the length of a program for the n-input AND.

Proof of Proposition 8. We choose $K = (n/C)^{1/c}$. For simplicity, let us first assume that K is an integer. Moreover, we assume that K is large enough such that $H_i = [_K\, H_{i+1}]$ holds whenever $k_i = \infty$ and that $K \geq k_i + 1$ for all $k_i < \infty$.

We define sets $A_i \subseteq G$ inductively by $A_m = G$ and $A_i = [_K\, A_{i+1}]_{\mathrm{set}}$ if $k_i = \infty$ and $A_i = [_{k_i+1}\, A_{i+1}]_{\mathrm{set}}$ if $k_i < \infty$. By Lemma 1 and induction it follows that $H_i = \langle A_i \rangle$ for all $i \in 0, \ldots, m$. Since $H_1 \neq 1$, we find a non-trivial element $g \in A_1$. We can decompose g recursively. For this, we need some more notation: for $\ell \in [1 \mathinner{..} m]$ consider the set of words

$$V_\ell = \left\{ v = v_1 \cdots v_{\ell-1} \in [1 \mathinner{..} K]^{\ell-1} \;\middle|\; v_i \leq k_i + 1 \text{ for all } i \in [1 \mathinner{..} \ell-1] \right\}.$$

We have $|V_m| = C \cdot K^c = n$, so we can fix a bijection $\kappa \colon V_m \to [1 \mathinner{..} n]$.

Now, we can describe the recursive decomposition of $g = g_\epsilon$:

[2] This group can be found in the GAP small group library under the index [72, 40]. It has been suggested as an example by Barrington (private communication).

- g_v for $v \in V_m$ are arbitrary element from G, and
- $g_v = [g_{v1}, \dots, g_{vK}]$ for $v \in V_\ell$ with $k_\ell = \infty$, and
- $g_v = [g_{v1}, \dots, g_{v(k_\ell+1)}]$ for $v \in V_\ell$ with $k_\ell < \infty$.

For $v \in V_\ell$ we have $|g_v| \leq \sum_{i=1}^{K} 2^{K+1-i} |g_{vi}| \leq 2^{K+1} \max_i |g_{vi}|$ whenever $k_\ell = \infty$ and $|g_v| \leq 2^{k_\ell+2} \max_i |g_{vi}|$ if $k_\ell < \infty$. Therefore, setting $D = \frac{c}{C^{1/c}}$ we obtain by induction

$$|g_\varepsilon| \leq 2^{\sum_{k_\ell<\infty}(k_\ell+2)}(2^{K+1})^c \in \mathcal{O}(2^{Dn^{1/c}}).$$

In order to obtain a G-program for the n-input AND, we define G-programs P_v for $v \in \bigcup_{\ell \leq m} V_\ell$. In the commutators we need also programs for inverses: for a G-program $P = \langle B_{i_1}, a_1, b_1 \rangle \langle B_{i_2}, a_2, b_2 \rangle \cdots \langle B_{i_\ell}, a_\ell, b_\ell \rangle$ we set $P^{-1} = \langle B_{i_\ell}, a_\ell^{-1}, b_\ell^{-1} \rangle \cdots \langle B_{i_1}, a_1^{-1}, b_1^{-1} \rangle$. Clearly $(\sigma(P))^{-1} = \sigma(P^{-1})$ for all assignments σ.

- for $v \in V_m$ we set $P_v = \langle B_{\kappa(v)}, 1, g_v \rangle$,
- for $v \in V_\ell$ with $1 \leq \ell < m$ we set $P_v = [P_{v1}, \dots, P_{vK}]$ if $k_\ell = \infty$, and
- for $v \in V_\ell$ with $1 \leq \ell < m$ we set $P_v = [P_{v1}, \dots, P_{v(k_\ell+1)}]$ if $k_\ell < \infty$.

For $v \in V_\ell$ let $V(v)$ denote the set of those words $w \in V_m$ having v as a prefix. By induction we see that

$$\sigma(P_v) = \begin{cases} g_v & \text{if } \sigma(B_{\kappa(w)}) = 1 \text{ for all } w \in V(v), \\ 1 & \text{otherwise.} \end{cases}$$

This shows the correctness of our construction.

It remains to consider the case that $(n/C)^{1/c}$ is not an integer. Then we set $K = \lceil (n/C)^{1/c} \rceil$. It follows that $|V_{m-1}| = C \cdot K^c \geq n$, so we can fix a bijection $\kappa \colon U \to [1 \,..\, n]$ for some subset $U \subseteq V_{m-1}$. We still have $|g_\varepsilon| \leq 2^{\sum_{k_i<\infty}(k_i+1)}(2^{K+1})^c \in \mathcal{O}(2^{cK}) = \mathcal{O}(2^{Dn^{1/c}})$ with D as above. This concludes the proof of Proposition 8. ◀

▶ Remark 11. In the light of Proposition 8 it is natural to ask for a refined version of the AND-weakness conjecture. A natural candidate would be to conjecture that every G-program for the n-input AND has length $2^{\Omega(n^{1/(d-1)})}$ where d is the Fitting length of G.

However, this also weaker version of the AND-weakness conjecture is wrong! Indeed, in [4, Section 2.4] Barrington, Beigel and Rudich show that the n-input AND can be computed by circuits using only MOD_m gates of depth 3 and size $2^{\mathcal{O}(n^{1/r} \log n)}$ where r is the number of different prime factors of m. Translating the circuit into a G-program yields a group G of Fitting length 3. Since there is no bound on r, we see that there is no lower bound on the exponent δ such that there are G-programs of length $2^{\mathcal{O}(n^\delta)}$ for the n-input AND in groups of Fitting length 3.

In [17] it is shown that the AND function can be computed by probabilistic CC^0 circuits using only a logarithmic number of random bits, which "may be viewed as evidence contrary to the conjecture" [17]. In the light of this, we do not feel confident to judge which form of the AND-weakness conjecture might be true. The following version seems possible.

▶ **Conjecture 1** (AND-weakness [6]). *Let G be finite solvable. Then every G-program for the n-input* AND *has length $2^{n^{\Omega(1)}}$.*

Notice that [5, Theorem 2] (if G is AND-weak, ProgramSAT over G can be decided in quasi-polynomial time) still holds with this version of the AND-weakness conjecture.

4 Reducing C-Coloring to equations

In this section we describe the reduction of C-Coloring to EQN-SAT(G) and EQN-ID(G) in the spirit of [15, 31]. For this, we rely on the fact that G has some normal subgroups meeting some special requirements. In Section 5, we show that all sufficiently complicated finite solvable groups meet the requirements of Theorem 14.

For a normal subgroup $H \trianglelefteq G$ and $g \in G$, we define $\eta_g(H) = \left[H, {}_M\, g^G\right]$. Recall that M is chosen large enough such that $[X, {}_M\, Y] = [X, {}_i\, Y]$ for all $i \geq M$ and all $X, Y \subseteq G$ with $X^G = X$ and $Y^G = Y$. Since H is normal, we have $\eta_g(H) \leq H$ and $\eta_g(H)$ is normal in G.

▶ **Lemma 12.** *Let $H \trianglelefteq G$ be a normal subgroup and $g, h \in G$. Then*

(i) $\eta_g(\eta_g(H)) = \eta_g(H)$, *and*

(ii) $\eta_{gh}(H) \leq \eta_g(H)\eta_h(H)$, *and*

(iii) $\mathrm{FitLen}(\eta_{gh}(H)) \leq \max\{\, \mathrm{FitLen}(\eta_g(H)), \mathrm{FitLen}(\eta_h(H))\,\}$.

Proof. We use the fact that M is chosen such that $[X, {}_M\, Y] = [X, {}_i\, Y]$ for all $i \geq M$ and all $X, Y \subseteq G$ with $X^G = G$ and $Y^G = Y$:

$$\eta_g(H) = \left[H, {}_M\, g^G\right] = \left[H, {}_{2M}\, g^G\right] = \left[\left[H, {}_M\, g^G\right], {}_M\, g^G\right] = \eta_g(\eta_g(H)).$$

The second point follows with the same kind of argument:

$$\begin{aligned} \eta_{gh}(H) &= [H, {}_{2M} (gh)^G] \leq [H, {}_{2M} \left\langle g^G \cup h^G \right\rangle] \\ &= \left\langle [H, {}_{2M} g^G \cup h^G]_{\mathrm{set}} \right\rangle && \text{(by Lemma 1)} \\ &\leq \eta_g(H)\eta_h(H). \end{aligned}$$

The last step is because each of the commutators in $[H, {}_{2M} g^G \cup h^G]_{\mathrm{set}}$ either contains at least M terms from g^G and, thus, is in $\eta_g(H)$ or it contains at least M terms from h^G.

The third point is an immediate consequence of the second point and Lemma 3. ◀

▶ **Lemma 13.** *Suppose that $K \trianglelefteq G$ is a normal subgroup satisfying $\eta_g(K) = K$ for some $g \in G$. Then K is inducible.*

Proof. Because $\eta_g(K) = K$ for some $g \in G$ implies that $K = [K, G]$, it follows from Lemma 4 that K is inducible. ◀

▶ **Theorem 14.** *Let G be a finite solvable group of Fitting length three and assume there are normal subgroups $K \trianglelefteq H \trianglelefteq G$ such that $\mathrm{FitLen}(K) = 2$, $\mathcal{U}_2 G \leq H$, and $|G/H| \geq 3$. Moreover, assume that*

(I) *for all $g \in G \setminus H$ we have $\eta_g(K) = K$,*

(II) *for all $h \in H$ we have $\mathrm{FitLen}(\eta_h(K)) \leq 1$.*

Then EQN-SAT(G) and EQN-ID(G) cannot be decided in deterministic time $2^{o(\log^2 N)}$ under ETH where N is the length of the input expression. In particular, EQN-SAT(G) and EQN-ID(G) are not in P *under ETH.*

Proof outline. The crucial observation for this theorem is the same as for Proposition 8: that, roughly speaking, the n-input AND can be decomposed into the conjunction of $\sqrt{n}$ many $\sqrt{n}$-input ANDs. We use this observation in order to reduce the C-Coloring problem to EQN-SAT. More precisely, given a graph Γ with n vertices and m edges, we construct an expression δ and an element $\tilde{h} \in G$ such that

(A) the length of δ is in $2^{\mathcal{O}(\sqrt{m+n})}$,
(B) δ can be computed in time polynomial in its length,
(C) $\delta = \tilde{h}$ is satisfiable if and only if Γ has a valid C-coloring, and
(D) $\sigma(\delta) = 1$ holds for all assignments σ if and only if Γ does *not* have a valid C-coloring.

For the number of colors we use $C = |G/H|$. Let N denote the input length for EQN-SAT (resp. EQN-ID). A $2^{o(\log^2 N)}$-time algorithm for EQN-SAT (resp. EQN-ID), thus, would imply a $2^{o(n+m)}$-time algorithm for C-Coloring contradicting ETH. Hence, it is enough to show points (A)–(D).

In order to construct the expression δ, we assign a variable X_i to every vertex v_i of Γ. Every assignment σ to the variables X_i will give us a coloring χ_σ of Γ (to be defined later). During the proof, we also introduce some auxiliary variables. The aim is to construct δ in a way that an assignment σ to the variables X_i can be extended to a satisfying assignment for $\delta = \tilde{h}$ if and only if χ_σ is a valid coloring of Γ (see Lemma 17).

We start by grouping the edges into roughly $\sqrt{m}$ batches of $\sqrt{m}$ edges each. For each batch of edges, we construct an expression γ_r (where r is the number of the batch) such that for every assignment σ to the variables X_i we have

- if χ_σ assigns the same color to two endpoints of an edge in the r-th batch, then for every assignment to the auxiliary variables, γ_r evaluates to something in $\mathcal{U}_1 K$,
- otherwise, for every element $h \in K$, there is an assignment to the auxiliary variables such that γ_r evaluates to h.

A more formal statement of this can be found in Lemma 15. The expression δ combines all the γ_r as an iterated commutator such that if one of the γ_r evaluates to something in $\mathcal{U}_1 K$, then δ evaluates to 1, and, otherwise, there is some assignment to the auxiliary variables such that δ evaluates to the fixed element $\tilde{h}$.

Proof. Let $C = |G/H|$. Let us describe how the C-Coloring problem for a given graph $\Gamma = (V, E)$ is reduced to an instance of EQN-SAT (resp. EQN-ID). We denote $V = \{ v_1, \dots, v_n \}$. For every vertex v_i we introduce a variable X_i and we set $\mathcal{X} = \{ X_1, \dots, X_n \}$. By fixing a bijection $|G/H| \to [1 .. C]$, we obtain a correspondence between assignments $\mathcal{X} \to G$ and colorings $V \to [1 .. C]$ (be aware that it is *not* one-to-one). During the construction we will also introduce a set $\mathcal{Y}$ of auxiliary variables. As outlined above, the idea is that an assignment $\mathcal{X} \to G$ represents a valid coloring if and only if there is an assignment to the auxiliary variables under which the equation evaluates to a non-identity element.

For each edge $\{ v_i, v_j \} \in E$, we introduce one edge gadget $X_i X_j^{-1}$ (it does not matter which one is the positive variable). Now, we group these gadgets into R batches of R elements each (if the number of gadgets is not a square, we duplicate some gadgets) – i.e., we choose $R = \lceil \sqrt{m} \rceil$. How the gadgets exactly are grouped together does not matter.

For $r \in [1 .. R]$ and $k \in [1 .. |K|]$ let $\alpha_{r,k}$ be an expression which induces K (i.e., all $\alpha_{r,k}$ are the same expressions but with disjoint sets of variables). Such expressions exist by Lemma 13. Let the variables of $\alpha_{r,k}$ be $Y_{r,k,t}$ for $t \in [1 .. T]$ for some $T \in \mathbb{N}$. Moreover, we introduce more auxiliary variables $Z_{r,k,s,\nu}$ for $r \in [1 .. R]$, $k \in [1 .. |K|]$, $s \in [1 .. R]$, and $\nu \in [1 .. M]$ (recall that M is chosen such that, in particular, $[H_1, {}_M\, H_2] = [H_1, {}_{M+1}\, H_2]$ for arbitrary normal subgroups H_1, H_2 of G) and we set

$$\mathcal{Y}'_r = \left\{ Z_{r,k,s,\nu},\ Y_{r,k,t} \;\middle|\; k \in [1 .. |K|], s \in [1 .. R], \nu \in [1 .. M], t \in [1 .. T] \right\}.$$

Let $\beta_{r,1}, \dots, \beta_{r,R}$ be the gadgets of the r-th batch for some $r \in [1 .. R]$. We define

$$\gamma_r = \prod_{k=1}^{|K|} \left[\alpha_{r,k}, \beta_{r,1}^{Z_{r,k,1,1}}, \dots, \beta_{r,1}^{Z_{r,k,1,M}}, \dots, \beta_{r,R}^{Z_{r,k,R,1}}, \dots, \beta_{r,R}^{Z_{r,k,R,M}} \right]. \tag{1}$$

We do this for every batch of gadgets. The following observation is crucial:

▶ **Lemma 15.** *Let $\sigma\colon \mathcal{X} \to G$ be an assignment and let $r \in [1 \mathinner{..} R]$.*

- *If $\sigma(\beta_{r,s}) \in G \setminus H$ for all s, then $\left\{\, (\sigma \cup \sigma')(\gamma_r) \;\middle|\; \sigma' : \mathcal{Y}'_r \to G \,\right\} = K$,*
- *Otherwise, $\left\{\, (\sigma \cup \sigma')(\gamma_r) \;\middle|\; \sigma' : \mathcal{Y}'_r \to G \,\right\} \leq \mathcal{U}_1 K$.*

Proof. By construction, we have $(\sigma \cup \sigma')(\alpha_{r,k}) \in K$ for all r and k and all assignments σ and σ'. Since K is normal, it follows that $(\sigma \cup \sigma')(\gamma_r) \in K$ for all assignments σ and σ'.

Consider the case that $g_s \coloneqq \sigma(\beta_{r,s}) \in G \setminus H$ for all $s \in [1 \mathinner{..} R]$. By assumption (I), we have $K = \eta_{g_1}(K) = \eta_{g_2}(\eta_{g_1}(K)) = \cdots = \eta_{g_R} \ldots \eta_{g_2}(\eta_{g_1}(K)) \cdots)$. By Lemma 1, it follows that $K = \left\langle [K, {}_M\, g_1^G, \ldots, {}_M\, g_R^G]_{\mathrm{set}} \right\rangle$. Since $1 \in [K, {}_M\, g_1^G, \ldots, {}_M\, g_R^G]_{\mathrm{set}}$ and every element in K can be written as a product of length at most $|K|$ over any generating set, we conclude $K = \left([K, {}_M\, g_1^G, \ldots, {}_M\, g_R^G]_{\mathrm{set}}\right)^{|K|}$. This is exactly the form how γ_r was defined in Equation (1) (recall that $\alpha_{r,s}$ can evaluate to every element of K). Therefore, for each $h \in K$, there is an assignment $\sigma'\colon \mathcal{Y}'_r \to G$ such that $(\sigma \cup \sigma')(\gamma_r) = h$.

On the other hand, let $g_s \coloneqq \sigma(\beta_{r,s}) \in H$ for some s. Then, by assumption (II) we have $\mathrm{FitLen}(\eta_{g_s}(K)) \leq 1$. Since $(\sigma \cup \sigma')(\gamma_r) \in \eta_{g_s}(K)$, we obtain $(\sigma \cup \sigma')(\gamma_r) \in \mathcal{U}_1 K$ by Lemma 3. ◀

Now, for every set of auxiliary variables $\mathcal{Y}'_r$ we introduce M disjoint copies, which we call $\mathcal{Y}_r^{(\mu)}$ for $\mu \in [1 \mathinner{..} M]$. We write $\gamma_r^{(\mu)}$ for the copy of γ_r where the variables of $\mathcal{Y}'_r$ are substituted by the corresponding ones in $\mathcal{Y}_r^{(\mu)}$ (the variables $\mathcal{X}$ are shared over all $\gamma_r^{(\mu)}$). We set

$$\delta = \left[\gamma_1^{(1)}, \ldots, \gamma_1^{(M)}, \ldots, \gamma_R^{(1)}, \ldots, \gamma_R^{(M)}\right].$$

Finally, fix some $\tilde{h} \in K \setminus 1$ with $\tilde{h} \in [{}_{M \cdot R}\, K]_{\mathrm{set}}$ and set $\mathcal{Y} = \bigcup_{r,\mu} \mathcal{Y}_r^{(\mu)}$.

▶ **Lemma 16.** *Let $\sigma\colon \mathcal{X} \to G$ be an assignment. If $\sigma(\beta_{r,s}) \in G \setminus H$ for all r and s, then there is some assignment $\sigma'\colon \mathcal{Y} \to G$ such that $(\sigma \cup \sigma')(\delta) = \tilde{h}$. Otherwise $(\sigma \cup \sigma')(\delta) = 1$ for all $\sigma'\colon \mathcal{Y} \to G$.*

Proof. If $\sigma(\beta_{r,s}) \in G \setminus H$ for all r and s, then by Lemma 15, $\left\{(\sigma \cup \sigma')(\gamma_r^{(\mu)}) \;\middle|\; \sigma' : \mathcal{Y}_r^{(\mu)} \to G\right\} = K$ for all $r \in [1 \mathinner{..} R]$ and $\mu \in [1 \mathinner{..} M]$. Hence, since we chose the auxiliary variables $\mathcal{Y}_r^{(\mu)}$ to be all disjoint, we obtain

$$\tilde{h} \in [{}_{M \cdot R}\, K]_{\mathrm{set}} \subseteq \left\{\, (\sigma \cup \sigma')(\delta) \;\middle|\; \sigma' : \mathcal{Y}_r^{(\mu)} \to G \,\right\}.$$

On the other hand, if $\sigma(\beta_{r,s}) \in H$, then, by Lemma 15, for all $\sigma'\colon \mathcal{Y} \to G$ and all $\mu \in [1 \mathinner{..} M]$ we have $(\sigma \cup \sigma')(\gamma_r^{(\mu)}) \in \mathcal{U}_1 K$. Hence, $(\sigma \cup \sigma')(\delta) \in [{}_M\, \mathcal{U}_1 K] = 1$. ◀

Now we are ready to define our equation as $\delta \tilde{h}^{-1}$ for the reduction of C-Coloring to EQN-SAT(G) and δ for the reduction to EQN-ID(G).

The final step is to show points (A)–(D) from above.

For (A) observe that the length of γ_r is $\mathcal{O}(2^{M \cdot R})$ for all r. Thus, the length of δ is $\mathcal{O}(2^{M \cdot R}) \cdot \mathcal{O}(2^{M \cdot R}) \subseteq 2^{\mathcal{O}(R)} = 2^{\mathcal{O}(\sqrt{m})}$ as desired. Point (B) is straightforward from the construction of δ.

In order to see (C) and (D), we use Lemma 16 to prove another lemma. We fix a bijection $\xi : G/H \to [1 \mathinner{..} C]$. For an assignment $\sigma : \mathcal{X} \to G$, we define a corresponding coloring $\chi_\sigma : V \to [1 \mathinner{..} C]$ by $\chi_\sigma(v_i) = \xi(\sigma(X_i)H)$.

▶ **Lemma 17.** *Let $\sigma : \mathcal{X} \to G$ be an assignment. Then*

- *if χ_σ is valid, then there is an assignment $\sigma' : \mathcal{Y} \to G$ such that $(\sigma \cup \sigma')(\delta) = \tilde{h} \neq 1$,*
- *if χ_σ is* not *valid, then for all assignments $\sigma' : \mathcal{Y} \to G$ we have $(\sigma \cup \sigma')(\delta) = 1$.*

Proof. Let χ_σ be a valid coloring. First, observe that the gadgets all evaluate to some element outside of H under σ. This is because, if there is a gadget $X_i X_j^{-1}$ that means that $\{ v_i, v_j \} \in E$ and so $\chi_\sigma(v_i) \neq \chi_\sigma(v_j)$; hence, $\sigma(X_i) \neq \sigma(X_j)$ in G/H (since ξ is a bijection). Therefore, by Lemma 16, it follows that δ evaluates to $\tilde{h}$ under some proper assignment for $\mathcal{Y}$.

On the other hand, if χ_σ is not a valid coloring, then there is an edge $\{ v_i, v_j \} \in E$ with $\chi_\sigma(v_i) = \chi_\sigma(v_j)$. Then we have $\sigma(X_i)H = \sigma(X_j)H$. Hence, by Lemma 16, we obtain that $(\sigma_\chi \cup \sigma')(\delta) = 1$ in G for every $\sigma' : \mathcal{Y} \to G$. ◀

This concludes the proof of Theorem 14. ◀

5 Consequences

In this section we derive our main result Corollary A. We start again with a lemma.

▶ **Lemma 18.** *For every finite solvable, non-nilpotent group G of Fitting length d, there are proper normal subgroups $K \trianglelefteq H \triangleleft G$ with $\mathrm{FitLen}(K) = d - 1$ and $\mathcal{U}_{d-1}G \leq H$ such that*

- *for all $g \in G \setminus H$ we have $\eta_g(K) = K$,*
- *for all $h \in H$ we have $\mathrm{FitLen}(\eta_h(K)) < \mathrm{FitLen}(K)$.*

The construction for Lemma 18 resembles the ones in Lemmas 5 and 6 of [31]. However, while in [31] a minimal normal subgroup N of a quotient G/K is constructed such that r_g with $r_g(x) = [x, g]$ is an automorphism of N (and N is abelian), in our case this is not enough since we need to apply commutator constructions to our analog of N in the spirit of the divide-and-conquer approach of Proposition 8.

Proof. Let $g_1 \in G \setminus \mathcal{U}_{d-1}G$ where d is the Fitting length of G. We construct a sequence of normal subgroups $K_1, K_2, \ldots$ of G as follows: we set $K_1 = \eta_{g_1}(G)$. By Lemma 2, $K_1 = \gamma_\infty \langle g_1^G \rangle$, so it has Fitting length $d - 1$.

Now, while there is some $g_i \in G$ such that $\eta_{g_i}(K_{i-1}) < K_{i-1}$ and $\mathrm{FitLen}(\eta_{g_i}(K_{i-1})) = \mathrm{FitLen}(K_{i-1})$, we set $K_i = \eta_{g_i}(K_{i-1})$ and continue. Since K_i is a proper subgroup of K_{i-1}, this process eventually terminates. We call the last term K. We claim that K satisfies the statement of Lemma 18. By construction for every $g \in G$ one of the two cases

- $\eta_g(K) = K$ or
- $\mathrm{FitLen}(\eta_g(K)) < \mathrm{FitLen}(K)$

applies. Moreover, since $K = \eta_g(K')$ for some $K' \leq G$ and some $g \in G$, we have $K = \eta_g(K') = \eta_g(\eta_g(K')) = \eta_g(K)$ by Lemma 12 (i). By Lemma 12 (iii), the elements $\{ h \in G \mid \mathrm{FitLen}(\eta_h(K)) < \mathrm{FitLen}(K) \}$ form a subgroup H of G. Clearly H is normal (by the definition of η_h) and $\mathcal{U}_{d-1}G \leq H$ because $\mathrm{FitLen}([K, {}_M\, \mathcal{U}_{d-1}G]) = \mathrm{FitLen}(K) - 1$. Since there is some $g \in G$ with $K = \eta_g(K)$, we have $H \neq G$. ◀

Be aware that K depends on the order the g_i were chosen. Indeed, if G is a direct product of two groups G_1 and G_2 of equal Fitting length, then K will either be contained in G_1 or in G_2 – in which factor depends on the choice of the g_i.

▶ **Theorem 19** (Corollary A). *Let G be a finite solvable group such that either* $\mathrm{FitLen}(G) = 3$ *and $|G/\mathcal{U}_2G|$ has a prime divisor 3 or greater (i.e., $G/\mathcal{U}_2G$ is not a 2-group) or* $\mathrm{FitLen}(G) \geq 4$. *Then EQN-SAT$(G)$ and EQN-ID(G) cannot be decided in deterministic time $2^{o(\log^2 N)}$ under ETH. In particular, EQN-SAT(G) and EQN-ID(G) are not in* P *under ETH.*

Proof. Consider the case that G has Fitting length 3 and $|G/\mathcal{U}_2G|$ has a prime divisor 3 or greater. Let 2^ν for some $\nu \in \mathbb{N}$ be the greatest power of two dividing $|G/\mathcal{U}_2G|$. Then, the subgroup $\widetilde{G}$ generated by all 2^ν-th powers is normal and it is not contained in $\mathcal{U}_2G$. Therefore, by Lemma 3 it has Fitting length 3 as well. Also, by Lemma 3, we know that $\mathcal{U}_2\widetilde{G} = \widetilde{G} \cap \mathcal{U}_2G$. Hence, $\widetilde{G}/\mathcal{U}_2\widetilde{G}$ is a subgroup of $G/\mathcal{U}_2G$. Moreover, since $\widetilde{G}$ is generated by 2^ν-th powers, the generators of $\widetilde{G}$ have odd order in $\widetilde{G}/\mathcal{U}_2\widetilde{G}$. Since $\widetilde{G}/\mathcal{U}_2\widetilde{G}$ is nilpotent, it follows that $|\widetilde{G}/\mathcal{U}_2\widetilde{G}|$ is odd (recall that a nilpotent group is a direct product of p-groups).

Since $\widetilde{G}$ is inducible in G, by Lemma 5, it suffices to show that $\widetilde{G}$ satisfies the requirements of Theorem 14. For this, we use Lemma 18, which gives us normal subgroups $K \trianglelefteq H \triangleleft \widetilde{G}$ with $\mathcal{U}_2\widetilde{G} \leq H$, $\mathrm{FitLen}(K) = 2$ and such that for all $g \in \widetilde{G} \setminus H$ we have $\eta_g(K) = K$, and for all $h \in H$ we have $\mathrm{FitLen}(\eta_h(K)) \leq 1$.

It only remains to show that $|\widetilde{G}/H| \geq 3$. Since $H \neq \widetilde{G}$ and $|\widetilde{G}/H|$ is odd, this holds trivially. Thus, both EQN-SAT(G) and EQN-ID(G) are not in P under ETH if G has Fitting length 3 and $|G/\mathcal{U}_2G|$ a prime divisor 3 or greater.

The second case can be reduced to the first case as follows: Assume that G has Fitting length $d \geq 4$. If $|G/\mathcal{U}_{d-1}G|$ has a prime factor 3 or greater, we can apply the Fitting length 3 case to $G/\mathcal{L}_3G$ for EQN-SAT and to $G/\mathcal{U}_{d-3}G$ for EQN-ID. By Lemma 4 and Lemma 5 this implies the corollary for EQN-SAT. For EQN-ID, the statement follows form Lemma 6 and Lemma 7.

On the other hand, if $|G/\mathcal{U}_{d-1}G| = 2^\nu$ for some $\nu \geq 1$, as in the first case, we consider the subgroup $\widetilde{G}$ generated by all 2^ν-th powers. Then the index of $\widetilde{G}$ in G is again a power of two (since the order of every element in $G/\widetilde{G}$ is a power of two). Moreover, $\widetilde{G} \leq \mathcal{U}_{d-1}G$ and, by Lemma 3, we have

$$\widetilde{G}/\mathcal{U}_{d-2}\widetilde{G} = \widetilde{G}/(\mathcal{U}_{d-2}G \cap \widetilde{G}) \cong (\widetilde{G} \cdot \mathcal{U}_{d-2}G)/\mathcal{U}_{d-2}G \leq \mathcal{U}_{d-1}G/\mathcal{U}_{d-2}G.$$

Now, $|\mathcal{U}_{d-1}G/\mathcal{U}_{d-2}G|$ cannot be a power of two because, otherwise, $G/\mathcal{U}_{d-2}G$ would be a 2-group and, thus, nilpotent – contradicting the fact that the upper Fitting series is a shortest Fitting series. Since the index of $\widetilde{G}$ in $\mathcal{U}_{d-1}G$ is a power of two, we see that $\widetilde{G} \not\subseteq \mathcal{U}_{d-2}G$ and that the index of $\mathcal{U}_{d-2}\widetilde{G}$ in $\widetilde{G}$ has a prime factor other than 2. Therefore, we can apply the Fitting length 3 case to $\widetilde{G}/\mathcal{L}_3\widetilde{G}$ (resp. $\widetilde{G}/\mathcal{U}_{d-3}\widetilde{G}$). ◀

The case that $G/\mathcal{U}_2G$ is a 2-group. As mentioned above, in the recent paper [24] Idziak, Kawałek, and Krzaczkowski proved a $2^{\mathcal{O}(\log^2(n))}$-lower bound under ETH for EQN-SAT(S_4). They apply a reduction of 3SAT to EQN-SAT(S_4). Instead of using commutators to simulate conjunctions in the group, the more complicated logical function $(X, Y_1, Y_2, Y_3) \mapsto X \wedge (Y_1 \vee Y_2 \vee Y_3)$ is encoded into the group. Indeed, under suitable assumptions on the group and the range of the variables, both the expressions $w(X, Y_1, Y_2, Y_3) = X^8[X, Y_1, Y_2, Y_3]$ (see [31]) and $s(X, Y_1, Y_2, Y_3) = X\,[X, Y_1, Y_2, Y_3]^{-1}$ (see [16] – referred to by [24]) simulate this logical function. A new paper unifying our approaches and proving Theorem 19 for *all* groups of Fitting length 3 is under preparation.

Consequences for ProgramSAT. We have EQN-SAT$(G) \leq_{\mathrm{m}}^{\mathsf{AC}^0}$ ProgramSAT(G) for every finite group G by [5, Lem. 1] (while not explicitly stated, it is clear that this reduction is an AC^0-reduction). Thus, by Theorem 14, ProgramSAT(G) is not in P under ETH if G is of Fitting length at least 4 or G is of Fitting length 3 and $G/\mathcal{U}_2G$ is not a 2-group.

Table 1 Groups up to order 767 for which Theorem 19 gives lower bounds.

Index in Small Groups Library	Fitting length	GAP Structure description
[168, 43]	3	(C2 x C2 x C2) : (C7 : C3)
[216, 153]	3	((C3 x C3) : Q8) : C3
[324, 160]	3	((C3 x C3 x C3) : (C2 x C2)) : C3
[336, 210]	3	C2 x ((C2 x C2 x C2) : (C7 : C3))
[432, 734]	4	(((C3 x C3) : Q8) : C3) : C2
[432, 735]	3	C2 x (((C3 x C3) : Q8) : C3)
[504, 52]	3	(C2 x C2 x C2) : (C7 : C9)
[504, 158]	3	C3 x ((C2 x C2 x C2) : (C7 : C3))
[600, 150]	3	(C5 x C5) : SL(2,3)
[648, 531]	3	C3 . (((C3 x C3) : Q8) : C3) = (((C3 x C3) : C3) : Q8) . C3
[648, 532]	3	(((C3 x C3) : C3) : Q8) : C3
[648, 533]	3	(((C3 x C3) : C3) : Q8) : C3
[648, 534]	3	((C3 x C3) : Q8) : C9
[648, 641]	3	((C3 x C3 x C3) : Q8) : C3
[648, 702]	3	C3 x (((C3 x C3) : Q8) : C3)
[648, 703]	4	(((C3 x C3 x C3) : (C2 x C2)) : C3) : C2
[648, 704]	4	(((C3 x C3 x C3) : (C2 x C2)) : C3) : C2
[648, 705]	3	(S3 x S3 x S3) : C3
[648, 706]	3	C2 x (((C3 x C3 x C3) : (C2 x C2)) : C3)
[672, 1049]	3	C4 x ((C2 x C2 x C2) : (C7 : C3))
[672, 1256]	3	C2 x C2 x ((C2 x C2 x C2) : (C7 : C3))
[672, 1257]	3	(C2 x C2 x C2 x C2 x C2) : (C7 : C3)

Small groups for which Theorem 19 gives a lower bound. In [19] lists of groups are given where the complexity of EQN-SAT and EQN-ID is unknown. The paper refers to a more comprehensive list available on the author's website `http://math.unideb.hu/horvath-gabor/research.html`. We downloaded the lists of groups and ran tests in GAP for which of these groups Theorem 19 provides lower bounds. In the list with unknown complexity for EQN-ID there are 2331 groups of order less than 768 out of which 1559 are of Fitting length three or greater. Theorem 19 applies to 22 of them: 3 groups of Fitting length 4 and 19 groups G of Fitting length 2 where $G/\mathcal{U}_2G$ is not a 2-group. A list of the groups for which we could prove lower bounds can be found in Table 1.

5.1 Equations in finite semigroups

For a semigroup S, the problems EQN-SAT(S) and EQN-ID(S) both receive two expressions as input. The questions is whether the two expressions evaluate to the same element under some (resp. all) assignments. For semigroups R, S we say that R *divides* S if R is a quotient of a subsemigroup of S. The following lemmas are straightforward to prove using basic semigroup theory.

For the proofs, we need Green's relations $\mathcal{H}$ and $\mathcal{J}$. For a definition, we refer to [35, Appendix A]. For a semigroup S we write S^1 for S with an identity adjoined if there is none.

▶ **Lemma 20.** *If G is a maximal subgroup of a finite semigroup S, then EQN-SAT$(G) \le_{\mathrm{m}}^{\mathsf{AC}^0}$ EQN-SAT(S).*

Proof. Let $e \in G$ denote the identity of G. Clearly, $G = eGe \le eSe$ and eSe is a submonoid of S with identity e. The reduction simply replaces every variable X by eXe (and likewise for constants). Let $\tilde{\alpha}$ denote the equation we obtain from an input equation α this way. Now the question is whether $\tilde{\alpha} = e$ in S. Clearly, if α has a solution in G, the resulting equation $\tilde{\alpha}$ has a solution in S. On the other hand, if $\tilde{\alpha}$ has a solution in S, we obtain a solution of $\alpha = e$ in S where every variable takes values in eSe.

Assume we have $\sigma(X) = x \notin G$ for a satisfying assignment σ and some variable X of α. Since $\sigma(\alpha) = e$, we have that e is in the two-sided ideal $S^1 x S^1$ generated by $x = exe$. By point 2. of [35, Exercise A.2.2] it follows that $x \in H_e = G$ where H_e denotes the $\mathcal{H}$-class of e under Green's relations (for a definition, we refer to [35]) and G agrees with H_e because G is a maximal subgroup. ◀

▶ **Lemma 21.** *If a group G divides a semigroup S, then G divides already one of the maximal subgroups (i.e., regular $\mathcal{H}$-classes) of S.*

Proof. Let $U \leq S$ a subsemigroup and $\varphi : U \to G$ a surjective semigroup homomorphism. Pick some arbitrary element $s \in U$ and let $e = s^{\omega}$ be the idempotent generated by s. Clearly, we have $\varphi(e) = 1$. Now, the subsemigroup $eUe \leq U$ still maps surjectively onto G under φ: by assumption for every $g \in G$ there is some $u_g \in U$ with $\varphi(u_g) = g$; hence, $g = 1g1 = \varphi(e)\varphi(u_g)\varphi(e) \in \varphi(eUe)$.

If eUe is not contained in a maximal subgroup, then by point 2. of [35, Exercise A.2.2], there is some $t \in eUe$ which is not $\mathcal{J}$-equivalent to e. Now, we can repeat the above process starting with t. This will decrease the size of U, so it eventually terminates. ◀

▶ **Corollary 22.** *Let S be a finite semigroup and G a group dividing S. If $\mathrm{FitLen}(G) \geq 4$ or $\mathrm{FitLen}(G) = 3$ and $G/\mathcal{U}_2 G$ is not a 2-group, then EQN-SAT(S) is not in P under ETH.*

Proof. If G with $\mathrm{FitLen}(G) \geq 4$ or $\mathrm{FitLen}(G) = 3$ and $G/\mathcal{U}_2 G$ divides S, then it follows from Lemma 21 that there is a group $\widetilde{G}$ with the same properties and which is a maximal subgroup of S. Hence, the statement follows from Lemma 20. ◀

[2, Theorem 1] states that identity checking over $\widetilde{G}$ reduces to identity checking over S where $\widetilde{G}$ is the direct product of all maximal subgroups of S. However, be aware that in this context the identity checking problem does not allow constants. Since the proof of Theorem 14 essentially relies on the fact that the subgroup K is inducible and this can be only shown using constants, this does not allow us to show hardness of EQN-ID(S).

6 Conclusion

We have shown that assuming the exponential time hypothesis there are solvable groups with equation satisfiability problem not decidable in polynomial time. Thus, under standard assumptions from complexity theory this means a negative answer to [9, Problem 1] (also conjectured in [18]). Theorem 19 yields a quasipolynomial time lower bound under ETH. Thus, a natural weakening of [9, Problem 1] is as follows:

▶ **Conjecture 2.** *If G is a finite solvable group, then EQN-SAT(G) and EQN-ID(G) are decidable in quasipolynomial time.*

In [5, Theorem 2] it is proved that ProgramSAT(G) and, hence, also EQN-SAT(G) can be decided in quasipolynomial time given that G is AND-weak. As remarked in Section 3 this theorem remains valid with our slightly less restrictive definition of AND-weakness in Conjecture 1. Thus, Conjecture 1 implies Conjecture 2. In particular, under the assumption of both ETH and the AND-weakness conjecture (Conjecture 1), for every finite solvable group G meeting the requirements of Theorem 19 there are quasipolynomial upper and lower bounds for EQN-SAT(G) and EQN-ID(G) – so under these assumptions both problems are neither in P nor NP-complete. This contrasts the situation for solving systems of equations: there is a clear P versus NP-complete dichotomy [15].

Theorem 19 proves lower bounds on EQN-SAT and EQN-ID for all sufficiently complicated finite solvable groups. As outlined above, together with the authors of [24] the extension to *all* groups of Fitting length three is under preparation. As a refinement we plan to show that under ETH there is no $2^{o(n^{1/(d-1)})}$-time algorithm for EQN-SAT(G) and EQN-ID(G) where d is the Fitting length of G. Possible further research might address the complexity of EQN-SAT and EQN-ID in groups of Fitting length two. The results presented in the introduction suggest that these cases can be solved in polynomial time.

▶ **Conjecture 3.** *If G is a finite solvable group of Fitting length two, then EQN-SAT(G) and EQN-ID(G) are decidable in polynomial time.*

Another direction for future work is the complexity of EQN-ID for expressions without constants.

References

1 Scott Aaronson, Russell Impagliazzo, and Dana Moshkovitz. AM with multiple merlins. In *IEEE 29th Conference on Computational Complexity, CCC 2014, Vancouver, BC, Canada, June 11-13, 2014*, pages 44–55. IEEE Computer Society, 2014. doi:10.1109/CCC.2014.13.

2 Jorge Almeida, M. V. Volkov, and S. V. Goldberg. Complexity of the identity checking problem for finite semigroups. *Journal of Mathematical Sciences*, 158(5):605–614, 2009. doi:10.1007/s10958-009-9397-z.

3 David A. Mix Barrington. Width-3 permutation branching programs. Technical Report TM-293, MIT Laboratory for Computer Science, 1985.

4 David A. Mix Barrington, Richard Beigel, and Steven Rudich. Representing Boolean functions as polynomials modulo composite numbers. *Computational Complexity*, 4:367–382, 1994. doi:10.1007/BF01263424.

5 David A. Mix Barrington, Pierre McKenzie, Cristopher Moore, Pascal Tesson, and Denis Thérien. Equation satisfiability and program satisfiability for finite monoids. In *Mathematical Foundations of Computer Science 2000, 25th International Symposium, MFCS 2000, Proceedings*, volume 1893 of *Lecture Notes in Computer Science*, pages 172–181. Springer, 2000. doi:10.1007/3-540-44612-5_13.

6 David A. Mix Barrington, Howard Straubing, and Denis Thérien. Non-uniform automata over groups. *Inf. Comput.*, 89(2):109–132, 1990. doi:10.1016/0890-5401(90)90007-5.

7 Mark Braverman, Young Kun-Ko, Aviad Rubinstein, and Omri Weinstein. ETH hardness for densest-k-subgraph with perfect completeness. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 1326–1341. SIAM, 2017. doi:10.1137/1.9781611974782.86.

8 Mark Braverman, Young Kun-Ko, and Omri Weinstein. Approximating the best nash equilibrium in $n^{o(\log n)}$-time breaks the exponential time hypothesis. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 970–982. SIAM, 2015. doi:10.1137/1.9781611973730.66.

9 Stanley Burris and J. Lawrence. Results on the equivalence problem for finite groups. *Algebra Universalis*, 52(4):495–500 (2005), 2004. doi:10.1007/s00012-004-1895-8.

10 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

11 Volker Diekert and Murray Elder. Solutions of twisted word equations, EDT0L languages, and context-free groups. In *ICALP 2017, Proceedings*, volume 80 of *LIPIcs*, pages 96:1–96:14, Dagstuhl, Germany, 2017. doi:10.4230/LIPIcs.ICALP.2017.96.

12 Attila Földvári. The complexity of the equation solvability problem over semipattern groups. *IJAC*, 27(2):259, 2017. doi:10.1142/S0218196717500126.

13 Attila Földvári and Gábor Horváth. The complexity of the equation solvability and equivalence problems over finite groups. *International Journal of Algebra and Computation*, 30(03):607–623, 2020. doi:10.1142/S0218196720500137.

14 Albert Garreta, Alexei Miasnikov, and Denis Ovchinnikov. Diophantine problems in solvable groups. *Bulletin of Mathematical Sciences*, January 2020. doi:10.1142/S1664360720500058.

15 Mikael Goldmann and Alexander Russell. The complexity of solving equations over finite groups. *Inf. Comput.*, 178(1):253–262, 2002. doi:10.1006/inco.2002.3173.

16 Tomasz A. Gorazd and Jacek Krzaczkowski. Term equation satisfiability over finite algebras. *IJAC*, 20(8):1001–1020, 2010. doi:10.1142/S021819671000600X.

17 Kristoffer Arnsfelt Hansen and Michal Koucký. A new characterization of ACC^0 and probabilistic CC^0. *Computational Complexity*, 19(2):211–234, 2010. doi:10.1007/s00037-010-0287-z.

18 Gábor Horváth. The complexity of the equivalence and equation solvability problems over nilpotent rings and groups. *Algebra Universalis*, 66(4):391–403, 2011. doi:10.1007/s00012-011-0163-y.

19 Gábor Horváth. The complexity of the equivalence and equation solvability problems over meta-Abelian groups. *J. Algebra*, 433:208–230, 2015. doi:10.1016/j.jalgebra.2015.03.015.

20 Gábor Horváth and Csaba Szabó. The extended equivalence and equation solvability problems for groups. *Discrete Math. Theor. Comput. Sci.*, 13(4):23–32, 2011.

21 Gábor Horváth and Csaba Szabó. Equivalence and equation solvability problems for the alternating group $\mathbf{A}_4$. *J. Pure Appl. Algebra*, 216(10):2170–2176, 2012. doi:10.1016/j.jpaa.2012.02.007.

22 Gábor Horváth and Csaba A. Szabó. The complexity of checking identities over finite groups. *IJAC*, 16(5):931–940, 2006. doi:10.1142/S0218196706003256.

23 B. Huppert. *Endliche Gruppen. I.* Die Grundlehren der Mathematischen Wissenschaften, Band 134. Springer-Verlag, Berlin-New York, 1967.

24 Pawel M. Idziak, Piotr Kawalek, and Jacek Krzaczkowski. Intermediate problems in modular circuits satisfiability. In *LICS 2020, Proceedings.* ACM, 2020. Preprint at arXiv:2002.08626. doi:10.1145/3373718.3394780.

25 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.

26 Marcel Jackson and Ralph McKenzie. Interpreting graph colorability in finite semigroups. *IJAC*, 16(1):119–140, 2006. doi:10.1142/S0218196706002846.

27 Andrzej Kisielewicz. Complexity of semigroup identity checking. *IJAC*, 14(4):455–464, 2004. doi:10.1142/S0218196704001840.

28 Ondrej Klíma, Pascal Tesson, and Denis Thérien. Dichotomies in the complexity of solving systems of equations over finite semigroups. *Theory Comput. Syst.*, 40(3):263–297, 2007. doi:10.1007/s00224-005-1279-2.

29 Ondřej Klíma. Complexity issues of checking identities in finite monoids. *Semigroup Forum*, 79(3):435–444, 2009. doi:10.1007/s00233-009-9180-y.

30 Michael Kompatscher. CC-circuits and the expressive power of nilpotent algebras. *CoRR*, abs/1911.01479, 2019. arXiv:1911.01479.

31 Michael Kompatscher. Notes on extended equation solvability and identity checking for groups. *Acta Math. Hungar.*, 159(1):246–256, 2019. doi:10.1007/s10474-019-00924-7.

32 Markus Lohrey and Géraud Sénizergues. Theories of HNN-extensions and amalgamated products. In *ICALP 2006, Proceedings*, pages 504–515, 2006. doi:10.1007/11787006_43.

33 Gennadií Semyonovich Makanin. The problem of solvability of equations in a free semigroup. *Math. Sbornik*, 103:147–236, 1977. English transl. in Math. USSR Sbornik 32 (1977).

34 Pierre McKenzie, Pierre Péladeau, and Denis Thérien. NC^1: The automata-theoretic viewpoint. *Computational Complexity*, 1:330–359, 1991. doi:10.1007/BF01212963.

35 John L. Rhodes and Benjamin Steinberg. *The* q*-theory of finite semigroups*. Springer Monographs in Mathematics. Springer, 2009.

36 Derek J. S. Robinson. *A course in the theory of groups*, volume 80 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, second edition, 1996. doi:10.1007/978-1-4419-8594-1.

37 Vitaly Roman'kov. Equations in free metabelian groups. *Siberian Mathematical Journal*, 20, May 1979. doi:10.1007/BF00969959.

38 Steve Seif. The Perkins semigroup has co-NP-complete term-equivalence problem. *Internat. J. Algebra Comput.*, 15(2):317–326, 2005. doi:10.1142/S0218196705002293.

39 Steve Seif and Csaba Szabó. Computational complexity of checking identities in 0-simple semigroups and matrix semigroups over finite fields. *Semigroup Forum*, 72(2):207–222, 2006. doi:10.1007/s00233-005-0510-4.

40 Csaba Szabó and Vera Vértesi. The complexity of checking identities for finite matrix rings. *Algebra Universalis*, 51(4):439–445, 2004. doi:10.1007/s00012-004-1873-1.

41 Yanior Weg. Normal subgroup of Fitting length i contained in i-th term of upper Fitting series? (answer). MathOverflow. URL: https://mathoverflow.net/questions/350552/ (visited on: 2020-04-24).

Graph Isomorphism in Quasipolynomial Time Parameterized by Treewidth

Daniel Wiebking
RWTH Aachen University, Germany
wiebking@informatik.rwth-aachen.de

Abstract

We extend Babai's quasipolynomial-time graph isomorphism test (STOC 2016) and develop a quasipolynomial-time algorithm for the multiple-coset isomorphism problem. The algorithm for the multiple-coset isomorphism problem allows to exploit graph decompositions of the given input graphs within Babai's group-theoretic framework.

We use it to develop a graph isomorphism test that runs in time $n^{\operatorname{polylog}(k)}$ where n is the number of vertices and k is the minimum treewidth of the given graphs and $\operatorname{polylog}(k)$ is some polynomial in $\log(k)$. Our result generalizes Babai's quasipolynomial-time graph isomorphism test.

2012 ACM Subject Classification Mathematics of computing → Combinatorial algorithms; Mathematics of computing → Hypergraphs; Mathematics of computing → Graph algorithms; Theory of computation → Graph algorithms analysis

Keywords and phrases Graph isomorphism, canonization, treewidth, hypergraphs

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.103

Category Track A: Algorithms, Complexity and Games

Related Version A full version of the paper is available at `https://arxiv.org/abs/1911.11257`.

1 Introduction

The graph isomorphism problem asks for a structure preserving bijection between two given graphs G and H, i.e., a bijection $\varphi : V(G) \to V(H)$ such that $vw \in E(G)$ if and only if $\varphi(v)\varphi(w) \in E(H)$. One central open problem in theoretical computer science is the question whether the graph isomorphism problem can be solved in polynomial time. There are a few evidences that the problem might not be NP-hard. For example, NP-hardness of the problem implies a collapse of the polynomial hierarchy [32]. Moreover, NP-hardness of the graph isomorphism problem would refute the exponential time hypothesis since the problem can be decided in quasipolynomial time [1].

The research of the graph isomorphism problem started with two fundamental graph classes, i.e., the class of trees and the class of planar graphs. In 1970, Zemlyachenko gave a polynomial-time isomorphism algorithm for trees [37]. One year later, Hopcroft and Tarjan extended a result of Weinberg and designed a polynomial-time isomorphism algorithm for planar graphs [16],[34]. In 1980, Filotti, Mayer and Miller extended the polynomial-time algorithm to graphs of bounded genus [24],[10][1]. The genus is a graph parameter that measures how far away the graph is from being planar.

In Luks's pioneering work in 1982, he gave a polynomial-time isomorphism algorithm for graphs of bounded degree [22]. His group-theoretic approach laid the foundation of many other algorithms that were developed ever since. It turns out that the research in the

[1] Myrvold and Kocay pointed out an error in Filotti's techniques [26]. However, different algorithms have been given which show that the graph isomorphism problem for graphs of bounded genus is indeed decidable in polynomial time [25, 11, 17].

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 103; pp. 103:1–103:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

graph isomorphism problem for restricted graph classes was a promising approach in tackling the graph isomorphism problem in general. Shortly after Luks's result, a combinatorial partitioning lemma by Zemlyachenko was combined with Luks's framework. This resulted in an isomorphism algorithm for graphs with n vertices in general that runs in time $2^{\mathcal{O}(\sqrt{n \log n})}$ [38],[4]. This algorithm was the fastest for decades.

In 1983, the seminal work of Robertson and Seymour in graph minors started a new era of graph theory [30]. At the same time, Miller extended Luks's group-theoretic framework to hypergraphs [25]. It turned out that the study of general structures such as hypergraphs was also a promising approach in tackling the graph isomorphism problem. In 1991, Ponomarenko could in fact use Miller's hypergraph algorithm to design a polynomial-time isomorphism algorithm for graphs excluding a minor [29].

The work of Robertson and Seymour also rediscovered the notion of treewidth [8], a graph parameter that measures how far away the graph is from being a tree. The treewidth parameter was reborn and has been studied ever since. So, researchers went back to the roots and studied the isomorphism problem for graphs of bounded treewidth. In 1990, Bodlaender gave a simple isomorphism-algorithm for graphs of treewidth k with n vertices that runs in time $n^{\mathcal{O}(k)}$ [5]. However, no FPT-algorithm was known, i.e., an isomorphism algorithm with a running time of the form $f(k) \cdot n^{\mathcal{O}(1)}$. The search of a FPT-algorithm occupied researchers over years and this open problem was explicitly stated by several authors [36, 6, 18, 19, 28, 7, 9, 12]. In 2017, Lokshtanov, Pilipczuk, Pilipczuk and Saurabh finally solved this open problem and designed a FPT-algorithm for the graph isomorphism problem [21]. Their algorithm runs in time $2^{\mathcal{O}(k^5 \log k)} n^{\mathcal{O}(1)}$ where n is the number of vertices and k is the minimum treewidth of the given graphs.

At the same time, Babai made a breakthrough and designed a quasipolynomial-time algorithm for the graph isomorphism problem in general [1]. His algorithm runs in time $n^{\mathrm{polylog}(n)}$ where n is the number of vertices and $\mathrm{polylog}(n)$ is some polynomial in $\log(n)$ (according to Helfgott's analysis the function $\mathrm{polylog}(n)$ can chosen to be quadratic in $\log(n)$ [15]). To achieve this result, Babai built on Luks's group-theoretic framework, which actually solves the more general string isomorphism problem. One of the main questions is how to combine Babai's group-theoretic algorithm with the graph-theoretic techniques that have been developed. For example, it is unclear how to exploit a decomposition of the given graphs within Babai's framework since his algorithm actually processes strings rather than graphs.

Recently, Grohe, Neuen and Schweitzer were able to extend Babai's algorithm to graphs of maximum degree d and an isomorphism algorithm was developed that runs in time $n^{\mathrm{polylog}(d)}$ [13]. They suggest that their techniques might be useful also for graphs parameterized by treewidth and conjectured that the isomorphism problem for graphs of treewidth k can be decided in time $n^{\mathrm{polylog}(k)}$.

In [14], the graph-theoretic FPT-algorithm of Lokshtanov et al. was improved by using Babai's group-theoretic algorithm and the extension given by Grohe et al. as a black box. They decomposed a graph of bounded treewidth into subgraphs with particular properties. They were able to design a faster algorithm that computes the isomorphisms between these subgraphs. However, they pointed out a central problem that arises when dealing with graph decompositions: When the isomorphisms between these subgraphs are already computed, how can they be efficiently merged in order to compute the isomorphisms between the entire graphs? This problem was named as *multiple-coset isomorphism problem* and is formally defined as follows. Given two sets $J = \{\rho_1 \Delta_1^{\mathrm{Can}}, \ldots, \rho_t \Delta_t^{\mathrm{Can}}\}$ and $J' = \{\rho'_1 \Delta'^{\mathrm{Can}}_1, \ldots, \rho'_t \Delta'^{\mathrm{Can}}_t\}$ where $\rho_i : V \to n, \rho'_i : V' \to n$ are bijections and $\Delta_i^{\mathrm{Can}}, \Delta'^{\mathrm{Can}}_i \leq \mathrm{Sym}([n])$ are permutation groups for all $i \in [t]$, the problem is to decide whether there are bijections $\varphi : V \to V', \psi : [t] \to [t]$ such that $\Delta_i^{\mathrm{Can}} = \Delta'^{\mathrm{Can}}_{\psi(i)}$ and $\varphi \in \rho_i \Delta_i^{\mathrm{Can}}(\rho'_{\psi(i)})^{-1}$ for all $i \in [t]$. By applying the group-theoretic black box algorithms, they achieved an improved isomorphism test for graphs of treewidth k that runs in time $2^{k \cdot \mathrm{polylog}(k)} n^{\mathcal{O}(1)}$. However, for further improvements, it did not seem to be enough to use the group-theoretic algorithms as a black box only. The question of an isomorphism algorithm that runs in time $n^{\mathrm{polylog}(k)}$ remained open.

In [31], the study of the multiple-coset isomorphism problem continued. Rather than using group-theoretic algorithms as a black box, they were able to extend Luks's group-theoretic framework to the multiple-coset isomorphism problem. In order to facilitate their recursion, they introduced the class of combinatorial objects. Their class of combinatorial objects contains hypergraphs, colored graphs, relational structures, explicitly given codes and more. However, the key idea in order to handle the involved structures recursively, was to add so-called labeling cosets to their structures. By doing so, they could combine combinatorial decomposition techniques with Luks's group-theoretic framework. This led to a simply-exponential time algorithm for the multiple-coset isomorphism problem. Although the achieved running time was far away from being quasipolynomial, their result led to improvements of several algorithms. For example, it led to the currently best algorithm for the normalizer problem (a central problem in computational group theory) [35]. However, they were not able to extend also Babai's techniques to their framework and the question of a graph isomorphism algorithm running in time $n^{\mathrm{polylog}(k)}$ remained open.

Our Contribution. In this paper, we give a quasipolynomial-time algorithm for the multiple-coset isomorphism problem. This leads to an answer of the conjecture in [13] mentioned above.

▶ **Theorem** (Theorem 10). *The graph isomorphism problem can be decided in time $n^{\mathrm{polylog}(k)}$ where n is the number of vertices and k is the minimum treewidth of the input graphs.*

When $k = \mathrm{polylog}(n)$, our algorithm runs in time $n^{\mathcal{O}(\log(\log n)^c)}$ (for some constant c) and is significantly faster than Babai's algorithm and existing FPT-algorithms for graphs parameterized by treewidth.

For the present work, we exploit the fact that Babai's algorithm was recently extended to canonization [3]. A canonical labeling of a graph is a function that labels the vertices V of the graph with integers $1, \dots, |V|$ in such a way that the labeled versions of two isomorphic graphs are equal (rather than isomorphic). The computation of canonical forms and labelings, rather than isomorphism testing, is an important task in the area of graph isomorphism and is especially useful for practical applications. Also the framework given in [31] is actually designed for the canonization problem. The present paper is based on these works and our algorithms provide canonical labelings as well. Only the algorithm given in the last section depends on the bounded-degree isomorphism algorithm of Grohe et al. for which no adequate canonization version is known.

The first necessary algorithm that we provide in our work is a simple canonization algorithm for hypergraphs.

▶ **Theorem** (Theorem 6). *Canonical labelings for hypergraphs (V, H) can be computed in time $(|V| + |H|)^{\mathrm{polylog}|V|}$.*

There is a simple argument why this algorithm is indeed necessary for our main result. It is well-known that a hypergraph $X = (V, H)$ can be encoded as a bipartite graph $G_X = (V \cup H, E)$ (the bipartite graph G_X has an edge $(v, S) \in E$, if and only if $v \in S$). It is not hard to show that the treewidth k of this bipartite graph G_X is at most $|V|$. The bipartite graph G_X uniquely encodes the hypergraph X, in particular, two hypergraphs are isomorphic if and only if their corresponding bipartite graphs are isomorphic. This means that an isomorphism algorithm for graphs of treewidth k running in time $n^{\mathrm{polylog}(k)}$ would imply an isomorphism algorithm for hypergraphs running in time $(|V|+|H|)^{\mathrm{polylog}|V|}$. However, applying Babai's algorithm to the bipartite graph would lead to a running time of $(|V|+|H|)^{\mathrm{polylog}(|V|+|H|)}$. Instead of applying Babai's algorithm to the bipartite graph directly, we decompose the hypergraph and canonize the substructures recursively. To merge the canonical labelings of all subhypergraphs, we use a canonical version of the multiple-coset isomorphism problem. However, for the hypergraph algorithm, it suffices to use Babai's algorithm as a black box only.

Our decomposition technique for hypergraphs can also be used to design a simple canonization algorithm for k-ary relations.

▶ **Theorem** (Theorem 5). *Canonical labelings for k-ary relations $R \subseteq V^k$ can be computed in time $2^{\mathrm{polylog}|V|}|R|^{\mathcal{O}(1)}$.*

The algorithm improves the currently best algorithm from [13]. As graphs can be seen as binary relations, our algorithm generalizes the quasipolynomial-time bound for graphs. The achieved running time is the best one can hope for as long as the graph isomorphism problem has no solution better than quasipolynomial time.

Our main algorithm finally solves the multiple-coset isomorphism problem. In fact, the algorithm computes canonical labelings as well.

▶ **Theorem** (Theorem 7). *Canonical labelings for a set J consisting of labeling cosets can be computed in time $(|V|+|J|)^{\mathrm{polylog}|V|}$.*

This result is actually of independent interest as it also implies a faster canonization algorithm for the entire class of combinatorial objects.

To solve this problem, the simple hypergraph canonization algorithm can be used as a subroutine in some places. However, we do not longer use Babai's and Luks's techniques as a black box only. To extend their methods, we follow the route of [31] and consider combinatorial objects that allows to combine combinatorial structures with permutation group theory. In particular, we can extend Luks's subgroup reduction and Babai's method and aggregation of local certificates to our framework. All these methods were designed for the string isomorphism problem and need non-trivial extensions when dealing with a set of labeling cosets rather than a string.

Related Work. Another extension of Babai's quasipolynomial time algorithm has been independently proposed by Daniel Neuen [27] who provided another algorithm for the isomorphism problem of hypergraphs. However, Neuen can exploit groups with restricted composition factors that are given as additional input in order to speed up his algorithm. This can be exploited in the setting of graphs of bounded Euler genus. He provides a graph isomorphism algorithm that runs in time $n^{\mathrm{polylog}(g)}$ where n is the number of vertices and g is the minimum genus of the given graphs.

On the other hand, his algorithm is not able to handle labeling cosets occurring in the combinatorial structures. In particular, his algorithm is not able to solve the multiple-coset isomorphism problem in the desired time bound, which we require for our isomorphism algorithm for graphs parameterized by treewidth. Moreover, his techniques do not provide canonical labelings.

We hope that both algorithms can be combined to give a faster isomorphism test for the large class of graphs excluding a topological subgraph. This large class of graphs includes the graphs of bounded treewidth, graphs of bounded genus, graphs of bounded degree and graphs excluding a minor. In fact, Grohe and Marx provide a structure theorem which shows that the graph classes mentioned above also characterize graphs excluding a topological subgraph. Informally, they showed that graphs excluding a topological subgraph can be decomposed into almost bounded-degree parts and minor-free parts which in turn can be decomposed into almost-embeddable parts [12]. Therefore, we hope that the improved algorithms for the isomorphism problem for bounded-degree graphs and bounded-genus graphs can be combined with our algorithm to exploit the occurring graph decomposition.

Organization of the Paper. In Section 3, we show how (sufficiently small) instances of the multiple-coset isomorphism and canonization problem can be processed with Babai's algorithm. In Section 4, we present a partitioning technique to reduce the canonization problem for k-ary relations to instances of small size in each decomposition level. In Section 5, we extend our technique to canonization of hypergraphs, which is an important subroutine used in the next section. In Section 6, we finally present our main algorithm which canonizes a set of labeling cosets and is divided into five subroutines. In the first subroutine, we extend the partitioning technique to families of partitions. The second and third subroutine extends Luks's subgroup reduction to our framework and reduces the problem to the barrier configuration characterized by a giant representation. The fourth and fifth subroutine extend Babai's method and aggregation of local certificates to our framework. In Section 7, a straightforward application of the multiple-coset isomorphism problem leads to an isomorphism algorithm that runs in time $n^{\mathrm{polylog}(k)}$ where n is the number of vertices and k is the treewidth of the given graphs.

2 Preliminaries

For an integer t, we write $[t]$ for $\{1,\dots,t\}$. For a set S and an integer k, we write $\binom{S}{k}$ for the k-element subsets of S and 2^S for the power set of S. The composition of two functions $f : V \to U$ and $g : U \to W$ is denoted by fg and is defined as the function that first applies f and then applies g.

Labeling Cosets. A *labeling* of a set V is a bijection $\rho : V \to \{1,\dots,|V|\}$. A *labeling coset* of a set V is a set of bijections Λ such that $\Lambda = \Delta\rho = \{\delta\rho \mid \delta \in \Delta\}$ for some subgroup $\Delta \le \mathrm{Sym}(V)$ and some labeling $\rho : V \to \{1,\dots,|V|\}$. We write $\mathrm{Label}(V)$ to denote the labeling coset $\mathrm{Sym}(V)\rho = \{\sigma\rho \mid \sigma \in \mathrm{Sym}(V)\}$ where ρ is an arbitrary labeling of V. Analogous to subgroups, a set $\Theta\tau$ is called a *labeling subcoset* of $\Delta\rho$, written $\Theta\tau \le \Delta\rho$, if the labeling coset $\Theta\tau$ is a subset of $\Delta\rho$.

Generating Sets. For the basic theory of handling permutation groups given by generating sets, we refer to [33]. Indeed, most algorithms are based on strong generating sets. However, given an arbitrary generating set, the Schreier-Sims algorithm is used to compute a strong generating set (of size quadratic in the degree) in polynomial time.

Hereditarily Finite Sets and Combinatorial Objects. Inductively, we define *hereditarily finite sets*, denoted by $\mathrm{HFS}(V)$, over a ground set V.

- A vertex $v \in V$ is an atom and a hereditarily finite set $v \in \mathrm{HFS}(V)$,
- a labeling coset $\Delta\rho \le \mathrm{Label}(V)$ is an atom and a hereditarily finite set $\Delta\rho \in \mathrm{HFS}(V)$,
- if $X_1,\dots,X_t \in \mathrm{HFS}(V)$, then also $\mathcal{X} = \{X_1,\dots,X_t\} \in \mathrm{HFS}(V)$ where $t \in \mathbb{N} \cup \{0\}$, and
- if $X_1,\dots,X_t \in \mathrm{HFS}(V)$, then also $\mathcal{X} = (X_1,\dots,X_t) \in \mathrm{HFS}(V)$ where $t \in \mathbb{N} \cup \{0\}$.

A *(combinatorial) object* is a pair $(V,\mathcal{X})$ consisting of a ground set V and a hereditarily finite set $\mathcal{X}\in\mathrm{HFS}(V)$. The ground set V is usually apparent from context and the combinatorial object $(V,\mathcal{X})$ is identified with the hereditarily finite set $\mathcal{X}$. The set $\mathrm{Obj}(V)$ denotes the set of all objects over V. The *transitive closure* of an object $\mathcal{X}$, denoted by $\mathrm{TC}(\mathcal{X})$, is defined as all objects that recursively occur in $\mathcal{X}$. All labeling cosets that occur in $\mathcal{X}$ are succinctly represented via generating sets. The encoding size of an object $\mathcal{X}$ can be chosen polynomial in $|\mathrm{TC}(\mathcal{X})|+|V|+t_{\max}$ where $t_{\max}$ is the maximal length of a tuple in $\mathrm{TC}(\mathcal{X})$.

Isomorphisms and Automorphisms of Objects. The image of an object $\mathcal{X}\in\mathrm{Obj}(V)$ (where V is disjoint from $\mathbb{N}$) under a bijection $\mu:V\to V'$ is an object $\mathcal{X}^{\mu}\in\mathrm{Obj}(V')$ that is defined as follows. Define $v^{\mu}:=\mu(v)$ for an atom $\mathcal{X}=v$ and define $(\Delta\rho)^{\mu}:=\mu^{-1}\Delta\rho$ for an atom $\mathcal{X}=\Delta\rho$, and inductively define $\{X_1,\dots,X_t\}^{\mu}:=\{X_1^{\mu},\dots,X_t^{\mu}\}$ and $(X_1,\dots,X_t)^{\mu}:=(X_1^{\mu},\dots,X_t^{\mu})$.

The set of all isomorphisms from an object $\mathcal{X}\in\mathrm{Obj}(V)$ and to an object $\mathcal{X}'\in\mathrm{Obj}(V')$ is denoted by $\mathrm{Iso}(\mathcal{X};\mathcal{X}'):=\{\varphi:V\to V'\mid\mathcal{X}^{\varphi}=\mathcal{X}'\}$. The set of all automorphisms of an object $\mathcal{X}$ is denoted by $\mathrm{Aut}(\mathcal{X}):=\mathrm{Iso}(\mathcal{X};\mathcal{X})$.

▶ **Definition 1** ([31]). *Let $\mathcal{C}$ be an isomorphisms-closed class of (unordered) objects, i.e., for all $\mathcal{X}\in\mathcal{C}$ over a set V and all bijections $\varphi:V\to V'$ it holds that $\mathcal{X}^{\varphi}\in\mathcal{C}$. A* canonical labeling function CL *is a function that assigns each object in $\mathcal{C}$ a labeling coset $\mathrm{CL}(\mathcal{X})=\Lambda\le\mathrm{Label}(V)$ such that:*

(CL1) $\mathrm{CL}(\mathcal{X})=\varphi\,\mathrm{CL}(\mathcal{X}^{\varphi})$ *for all $\varphi\in\mathrm{Iso}(V;V')$ (the set of bijections from V to V'), and*

(CL2) $\mathrm{CL}(\mathcal{X})=\mathrm{Aut}(\mathcal{X})\pi$ *for some (and thus for all) $\pi\in\mathrm{CL}(\mathcal{X})$.*

In this case, the labeling coset Λ is also called a canonical labeling *for $\mathcal{X}$.*

3 Handling Small Objects via String Canonization

Next, we define the central problem of this paper which is introduced in [14],[31]. This problem is a canonical version of the multiple-coset isomorphism problem.

▶ **Problem 2.** *Compute a function $\mathrm{CL}_{\mathrm{Set}}$ with the following properties:*

Input $J\in\mathrm{Obj}(V)$ *where* $J=\{\Delta_1\rho_1,\dots,\Delta_t\rho_t\}$, $\Delta_i\rho_i\le\mathrm{Label}(V), i\in[t]$ *and V is an (unordered) set.*

Output *A labeling coset* $\mathrm{CL}_{\mathrm{Set}}(J)=\Lambda\le\mathrm{Label}(V)$ *such that:*

(CL1) $\mathrm{CL}_{\mathrm{Set}}(J)=\varphi\,\mathrm{CL}_{\mathrm{Set}}(J^{\varphi})$ *for all* $\varphi\in\mathrm{Iso}(V;V')$.

(CL2) $\mathrm{CL}_{\mathrm{Set}}(J)=\mathrm{Aut}(J)\pi$ *for some (and thus for all)* $\pi\in\Lambda$.

Following the definition of the automorphism group for objects in general, we have that $\mathrm{Aut}(J)=\{\sigma\in\mathrm{Sym}(V)\mid\exists\psi\in\mathrm{Sym}(t)\forall i\in[t]:\sigma^{-1}\Delta_i\rho_i=\Delta_{\psi(i)}\rho_{\psi(i)}\}$.

To clarify the complexity status of Problem 2, it is important to note that the problem is actually polynomial-time equivalent to the string canonization problem. The string canonization problem in turn can be solved in quasipolynomial-time with Babai's algorithm [3]. However, the reduction increases the permutation domain V by a factor $|J|$, which leads to a running time of $2^{\mathrm{polylog}(|V|+|J|)}$ as stated in the following lemma.

▶ **Lemma 3.** *Canonical labelings for sets J can be computed in time $2^{\mathrm{polylog}(|V|+|J|)}$.*

The main task in this work is to remove the dependency of $|J|$ in the exponent. The main algorithm (Theorem 7) solves Problem 2 in a running time of $(|V|+|J|)^{\mathrm{polylog}|V|}$ and is presented in Section 6. This improvement will finally lead to an improved algorithm for the graph isomorphism problem from $n^{\mathrm{polylog}(n)}$ to $n^{\mathrm{polylog}(k)}$ where n is the number of vertices and k is the minimum treewidth of the input graphs. However, Lemma 3 is still used in our algorithms. Especially, when $|J|$ is bounded by some quasipolynomial in $|V|$, the algorithm from Lemma 3 already runs in the desired time bound.

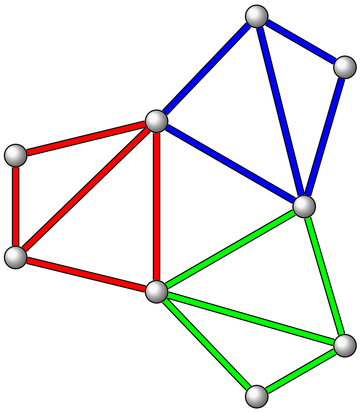


Figure 1 We see a graph G decomposed into 3 isomorphic subgraphs $H_1, H_2, H_3 \subseteq G$ shown in distinct colors.

The Intuition Behind this Central Problem. We explain why Problem 2 is the central problem when dealing with graph decompositions. We want to keep this subsection as simple as possible and do not want to introduce tree decompositions yet. For our purpose, we consider a simplified formulation of a graph decomposition. In this subsection, a graph decomposition of a graph $G = (V, E)$ is a family of subgraphs $\{H_i\}_{i \in [t]}$ that covers the edges of the entire graph, i.e., $E(G) = E(H_1) \cup \ldots \cup E(H_t)$. We say that a graph decomposition is defined in an isomorphism-invariant way if for two isomorphic graphs G, G' the decompositions $\{H_i\}_{i \in [t]}, \{H'_i\}_{i \in [t]}$ are defined in such a way that each isomorphism $\varphi \in \text{Iso}(G; G')$ also maps each subgraph H_i of the decomposition of G to a subgraph H'_j of the decomposition of G'. In particular, such a decomposition has to be invariant under automorphisms of the graph. A prime example of such a isomorphism-invariant decomposition in the setting of bounded-treewidth graphs is the decomposition into clique-separator-free parts. The clique-separator decomposition goes back to Leimer [20] and is also used in our final isomorphism algorithm.

Assume we have given a graph G for which we can construct a graph decomposition $\{H_i\}_{i \in [t]}$ in an isomorphism-invariant way and our task is the computation of a canonical labeling for G. A priori, it is unclear how to exploit our graph decomposition. In a first step, we could compute canonical labelings $\Delta_i \rho_i := \text{CL}(H_i)$ for each subgraph H_i recursively. The central question is how to merge these labeling cosets $\Delta_i \rho_i$ for H_i in order to obtain a canonical labeling $\Delta \rho$ for the entire graph G.

The easy case occurs when all subgraphs H_i, H_j are pairwise non-isomorphic. In this case, the subgraphs cannot be mapped to each other and indeed $\text{Aut}(G) = \text{Aut}(H_1) \cap \ldots \cap \text{Aut}(H_t)$. Therefore, the computation of $\Delta \rho$ reduces to a canonical intersection-problem. In fact, Babai's quasipolynomial-time algorithm [3] can be used to intersect labeling cosets canonically.

We consider the second extreme case in which all subgraphs H_i, H_j are pairwise isomorphic, see Figure 1.

In such a case, we have that $\text{Aut}(G) = \{\sigma \in \text{Sym}(V) \mid \exists \psi(t) \forall i \in [t] : \sigma \in \text{Iso}(H_i; H_{\psi(i)})\}$. Equivalently, we have that $\text{Aut}(G) = \text{Aut}(\{\Delta_1 \rho_1, \ldots, \Delta_t \rho_t\})$. Therefore, by the definition of Problem 2, the canonical labeling $\Delta \rho := \text{CL}_{\text{Set}}(\{\Delta_1 \rho_1, \ldots, \Delta_t \rho_t\})$ defines a canonical labeling for the entire graph G.

Alternatively, one can use the following lemma which intuitively says that for the purpose of canonization the subgraphs H_i can be replaced with their labeling cosets $\Delta_i \rho_i$ while preserving all symmetry information. Formally, it says the following.

▶ **Lemma 4** ([31], Object Replacement Lemma)**.** *Let $\mathcal{X} = \{X_1, \ldots, X_t\}$ be an object and let* CL *and* $\mathrm{CL}_{\mathrm{Set}}$ *be canonical labeling functions. Define $\mathcal{X}^{\mathrm{Set}} := \{\Delta_1\rho_1, \ldots, \Delta_t\rho_t\}$ where $\Delta_i\rho_i := \mathrm{CL}(X_i)$ is a canonical labeling for $X_i \in \mathcal{X}$. Assume that $X_i, X_j \in \mathcal{X}$ are pairwise isomorphic. Then, $\mathrm{CL}_{\mathrm{Object}}(\mathcal{X}) := \mathrm{CL}_{\mathrm{Set}}(\mathcal{X}^{\mathrm{Set}})$ defines a canonical labeling for $\mathcal{X}$.*

Roughly speaking, Problem 2 can be seen as the task of merging the given labeling cosets. The mixed case in which some (but not all) subgraphs H_i, H_j are isomorphic can be handled by a mixture of the above cases.

In Section 7, we apply this problem to graphs G with n vertices of treewidth k. By exploiting that the subgraphs in our application can only intersect in cliques of size at most k, we are able to restrict our attention to vertex sets of size $|V| \leq k$ (with at most $|J| \leq n$ labeling cosets). This finally leads to the desired running time of $n^{\mathrm{polylog}(k)}$.

Application to Combinatorial Objects. A second application of Problem 2 is the canonization framework for combinatorial objects in general given in [31]. In fact, our algorithms build on this canonization framework as it allows a recursive approach to solve the problem. Our improved running time for Problem 2 then implies an improved canonization algorithm for combinatorial objects in general that runs in time $n^{\mathrm{polylog}|V|}$ (Corollary 8).

4 Canonization of k-ary Relations

In this section, we provide an algorithm for canonical labeling of k-ary relations $R \subseteq V^k$. As graphs can be seen as binary relations, this problem clearly generalizes the graph canonization problem. One way to canonize k-ary relations is by using a well-known reduction to the graph canonization problem [23]. However, this approach leads to a running time that is quasipolynomial in $|V| + |R|$. In this section, we will give a polynomial-time reduction to the canonization problem for objects that are of input size polynomial in $|V|$ (which does not depend on $|R|$). With this reduction, we obtain an improved algorithm that runs in time $2^{\mathrm{polylog}|V|}|R|^{\mathcal{O}(1)}$. Our bound improves the currently best algorithm from [13]. Moreover, our time bound is also optimal (when measured in $|V|$ and $|R|$) as long as the graph isomorphism problem can not be solved faster than quasipolynomial time.

A *partition* of a set $\mathcal{X} \in \mathrm{Obj}(V)$ is a set $\mathcal{P} = \{P_1, \ldots, P_p\}$ such that $\mathcal{X} = P_1 \cup \ldots \cup P_p$ where $\emptyset \neq P_i \subseteq \mathcal{X}$ for all $P_i \in \mathcal{P}$. We suggest a general technique for exploiting partitions.

The Partitioning Technique. In this setting, we assume that we are given some object $\mathcal{X} \in \mathrm{Obj}(V)$ for which we can construct a partition $\mathcal{P} = \{P_1, \ldots, P_p\}$ in an isomorphism-invariant way such that $2 \leq |\mathcal{P}| \leq 2^{\mathrm{polylog}|V|}$. The goal is the computation of a canonical labeling for $\mathcal{X}$ by using an efficient recursion.

For example, $\mathcal{X} = R \subseteq V^k$ might be a k-ary relation for which we can easily construct a partition in an isomorphism-invariant way, as seen next. Assume $|R| \geq 2$ (otherwise the canonization problem is easy to solve) and let r be the first position in which R differs, i.e., the smallest $r \in [k]$ such that there are $(x_1, \ldots, x_k), (y_1, \ldots, y_k) \in R$ with $x_r \neq y_r$. Then, we partition $R = P_1 \cup \ldots \cup P_p$ by saying that two tuples $(x_1, \ldots, x_k), (y_1, \ldots, y_k)$ are in the same part P_i if and only if $x_r = y_r$. This gives a non-trivial partition $\mathcal{P} = \{P_1, \ldots, P_p\}$ with $2 \leq |\mathcal{P}| \leq |V| \leq 2^{\mathrm{polylog}|V|}$ which is preserved under automorphisms and isomorphisms.

Using recursion, we compute a canonical labeling $\Delta_i\rho_i$ for each part $P_i \subseteq \mathcal{X}$ recursively (assumed that we can define a partition for each part again). In our example, $P_i \subseteq R$ is a subrelation and therefore we can apply our approach recursively.

So far, we computed canonical labelings for each part $P_i \subseteq \mathcal{X}$ independently. The main idea is to use our central problem (Problem 2) to merge all these labeling cosets. Let us restrict our attention to the case in which the parts $P_i, P_j \in \mathcal{P}$ are pairwise isomorphic. In this case, we define the set $\mathcal{P}^{\text{Set}} := \{\Delta_i \rho_i \mid P_i \in \mathcal{P}\}$ consisting of the canonical labelings $\Delta_i \rho_i$ for each part. Moreover, by object replacement (Lemma 4), a canonical labeling for $\mathcal{P}^{\text{Set}}$ defines a canonical labeling for $\mathcal{P}$ as well. A canonical labeling for $\mathcal{P}$ in turn defines a canonical labeling for $\mathcal{X}$ since we assume the partition to be defined in an isomorphism-invariant way. Therefore, it is indeed true that a canonical labeling for $\mathcal{P}^{\text{Set}}$ would define a canonical labeling for $\mathcal{X}$. For this reason, we can use the algorithm from Lemma 3 to compute a canonical labeling for $\mathcal{P}^{\text{Set}}$. Intuitively, this algorithm merges all the labeling cosets in $\mathcal{P}^{\text{Set}}$ into one single canonical labeling. In our example, this single labeling coset is a canonical labeling for the relation R. The algorithm Lemma 3 runs in the desired time bound since $|\mathcal{P}^{\text{Set}}| = |\mathcal{P}| \leq 2^{\text{polylog}|V|}$ is bounded by some quasipolynomial.

Let us consider the number of recursive calls $R(\mathcal{X})$ of this approach for a given object $\mathcal{X}$. Since we recurse on each part $P_i \in \mathcal{P}$, we have a recurrence of $R(\mathcal{X}) = 1 + \sum_{P_i \in \mathcal{P}} R(P_i)$ leading to at most $|\mathcal{X}|^{\mathcal{O}(1)}$ recursive calls. The running time for one single recursive call is bounded by $2^{\text{polylog}|V|}$. For this reason, the total running time is bounded by $2^{\text{polylog}|V|}|\mathcal{X}|^{\mathcal{O}(1)}$.

▶ **Theorem 5.** *Canonical labelings for k-ary relations $R \subseteq V^k$ can be computed in time $2^{\text{polylog}|V|}|R|^{\mathcal{O}(1)}$.*

5 Canonization of Hypergraphs

In this section, we provide an algorithm for canonical labeling of hypergraphs (V, H) where $H \subseteq 2^V$.

We want to extend the previous partitioning technique to hypergraphs. However, for hypergraphs a non-trivial isomorphism-invariant partition $H = H_1 \uplus \ldots \uplus H_s$ of the edge set does not always exist, e.g., the hypergraph $(V, \{S \subseteq V \mid |S| = 2\})$ does not have a non-trivial partition of the edge set that is preserved under automorphisms. Therefore, we can not apply the partitioning technique to this setting. For this reason, we introduce a generalized technique in order to solve this problem. This generalized technique results in a slightly weaker time bound of $(|V| + |H|)^{\text{polylog}|V|}$ (where the dependency on $|H|$ is not polynomial). Indeed, it is an open problem whether the running time for the hypergraph isomorphism problem can be improved to $2^{\text{polylog}|V|} \cdot |H|^{\mathcal{O}(1)}$ [2].

A *cover* of a set $\mathcal{X} \in \text{Obj}(V)$ is a set $\mathcal{C} = \{C_1, \ldots, C_c\}$ such that $\mathcal{X} = C_1 \cup \ldots \cup C_c$ where $\emptyset \neq C_i \subseteq \mathcal{X}$ for all $C_i \in \mathcal{C}$. In contrast to a partition, the sets C_i, C_j are not necessarily disjoint for $i \neq j$. A cover $\mathcal{C}$ of $\mathcal{X}$ is called *sparse* if $|C_i| \leq \frac{1}{2}|\mathcal{X}|$ for all $C_i \in \mathcal{C}$. Extending the partitioning technique, we suggest a technique to handle covers.

The Covering Technique. In this setting, we assume that we have given some object $\mathcal{X} \in \text{Obj}(V)$ for which we can define a cover $\mathcal{C} = \{C_1, \ldots, C_c\}$ in an isomorphism-invariant way. Also here, we assume that $2 \leq |\mathcal{C}| \leq 2^{\text{polylog}|V|}$. The goal is the computation of a canonical labeling of $\mathcal{X}$ using an efficient recursion.

For example, $\mathcal{X} = H$ is a hypergraph for which we can easily define a cover in an isomorphism-invariant way, as seen next. We assume that $|H| \geq 2$ (otherwise the canonization problem is easy to solve) and that $\emptyset \notin H$ (otherwise we remove the empty set from H). Then, we cover $H = \bigcup_{v \in V} C_v$ by setting $C_v := \{S \in H \mid v \in S\}$ and define a cover $\mathcal{C} := \{C_v \mid v \in V\}$. Since each hyperedge $S \in H$ contains at least one vertex $v \in V$, each hyperedge S is contained in at least one set $C_v \in \mathcal{C}$. Moreover, the cover $\mathcal{C}$ is preserved under automorphisms and isomorphisms.

First, we reduce to the setting in which $\mathcal{C}$ is a sparse cover of $\mathcal{X}$. This can be done as follows. We define $C_i^* := C_i$ if $|C_i| \leq \frac{1}{2}|\mathcal{X}|$ and we define $C_i^* := \mathcal{X} \setminus C_i$ if $|C_i| > \frac{1}{2}|\mathcal{X}|$. By definition, we ensured that $|C_i^*| \leq \frac{1}{2}|\mathcal{X}|$ for all $i \in [c]$. Let $\mathcal{X}^* := \bigcup_{i \in [c]} C_i^*$. Next, we consider two cases.

If $\mathcal{X}^* \subsetneq \mathcal{X}$, then we have found a non-trivial partition $\mathcal{X} = \mathcal{X}^* \uplus \mathcal{X}^\circ$ where $\mathcal{X}^\circ := \mathcal{X} \setminus \mathcal{X}^*$. In the hypergraph example, we would have a non-trivial partition of the hyperedges defined in an isomorphism-invariant way (which can be exploited very easily). We proceed analogously as in the partitioning technique explained in Section 4.

Otherwise, if $\mathcal{X}^* = \mathcal{X}$, then $\mathcal{C}^* := \{C_1^*, \dots, C_c^*\}$ is also a cover of $\mathcal{X}$. But more importantly, the cover $\mathcal{C}^*$ is indeed sparse. In the case of a sparse cover, we also proceed analogously as in the partition technique explained in Section 4. However, the key difference of the covering technique compared to the partitioning technique lies in the recurrence for the number of recursive calls since the sets $C_i^*, C_j^* \in \mathcal{C}^*$ are not necessarily pairwise disjoint. The recurrence we have is $R(\mathcal{X}) = 1 + \sum_{C_i^* \in \mathcal{C}^*} R(C_i^*)$. By using that $|\mathcal{C}^*| = |\mathcal{C}| \leq 2^{\operatorname{polylog}|V|}$ and that $|C_i^*| \leq \frac{1}{2}|\mathcal{X}|$, we obtain at most $|\mathcal{X}|^{\operatorname{polylog}|V|}$ recursive calls. This is exactly the reason why the algorithm for relations is faster than the algorithm for hypergraphs.

▶ **Theorem 6.** *Canonical labelings for hypergraphs* (V, H) *can be computed in time* $(|V| + |H|)^{\operatorname{polylog}|V|}$.

6 Canonization of Sets and Objects

Our main theorem provides an algorithm that canonizes a set $J = \{\Delta_1\rho_1, \dots, \Delta_t\rho_t\}$ consisting of labelings cosets $\Delta_i\rho_i \leq \operatorname{Label}(V), i \in [t]$.

▶ **Theorem 7.** *A function* $\mathrm{CL}_{\mathrm{Set}}$ *solving Problem 2 can be computed in time* $(|V|+|J|)^{\operatorname{polylog}|V|}$.

▶ **Corollary 8.** *Canonical labelings for combinatorial objects can be computed in time* $n^{\operatorname{polylog}|V|}$ *where* n *is the input size and* V *is the ground set of the object.*

Proof Outline. For the purpose of recursion, our main algorithm $\mathrm{CL}_{\mathrm{Set}}$ needs some additional input parameters. The input of the main algorithm is a tuple $(J, A, \Delta^{\mathrm{Can}}, g^{\mathrm{Can}})$ consisting of the following input parameters.

- J is a set consisting of labeling cosets,
- $A \subseteq V$ is a subset which is Δ_i-invariant for all $\Delta_i\rho_i \in J$. We require that Property (A) holds: $(\Delta_i\rho_i)|_{V \setminus A} = (\Delta_j\rho_j)|_{V \setminus A}$ for all $\Delta_i\rho_i, \Delta_j\rho_j \in J$ (initially, we set $A := V$),
- $\Delta^{\mathrm{Can}} \leq \operatorname{Sym}(V^{\mathrm{Can}})$ is a group over the linearly ordered set $V^{\mathrm{Can}} = \{1, \dots, |V|\}$. We require that for all $\Delta_i\rho_i \in J$ it holds that $\rho_i^{-1}\Delta_i\rho_i = \Delta^{\mathrm{Can}}$ (if this would not be the case, in can be shown that J can be partitioned such that progress can be measured), and
- $g^{\mathrm{Can}} : \Delta^{\mathrm{Can}} \to \operatorname{Sym}(W^{\mathrm{Can}})$ is a giant representation where $W^{\mathrm{Can}} = \{1, \dots, |W^{\mathrm{Can}}|\}$ is a linearly ordered set (a homomorphism $h : \Delta \to \operatorname{Sym}(W)$ is called a *giant representation* if the image of Δ under h is a giant, i.e., $\operatorname{Alt}(W) \leq h(\Delta) \leq \operatorname{Sym}(W)$). It is allowed that g^{Can} is undefined ($g^{\mathrm{Can}} = \bot$).

The Subgroup Recursion. First of all we design a subgroup reduction that, given a tuple $(J, A, \Delta^{\mathrm{Can}}, \bot)$ and a subgroup $\Psi^{\mathrm{Can}} \leq \Delta^{\mathrm{Can}}$, reduces the canonical labeling problem of $(J, A, \Delta^{\mathrm{Can}}, \bot)$ to s-many instances $(\widehat{J}_i, A, \Psi^{\mathrm{Can}}, \bot)$ where $|\widehat{J}_i| \leq |J|$. Here, s corresponds to the index of Ψ^{Can} in Δ^{Can}. In contrast to Luks's subgroup reduction, the present reduction splits all labeling cosets in J simultaneously. We describe the idea of this algorithm.

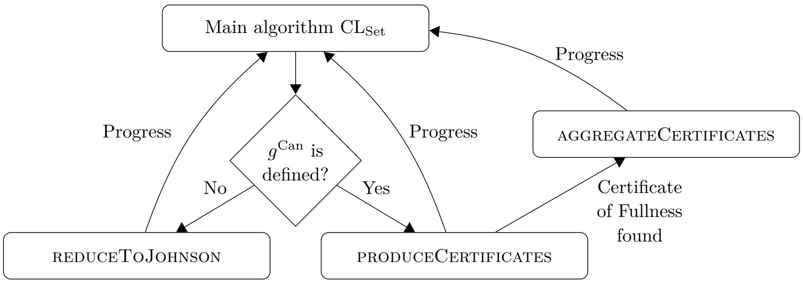

Main algorithm $\mathrm{CL}_{\mathrm{Set}}$
g^{Can} is defined?
No
Yes
Progress
Progress
Progress
REDUCETOJOHNSON
PRODUCECERTIFICATES
AGGREGATECERTIFICATES
Certificate of Fullness found

Figure 2 Flowchart of the algorithm for Theorem 7.

We consider the decomposition into left cosets of $\Delta^{\mathrm{Can}} = \bigcup_{\ell\in[s]} \delta_\ell^{\mathrm{Can}}\Psi^{\mathrm{Can}}$ and define $\widehat{\mathcal{J}} := \{\rho_i\delta_\ell^{\mathrm{Can}}\Psi^{\mathrm{Can}} \mid i \in [t], \ell \in [s]\}$. Surprisingly, we can show that $\mathrm{Aut}(\widehat{\mathcal{J}}) = \mathrm{Aut}(J)$. This means that a canonical labeling for $\widehat{\mathcal{J}}$ defines a canonical labeling for J as well and vice versa. Therefore, the first idea that comes to mind would be a recursion on the instance $(\widehat{\mathcal{J}}, A, \Psi^{\mathrm{Can}}, \bot)$. However, there are two problems when recursing on $\widehat{\mathcal{J}}$. First, the instance $\widehat{\mathcal{J}}$ does not necessarily preserve Property (A) that we have for the subset $A \subseteq V$ (this requirement is important for the subroutines that follow).

Second, it holds that $|\widehat{\mathcal{J}}| > |J|$ (assumed that $\Psi^{\mathrm{Can}} < \Delta^{\mathrm{Can}}$ is a proper subgroup). Also this blow-up in the instance size would not lead to the desired recursion. However, we are able to construct a decomposition of $\widehat{\mathcal{J}} = \widehat{\mathcal{J}}_1 \uplus \ldots \uplus \widehat{\mathcal{J}}_r$ such that $r \le s$ and $|\widehat{\mathcal{J}}_i| \le |J|$ and such that Property (A) holds for each instance $(\widehat{\mathcal{J}}_i, A, \Psi^{\mathrm{Can}}, \bot)$.

The Johnson Reduction. We design an algorithm that, given an instance $(J, A, \Delta^{\mathrm{Can}}, \bot)$, either finds a giant representation $g^{\mathrm{Can}} : \Delta^{\mathrm{Can}} \to \mathrm{Sym}(W^{\mathrm{Can}})$ or reduces the canonical labeling problem of $(J, A, \Delta^{\mathrm{Can}}, \bot)$ to instances that are smaller (according to some function that measures progress).

First of all, we want to reduce to the case in which all $\Delta_i \le \mathrm{Sym}(V)$ are transitive on $A \subseteq V$. To achieve transitivity, Babai's algorithm uses Luks's idea of orbit-by-orbit processing. However, the orbit-by-orbit recursion is a tool that is developed for strings and needs a non-trivial adaption when dealing with a set of labeling cosets J. To achieve transitivity, the present algorithm uses an extension of the orbit-by-orbit recursion that was developed in [31]. In the transitive case, we follow Babai's idea. First, we define a block system $\mathcal{B}^{\mathrm{Can}}$ on which Δ^{Can} acts primitively. If the primitive group acting on $\mathcal{B}^{\mathrm{Can}}$ is small, we use the subgroup reduction to reduce to a subgroup $\Psi^{\mathrm{Can}} \le \Delta^{\mathrm{Can}}$ that is defined as the kernel of that action. In case that the primitive group is large, we use Cameron's classification of large primitive groups which implies that the primitive group is a Cameron group. We reduce the Cameron group to a Johnson group by using the subgroup reduction again. The Johnson group (acting on subsets of a set W^{Can}) in turn can be used to define a giant representation $g^{\mathrm{Can}} : \Delta^{\mathrm{Can}} \to \mathrm{Sym}(W^{\mathrm{Can}})$.

▶ **Definition 9** (Certificates of Fullness). *A group $G \le \mathrm{Sym}(V)$ is called* certificate of fullness *for an instance $(J, A, \Delta^{\mathrm{Can}}, g^{\mathrm{Can}})$ if*

1. $G \le \mathrm{Aut}(J)$,
2. $G^{\mathrm{Can}} := G^{\rho_i} \le \Delta^{\mathrm{Can}}$ *does not depend on the choice of $\Delta_i\rho_i \in J$, and*
3. $g^{\mathrm{Can}} : G^{\mathrm{Can}} \to \mathrm{Sym}(W^{\mathrm{Can}})$ *is still a giant representation.*

The Certificate Producing Algorithm. We design an algorithm that, given an instance $(J, A, \Delta^{\mathrm{Can}}, g^{\mathrm{Can}})$ (where g^{Can} is defined), either finds a certificate of fullness or makes progress (according to some function that measures progress).

The algorithm picks a subset $T^{\mathrm{Can}} \subseteq W^{\mathrm{Can}}$ of logarithmic size. We call this set T^{Can} a *canonical test set.* Next, we define the group $\Delta_T^{\mathrm{Can}} \leq \Delta^{\mathrm{Can}}$ which stabilizes T^{Can} in the image under g^{Can}. By doing so, we can define a giant representation $g_T^{\mathrm{Can}} : \Delta_T^{\mathrm{Can}} \to \mathrm{Sym}(T^{\mathrm{Can}})$. We say that $v^{\mathrm{Can}} \in V^{\mathrm{Can}}$ is *affected* by g_T^{Can} if g_T^{Can} is not a giant representation when restricted to $(\Delta_T^{\mathrm{Can}})_{(v^{\mathrm{Can}})}$. Let $S^{\mathrm{Can}}, U^{\mathrm{Can}} \subseteq V^{\mathrm{Can}}$ be set of elements affected and unaffected by g_T^{Can}, respectively. We have a technical difference in our algorithm in contrast to Babai's method. In Babai's method of local certificates, he processes a giant representation $g : \Delta \to \mathrm{Sym}(W)$ and considers multiple test sets $T \subseteq W$ (one test set for each subset of logarithmic size). In our framework, we define the giant representation for a group Δ^{Can} over a linearly ordered set V^{Can}. This allows us to choose one single (canonical) test set $T^{\mathrm{Can}} \subseteq W^{\mathrm{Can}}$ only. Here, canonical means that the subset is chosen minimal with respect to the ordering of the natural numbers. However, when we translate the ordered structures V^{Can} to unordered structures over V, we implicitly consider multiple test sets and giant representations. More precise, by applying inverses of labelings in $\Delta_i\rho_i \in J$ to the ordered group $\Delta_T^{\mathrm{Can}} \leq \mathrm{Sym}(V^{\mathrm{Can}})$, we obtain a set of groups over V, i.e., $\{\lambda_i \Delta_T^{\mathrm{Can}} \lambda_i^{-1} \mid \lambda_i \in \Delta_i\rho_i\}$. Similarly, we can define a set of giant representations $\{(g_T^{\mathrm{Can}})^{\lambda_i^{-1}} \mid \lambda_i \in \Delta_i\rho_i\}$ (where $(g_T^{\mathrm{Can}})^{\lambda_i^{-1}}(\delta_i) := g_T^{\mathrm{Can}}(\lambda_i^{-1}\delta_i\lambda_i)$ for $\delta_i \in \Delta_i$) and a set of affected points $H_i := \{S \subseteq V \mid S^{\lambda_i} = S^{\mathrm{Can}}$ for some $\lambda_i \in \Delta_i\rho_i\}$. Therefore, when dealing over unordered structures, we need to consider multiple groups and homomorphisms. It becomes even more complex, since we are dealing with a set J consisting of labeling cosets rather than one single group only. In fact, we obtain a set of affected point sets H_i for each labeling coset $\Delta_i\rho_i \in J$. However, it turns out that the hardest case occurs when $H_i = H_j$ for all $\Delta_i\rho_i, \Delta_j\rho_j \in J$. Roughly speaking, we will apply the following strategy.

We restrict each labeling coset in J to some set of affected points $S \in H_i$ and define a set of local restrictions J_S^* that ignore the vertices outside S. The precise definition of J_S^* is given in our algorithm. Intuitively, the algorithms tries to analyze the labeling cosets locally.

Case 1: The local restrictions J_S^* are pairwise distinct. In this case, we canonize the local restrictions J_S^* recursively. Observe that a canonical labeling $\Delta\rho$ for J_S^* does not necessarily define a canonical labeling for J. However, we can define a function $\alpha : J_S^* \to J$ that assigns each local restriction its corresponding labeling coset $\Delta_i\rho_i \in J$. This function is well-defined since we assumed the local restrictions to be pairwise distinct. Now, each automorphism in $\mathrm{Aut}(J_S^*)$ induces a permutation of J_S^* which in turn induces a permutation of J. We are able to use the permutations on J to canonize the set J efficiently (without even applying further recursive calls).

Case 2: Some local restrictions in J_S^* are pairwise different and some local restrictions in J_S^* are pairwise equal. In this case, we can define a non-trivial partition of J in the following way. We say that two labeling cosets $\Delta_i\rho_i, \Delta_j\rho_j$ are in the same part, if and only if the corresponding local restrictions in J_S^* coincide. Actually, this leads to a family of partitions since we obtain one partition for each choice of an affected set $S \in H_i$. We exploit this partition family by using an extension of the partitioning techniques from Sections 4 and 5.

Case 3: The local restrictions J_S^* are pairwise equal. In this case, it is possible to find automorphisms $G_S \leq \mathrm{Sym}(V)$ of J which fix the unaffected points $V \setminus S$. In fact, we can find such automorphisms for all choices of $S \in H_i$, otherwise we are in a situation of a previous case. Finally, we consider the group of automorphisms $G \leq \mathrm{Aut}(J)$ generated by all G_S for $S \in H_i$. We can show that G is indeed a certificate of fullness.

The Certificate Aggregation. We finally design an algorithm that, given an instance $(J, A, \Delta^{\mathrm{Can}}, g^{\mathrm{Can}})$ and a certificate of fullness $G \leq \mathrm{Sym}(V)$ makes progress (according to some function that measures progress).

Let us consider the less technical case in which $g^{\mathrm{Can}}(G^{\mathrm{Can}})$ is the symmetric group (rather than the alternating group). In this case, it holds that $G^{\mathrm{Can}}\Psi^{\mathrm{Can}} = \Delta^{\mathrm{Can}}$ where Ψ^{Can} is the kernel of g^{Can}. Similarly to the subgroup recursion, we consider the decomposition of $\Delta^{\mathrm{Can}} = \bigcup_{\ell\in[s]} \delta_\ell^{\mathrm{Can}}\Psi^{\mathrm{Can}}$ into left cosets of the kernel and define $\widehat{J} := \{\rho_i\delta_\ell^{\mathrm{Can}}\Psi^{\mathrm{Can}} \mid i \in [t], \ell \in [s]\}$. Again, we have $\mathrm{Aut}(\widehat{J}) = \mathrm{Aut}(J)$. The key observation is that G is transitive on $\widehat{J}$ since $(\rho_i\delta_\ell^{\mathrm{Can}}\Psi^{\mathrm{Can}})^{g^{-1}} = \rho_i g^{\rho_i}\delta_\ell^{\mathrm{Can}}\Psi^{\mathrm{Can}}$ for all $g \in G$ and $G^{\mathrm{Can}}\Psi^{\mathrm{Can}} = \Delta^{\mathrm{Can}}$.

First, consider an easy case in which $J = \{\Delta_1\rho_1\}$ consists of one single labeling coset. In this case, we have a set of automorphisms G acting transitively on the subcosets $\widehat{J} = \{\rho_1\delta_\ell^{\mathrm{Can}}\Psi^{\mathrm{Can}} \mid \ell \in [s]\}$. Moreover, each subcoset satisfies $\mathrm{Aut}(\rho_1\delta_\ell^{\mathrm{Can}}\Psi^{\mathrm{Can}}) \leq \mathrm{Aut}(J)$ and can be seen as an individualization of J. This means, we can choose (arbitrarily) a subcoset $\rho_1\delta_\ell^{\mathrm{Can}}\Psi^{\mathrm{Can}} \leq \Delta_1\rho_1$ and recurse on that. Since the automorphisms in G can map each subcoset to each other subcoset it does not matter which subcoset we choose. By recursing on one single subcoset only, we can measure significant progress. At the end, we return $G\widehat{\Lambda}$ where $\widehat{\Lambda}$ is a canonical labeling for the (arbitrarily) chosen subcoset and G is the group of automorphisms (acting transitively on the set of all subcosets).

However, the situation becomes more difficult when dealing with more labeling cosets $J = \{\Delta_1\rho_1,\dots,\Delta_t\rho_t\}$ for $t \geq 2$. The first idea that comes to mind is the following generalization. We choose (arbitrarily) some $\ell \in [s]$ and define the set of subcosets $\widehat{J_\ell} := \{\rho_i\delta_\ell^{\mathrm{Can}}\Psi^{\mathrm{Can}} \mid i \in [t]\} \subseteq \widehat{J}$. The set $\widehat{J_\ell}$ contains exactly one subcoset $\rho_i\delta_\ell^{\mathrm{Can}}\Psi^{\mathrm{Can}} \leq \Delta_i\rho_i$ of each $\Delta_i\rho_i \in J$. However, the partition $\widehat{\mathcal{J}} := \{\widehat{J_\ell} \mid \ell \in [s]\}$ might not be G-invariant and G might not be transitive on it. In fact, we are able to find a suitable partition $\widehat{\mathcal{J}} := \{\widehat{J}_1,\dots,\widehat{J}_r\}$ of the subcosets $\widehat{J}$ on which G is transitive.

7 Isomorphism of Graphs Parameterized by Treewidth

▶ **Theorem 10.** *Let G_1, G_2 be two connected graphs. There is an algorithm that, given a pair (G_1, G_2), computes the set of isomorphisms $\mathrm{Iso}(G_1; G_2)$ in time $|V(G_1)|^{\mathrm{polylog}(\mathrm{tw}\, G_1)}$.*

Proof Outline. We follow the graph decomposition approach from [14] building on [21]. It was shown that graphs of treewidth k can be decomposed in an isomorphism-invariant way into parts that have restrictions on their automorphism groups. More precisely, the automorphism group of each part has composition-width at most k after fixing one vertex (the composition-width of a group Δ is the smallest integer k such that all composition factors of Δ are isomorphic to a subgroup of $\mathrm{Sym}(k)$). This fact allows us to use the bounded-degree graph isomorphism algorithm given in [13] to compute the isomorphisms between the parts. Finally, we use our main theorem (Theorem 7) to merge the isomorphisms.

Since Grohe, Neuen and Schweitzer provide an isomorphism algorithm, rather than a canonization algorithm, our final algorithm from the previous theorem does not lead to canonical forms. However, this is the only part that depends on their isomorphism algorithm.

8 Outlook and Open Questions

One could ask the question whether our isomorphism algorithm for graphs can be improved to a FPT-algorithm that runs in time $2^{\mathrm{polylog}(k)}n^{\mathcal{O}(1)}$ where n is the number of vertices and k is the maximum treewidth of the given graphs. There are various reasons why this might be difficult. One reason is that our approach would require a FPT-algorithm for the isomorphism problem of graphs of maximum degree d that runs in time $2^{\mathrm{polylog}(d)}n^{\mathcal{O}(1)}$. However, it is an open question whether any FPT-algorithm for the graph isomorphism problem parameterized by maximum degree exists. Another reason is that an algorithm for graphs running in time $2^{\text{polylog}(k)}n^{\mathcal{O}(1)}$ would imply an isomorphism algorithm for hypergraphs (V,H) running in time $2^{\text{polylog}|V|}|H|^{\mathcal{O}(1)}$. It is also an open question, whether such a hypergraph isomorphism algorithm exists [2]. If this were indeed the case, one could hope for an improvement of our canonization algorithm for a set J consisting of labeling cosets that runs in time $2^{\text{polylog}|V|}|J|^{\mathcal{O}(1)}$.

Recently, Babai extended his quasipolynomial-time algorithm to the canonization problem for graphs [3]. With Babai's result, it is a natural question whether the bounded-degree isomorphism algorithm of [13] extends to canonization as well. The present isomorphism algorithm for graphs parameterized by treewidth should then be amenable to canonization as well.

Another question that arises is about permutation groups $G \leq \text{Sym}(V)$. The canonical labeling problem for permutation groups is of great interest because it also solves the normalizer problem. In our recent work, we gave a canonization algorithm for explicitly given permutation groups running in time $2^{\mathcal{O}(|V|)}|G|^{\mathcal{O}(1)}$ [31]. Recently, the framework was extended to permutation groups that are implicitly given and the running time was improved to $2^{\mathcal{O}(|V|)}$ [35]. The present work implies a canonization algorithm running in time $(|V|+|G|)^{\text{polylog}|V|}$. An important question is whether the present techniques can be combined with the canonization techniques for implicitly given permutation groups to obtain a canonization algorithm running in time $2^{\text{polylog}|V|}$.

Finally, we ask whether the isomorphism problem can be solved in time $n^{\text{polylog}(|V(H)|)}$ where n is the number of vertices and H is an excluded topological subgraph H of the given graphs. Even for excluded minors H, we do not have such an algorithm.

References

1 László Babai. Graph isomorphism in quasipolynomial time [extended abstract]. In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 684–697. ACM, 2016. doi:10.1145/2897518.2897542.

2 László Babai. Groups, graphs, algorithms: The graph isomorphism problem. In *Proc. ICM*, pages 3303–3320, 2018.

3 László Babai. Canonical form for graphs in quasipolynomial time: preliminary report. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019.*, pages 1237–1246, 2019. doi:10.1145/3313276.3316356.

4 László Babai and Eugene M. Luks. Canonical labeling of graphs. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing, 25-27 April, 1983, Boston, Massachusetts, USA*, pages 171–183, 1983. doi:10.1145/800061.808746.

5 Hans L. Bodlaender. Polynomial algorithms for graph isomorphism and chromatic index on partial k-trees. *J. Algorithms*, 11(4):631–643, 1990. doi:10.1016/0196-6774(90)90013-5.

6 Hans L. Bodlaender, Leizhen Cai, Jianer Chen, Michael R. Fellows, Jan Arne Telle, and Dániel Marx. Open problems in parameterized and exact computation-iwpec 2006. *UU-CS*, 2006, 2006.

7 Adam Bouland, Anuj Dawar, and Eryk Kopczynski. On tractable parameterizations of graph isomorphism. In *Parameterized and Exact Computation - 7th International Symposium, IPEC 2012, Ljubljana, Slovenia, September 12-14, 2012. Proceedings*, pages 218–230, 2012. doi:10.1007/978-3-642-33293-7_21.

8 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

9 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. doi:10.1007/978-1-4471-5559-1.

10 I. S. Filotti and Jack N. Mayer. A polynomial-time algorithm for determining the isomorphism of graphs of fixed genus (working paper). In *Proceedings of the 12th Annual ACM Symposium on Theory of Computing, April 28-30, 1980, Los Angeles, California, USA*, pages 236–243, 1980. doi:10.1145/800141.804671.

11 Martin Grohe. Isomorphism testing for embeddable graphs through definability. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, pages 63–72, 2000. doi:10.1145/335305.335313.

12 Martin Grohe and Dániel Marx. Structure theorem and isomorphism test for graphs with excluded topological subgraphs. *SIAM J. Comput.*, 44(1):114–159, 2015. doi:10.1137/120892234.

13 Martin Grohe, Daniel Neuen, and Pascal Schweitzer. A faster isomorphism test for graphs of small degree. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 89–100, 2018. doi:10.1109/FOCS.2018.00018.

14 Martin Grohe, Daniel Neuen, Pascal Schweitzer, and Daniel Wiebking. An improved isomorphism test for bounded-tree-width graphs. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, pages 67:1–67:14, 2018. doi:10.4230/LIPIcs.ICALP.2018.67.

15 Harald Andrés Helfgott. Isomorphismes de graphes en temps quasi-polynomial (d'après babai et luks, weisfeiler-leman...), 2017. arXiv:1701.04372.

16 John E. Hopcroft and Robert Endre Tarjan. A v^2 algorithm for determining isomorphism of planar graphs. *Inf. Process. Lett.*, 1(1):32–34, 1971. doi:10.1016/0020-0190(71)90019-6.

17 Ken-ichi Kawarabayashi. Graph isomorphism for bounded genus graphs in linear time. *CoRR*, abs/1511.02460, 2015. arXiv:1511.02460.

18 Ken-ichi Kawarabayashi and Bojan Mohar. Graph and map isomorphism and all polyhedral embeddings in linear time. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 471–480, 2008. doi:10.1145/1374376.1374443.

19 Stefan Kratsch and Pascal Schweitzer. Isomorphism for graphs of bounded feedback vertex set number. In *Algorithm Theory - SWAT 2010, 12th Scandinavian Symposium and Workshops on Algorithm Theory, Bergen, Norway, June 21-23, 2010. Proceedings*, pages 81–92, 2010. doi:10.1007/978-3-642-13731-0_9.

20 Hanns-Georg Leimer. Optimal decomposition by clique separators. *Discrete Mathematics*, 113(1-3):99–123, 1993. doi:10.1016/0012-365X(93)90510-Z.

21 Daniel Lokshtanov, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. Fixed-parameter tractable canonization and isomorphism test for graphs of bounded treewidth. *SIAM J. Comput.*, 46(1):161–189, 2017. doi:10.1137/140999980.

22 Eugene M. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. *J. Comput. Syst. Sci.*, 25(1):42–65, 1982. doi:10.1016/0022-0000(82)90009-5.

23 Gary L. Miller. Graph isomorphism, general remarks. *J. Comput. Syst. Sci.*, 18(2):128–142, 1979. doi:10.1016/0022-0000(79)90043-6.

24 Gary L. Miller. Isomorphism testing for graphs of bounded genus. In *Proceedings of the 12th Annual ACM Symposium on Theory of Computing, April 28-30, 1980, Los Angeles, California, USA*, pages 225–235, 1980. doi:10.1145/800141.804670.

25 Gary L. Miller. Isomorphism of k-contractible graphs. A generalization of bounded valence and bounded genus. *Information and Control*, 56(1/2):1–20, 1983. doi:10.1016/S0019-9958(83)80047-3.

26 Wendy Myrvold and William L. Kocay. Errors in graph embedding algorithms. *J. Comput. Syst. Sci.*, 77(2):430–438, 2011. doi:10.1016/j.jcss.2010.06.002.

27 Daniel Neuen. Hypergraph isomorphism for groups with restricted composition factors. In *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (virtual conference)*, 2020. To appear.

28 Yota Otachi. Isomorphism for graphs of bounded connected-path-distance-width. In *Algorithms and Computation - 23rd International Symposium, ISAAC 2012, Taipei, Taiwan, December 19-21, 2012. Proceedings*, pages 455–464, 2012. doi:10.1007/978-3-642-35261-4_48.

29 I. N. Ponomarenko. The isomorphism problem for classes of graphs closed under contraction. *Journal of Soviet Mathematics*, 55(2):1621–1643, June 1991. doi:10.1007/BF01098279.

30 Neil Robertson and Paul D. Seymour. Graph minors. i. excluding a forest. *J. Comb. Theory, Ser. B*, 35(1):39–61, 1983. doi:10.1016/0095-8956(83)90079-5.

31 Pascal Schweitzer and Daniel Wiebking. A unifying method for the design of algorithms canonizing combinatorial objects. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019.*, pages 1247–1258, 2019. doi:10.1145/3313276.3316338.

32 Uwe Schöning. Graph isomorphism is in the low hierarchy. *J. Comput. Syst. Sci.*, 37(3):312–323, 1988. doi:10.1016/0022-0000(88)90010-4.

33 Ákos Seress. *Permutation group algorithms*, volume 152 of *Cambridge Tracts in Mathematics*. Cambridge University Press, Cambridge, 2003. doi:10.1017/CBO9780511546549.

34 Louis Weinberg. A simple and efficient algorithm for determining isomorphism of planar triply connected graphs. *IEEE Transactions on Circuit Theory*, 13(2):142–148, 1966.

35 Daniel Wiebking. Normalizes and permutational isomorphisms in simply-exponential time. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 230–238, 2020. doi:10.1137/1.9781611975994.14.

36 Koichi Yamazaki, Hans L. Bodlaender, Babette de Fluiter, and Dimitrios M. Thilikos. Isomorphism for graphs of bounded distance width. *Algorithmica*, 24(2):105–127, 1999. doi:10.1007/PL00009273.

37 Viktor N. Zemlyachenko. Canonical numbering of trees. In *Proc. Seminar on Comb. Anal. at Moscow State University*, page 55, 1970.

38 Viktor N. Zemlyachenko, Nickolay M. Korneenko, and Regina I. Tyshkevich. Graph isomorphism problem. *Journal of Soviet Mathematics*, 29(4):1426–1481, 1985.

Parameterized Inapproximability for Steiner Orientation by Gap Amplification

Michał Włodarczyk
Eindhoven University of Technology, The Netherlands
m.wlodarczyk@tue.nl

Abstract

In the k-Steiner Orientation problem, we are given a mixed graph, that is, with both directed and undirected edges, and a set of k terminal pairs. The goal is to find an orientation of the undirected edges that maximizes the number of terminal pairs for which there is a path from the source to the sink. The problem is known to be W[1]-hard when parameterized by k and hard to approximate up to some constant for FPT algorithms assuming Gap-ETH. On the other hand, no approximation factor better than $\mathcal{O}(k)$ is known.

We show that k-Steiner Orientation is unlikely to admit an approximation algorithm with any constant factor, even within FPT running time. To obtain this result, we construct a self-reduction via a hashing-based gap amplification technique, which turns out useful even outside of the FPT paradigm. Precisely, we rule out any approximation factor of the form $(\log k)^{o(1)}$ for FPT algorithms (assuming FPT $\neq$ W[1]) and $(\log n)^{o(1)}$ for purely polynomial-time algorithms (assuming that the class W[1] does not admit randomized FPT algorithms). This constitutes a novel inapproximability result for polynomial-time algorithms obtained via tools from the FPT theory. Moreover, we prove k-Steiner Orientation to belong to W[1], which entails W[1]-completeness of $(\log k)^{o(1)}$-approximation for k-Steiner Orientation. This provides an example of a natural approximation task that is complete in a parameterized complexity class.

Finally, we apply our technique to the maximization version of directed multicut – Max (k,p)-Directed Multicut – where we are given a directed graph, k terminals pairs, and a budget p. The goal is to maximize the number of separated terminal pairs by removing p edges. We present a simple proof that the problem admits no FPT approximation with factor $\mathcal{O}(k^{\frac{1}{2}-\varepsilon})$ (assuming FPT $\neq$ W[1]) and no polynomial-time approximation with ratio $\mathcal{O}(|E(G)|^{\frac{1}{2}-\varepsilon})$ (assuming NP $\not\subseteq$ co-RP).

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability

Keywords and phrases approximation algorithms, fixed-parameter tractability, hardness of approximation, gap amplification

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.104

Category Track A: Algorithms, Complexity and Games

Related Version A full version of the paper is available at https://arxiv.org/abs/1907.06529.

Funding The main part of the work has been done when the author was a Ph.D. student at the University of Warsaw and it was a part of the project TOTAL, that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 677651). The author was also supported by the Foundation for Polish Science (FNP).

1 Introduction

In the recent years new research directions emerged in the intersection of the two theories aimed at tackling NP-hard problem: parameterized complexity and approximation algorithms. This led to numerous results combining techniques from both toolboxes. The main goal in this area is to obtain an algorithm running in time $f(k) \cdot |I|^{\mathcal{O}(1)}$ for an instance I with

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 104; pp. 104:1–104:19
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

parameter k, that finds a solution of value not worse than α (the approximation factor) times the value of the optimal solution. They are particularly interesting for problems that are both W[1]-hard and at the same time cannot be well approximated in polynomial time [1, 10, 13, 22, 30]. On the other hand, some problems remain resistant to approximation even in this paradigm.

Obtaining polynomial-time approximation lower bounds under the assumption of P $\neq$ NP is challenging, because it usually requires to prove NP-hardness of a *gap problem.* In a gap problem one only needs to distinguish instances with the value of optimal solution at least C_1 from those with this value at most C_2. This provides an argument that one cannot obtain any approximation factor better than the gap, i.e., $\frac{C_1}{C_2}$, as long as P $\neq$ NP.

A road to such lower bounds has been paved by the celebrated PCP theorem [4], which gives an alternative characterization of the class NP. The original complicated proof has been simplified by Dinur [16] via the technique of *gap amplification*: an iterated reduction from a gap problem with a small gap to one with a larger gap. When the number of iterations depends on the input size, this allows us to start the chain of reductions from a problem with no constant gap. However, this is only possible when we can guarantee that the size of all created instances does not grow super-polynomially.

The process of showing approximation lower bounds becomes easier with an additional assumption of the *Unique Games Conjecture* [28], which states that a particular gap version of the Unique Games problem is NP-hard. This makes it possible to start a reduction from a problem with an already relatively large gap. The reductions based on Unique Games Conjecture provided numerous tight approximation lower bounds [5, 23, 31].

A parameterized counterpart of the hardness assumption P $\neq$ NP is FPT $\neq$ W[1], which is equivalent to the statement that k-Clique $\notin$ FPT, that is, k-Clique[1] does not admit an algorithm with running time of the form $f(k) \cdot |I|^{\mathcal{O}(1)}$. Similarly to the classical complexity theory, proving hardness of an approximate task relying only on FPT $\neq$ W[1] is difficult but possible. A recent result stating that the gap version of k-Dominating Set is W[1]-hard (for the gap being any computable function $F(k)$) required gap amplification through a distributed PCP theorem [26].

Again, the task becomes easier when working with a stronger hardness assumption: *Gap Exponential Time Hypothesis*[2] (Gap-ETH) states that there exists $\varepsilon > 0$ so that one requires exponential time to distinguish satisfiable 3-CNF-SAT formulas from those where only a fraction of $(1-\varepsilon)$ clauses can be satisfied at once [17, 34]. Gap-ETH is a stronger assumption than FPT $\neq$ W[1], i.e., the first implies the second, and it sometimes turns out more convenient since it already provides hardness for a problem with a gap. There are many recent examples of using Gap-ETH for showing hardness of parameterized approximation [6, 9, 10, 11, 13, 30].

Our contribution is a novel gap amplification technique which exploits the fact that in a parameterized reduction we can afford an exponential blow-up with respect to the parameter. It circumvents the obstacles related to PCP protocols and, together with a hashing-based lemma, allows us to construct relatively simple self-reductions for problems on directed graphs.

[1] We attach the parameter to the problem name when we refer to a parameterized problem.

[2] Gap-ETH is a stronger version of the *Exponential Time Hypothesis* (ETH), according to which one requires exponential time to solve 3-CNF-SAT [25].

Steiner Orientation

In the k-Steiner Orientation problem we are given a mixed graph, that is, with both directed and undirected edges, and a set of k terminal pairs. The goal is to find an orientation of the undirected edges that maximizes the number of terminal pairs for which there is a path from the source to the sink.

Some of the first studies on the k-Steiner Orientation problem (also referred to as the Maximum Graph Orientation problem) were motivated by modeling protein-protein interactions (PPI) [36] and protein-DNA interactions (PDI) [19, 20]. Whereas PPIs interactions could be represented with undirected graphs, PDIs required introducing mixed graphs. Arkin and Hassin [3] showed the problem to be NP-hard, but polynomially solvable for $k = 2$. This result was generalized by Cygan et al. [15], who presented an $n^{\mathcal{O}(k)}$-time algorithm, which implies that the problem belongs to the class XP (see Section 3) when parameterized by k (cf. [18] for different choices of parameterization).

The k-Steiner Orientation problem has been proved to be W[1]-hard by Pilipczuk and Wahlström [38], which makes it unlikely to be solvable in time $f(k)\cdot|I|^{\mathcal{O}(1)}$. The W[1]-hardness proof has been later strengthened to work on planar graphs and to give a stronger running time lower bound based on ETH [11], which is essentially tight with respect to the $n^{\mathcal{O}(k)}$-time algorithm.

The approximation of Steiner Orientation has been mostly studied on undirected graphs, where the problem reduces to optimization over trees by contracting 2-connected components [15]. Medvedovsky et al. [36] presented an $\mathcal{O}(\log n)$-approximation and actually proved that one can always find an orientation satisfying $\Omega(\frac{k}{\log n})$ terminal pairs. The approximation factor has been improved to $\mathcal{O}(\log n/\log\log n)$ [20] and later to $\mathcal{O}(\log k/\log\log k)$ [15] by observing that one can compress an undirected instance to a tree of size $\mathcal{O}(k)$. A lower bound of $\frac{12}{11} - \varepsilon$ (based on P $\neq$ NP) has been obtained via a reduction from Max Directed Cut [36]. Medvedovsky et al. [36] posed a question of tackling the maximization problem on mixed graphs, which was partially addressed by Gamzu et al. [20] who provided a polylogarithmic approximation in the case where the number of undirected components on each source-sink path is bounded by a constant.

The decision problem whether all the terminal pairs can be satisfied is polynomially solvable when restricting input graphs to be undirected [24], which makes the maximization version fixed-parameter tractable, by simply enumerating all subsets of terminals. The maximization version on mixed graphs is far less understood from the FPT perspective. It is unlikely to be exactly solvable since the decision problem is W[1]-hard, but can we approximate it within a reasonable factor? The reduction by Chitnis et al. [11] implies that, assuming Gap-ETH, k-Steiner Orientation cannot be approximated within factor $\frac{20}{19} - \varepsilon$ on mixed graphs, in running time $f(k) \cdot n^{\mathcal{O}(1)}$. Using new techniques introduced in this paper, we are able to provide stronger lower bounds based on a weaker assumption.

Related work

Some examples of the new advancements in parameterized approximations are 1.81-approximation for k-Cut [22] (recently improved to $(\frac{5}{3} + \varepsilon)$ [27]), which beats the factor 2 that is believed to be optimal within polynomial running time [33], or $(1 + \frac{2}{e} + \varepsilon)$-approximation for k-Median [13], all running in time $f(k) \cdot n^{\mathcal{O}(1)}$. For Capacitated k-Median, a constant factor FPT approximation has been obtained [1, 14], whereas the best-known polynomial-time approximation factor is $\mathcal{O}(\log k)$. Another example is an FPT approximation scheme for the planar case of Bidirected Steiner Network, which does not admit a polynomial-time approximation scheme unless P = NP [10].

On the other hand several problems have proven resistant to such improvements. Chalermsook et al. [9] showed that under the assumption of Gap-ETH there can be no parameterized approximations with ratio $o(k)$ for k-Clique or k-Biclique and none with ratio $F(k)$ for k-Dominating Set (for any computable function F). They have also ruled out $k^{o(1)}$-approximation for Densest k-Subgraph. The cited FPT approximation for k-Median has a tight approximation factor assuming Gap-ETH [13].

Subsequently, efforts have been undertaken to weaken the complexity assumptions on which the lower bounds are based. For the k-Dominating Set problem Gap-ETH has been replaced with a more established hardness assumption that FPT $\neq$ W[1] [26]. Marx [35] has proven parameterized inapproximability of Monotone k-Circuit SAT under the even weaker assumption that FPT $\neq$ W[P]. Lokshtanov et al. [30] introduced the *Parameterized Inapproximability Hypothesis* (PIH), that is weaker than Gap-ETH and stronger than FPT $\neq$ W[1], and used it to rule out an FPT approximation scheme for Directed k-Odd Cycle Transversal. PIH turned out to be a sufficient assumption to argue there can be no FPT algorithm for k-Even Set [6].

2 Overview of the results

Our main inapproximability result is a W[1]-hardness proof for the gap version of k-Steiner Orientation with the gap $q = (\log k)^{o(1)}$. This means that the problem is unlikely to admit a $(\log k)^{o(1)}$-approximation algorithm with running time $f(k) \cdot |I|^{\mathcal{O}(1)}$.

▶ **Theorem 2.1.** *Consider a function $\alpha(k) = (\log k)^{\beta(k)}$, where $\beta(k) \to 0$ is computable and non-increasing. It is* W[1]*-hard to distinguish whether for a given instance of k-Steiner Orientation:*

1. *there exists an orientation satisfying all k terminal pairs, or*
2. *for all orientations the number of satisfied pairs is at most $\frac{1}{\alpha(k)} \cdot k$.*

The previously known approximation lower bound for FPT algorithms, $\frac{20}{19} - \varepsilon$, was obtained via a linear reduction from k-Clique and was based on Gap-ETH [11]. Our reduction not only raises the inapproximability bar significantly, but also weakens the hardness assumption (although we are not able to enforce the planarity of the produced instances, as in [11]). In fact, we begin with the decision version of k-Steiner Orientation and introduce a gap inside the self-reduction. What is interesting, we rely on totally different properties of the problem than in the W[1]-hardness proof [38]: that one required gadgets with long undirected paths and we introduce only new directed edges.

This result is also interesting from the perspective of the classical (non-parameterized) approximation theory. The best approximation lower bound known so far has been $\frac{12}{11} - \varepsilon$ [36], valid also for undirected graphs. Therefore we provide a new inapproximability result for polynomial algorithms, which is based on an assumption from parameterized complexity. Restricting to a purely polynomial running time allows us to rule out also approximation factors depending on n (rather than on k) with a slightly stronger assumption, which is required because the reduction is randomized (see Section 3 for the formal definition of a false-biased FPT algorithm).

▶ **Theorem 2.2.** *Consider a function $\alpha(n) = (\log n)^{\beta(n)}$, where $\beta(n) \to 0$ is computable and non-increasing. Unless the class W[1] admits false-biased FPT algorithms, there is no polynomial-time algorithm that, given an instance of Steiner Orientation with n vertices and k terminal pairs, distinguishes between the following cases:*

1. *there exists an orientation satisfying all k terminal pairs, or*
2. *for all orientations the number of satisfied pairs is at most $\frac{1}{\alpha(n)} \cdot k$.*

A similar phenomenon, that is, novel polynomial-time hardness based on an assumption from parameterized complexity, has appeared in the work on Monotone k-Circuit SAT [35]. Another example of this kind is polynomial-time approximation hardness for Densest k-Subgraph based on ETH [32].

W[1]-completeness

So far, the decision version of k-Steiner Orientation has only been known to be W[1]-hard [38] and to belong to XP [15]. We establish its exact location in the W-hierarchy. A crucial new insight is that we can assume the solution to be composed of $f(k)$ pieces, for which we only need to check if they match each other, and this task reduces to k-Clique.

▶ **Theorem 2.3.** *k-Steiner Orientation is* W[1]*-complete.*

We hereby solve an open problem posted by Chitnis et al. [11]. What is more, this implies that $(\log k)^{o(1)}$-Gap k-Steiner Orientation belongs to W[1] (see Section 3 for formal definitions of problems). Together with Theorem 2.1 this entails W[1]-completeness. Another gap problem with this property is Maximum k-Subset Intersection[3], introduced for the purpose of proving W[1]-hardness of k-Biclique [29]. We are not aware of any other natural gap problem being complete in a parameterized complexity class. Note that although W[1]-hardness of the gap version of k-Dominating Set is known [26], k-Dominating Set is W[2]-complete.

Directed Multicut

As another application of our technique, we present a simple hardness result for the gap version of Max (k,p)-Directed Multicut with the gap $q = k^{\frac{1}{2}-\varepsilon}$. We show that even if we parameterize the problem with both the number of terminal pairs k and the size of the cutset p, then we essentially cannot obtain any approximation ratio better than $\sqrt{k}$.

▶ **Theorem 2.4.** *For any $\varepsilon > 0$ and function $\alpha(k) = \mathcal{O}\left(k^{\frac{1}{2}-\varepsilon}\right)$, it is* W[1]*-hard to distinguish whether for a given instance of Max (k,p)-Directed Multicut:*

1. *there is a cut of size p that separates all k terminal pairs, or*
2. *all cuts of size p separate at most $\frac{1}{\alpha(k)} \cdot k$ terminal pairs.*

When restricted to polynomial running time, the lower bound of $\Omega(k^{\frac{1}{2}-\varepsilon})$ can be improved to $\Omega(|E(G)|^{\frac{1}{2}-\varepsilon})$, however unlike the case of k-Steiner Orientation, this time the reduction is polynomial and we need to assume only NP $\not\subseteq$ co-RP (recall that a problem is in co-RP if it admits a polynomial-time false-biased algorithm, i.e., an algorithm which is always correct for YES-instances and for NO-instances returns the correct answer with probability greater than some constant).

▶ **Theorem 2.5.** *Assuming* NP $\not\subseteq$ co-RP*, for any $\varepsilon > 0$ and function $\alpha(m) = \mathcal{O}\left(m^{\frac{1}{2}-\varepsilon}\right)$, there is no polynomial-time algorithm that, given an instance $(G, \mathcal{T}, p)$, $|\mathcal{T}| = k$, $|E(G)| = m$, of Max Directed Multicut, distinguishes between the following cases:*

1. *there is a cut of size p that separates all k terminal pairs, or*
2. *all cuts of size p separate at most $\frac{1}{\alpha(m)} \cdot k$ terminal pairs.*

[3] To give a concrete example of a W[1]-complete gap problem, consider the task of distinguishing graphs with $K_{k,F_1(k)}$ from those with no $K_{k,F_2(k)}$. The reduction in [29] implies that this is W[1]-hard for functions F_1, F_2 with large gap. On the other hand, the problem of finding $K_{k,F_1(k)}$ belongs to W[1] via a color-coding reduction to $(F_1(k)+k)$-Clique.

As far as we know, the approximation status of this variant has not been studied yet. If we want to minimize the number of removed edges to separate all terminal pairs or minimize the ratio of the cutset size to the number of separated terminal pairs (this problem is known as Directed Sparsest Multicut), those cases admit a polynomial-time $\tilde{\mathcal{O}}(n^{\frac{11}{23}})$-approximation algorithm [2] and a lower bound of $2^{\Omega(\log^{1-\varepsilon} n)}$ [12]. Since $\frac{11}{23} < \frac{1}{2}$ and $n \le m$, the maximization variant with a hard constraint on the cutset size turns out to be harder.

In the undirected case p-Multicut is FPT, even when parameterized only by the size of the cutset p and allowing arbitrarily many terminals [7]. This is in contrast with the directed case, which becomes W[1]-hard already for 4 terminals, when parameterized by p. It is worth mentioning that k-Steiner Orientation and p-Directed Multicut were proven to be W[1]-hard with a similar gadgeting machinery [38].

Organization of the paper

We begin with the necessary definitions in Section 3. As our gap amplification technique is arguably the most innovative ingredient of the paper, we precede the proofs with informal Section 4, which introduces the ideas gradually. It is followed by the detailed constructions for k-Steiner Orientation in Section 5 and for Max (k, p)-Directed Multicut in Section 6. Each contains a self-reduction lemma and applications to polynomial and FPT running time. The proof of W[1]-completeness of k-Steiner Orientation can be found in the full version of the article.

3 Preliminaries

Fixed parameter tractability

A parameterized problem instance is created by associating an integer parameter k with an input instance. Formally, a parameterized language is a subset of $\Sigma^* \times \mathbb{N}$. We say that a language (or a problem) is *fixed parameter tractable* (FPT) if it admits an algorithm solving an instance (I, k) (i.e., deciding if it belongs to the language) in running time $f(k) \cdot |I|^{\mathcal{O}(1)}$, where f is a computable function. Such a procedure is called an *FPT algorithm* and we say concisely that it runs in *FPT time*. A language belongs to the broader class XP if it admits an algorithm with running time of the form $|I|^{f(k)}$.

There is no widely recognized class describing problems which admit randomized FPT algorithms. Instead of defining such a class, we will directly use the notion of a *false-biased* algorithm, which is always correct for YES-instances and for NO-instances returns the correct answer with probability greater than some constant (equivalently, when the algorithm returns *false* then it is always correct). Similarly, a *true-biased* algorithm is always correct for NO-instances but may be wrong for YES-instances with bounded probability. A false-biased (resp. true-biased) FPT algorithm satisfies the condition above and runs in FPT time.

To argue that a problem is unlikely to be FPT, we use parameterized reductions analogous to those employed in the classical complexity theory. Here, the concept of W-hardness replaces NP-hardness, and we need not only to construct an equivalent instance in FPT time, but also ensure that the parameter in the new instance depends only on the parameter in the original instance. If there exists a parameterized reduction from a W[1]-hard problem (e.g., k-Clique) to another problem Π, then the problem Π is W[1]-hard as well. This provides an argument that Π does not admit an algorithm with running time $f(k) \cdot |I|^{\mathcal{O}(1)}$ under the assumption that FPT $\neq$ W[1].

Approximation algorithms and gap problems

We define an optimization problem (resp. parameterized optimization problem) as a task of optimizing function $L \to \mathbb{N}$, where $L \subseteq \Sigma^*$ (resp. $L \subseteq \Sigma^* \times \mathbb{N}$), representing the value of the optimal solution. An α-approximation algorithm for a maximization task must return a solution of value no less than the optimum divided by α (we follow the convention with $\alpha \ > 1$). The approximation factor α can be a constant or it can depend on the input size. In the most common setting, the running time is required to be polynomial. An FPT approximation algorithm works with a parameterized optimization problem and is required to run in FPT time. It is common that its approximation factor can depend on the parameter.

A gap problem (resp. parameterized gap problem) is given by two disjoint languages $L_1, L_2 \subseteq \Sigma^*$ (resp. $L_1, L_2 \subseteq \Sigma^* \times \mathbb{N}$). An algorithm should decide whether the input belongs to L_1 or to L_2. If neither holds, then the algorithm is allowed to return anything. Usually L_1, L_2 are defined respectively as the sets of instances (of an optimization problem) with a solution of value at least C_1 and instances with no solution with value greater than C_2. An α-approximation algorithm with $\alpha \ < \frac{C_1}{C_2}$ can distinguish L_1 from L_2, therefore hardness of an approximation task is implied by hardness for the related gap problem.

Problem definitions

We now formally describe the problems we work with. Since we consider parameterized algorithms it is important to specify how we define the parameter of an instance.

A mixed graph is a triple (V, A, E), where V is the vertex set, A is the set of directed edges, and E stands for the set of undirected edges. An orientation of a mixed graph is given by replacing each undirected edge $uv \in E$ with one of the directed ones: (u, v) or (v, u). This creates a directed graph $\tilde{G} = (V, A \cup \tilde{E})$, where $\tilde{E}$ is the set of newly created directed edges. We assume that $uv \notin E$ for each $(u, v) \in A$, so $\tilde{G}$ is always a simple graph.

k-Steiner Orientation

Input: mixed graph $G = (V, A, E)$, list of terminal pairs $\mathcal{T} = ((s_1, t_1), \ldots, (s_k, t_k))$,
Parameter: k
Task: find an orientation $\tilde{G}$ of G that maximizes the number of pairs (s_i, t_i), such that t_i is reachable from s_i in $\tilde{G}$

We add „Max" to the name of the following problem in order to distinguish it from the more common version of Directed Multicut, where one minimizes the number of edges in the cut.

Max (k, p)-Directed Multicut

Input: directed graph $G = (V, A)$, list of terminal pairs $\mathcal{T} = ((s_1, t_1), \ldots, (s_k, t_k))$, integer p
Parameter: $k + p$
Task: find a subset of edges $A' \subseteq A$, $|A'| \leq p$, in order to maximize the number of pairs (s_i, t_i), such that t_i is unreachable from s_i in $G \setminus A'$

If a solution to either problem satisfies the reachability (resp. unreachability) condition for a particular terminal pair, we say that this pair is satisfied by this solution. The decision versions of both problems ask whether there is a solution of value k, that is, satisfying all the terminal pairs. We call such an instance fully satisfiable, or a YES-instance, and a NO-instance otherwise. For the sake of proving approximation hardness we introduce the gap versions: q-Gap k-Steiner Orientation and q-Gap Max (k,p)-Directed Multicut, where we are promised that the value of the optimal solution is either k or at most $\frac{k}{q}$, and we have to distinguish between these cases.

When referring to non-parameterized problems, we drop the parameters in the problem name. We use notation $[n] = \{1, 2, \dots, n\}$. All logarithms are 2-based.

4 The gap amplification technique

We begin with an informal thought experiment that helps to understand the main ideas behind the reduction. For an instance $(G, \mathcal{T} = ((s_1, t_1), \dots, (s_k, t_k)))$ of k-Steiner Orientation we refer to the vertices $s_i, t_i \in V(G)$ as $G\{s, i\}$ and $G\{t, i\}$. We want to construct a larger instance $(H, \mathcal{T}_H)$ so that if $(G, \mathcal{T})$ is fully satisfiable then $(H, \mathcal{T}_H)$ is as well, but otherwise the maximal fraction of satisfiable pairs in $(H, \mathcal{T}_H)$ is strictly less than $\frac{k-1}{k}$. Consider k vertex-disjoint copies of the original instance: $(G_1, \mathcal{T}_1), (G_2, \mathcal{T}_2), \dots, (G_k, \mathcal{T}_k)$, that will be treated as the first layer. Assume that $(G, \mathcal{T})$ is a NO-instance (i.e., one cannot satisfy all pairs at once), so for any orientation of the copies $\tilde{G}_1, \tilde{G}_2, \dots, \tilde{G}_k$, there is a tuple $(j_1, j_2, \dots, j_k)$ such that $\tilde{G}_i\{t, j_i\}$ is unreachable from $\tilde{G}_i\{s, j_i\}$ in $\tilde{G}_i$. Suppose for now that we have fixed the values of (j_i), even before we have finished building our instance.

Let $R = (r_1, r_2, \dots, r_k)$ be a tuple sampled randomly from $[k]^k$. We connect the sinks in the first layer to the sources in another copy of the same instance – let us refer to it as $(G_R, \mathcal{T}_R)$. We add a directed edge from $G_i\{t, r_i\}$ to $G_R\{s, i\}$ for each $i \in [k]$, thus connecting a random sink of G_i to the source $G_R\{s, i\}$, as shown in Figure 1. We refer to the union of all $k+1$ copies of G with k added connecting edges as the graph H. We define $\mathcal{T}_H = ((G_1\{s, r_1\}, G_R\{t, 1\}), \dots, (G_k\{s, r_k\}, G_R\{t, k\}))$, so we want to satisfy those k terminal pairs that got connected randomly. Let X be a random variable equal to the value of the optimal solution for $(H, \mathcal{T}_H)$ under the restriction that the solution orients $G_1, G_2 \dots, G_k$ as $\tilde{G}_1, \tilde{G}_2, \dots, \tilde{G}_k$. What would be the expected value of X?

Let Y denote another random variable being the number of indices i for which $r_i \neq j_i$. By linearity of expectation we have $\mathbb{E}Y = k - 1$. It holds that $X \leq Y$ and so far we still have only the bound $\mathbb{E}X \leq k - 1$. However, with probability $(\frac{k-1}{k})^k$ we have $r_i \neq j_i$ for all i, therefore $Y = k$, but we cannot connect all pairs within G_R (because it is a copy of a NO-instance), so $X \leq k - 1$. This means that $\mathbb{E}(Y - X) \geq (\frac{k-1}{k})^k$ and so $\mathbb{E}X \leq k - 1 - (\frac{k-1}{k})^k$: the gap has been slightly amplified.

Of course in the proper reduction we cannot fix the orientation before adding the connecting edges. However, we can afford an exponential blow-up with respect to k. We can include in the second layer the whole probabilistic space, that is, k^k copies of $(G, \mathcal{T})$ (rather than a single $(G_R, \mathcal{T}_R)$), each connected to the first layer with respect to a different tuple $(r_1, r_2, \dots, r_k)$, thus creating a large instance $(H, \mathcal{T}_H)$ with $k^k \cdot k$ terminal pairs (see Figure 1). For any orientation of H the fraction of satisfied terminal pairs equals the average over the fractions for all k^k groups of terminal pairs, so we can emulate the construction above without fixing $(j_1, j_2, \dots, j_k)$. The maximal fraction of satisfiable terminal pairs in such (no longer random) $(H, \mathcal{T}_H)$ would be the same as before, that is, $\frac{\mathbb{E}X}{k} < k - 1$. However, the smaller instance we create the better lower bounds we get, so we will try to be more economical while constructing $(H, \mathcal{T}_H)$.

An important observation is that we do not have to include all k^k choices of R in the construction. We just need a sufficient combination of them, so that the gap amplification occurs for any choice of $(j_1, j_2, \dots, j_k)$. This can be ensured by picking just $k^{\mathcal{O}(1)}$ choices of R and using an argument based on the Chernoff bound (see Section 5 for a detailed construction).

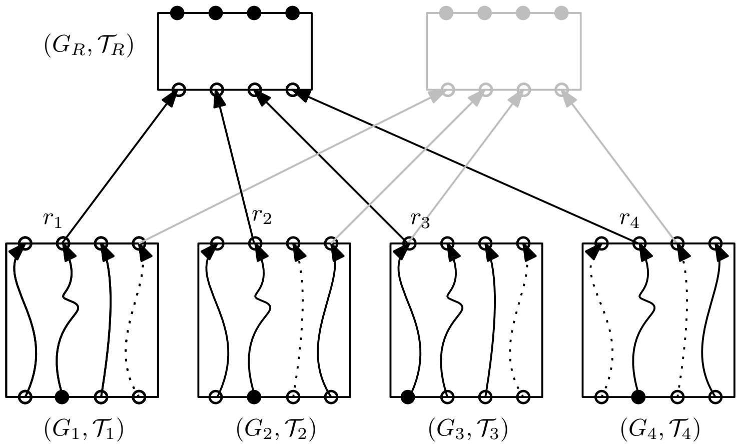


Figure 1 The construction of $(H, \mathcal{T}_H)$ for $k = 4$. The black copy $(G_R, \mathcal{T}_R)$ is connected according to the choice of $R = (r_1, r_2, r_3, r_4) = (2, 2, 1, 2)$ and the terminals (sources and sinks) are marked with black dots. The dotted lines indicate which pairs of terminals are unreachable in each $\tilde{G}_i$ and we can set $(j_1, j_2, j_3, j_4) = (4, 3, 4, 3)$ (another valid choice is $(4, 3, 4, 1)$). We have $r_i \neq j_i$ for each i, so $Y = 4$ but $X \leq 3$. The grey copy illustrates another random choice of connection: $R' = (r_1, r_2, r_3, r_4) = (4, 4, 1, 3)$ and $X \leq Y = 2$. If we wanted to construct $(H, \mathcal{T}_H)$ emulating the whole probabilistic space, we would include all 4^4 copies of $(G_R, \mathcal{T}_R)$ for all choices of $R = (r_1, r_2, r_3, r_4)$ in the same way as the black and the grey copy.

We can iterate this construction by treating $(H, \mathcal{T}_H)$ as a new input and amplifying the gap further in each step. In further steps we need to add an exponential number of copies to the new layer, even when compressing the probabilistic space as above. This is why we get an exponential blow-up with respect to k and we need to work with a parameterized hardness assumption, even for ruling out polynomial-time approximations.

The construction for Max (k, p)-Directed Multicut is simpler because the layer stacking does not have to be iterated. Therefore to achieve polynomial-time hardness it suffices to assume that NP $\not\subseteq$ co-RP. The phenomenon that both problems admit such strong self-reduction properties can be explained by the fact that when dealing with directed reachability one can compose instances sequentially, which is the first step in both reductions.

5 Inapproximability of Steiner Orientation

We are going to present a formal construction of the argument sketched in Section 4. We first formulate and discuss the properties of the construction. Then we introduce a probabilistic tool, called a δ-biased sampler family, describe the reduction step, and prove the gap amplifying property. At the end of this section we present the proof of the following lemma. Let $S(G, \mathcal{T})$ denote the maximal number of pairs that can be satisfied by some orientation in an instance $(G, \mathcal{T})$.

▶ **Lemma 5.1.** *There is a procedure that, for an instance $(G, \mathcal{T}_G)$, $k = |\mathcal{T}_G|$, of Steiner Orientation, and a parameter q, constructs a new instance $(H, \mathcal{T}_H)$, $k_0 = |\mathcal{T}_H|$, such that:*
1. $k_0 = 2^{q^{\mathcal{O}(k)}}$,
2. $|V(H)| \leq |V(G)| \cdot k_0^2$,
3. *if $S(G, \mathcal{T}_G) = k$, then $S(H, \mathcal{T}_H) = k_0$ always (Completeness),*
4. *if $S(G, \mathcal{T}_G) < k$, then $S(H, \mathcal{T}_H) \leq \frac{1}{q} \cdot k_0$ with probability at least $\frac{1}{k_0}$ (Soundness).*

The randomized construction runs in time proportional to $|H|$. It can be derandomized within running time $f(k, q) \cdot |G|$.

It easily follows from these properties that the gap can get amplified to any constant q. It is more complicated though to rule out a superconstant approximation factor, e.g., $\alpha(k) = \log\log k$, because we need to keep track of the growth of $\alpha(k_0)$ when increasing q. We address this issue after proving Lemma 5.1.

Sampler families

As sketched in Section 4, given fixed orientations of the k copies of G, we are able to randomly sample k sinks and insert additional edges so that the expected optimum of the new instance is sufficiently upper bounded. We want to reverse this idea, so we could randomly sample a moderate number of additional connections once to ensure the upper bound works for any orientation. To this end, we need some kind of a hashing technique to mimic the behaviour of the probabilistic space with a structure of moderate size. Examples of such constructions are (generalized) universal hash families [8, 39, 40] or expander random walk sampling [21]. Even though the construction presented below is relatively simple, we are not aware of any occurrences of it in the literature.

For a set X_1 and a multiset X_2, we write $X_2 \subseteq X_1$ if every element from X_2 appears in X_1. Let $U(X)$ denote the uniform distribution over a finite multiset X. In particular, each distinct copy of the same element in X has the same probability of being chosen: $\frac{1}{|X|}$.

▶ **Definition 5.2.** *For a family $\mathcal{F}$ of functions $X \to [0,1]$, a δ-biased sampler family is a multiset $X_H \subseteq X$, such that for every $f \in \mathcal{F}$ it holds*

$$\left|\mathbb{E}_{x\sim U(X_H)} f(x) - \mathbb{E}_{x\sim U(X)} f(x)\right| \le \delta.$$

▶ **Lemma 5.3.** *For a given X, $\mathcal{F}$, and $\delta\ > 0$, a sample of $\mathcal{O}(\delta^{-2}\log(|\mathcal{F}|))$ elements from X (sampled independently with repetitions) forms a δ-biased sampler family with probability at least $\frac{1}{2}$.*

Proof. We sample independently $M = 10\cdot\delta^{-2}\log(|\mathcal{F}|)$ elements from X with repetitions. For the sake of analysis, note that this is a single sample from the space $\Omega_{X,M}$ being the family of all M-tuples of elements from X, equipped with a uniform distribution. Let X_H denote the random multiset of all elements in this M-tuple. For each $f \in \mathcal{F}$ we define $A_f \subseteq \Omega_{X,M}$ as the family of tuples for which $\left|\mathbb{E}_{x\sim U(X_H)} f(x) - \mathbb{E}_{x\sim U(X)} f(x)\right| > \delta$. For a fixed f we apply the Hoeffding's inequality.

$$\mathbb{P}(A_f) = \mathbb{P}\left(\left|\mathbb{E}_{x\sim U(X_H)} f(x) - \mathbb{E}_{x\sim U(X)} f(x)\right| > \delta\right) \le 2\exp(-2\delta^2 M).$$

For our choice of M this bound gets less than $\frac{1}{2|\mathcal{F}|}$. By union bound, the probability that X_F is not a δ-biased sampler family is $\mathbb{P}\left(\bigcup_{f\in\mathcal{F}} A_f\right) \le \sum_{f\in\mathcal{F}} \mathbb{P}(A_f) \le \frac{1}{2}$. The claim follows. ◀

We keep the concise notation from Section 4: for an instance $(G,\mathcal{T})$, $\mathcal{T} = ((s_1,t_1),\dots,(s_k,t_k))$ of k-Steiner Orientation we refer to the vertices $s_i, t_i \in V(G)$ as $G\{s,i\}$ and $G\{t,i\}$.

Building the layers

Given an instance $(G,\mathcal{T}_G)$, our aim is to build a larger instance, so that if $S(G,\mathcal{T}) = k$ then the new one is also fully satisfiable, but otherwise the maximal fraction of terminal pairs being simultaneously satisfiable in the new instance is at most $\frac{1}{q}$.

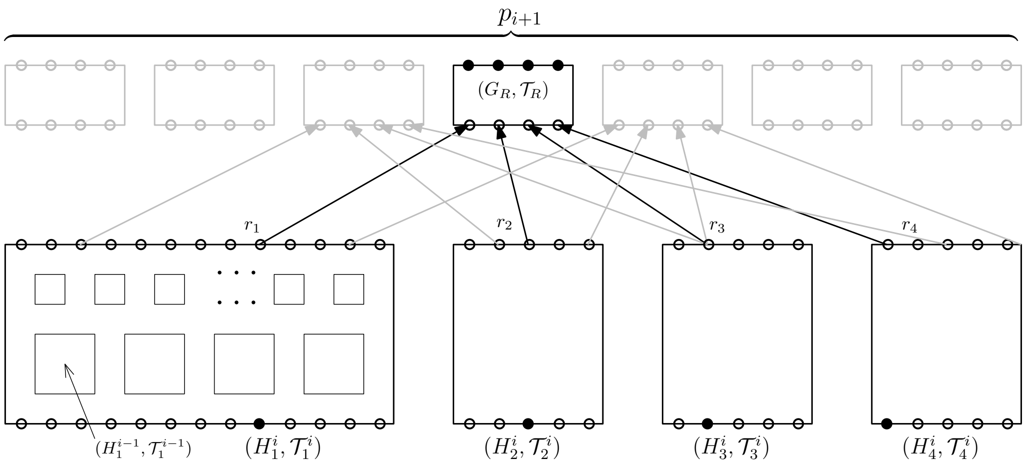

p_{i+1}
$(G_R, \mathcal{T}_R)$
r_1
r_2
r_3
r_4
$(H_1^{i-1}, \mathcal{T}_1^{i-1})$
$(H_1^i, \mathcal{T}_1^i)$
$(H_2^i, \mathcal{T}_2^i)$
$(H_3^i, \mathcal{T}_3^i)$
$(H_4^i, \mathcal{T}_4^i)$

Figure 2 The construction of $(H^{i+1}, \mathcal{T}^{i+1})$ for $k = 4$. The first copy of $(H^i, \mathcal{T}^i)$ is shown in greater detail to highlight the recursive construction, with circles representing its terminal pairs. The new layer consists of p_{i+1} copies of $(G, \mathcal{T})$. The vector $R = (r_1, r_2, r_3, r_4) \in [p_i]^4$ indicates which sinks are connected to the sources in $(G_R, \mathcal{T}_R)$. The black dots show 4 terminal pairs associated with this copy. For the sake of legibility only the edges incident to G_R (the black ones) and neighboring copies (the grey ones) are sketched.

We inductively construct a family of instances $(H^i, \mathcal{T}^i)_{i=1}^B$ with $(H^1, \mathcal{T}^1) = (G, \mathcal{T}_G)$. Let $k_i = |\mathcal{T}^i|$ and p_i indicate the number of copies of $(G, \mathcal{T}_G)$ in the last layer (to be explained below) of $(H^i, \mathcal{T}^i)$. We will have that $p_1 = 1$ and $k_i = |\mathcal{T}^i| = k \cdot p_i$. We construct $(H^{i+1}, \mathcal{T}^{i+1})$ by taking k vertex-disjoint copies of the i-th instance, denoted $(H_1^i, \mathcal{T}_1^i), \ldots, (H_k^i, \mathcal{T}_k^i)$ and forming a new layer of copies of $(G, \mathcal{T}_G)$ which will be randomly connected to the i-th layer through directed edges. Therefore graph H^{i+1} will have $i + 1$ layers of copies of G.

Let $\mathcal{R} = [k_i]^k$ be the family of k-tuples $(r_1, r_2, \ldots r_k)$ with elements from the set $[k_i]$. We sample a random tuple $R = (r_1, r_2, \ldots r_k)$ from $\mathcal{R}$ and create a new copy of the original instance – let us refer to it as $(G_R, \mathcal{T}_R)$. We add a directed edge from $H_j^i\{t, r_j\}$ to $G_R\{s, j\}$ for each $j \in [k]$, thus connecting a random sink of H_j^i to the source $G_R\{s, i\}$. We insert k pairs to $\mathcal{T}^{i+1}$: $(H_1^i\{s, r_1\}, G_R\{t, 1\}), \ldots, (H_k^i\{s, r_k\}, G_R\{t, k\})$. We iterate this subroutine $p_{i+1} = \mathcal{O}(k^4 q^{2k} p_i)$ times (a derivation of this quantity is postponed to Lemma 5.4), as shown in Figure 2.

▶ **Lemma 5.4.** *Let $y_i = S(H^i, \mathcal{T}^i) \,/\, k_i$ be the maximal fraction of terminal pairs that can be simultaneously satisfied in $(H^i, \mathcal{T}^i)$. Suppose that $S(G, \mathcal{T}_G) < k$ and $y_i \geq \frac{1}{q}$. Then with probability at least $\frac{1}{2}$ it holds $y_{i+1} \leq y_i - \frac{1}{2k} \cdot q^{-k}$.*

Proof. First observe that for each $(s_j, t_j) \in \mathcal{T}^i$, any (s_j, t_j)-path in $(H^i, \mathcal{T}^i)$ runs through i unique copies of $(G, \mathcal{T}_G)$. Therefore an (s_j, t_j)-pair is satisfied only if the corresponding i terminal pairs in those copies are satisfied. Recall that we have connected each copy of $(G, \mathcal{T}_G)$ in the i-th layer to the terminals from the previous layer according to a random tuple $R = (r_1, r_2, \ldots r_k) \in \mathcal{R}$.

We will now analyze how the possible orientations $\tilde{H}_1^i, \ldots, \tilde{H}_k^i$ influence the status of the terminal pairs in $\mathcal{T}^{i+1}$. Let $C_j \subseteq [k_i]$ encode which of the terminal pairs are reachable in $\tilde{H}_j^i$, that is, $C_j = \{\ell \in [k_i] \colon \tilde{H}_j^i\{t, \ell\}$ is reachable from $\tilde{H}_j^i\{s, \ell\}\}$. A tuple $C = (C_1, \ldots C_k)$ is called a *configuration* and we denote the family of all feasible configurations as $\mathcal{C}$. We have $|\mathcal{C}| \leq (2^{k_i})^k = 2^{p_i k^2}$. For a configuration $C \in \mathcal{C}$ let $f_C : \mathcal{R} \to [0, 1]$ be a function describing the maximal fraction of satisfiable terminal pairs from those with sinks in G_R connected through a tuple $R \in \mathcal{R}$. Note that $f_C(R)$ depends on C, R and the best possible orientation of G_R, whereas it is oblivious to the rest of the structure of $(\tilde{H}^i_j)_{j=1}^k$. We can thus think of C as an interface between the first i layers and G_R.

For fixed orientations $\tilde{H}^i_1, \dots, \tilde{H}^i_k$, and therefore fixed configuration $C \in \mathcal{C}$, we estimate the expected value of f_C. Let Y_j be a random Boolean variable indicating that $\tilde{H}^i_j[t, r_j]$ is reachable from $\tilde{H}^i_j[s, r_j]$ for a random $(r_1, r_2, \dots r_k) \in \mathcal{R}$, i.e., that $r_j \in C_j$. Let $Y = \sum_{j=1}^k Y_j/k$, so that we always have $f_C(R) \le Y$.

Let $c_j = |C_j|/k_i$. By linearity of expectation $\mathbb{E}Y = \sum_{j=1}^k c_j/k$ and by the assumption $c_j \le y_i$ for all $j \in [k]$. However, with probability $\prod_{j=1}^k c_j$ we have $r_j \in C_j$ for all j, so $Y = 1$, but we cannot connect all pairs within $(G_R, \mathcal{T}_R)$ (since we assumed $S(G, \mathcal{T}_G) < k$), so $f_C(R) \le 1 - \frac{1}{k}$. This means that

$$\mathbb{E}\left(Y - f_C(R)\right) \ge \frac{1}{k} \cdot \prod_{j=1}^k c_j, \quad \text{and so} \quad \mathbb{E}f_C(R) \le \frac{1}{k} \cdot \left(\sum_{j=1}^k c_j - \prod_{j=1}^k c_j \right).$$

The quantity $\sum_{j=1}^k c_j - \prod_{j=1}^k c_j$ can only increase when we increase some c_ℓ, because the ℓ-th partial derivative is $1 - \prod_{j \in [k],\, j \neq \ell} c_j \ge 0$. By the assumption $c_j \le y_i$ and $y_i \ge \frac{1}{q}$, hence

$$\mathbb{E}f_C(R) \le \frac{1}{k} \cdot \left(\sum_{j=1}^k y_i - \prod_{j=1}^k y_i \right) \le y_i - \frac{1}{k} \cdot q^{-k}.$$

Now we apply Lemma 5.3 for $\mathcal{F} = \{f_C : C \in \mathcal{C}\}$ and $\delta = \frac{1}{2k} \cdot q^{-k}$ to argue that for our choice of p_{i+1} – the number of copies in the last layer – the estimation works for all $C \in \mathcal{C}$ at once. The quantity $M = \mathcal{O}(\delta^{-2} \log(|\mathcal{F}|))$ in Lemma 5.3 becomes $\mathcal{O}(k^4 q^{2k} p_i)$, which is exactly as we defined p_{i+1}. We have sampled p_{i+1} tuples from $\mathcal{R}$ (let us denote this multiset as $\mathcal{R}_H \subseteq \mathcal{R}$) and added a copy $(G_R, \mathcal{T}_R)$ for each $R \in \mathcal{R}_H$. For a fixed $C \in \mathcal{C}$, the maximal fraction of satisfiable terminal pairs in $(H^{i+1}, \mathcal{T}^{i+1})$ equals the average of $f_C(R)$ over $R \in \mathcal{R}_H$. By Lemma 5.3 we know that, regardless of the choice of C, this quantity is at most

$$\mathbb{E}_{R \sim U(\mathcal{R}_H)} f_C(R) \le \mathbb{E}_{R \sim U(\mathcal{R})} f_C(R) + \frac{1}{2k} \cdot q^{-k} \le y_i - \frac{1}{2k} \cdot q^{-k},$$

with probability at least $\frac{1}{2}$ (that is, if we have chosen $\mathcal{R}_H$ correctly). Since the upper bound works for all $C \in \mathcal{C}$ simultaneously, the claim follows. ◀

Proof of Lemma 5.1. We define $(H, \mathcal{T}_H) = (H^B, \mathcal{T}^B)$ for $B = 2kq^k$. The completeness is straightforward: if $S(G, \mathcal{T}_G) = k$, then we can orient all copies of G so that $G\{t, j\}$ is always reachable from $G\{s, j\}$ and each requested path in $S(H^i, \mathcal{T}^i)$ is given as a concatenation of respective paths in B copies of G.

To see the soundness, suppose that $S(G, \mathcal{T}_G) < k$. The sequence $(y_i)_{i=0}^B$ is non-increasing and the value of y_i is being decreased by at least $\frac{i}{2k} \cdot q^{-k}$ in each iteration, as long as $y_i \ge \frac{1}{q}$, due to Lemma 5.4. Therefore after $B = 2kq^k$ iterations we are sure to have $y_B \le \frac{1}{q}$.

To estimate the size of the instance, recall that we have $k_i = p_i k$ and $p_{i+1} = \mathcal{O}(k^4 q^{2k} p_i)$. We can assume $q \ge 2$ and so $k \le q^k$. For $B = 2kq^k$, the value of p_B becomes $\left(k^4 q^{2k}\right)^{\mathcal{O}(kq^k)} = 2^{q^{\mathcal{O}(k)} \mathcal{O}(\log q)} = 2^{q^{\mathcal{O}(k)}}$. The size of $V(H)$ is at most $k_B \cdot |V(G)|$ times the number of layers, which is B. We trivially bound $B \le 2^B \le k_B$ to obtain $|V(H)| \le |V(G)| \cdot k_B^2$.

The presented construction is randomized because we randomly choose a biased sampler family in each of the B steps. If we start with a YES-instance, then we produce a YES instance regardless of these choices, and otherwise we produce a NO instance with probability at least $2^{-B} \geq \frac{1}{k_B}$. The construction can be derandomized within running time $f(k,q) \cdot |V(G)|$ as follows. In each application of Lemma 5.3 the sizes of X and $\mathcal{F}$ are $(k_i)^k$ and $2^{p_i k^2}$, respectively, and $\delta = \frac{1}{2k} \cdot q^{-k}$, which are all bounded by a function of k and q. Therefore instead of sampling a biased sampler family, we can enumerate all $\mathcal{O}(\delta^{-2}\log(|\mathcal{F}|))$-tuples of elements from X and find one giving a biased sampler family. ◀

Adjusting the parameters

The construction above implies that we can amplify the gap to any constant q by multiplying the size of an instance by a factor depending on k and q. However, when we want to rule out a superconstant approximation factor, e.g., $\alpha(k) = \log\log k$, we would like to apply the hypothetical approximation algorithm to the instance $(H, \mathcal{T}_H)$ with parameter k_0 depending on k and q, so we additionally need to adjust q so that $\alpha(k_0(k,q)) \leq q$.

▶ **Theorem 2.1.** *Consider a function $\alpha(k) = (\log k)^{\beta(k)}$, where $\beta(k) \to 0$ is computable and non-increasing. It is* W[1]*-hard to distinguish whether for a given instance of k-*Steiner Orientation*:*

1. *there exists an orientation satisfying all k terminal pairs, or*
2. *for all orientations the number of satisfied pairs is at most $\frac{1}{\alpha(k)} \cdot k$.*

Proof. We are going to reduce the exact version of k-Steiner Orientation, which is W[1]-hard, to the version with a sufficiently large gap, with Lemma 5.1. For a fixed k we can bound k_0 by a function of q: $k_0(q) \leq 2^{q^{c \cdot k}}$ for some constant c. On the other hand, $k_0(q) \to \infty$. Given an instance $(G, \mathcal{T}_G)$ we use Lemma 5.1 with q large enough, so that $\beta(k_0(q)) \cdot c \cdot k \leq 1$. The dependency $q(k)$ is also a computable function. We get $\alpha(k_0) = (\log k_0)^{\beta(k_0)} \leq q^{c \cdot k \cdot \beta(k_0)} \leq q$.

We have obtained a new instance $(H, \mathcal{T}_H)$ of k_0-Steiner Orientation of size $f(k) \cdot |V(G)|$ and k_0 being a function of k. If the original instance is fully satisfiable then the same holds for $(H, \mathcal{T}_H)$ and otherwise $S(H, \mathcal{T}_H) \leq \frac{1}{q} \cdot k_0 \leq \frac{1}{\alpha(k_0)} \cdot k_0$, which finishes the reduction. ◀

If we restrict the running time to be purely polynomial, we can slightly strengthen the lower bound, i.e., replace k with n in the approximation factor, while working with a similar hardness assumption. To make this connection, we observe that in order to show that a problem is in FPT, it suffices to solve it in polynomial time for some superconstant bound on the parameter.

▶ **Proposition 5.5.** *Consider a parameterized problem $\Pi \in$ XP that admits a polynomial-time algorithm (resp. false-biased polynomial-time algorithm) for the case $f(k) \leq |I|$, where f is some computable function. Then Π admits an* FPT *algorithm (resp. false-biased* FPT *algorithm).*

Proof. Since $\Pi \in$ XP, it admits a deterministic algorithm with running time $|I|^{g(k)}$. Whenever $f(k) \leq |I|$, we execute the polynomial-time algorithm. Otherwise we can solve it in time $f(k)^{g(k)}$. ◀

The hardness assumption below is slightly stronger than in Theorem 2.1, because the quantity 2^{k_0} can be super-polynomial and we cannot afford the time-expensive derandomization.

▶ **Theorem 2.2.** *Consider a function $\alpha(n) = (\log n)^{\beta(n)}$, where $\beta(n) \to 0$ is computable and non-increasing. Unless the class W[1] admits false-biased FPT algorithms, there is no polynomial-time algorithm that, given an instance of* Steiner Orientation *with n vertices and k terminal pairs, distinguishes between the following cases:*

1. *there exists an orientation satisfying all k terminal pairs, or*
2. *for all orientations the number of satisfied pairs is at most $\frac{1}{\alpha(n)} \cdot k$.*

Proof. Suppose there is such an algorithm with approximation factor $\alpha(n) = (\log n)^{\beta(n)}$. Let $\beta^*(\ell)$ be the smallest integer L, for which $\beta(L) \le \frac{1}{\ell}$. The function β^* is well defined and computable, because β is computable. Again, for fixed k and some constant c we have dependency $k_0(q) \le 2^{q^{c \cdot k}}$.

We are going to use the polynomial-time algorithm to solve k-Steiner Orientation in randomized $f(k) \cdot n^{\mathcal{O}(1)}$ time, which would imply the claim. Since the problem is in XP [15], by Fact 5.5 it suffices to solve instances satisfying $\beta^*((ck)^2) \le n$ in polynomial time. We can thus assume $(ck)^2 \cdot \beta(n) \le 1$, or equivalently $ck \cdot \beta(n) \le \sqrt{\beta(n)}$.

Given an instance of k-Steiner Orientation, we apply Lemma 5.1 with $q = (2 \log n)^{\beta(n)}$.

$$k_0 \le 2^{q^{ck}} = 2^{(2\log n)^{ck \cdot \beta(n)}} \le 2^{(2\log n)^{\sqrt{\beta(n)}}} = n^{o(1)},$$

$$\alpha(|V(H)|) \le \alpha(n \cdot k_0^2) = \alpha(n^{1+o(1)}) = ((1 + o(1)) \cdot \log n)^{\beta(n)}.$$

For large n we obtain $\alpha(|V(H)|) \le q$. Since $|V(H)| \le n \cdot k_0^2 = n^{1+o(1)}$, the size of the new instance of polynomially bounded and thus the randomized construction takes polynomial time. If we started with a YES instance, we always produce a YES instance, and otherwise $S(H, \mathcal{T}_H) \le \frac{1}{q} \cdot k_0 \le \frac{1}{\alpha(|V(H)|)} \cdot k_0$ with probability at least $\frac{1}{k_0}$, so we need to repeat the procedure $k_0 = n^{o(1)}$ times to get a constant probability of creating an instance with a small optimum. A hypothetical algorithm distinguishing these cases would therefore entail a false-biased polynomial-time algorithm for Steiner Orientation for the case $\beta^*((ck)^2) \le n$. The claim follows from Proposition 5.5. ◀

6 Inapproximability of Directed Multicut

We switch our attention to the Max (k,p)-Directed Multicut problem, for which we provide a slightly simpler reduction. We keep the same convention as before: within graph G we refer to sources and sinks $(s_i, t_i) \in \mathcal{T}$ shortly as $G\{s,i\}$, $G\{t,i\}$, and denote the maximal number of terminal pairs separable in $(G, \mathcal{T})$ by deleting p edges by $S(G, \mathcal{T}, p)$.

▶ **Lemma 6.1.** *There is a procedure that, for an instance $(G, \mathcal{T}_G, p)$, $|\mathcal{T}_G| = 4$ of* Directed Multicut *and parameter q, constructs a new instance $(H, \mathcal{T}_H, p_0)$, $k_0 = |\mathcal{T}_H|$, such that:*

1. *$k_0 = \Theta(p \cdot q^2 \log q)$,*
2. *$p_0 = \Theta(p^2 \log q)$,*
3. *$|E(H)| = |E(G)| \cdot p_0 + \mathcal{O}(k_0 \cdot p_0)$,*
4. *if $S(G, \mathcal{T}_G, p) = 4$, then $S(H, \mathcal{T}_H, p_0) = k_0$ always (Completeness),*
5. *if $S(G, \mathcal{T}_G, p) < 4$, then $S(H, \mathcal{T}_H, p_0) \le \frac{1}{q} \cdot k_0$ with probability at least $\frac{1}{2}$ (Soundness).*

The randomized construction takes time proportional to $|H|$. It can be derandomized in time $f(p,q) \cdot |G|$.

Proof. Consider $M = 3(p+1) \cdot \log q$ copies of $(G, \mathcal{T}_G)$, denoted $(G_1, \mathcal{T}_1), \dots, (G_M, \mathcal{T}_M)$. Let $\mathcal{R} = [4]^M$ be the family of all M-tuples with values in $[4]$. For a random sequence $R = (r_1, r_2, \dots r_M) \in \mathcal{R}$, we add a terminal pair s_R, t_R and for each $i \in [M]$ we add directed

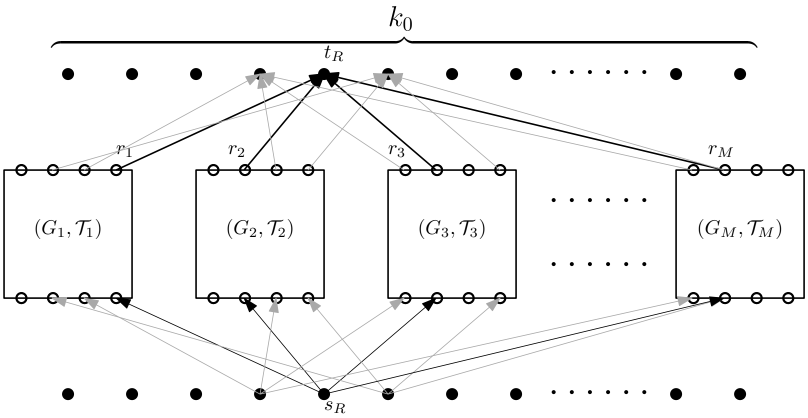


Figure 3 Building a new instance from M parallel copies of G. The tuple $R = (r_1, r_2, \dots r_M)$ encodes through which nodes the (s_R, t_R) pair is connected. For the sake of legibility only the edges incident to s_R, t_R (the black ones) and neighboring terminals (the grey ones) are sketched.

edges $(s_R, G_i\{s, r_i\})$ and $(G_i\{t, r_i\}, t_R)$. We repeat this subroutine $k_0 = \Theta(p \cdot q^2 \log q)$ times and create that many terminals pairs as depicted in Figure 3. We set the budget $p_0 = 3p(p+1) \cdot \log q$.

If $S(G, \mathcal{T}_G, p) = 4$, then the budget suffices to separate all terminal pairs in all copies of $(G, \mathcal{T}_G)$ (completeness). Otherwise, one needs to remove at least $p+1$ edges from each copy of $(G, \mathcal{T}_G)$ to separate all 4 pairs so we can afford that in at most $3p \cdot \log q$ copies. Therefore for any solution there are at least $3 \log q$ copies, where there is at least one terminal pair that is not separated.

Let $C_i \subseteq [4]$ represent information about the status of solution within $(G_i, \mathcal{T}_i)$: there is path from $G_i\{s, j\}$ to $G_i\{t, j\}$ only if $j \in C_i$. A tuple $C = (C_1, \dots, C_M)$ is called a configuration and we refer to the family of configurations induced by possible solutions as $\mathcal{C}$. Clearly, $|\mathcal{C}| \le 16^M$.

Recall that each terminal pair can be represented by a tuple $R = (r_1, r_2, \dots r_M) \in \mathcal{R}$ encoding through which terminal pair in G_i a path from s_R to t_R can go. For a fixed configuration $C \in \mathcal{C}$, function $f_C : \mathcal{R} \to \{0, 1\}$ is set to 1 if the pair s_R, t_R is separated, or equivalently: if for each $i \in [M]$ we have $r_i \notin C_i$. For $S(G, \mathcal{T}_G, p) < 4$ there are at least $3 \log q$ copies of G_i with $C_i \neq \emptyset$, therefore $\mathbb{E}_{R \sim U(\mathcal{R})} f_C(R) \le (\frac{3}{4})^{3 \log q} \le 2^{-\log(2q)} = \frac{1}{2q}$.

The size of $\mathcal{C}$ is at most $16^M = 2^{\mathcal{O}(p \log q)}$. We apply Lemma 5.3 for $\mathcal{F} = \{f_C : C \in \mathcal{C}\}$ and $\delta = \frac{1}{2q}$. It follows that $\mathcal{O}(\delta^{-2} \log(|\mathcal{F}|)) = \mathcal{O}(p \cdot q^2 \log q)$ random samples from $\mathcal{R}$ suffice to obtain a rounding error of at most $\frac{1}{2q}$ for all $C \in \mathcal{C}$ at once. Therefore with probability at least $\frac{1}{2}$ we have constructed an instance in which for any cutset of size p_0 (and thus for any configuration C) the fraction of separated terminal pairs is at most $\mathbb{E}_{R \sim U(\mathcal{R})} f_C(R) + \frac{1}{2q} \le \frac{1}{q}$. ◀

Remark on derandomization

As before, if we allow exponential running time with respect to p and q, we can find a correct sampler family by enumeration and derandomize the reduction. However, we cannot afford that in a polynomial-time reduction. To circumvent this, observe that we upper bound the expected value of f_C using independence of $3 \log q$ variables. We could alternatively take advantage of δ-biased ℓ-wise independent hashing [37] (cf. [8, 39, 40]) to construct biased ℓ-wise independent binary random variables with few random bits, instead of relying on Lemma 5.3. This technique provides an analogous bound on additive estimation error as in Lemma 5.3 for events that depend on at most ℓ variables. A family of N such variables can be constructed using $\mathcal{O}(\ell + \log\log N + \log(\frac{1}{\delta}))$ random bits [37, Lemma 4.2].

Since we are interested in having $N = \mathcal{O}(p \cdot \log q)$ variables, $\delta = \frac{1}{2q}$, and $(3\log q)$-wise independency, the size of the whole probabilistic space becomes $2^{\mathcal{O}(\log q + \log\log p)} = q^{\mathcal{O}(1)}(\log p)^{\mathcal{O}(1)}$. The problem is that we need to optimize the exponent at q in order to obtain better lower bounds. Unfortunately, we are not aware of any construction of a δ-biased ℓ-wise independent hash family, that would optimize this constant.

▶ **Theorem 2.4.** *For any $\varepsilon > 0$ and function $\alpha(k) = \mathcal{O}\left(k^{\frac{1}{2}-\varepsilon}\right)$, it is* W[1]*-hard to distinguish whether for a given instance of* Max (k,p)-Directed Multicut*:*

1. *there is a cut of size p that separates all k terminal pairs, or*
2. *all cuts of size p separate at most $\frac{1}{\alpha(k)} \cdot k$ terminal pairs.*

Proof. Let us fix $\varepsilon > 0$. We are going to reduce the exact version of p-Directed Multicut with 4 terminals, which is W[1]-hard, to the version with a sufficiently large gap, parameterized by both p and $k = |\mathcal{T}|$. Let L be an integer larger than $\frac{2}{\varepsilon}$.

For an instance $(G, \mathcal{T}, p)$ of p-Directed Multicut we apply Lemma 6.1 with $q = p^L$. If the original instance is fully solvable, the new one is as well. Otherwise the maximal fraction of separated terminal pairs is $\frac{k_0}{q} = \mathcal{O}(p \cdot q \log q) = \mathcal{O}(p^{L+2})$. On the other hand, $\frac{k_0}{\alpha(k_0)} = \Omega\left(k_0^{(\frac{1}{2}+\varepsilon)}\right) \geq \Omega\left(p^{2L\cdot(\frac{1}{2}+\varepsilon)}\right)$. The exponent at p in the latter formula is $L + 2\varepsilon L > L + 2$, so for large p it holds $\frac{k_0}{\alpha(k_0)} \geq \frac{k_0}{q}$, therefore the reduction maps NO-instances into those where all cuts of size p_0 separate at most $\frac{k_0}{\alpha(k_0)}$ terminal pairs. Both k_0 and p_0 are functions of p, therefore we have obtained a parameterized reduction. ◀

▶ **Theorem 2.5.** *Assuming* NP $\not\subseteq$ co-RP*, for any $\varepsilon > 0$ and function $\alpha(m) = \mathcal{O}\left(m^{\frac{1}{2}-\varepsilon}\right)$, there is no polynomial-time algorithm that, given an instance $(G, \mathcal{T}, p)$, $|\mathcal{T}| = k$, $|E(G)| = m$, of* Max Directed Multicut*, distinguishes between the following cases:*

1. *there is a cut of size p that separates all k terminal pairs, or*
2. *all cuts of size p separate at most $\frac{1}{\alpha(m)} \cdot k$ terminal pairs.*

Proof. Suppose there is such an algorithm for some $\varepsilon > 0$ and proceed as in the proof of Theorem 2.4 with L sufficiently large, so that $2\varepsilon L \geq 5$ and $q = m^L$. The reduction is polynomial because L is constant for fixed ε. We have $m_0 = |E(H)| = m \cdot p_0 + \mathcal{O}(k_0 \cdot p_0) = mp^2 \log q + \mathcal{O}(p^3 q^2 \log^2 q) = \mathcal{O}(m^{2L+5})$ because $p \leq m$. If the initial instance is fully satisfiable, then always $S(H, \mathcal{T}_H, p_0) = k_0$. For a NO-instance, we have $S(H, \mathcal{T}_H, p_0) \leq \frac{k_0}{q} = \mathcal{O}(m^{L+2})$ with probability at least $\frac{1}{2}$. On the other hand, $k_0 = \Omega(m^{2L})$ and

$$\frac{k_0}{\alpha(m_0)} = \Omega\left(\frac{1}{m_0^{\frac{1}{2}-\varepsilon}}\right) \cdot k_0 = \Omega\left(m^{2L-(2L+5)\cdot(\frac{1}{2}-\varepsilon)}\right) = \Omega(m^{L-\frac{5}{2}+2\varepsilon L}).$$

We have adjusted L to have $L - \frac{5}{2} + 2\varepsilon L > L + 2$, so for large m we get $\frac{k_0}{\alpha(m_0)} \geq \frac{k_0}{q}$. Therefore the reduction maps NO-instances into those where all cuts of size p_0 separate at most $\frac{k_0}{\alpha(m_0)}$ terminal pairs. When the reduction from Lemma 6.1 is correct (with probability at least $\frac{1}{2}$), we are able to detect the NO-instances. This implies that Directed Multicut $\in$ co-RP. ◀

7 Final remarks and open problems

I would like to thank Pasin Manurangsi for helpful discussions and, in particular, for suggesting the argument based on Chernoff bound in Lemma 5.1, which is surprisingly simple and powerful. A question arises whether one can derandomize this argument efficiently and construct a δ-biased sampler family in an explicit way. This would allow us to replace the assumption NP $\not\subseteq$ co-RP with P $\neq$ NP for Directed Multicut. This technique may also find use in other reductions in parameterized inapproximability.

An obvious question is if any of the studied problems admits an $o(k)$-approximation, or if the lower bounds can be strengthened. Note that for the maximization version of Directed Multicut we do not know anything better than $\frac{k}{2}$-approximation as we cannot solve the exact problem for $k > 2$. For Steiner Orientation, the reason why the value of the parameter in the self-reduction becomes so large, is that in each step we can add only an exponentially small term to the gap. Getting around this obstacle should lead to better lower bounds. Also, the approximation status for k-Steiner Orientation on planar graphs remains unclear [11]. Here we still cannot rule out a constant approximation and there are no upper bounds known.

Finally, it is an open quest to establish relations between other hard parameterized problems and their gap versions. Is $F(k)$-Gap k-Clique W[1]-hard for $F(k) = o(k)$ and is $F(k)$-Gap k-Dominating Set W[2]-hard for any function F (open questions in [26])? Or can it be possible that $F(k)$-Gap k-Dominating Set is in W[1] for some function F?

References

1 Marek Adamczyk, Jaroslaw Byrka, Jan Marcinkowski, Syed M. Meesum, and Michal Wlodarczyk. Constant-Factor FPT Approximation for Capacitated k-Median. In *27th Annual European Symposium on Algorithms (ESA 2019)*, volume 144 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 1:1–1:14, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ESA.2019.1.

2 Amit Agarwal, Noga Alon, and Moses S. Charikar. Improved approximation for directed cut problems. In *Proceedings of the Thirty-ninth Annual ACM Symposium on Theory of Computing*, STOC '07, pages 671–680, New York, NY, USA, 2007. ACM. doi:10.1145/1250790.1250888.

3 Esther M. Arkin and Refael Hassin. A note on orientations of mixed graphs. *Discrete Applied Mathematics*, 116(3):271–278, 2002. doi:10.1016/S0166-218X(01)00228-1.

4 Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *J. ACM*, 45(3):501–555, May 1998. doi:10.1145/278298.278306.

5 Nikhil Bansal and Subhash Khot. Optimal long code test with one free bit. In *2009 50th Annual IEEE Symposium on Foundations of Computer Science*, pages 453–462. IEEE, 2009.

6 Arnab Bhattacharyya, Suprovat Ghoshal, Karthik C. S., and Pasin Manurangsi. Parameterized intractability of Even Set and Shortest Vector Problem from Gap-ETH. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, pages 17:1–17:15, 2018. doi:10.4230/LIPIcs.ICALP.2018.17.

7 Nicolas Bousquet, Jean Daligault, and Stéphan Thomassé. Multicut is FPT. In *Proceedings of the Forty-third Annual ACM Symposium on Theory of Computing*, STOC '11, pages 459–468, New York, NY, USA, 2011. ACM. doi:10.1145/1993636.1993698.

8 J.Lawrence Carter and Mark N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143–154, 1979. doi:10.1016/0022-0000(79)90044-8.

9 Parinya Chalermsook, Marek Cygan, Guy Kortsarz, Bundit Laekhanukit, Pasin Manurangsi, Danupon Nanongkai, and Luca Trevisan. From Gap-ETH to FPT-inapproximability: Clique, Dominating Set, and more. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 743–754, October 2017. doi:10.1109/FOCS.2017.74.

10 Rajesh Chitnis, Andreas Emil Feldmann, and Pasin Manurangsi. Parameterized approximation algorithms for bidirected steiner network problems. In *26th Annual European Symposium on Algorithms (ESA 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

11 Rajesh Chitnis, Andreas Emil Feldmann, and Ondřej Suchý. A tight lower bound for Planar Steiner Orientation. *Algorithmica*, May 2019. doi:10.1007/s00453-019-00580-x.

12 Julia Chuzhoy and Sanjeev Khanna. Polynomial flow-cut gaps and hardness of directed cut problems. *J. ACM*, 56(2):6:1–6:28, April 2009. doi:10.1145/1502793.1502795.

13 Vincent Cohen-Addad, Anupam Gupta, Amit Kumar, Euiwoong Lee, and Jason Li. Tight FPT Approximations for k-Median and k-Means. In *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 42:1–42:14, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ICALP.2019.42.

14 Vincent Cohen-Addad and Jason Li. On the Fixed-Parameter Tractability of Capacitated Clustering. In *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 41:1–41:14, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ICALP.2019.41.

15 Marek Cygan, Guy Kortsarz, and Zeev Nutov. Steiner forest orientation problems. *SIAM Journal on Discrete Mathematics*, 27(3):1503–1513, 2013. doi:10.1137/120883931.

16 Irit Dinur. The PCP theorem by gap amplification. *J. ACM*, 54(3), June 2007. doi:10.1145/1236457.1236459.

17 Irit Dinur. Mildly exponential reduction from gap 3SAT to polynomial-gap label-cover. *Electronic Colloquium on Computational Complexity (ECCC)*, 23:128, 2016.

18 Britta Dorn, Falk Hüffner, Dominikus Krüger, Rolf Niedermeier, and Johannes Uhlmann. Exploiting bounded signal flow for graph orientation based on cause–effect pairs. In *Theory and Practice of Algorithms in (Computer) Systems*, pages 104–115, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

19 Guillaume Fertin, Hafedh Mohamed-Babou, and Irena Rusu. On the Complexity of two Problems on Orientations of Mixed Graphs. In *In Proc. 5èmes Journées Ouvertes Biologie Informatique Mathématiques (JOBIM 2012)*, pages 161–170, Rennes, France, July 2012. URL: https://hal.archives-ouvertes.fr/hal-00826863.

20 Iftah Gamzu, Danny Segev, and Roded Sharan. Improved orientations of physical networks. In *Algorithms in Bioinformatics*, pages 215–225, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

21 David Gillman. A Chernoff bound for random walks on expander graphs. *SIAM J. Comput.*, 27:1203–1220, 1998.

22 Anupam Gupta, Euiwoong Lee, and Jason Li. Faster exact and approximate algorithms for k-Cut. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 113–123, 2018. doi:10.1109/FOCS.2018.00020.

23 Venkatesan Guruswami, Rajsekar Manokaran, and Prasad Raghavendra. Beating the random ordering is hard: Inapproximability of maximum acyclic subgraph. In *2008 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 573–582. IEEE, 2008.

24 Rafael Hassin and Nimrod Megiddo. On orientations and shortest paths. *Linear Algebra and its Applications*, 114-115:589–602, 1989. doi:10.1016/0024-3795(89)90481-3.

25 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.

26 Karthik C. S., Bundit Laekhanukit, and Pasin Manurangsi. On the parameterized complexity of approximating Dominating Set. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 1283–1296, 2018. doi:10.1145/3188745.3188896.

27 Ken-ichi Kawarabayashi and Bingkai Lin. A nearly 5/3-approximation FPT algorithm for Min-k-Cut. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms*, pages 990–999, 2020. doi:10.1137/1.9781611975994.59.

28 Subhash Khot. On the power of unique 2-prover 1-round games. In *Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing*, STOC '02, pages 767–775, New York, NY, USA, 2002. ACM. doi:10.1145/509907.510017.

29 Bingkai Lin. The parameterized complexity of k-Biclique. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 605–615, 2015. doi:10.1137/1.9781611973730.41.

30 Daniel Lokshtanov, M. S. Ramanujan, Saket Saurabh, and Meirav Zehavi. Parameterized complexity and approximability of directed odd cycle transversal. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 2181–2200, 2020. doi:10.1137/1.9781611975994.134.

31 Rajsekar Manokaran, Joseph Naor, Prasad Raghavendra, and Roy Schwartz. SDP gaps and UGC hardness for multiway cut, 0-extension, and metric labeling. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 11–20, 2008.

32 Pasin Manurangsi. Almost-polynomial ratio ETH-hardness of approximating Densest k-Subgraph. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2017, pages 954–961, New York, NY, USA, 2017. ACM. doi:10.1145/3055399.3055412.

33 Pasin Manurangsi. Inapproximability of Maximum Edge Biclique, Maximum Balanced Biclique and Minimum k-Cut from the Small Set Expansion Hypothesis. In *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, volume 80 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 79:1–79:14, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ICALP.2017.79.

34 Pasin Manurangsi and Prasad Raghavendra. A Birthday Repetition Theorem and Complexity of Approximating Dense CSPs. In *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, volume 80 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 78:1–78:15, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ICALP.2017.78.

35 Daniel Marx. Completely inapproximable monotone and antimonotone parameterized problems. In *2010 IEEE 25th Annual Conference on Computational Complexity*, pages 181–187, June 2010. doi:10.1109/CCC.2010.25.

36 Alexander Medvedovsky, Vineet Bafna, Uri Zwick, and Roded Sharan. An algorithm for orienting graphs based on cause-effect pairs and its applications to orienting protein networks. In *Algorithms in Bioinformatics*, pages 222–232, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

37 Joseph Naor and Moni Naor. Small-bias probability spaces: Efficient constructions and applications. *SIAM Journal on Computing*, 22, March 2001. doi:10.1145/100216.100244.

38 Marcin Pilipczuk and Magnus Wahlström. Directed Multicut is W[1]-hard, even for four terminal pairs. *ACM Trans. Comput. Theory*, 10(3):13:1–13:18, May 2018. doi:10.1145/3201775.

39 Alan. Siegel. On universal classes of extremely random constant-time hash functions. *SIAM Journal on Computing*, 33(3):505–543, 2004. doi:10.1137/S0097539701386216.

40 Umesh Virkumar Vazirani. *Randomness, Adversaries and Computation (Random Polynomial Time)*. PhD thesis, University of California, Berkeley, 1986.

Near-Optimal Algorithm for Constructing Greedy Consensus Tree

Hongxun Wu
Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China
wuhx18@mails.tsinghua.edu.cn

Abstract

In biology, phylogenetic trees are important tools for describing evolutionary relations, but various data sources may result in conflicting phylogenetic trees. To summarize these conflicting phylogenetic trees, consensus tree methods take k conflicting phylogenetic trees (each with n leaves) as input and output a single phylogenetic tree as consensus.

Among the consensus tree methods, a widely used method is the greedy consensus tree. The previous fastest algorithms for constructing a greedy consensus tree have time complexity $\tilde{O}(kn^{1.5})$ [Gawrychowski, Landau, Sung, Weimann 2018] and $\tilde{O}(k^2n)$ [Sung 2019] respectively. In this paper, we improve the running time to $\tilde{O}(kn)$. Since k input trees have $\Theta(kn)$ nodes in total, our algorithm is optimal up to polylogarithmic factors.

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

Keywords and phrases phylogenetic trees, greedy consensus trees, splay tree

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.105

Category Track A: Algorithms, Complexity and Games

Acknowledgements We want to thank anonymous reviewers for many helpful comments.

1 Introduction

The problem of constructing consensus trees arises from bioinformatics. In biology, phylogenetic trees describe the biological evolutionary relations between species. But the phylogenetic trees from different biological data may conflict with each other. As mentioned in [22], even from the same data set, when certain resampling techniques are used, we could still get many different phylogenetic trees. This problem has long been studied, to name a few [1, 14, 28, 17, 13, 12, 18, 5].

Motivated by this, consensus tree methods were proposed [1] to summarize these phylogenetic trees into a single phylogenetic tree, which is viewed as the consensus of these phylogenetic trees and is called the consensus tree. Since then, many different consensus tree methods were proposed. As mentioned in [22], the majority rule consensus tree [28], the loose consensus tree [8], and the greedy consensus tree [9] are the most frequently used consensus trees.

As discussed in [11], while increasing the number of phylogenetic trees in the input, the greedy consensus tree converges faster than majority rule consensus tree and R^* consensus tree. Although the greedy consensus tree is not a consistent estimator, the region of parameter space in which greedy consensus tree fails is relatively small, hence greedy consensus tree offers more robustness [11]. The greedy consensus tree method is implemented in many software packages, such as PHYLIP [15], PAUP* [40], MrBayes [32], RAxML [37], and also widely used in numerous works in biology [6, 7, 11, 24, 26, 27, 30, 31, 33, 34, 36, 38].

For most consensus tree methods, optimal or near-optimal algorithms for construction have been found. One exception is the greedy consensus tree (See Table 1). For greedy consensus tree construction, the naïve algorithm takes $\tilde{O}(kn^3)$ time [9]. Then it was improved to $O(kn^2)$ time by [22]. Recently there are $\tilde{O}(kn^{1.5})$ [16] and $\tilde{O}(k^2n)$ [39] algorithms proposed for it.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 105; pp. 105:1–105:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Table 1 Running time of construction algorithms for different consensus tree methods.

Consensus tree method	Running time	Reference
Adam's consensus tree	$O(kn \log n)$	[19]
Strict consensus tree	$O(kn)$	[10]
Loose consensus tree	$O(kn)$	[22]
Frequency difference consensus tree	$O(kn \log^2 n)$	[16]
Majority-rule consensus tree	$O(kn \log k)$, Randomized $O(kn)$	[4, 22]
Majority-rule (+) consensus tree	$O(kn)$	[21]
Local consensus tree	$O(kn^3)$	[25, 20]
R^* consensus tree	$O(n^2 \log^{k+2} n)$	[23]
Greedy consensus tree	$O(kn^{1.5})$, $O(k^2 n)$	[16, 39]

In this paper, we present a near-optimal $\tilde{O}(kn)$ time algorithm for greedy consensus tree construction.

▶ **Theorem 1.** *Greedy consensus tree of k phylogenetic trees for n species can be constructed in $\tilde{O}(kn)$ time.*

High Level Idea

The previous $\tilde{O}(kn^{1.5})$ algorithm [16] builds the consensus tree by dynamically adding nodes to it. To find out the position to add a new node, they need to support least common ancestor query on the consensus tree. Since the consensus tree is dynamically changing, answering such queries is time-consuming and becomes the bottleneck of this previous algorithm. Motivated by it, we came up with an alternative approach that only requires least common ancestor query on the phylogenetic trees in the input. Since these are static trees, such queries can be efficiently answered. This leads to our improved algorithm.

2 Preliminaries

2.1 Phylogenetic Tree

Phylogenetic trees represent the evolution of species. Different leaves of a phylogenetic tree represent different species. From biological data, we can infer that some of these species share common ancestors. These common ancestors are represented by the nonleaf nodes, also called inner nodes. Each inner node represents the common ancestor of all leaves in its subtree.

Formally, a tree with n leaves is leaf-labeled if and only if its n leaves have distinct labels from 1 to n. A phylogenetic tree T is a rooted, unordered, leaf-labeled tree in which every inner node has at least two children. If there is a directed path from node u to node v, u is a descendant of v, and v is an ancestor of u. Thus node u is a descendant of itself. If $u \neq v$ and v is a descendant of u, v is a proper descendant of u, and u is a proper ancestor of v.

The subtree tree of an inner node u is the subtree rooted at u and formed by all its descendants. In the rest of the paper, whenever we say a subtree, we always refer to the subtree of an inner node.

Cluster and Signature

For inner node v in phylogenetic tree T, we define $L(v)$ to be the set of species within its subtree. Namely, $L(v) = \{x \in [n] : \text{There is a leaf labeled } x \text{ in the subtree of } v\}$. The set $L(v)$ is called a cluster.

Based on $L(v)$, we define the signature of phylogenetic tree T to be the set of all clusters in it. Namely, $\text{sign}(T) = \{L(v) : v \text{ is an inner node in } T\}$.

▶ **Observation 2.** *The signature of a phylogenetic tree completely captures its structure. Namely, given $sign(T)$, the phylogenetic tree T is uniquely determined.*

More specifically, suppose the cluster $c_1 \in sign(T)$ corresponds to node u on T. Namely, $c_1 = L(u)$. Let $par(u)$ denote the parent of u on T. $L(par(u))$ is the smallest cluster $c_2 \in sign(T)$ such that $c_1 \subset c_2$. This determines the parent of each node.

Consistency

But not every set S of clusters can be the signature of a phylogenetic tree. S is a valid signature if and only if it is a laminar family. In this case, we say S is consistent. Namely, S is consistent if and only if for every pair of distinct clusters $c_1, c_2 \in S$, one of the following holds:

- $c_1 \cap c_2 = \emptyset$
- $c_1 \subset c_2$
- $c_1 \supset c_2$

We say a cluster c is consistent with S if and only if $S \cup \{c\}$ is consistent.

2.2 Greedy Consensus Tree

Recall that the greedy consensus tree method takes k phylogenetic trees $T_1, T_2, \cdots, T_k$ as inputs. These k trees are over the same set of n species. Its objective is to output a single phylogenetic tree to be the consensus tree.

Definition

As its name suggests, a greedy consensus tree is defined as the output of a simple greedy algorithm. Let $\mathcal{F}$ be the set of all clusters that have appeared in $T_1, T_2, \cdots, T_k$. The frequency $f(c)$ of a cluster c is the number of times c appears in $\mathcal{F}$.

Let S denote the signature of the current consensus tree. Initially, the consensus tree only contains a root and n leaves. Thus $S = \{\{1, 2, \ldots, n\}, \{1\}, \{2\}, \ldots, \{n\}\}$. Each time we pick the cluster with the highest frequency in $\mathcal{F}$ and add it to S if they are consistent. From the tree point of view, we are adding a new inner node to the consensus tree. For the details, see Algorithm 1.

Algorithm 1 Greedy consensus tree.

```
1: Initially signature $S \leftarrow \{\{1, 2, \ldots, n\}, \{1\}, \{2\}, \ldots, \{n\}\}$
2: $\mathcal{F} \leftarrow \text{sign}(T_1) \cup \text{sign}(T_2) \cup \cdots \cup \text{sign}(T_k)$
3: For all clusters $c \in \mathcal{F}$, count its frequency $f(c) \leftarrow |\{i | c \in \text{sign}(T_i)\}|$
4: while $\mathcal{F} \neq \emptyset$ do
5:     Pick $c_0 \leftarrow \arg\max_{c \in \mathcal{F}} f(c)$ with the highest frequency (ties are broken arbitrarily)
6:     if $c_0$ is consistent with $S$ then $S \leftarrow S \cup \{c_0\}$
7:     $\mathcal{F} \leftarrow \mathcal{F} \backslash \{c_0\}$
8: $S$ is the signature of a greedy consensus tree
```

Since at Line 5, Algorithm 1, ties are broken arbitrarily, the output of the algorithm may not be unique. There may be more than one greedy consensus tree for a fixed input.

Running Time

Algorithm 1 is a naïve algorithm for constructing greedy consensus trees. Here we discuss this algorithm and its complexity in more detail. It essentially contains two phases:

1. Count the frequency $f(c)$ of clusters and sort them. (Line $1 \sim 3$ of Algorithm 1)
2. Repeatedly run the greedy procedure. (Line $4 \sim 7$ of Algorithm 1)

The first phase is easy to be handled in $\tilde{O}(kn)$ time. Suppose $|\mathcal{F}| = m$. Namely, there are m distinct clusters. For each of them, we assign a unique number in $[m]$ to it as its identifier. For each $u \in T_i$, let $id(u) \in [m]$ be the identifier of $L(u)$. Then, for $u \in T_i$ and $v \in T_j$, $id(u) = id(v)$ if and only if $L(u) = L(v)$.

In [16], they showed that these identifiers can be computed in $O(kn \log^2 n)$ time. To be self-contained, we state it in Lemma 4 and present a short proof here. We first need a data structure for dynamic set equality.

▶ **Lemma 3** (Lemma 1, [16]). *There is a deterministic dynamic set equality structure that supports:*

- $\textsc{Create}(s)$*: Create a new empty set* s*.*
- $\textsc{Add}(s, x)$*: Add an element* x *to set* s*. (create a new set* $s \cup \{x\}$ *without destroying* s*)*
- $ID(s)$*: Return the identifier of set* s *which is a positive integer smaller than the number of sets we have created. Two sets have the same identifier if and only if they are equal.*

Let n *denote the size of all sets created. Each operation takes* $\tilde{O}(1)$ *time.*

Proof. We apply the dynamic string equality structure in [29]. It supports the following operations in $\tilde{O}(1)$ time: (1) create a new string with a single character, (2) test if two strings are equal, (3) split a string into two new strings without destroying it, (4) concatenate two strings to form a new string without destroying them, (5) given i, return the i-th character in a string. Modifying a character of a string can be reduced to $O(1)$ split, create, and concatenate operations. Comparing the lexicographical order of two strings can be reduced to $O(\log n)$ split and equality testing operations using binary search. Thus they all take $\tilde{O}(1)$ time.

We encode each set s into a binary string. The i-th bit of the string is 1 if and only if $i \in s$. For empty set, we create a new string with n zeros in beginning. To add an element x to s, we only need to modify the x-th bit in the string which take $\tilde{O}(1)$ time.

To maintain the identifier of each set s, we maintain a global balanced binary search tree of the strings of all sets in their lexicographical order. When a new set is created, we add its string to this balanced binary search tree, and attach a new identifier to it. Since comparing lexicographical order takes $\tilde{O}(1)$ time, maintaining this binary search tree and finding the identifier of a set also takes $\tilde{O}(1)$ time. ◀

▶ **Lemma 4** (Theorem 3, [16]). *The identifier* $id(u)$ *can be found for every node* u *of phylogenetic trees* $T_1, T_2, \cdots, T_k$ *in* $\tilde{O}(kn)$ *time.*

Proof. We process each tree T_i bottom-up. For each inner node u, we obtain $L(u)$ and its identifier in the dynamic set equality structure by the following procedure:

1. Let v be the children of u with the largest subtree. If v is a leaf, we let s be the singleton $\{\text{label}(v)\}$. Otherwise, let $s = L(v)$ in dynamic set equality structure.
2. We traverse the subtrees of all other children of u and add the leaves we visited into s. Now s equals $L(u)$
3. $id(u) \leftarrow \text{ID}(s)$.

Since each time a node is visited at Step 2, the subtree it belongs to is doubled. Each node can be visited at most $O(\log n)$ times throughout the procedure. Thus this procedure takes $\tilde{O}(kn)$ time in total. ◀

After we get the identifiers $id(u)$, counting the frequency of identifiers and sorting within $\tilde{O}(kn)$ time are straight-forward.

The bottleneck of the naïve algorithm is the second phase. To test whether the signature S of the consensus tree is consistent with c_0, the naïve algorithm enumerates all clusters in S. There can be at most $O(n)$ clusters in S since each of them corresponds to a node in the consensus tree.

For every cluster in S testing whether it is consistent with c_0 takes $O(n)$ time. There are $O(n)$ clusters in S. Thus for each c_0, it takes at most $O(n^2)$ time. There are $O(kn)$ clusters in $\mathcal{F}$ (since each of them corresponds to at least one node in $T_1, T_2, \ldots, T_k$).

So in total, the naïve algorithm takes $O(kn^3)$ time. We will present our algorithm for the second phase in Section 3 and show how to make it efficient in Section 4.

2.3 Data Structures

Our algorithm relies on some classical data structure techniques introduced here.

DFS Sequence

Let T be a rooted tree. The Euler tour of T starts from its root, passing by each edge exactly twice (from opposite directions), and return to the root. We define $E(T)$ to be the Euler tour of T in which we only keep the first occurrence of each node. Or equivalently, $E(T)$ is the sequence produced by the following depth-first search: Initially let the sequence be empty. When we perform a depth-first search on T, each time we visit a node for the first time, we add it to the end of our sequence. In the end, this sequence equals $E(T)$.

▶ **Observation 5.** *Suppose v is a node on T. All nodes within the subtree of v form a continuous interval in $E(T)$.*

For the subtree of v on T, we denote its corresponding interval by $[l_T(v), r_T(v)]$. Here $l_T(v)$ is the position of v in $E(T)$, and $r_T(v)$ is the position of the last node that belongs to the subtree of v in $E(T)$.

Top Tree

A dynamic forest is a set of trees over disjoint sets of nodes that supports dynamic edge connection and deletion. Top tree [3] is a useful data structure for maintaining information in a dynamic forest. k-th ancestor of node u is the ancestor which is higher than u by k edges. When $k = 1$, it is the parent of u.

▶ **Lemma 6** ([3])**.** *Top tree supports the following operations in $\tilde{O}(1)$ time:*

1. *$connect(u, v)$: Add an edge connecting nodes u and v that belong to different trees.*
2. *$delete(u, v)$: Delete the edge connecting nodes u and v.*
3. *$lca(u, v)$: Return the least common ancestor of u and v.*
4. *$ancestor(u, k)$: Return the k-th ancestor of u.*

Splay tree

Splay tree [35] is a classical binary search tree maintaining a dynamic sequence. Each element in the sequence has two attributes, its key and value. The elements in a dynamic sequence are arranged in the increasing order of their keys. While keys specify the order of the elements, values are the information related to our queries.

▶ **Lemma 7.** *Let K be an ordered set, and let $G = (S, +)$ be a semigroup. Splay tree maintains n nodes each with its key and value, supporting the following operations:*

1. *$Insert(k, v)$: Insert a new node with key $k \in K$ and value $v \in S$.*
2. *$Delete(k)$: Delete the elements with key $k \in K$.*
3. *$Split(k)$: Return two splay trees T_1, T_2. T_1 contains all nodes whose keys are smaller than k, and T_2 contains all other nodes. The original splay tree is destroyed after the operation.*
4. *$Size()$: Return the number of nodes in the splay tree.*
5. *$Merge(T_1, T_2)$: Merge two splay trees T_1, T_2 where all nodes in T_1 have smaller keys than those in T_2.*
6. *$Sum(k_1, k_2)$: Suppose the values of nodes with keys $k \in [k_1, k_2]$ are $v_1, v_2, \ldots, v_t$ in the order of increasing keys. Then it returns $v_1 + v_2 + \cdots + v_t$.*

Suppose the $+$ operation of semigroup G takes $\tilde{O}(1)$ time. Then each operation here takes $\tilde{O}(1)$ time.

Specifically, it has the following two applications:

- *Let S be the set of integers, and let $+$ be addition. We can answer the summation of a continuous subsequence in $\tilde{O}(1)$ time.*
- *Let S be the set of nodes in a static tree, and let $a + b$ be the least common ancestor of a and b. We can answer the least common ancestor of a continuous subsequence in $\tilde{O}(1)$ time.*

Proof. The original paper [35] for splay tree showed how to handle insert, delete, split, and merge operations when nodes only have keys but no values.

Here at each node, in addition to its key, we also maintain its value and the summation of all values in its subtree. (Here summation refers to the operation of the semigroup) This summation can be calculated from the summation of its children by a $+$ operation. Whenever the children of a node in splay tree changes, we update its summation. Since for each operation, we only change the children of $O(\log n)$ nodes. Update the summation for them takes $O(\log n)$ many $+$ operations only.

To evaluate $Sum(k_1, k_2)$, we first split the splay tree into three trees T_1, T_2, T_3 using two splits. T_1 contains all nodes with keys smaller than k_1. T_2 contains all elements with keys in $[k_1, k_2]$. T_3 contains all other elements. Then we return the summation maintained at the root of T_2, and merge them back.

For $Size()$ query, we can let the value of each node be one and reduce it to the value summation query.

For the applications, since LCA operation is associative, $(S, +)$ is a semigroup. The LCA of two nodes can be answered in $\tilde{O}(1)$ time [2]. Thus each operation takes only $\tilde{O}(1)$ time, and we can answer the least common ancestor of a continuous subsequence by $Sum(k_1, k_2)$.

Similarly, we can answer the summation of a continuous subsequence in $\tilde{O}(1)$ time by $Sum(k_1, k_2)$. ◀

3 Algorithm

In the naïve algorithm, checking whether a cluster c_0 is consistent with signature S takes $O(n^2)$ time. To speed it up, the first step is to find a characterization of consistency that utilizes the tree structure. Here we start with the characterization from the previous $\tilde{O}(kn^{1.5})$ algorithm [16]. Then we develop it into an improved algorithm.

From this section, we will be using the following notations. $\mathcal{T}$ denotes the phylogenetic tree corresponding to signature S, namely our consensus tree. $\mathrm{LCA}_{\mathcal{T}}(c_0)$ is the least common ancestor of all species in cluster c_0 on consensus tree $\mathcal{T}$, while $LCA_i(c_0)$ is that on the input phylogenetic tree T_i. $subtree(v)$ denotes the set of all nodes (both inner nodes and leaves) within the subtree of v.

3.1 Characterization of consistency

In this subsection, the proof of Lemma 8 and 9 are already known from [16], but for clarity, we formally state and prove them here.

To utilize the tree structure, we will focus on consensus tree $\mathcal{T}$ instead of its signature S. The cluster c_0 is consistent with S if and only if for all nodes $u \in \mathcal{T}$, $L(u)$ is consistent with c_0. First, we begin with a lemma that says only those nodes within the subtree of $LCA_{\mathcal{T}}(c_0)$ matters.

▶ **Lemma 8** ([16]). *For node $u \in \mathcal{T}$ outside the subtree of $LCA_{\mathcal{T}}(c_0)$, $L(u)$ is always consistent with c_0.*

Proof. For simplicity, we use lca to denote $LCA_{\mathcal{T}}(c_0)$ here. By the definition of lca, we know $c_0 \subseteq L(lca)$. $u \notin subtree(lca)$ implies that $L(u) \not\subset L(lca)$. Since signature S is consistent, there are two possibilities, either $L(lca) \subset L(u)$ or $L(lca) \cap L(u) = \emptyset$.

- $L(lca) \subset L(u)$: Then $c_0 \subseteq L(lca) \subset L(u)$.
- $L(lca) \cap L(u) = \emptyset$: Then $c_0 \cap L(u) \subseteq L(lca) \cap L(u) = \emptyset$.

In both cases, $L(u)$ is consistent with c_0. ◀

Then we focus on nodes within the subtree of $LCA_{\mathcal{T}}(c_0)$, more specifically, the children of $LCA_{\mathcal{T}}(c_0)$. Following is the characterization.

▶ **Lemma 9** ([16]). *Cluster c_0 is consistent with the signature S of $\mathcal{T}$ if and only if every child w of $LCA_{\mathcal{T}}(c_0)$ satisfies one of the following:*

- $L(w) \subset c_0$
- $L(w) \cap c_0 = \emptyset$

Equivalently, $S \cup \{c_0\}$ is consistent if and only if

$$\sum_{\substack{\text{child } w \text{ of } LCA_{\mathcal{T}}(c_0) \\ L(w) \subset c_0}} |L(w)| = |c_0|$$

Proof. For simplicity, we use lca to denote $LCA_{\mathcal{T}}(c_0)$ here. By Lemma 8, we only have to consider every node u within the subtree of lca. If $c_0 = L(lca)$, c_0 is consistent with S (since $S \cup \{c_0\} = S$), and all children w satisfies $L(w) \subset c_0$. Now we assume $c_0 \neq L(lca)$ which implies $c_0 \subset L(lca)$.

We first prove that this condition is sufficient. For inner node u within the subtree of lca, if $u = lca$, we know $c_0 \subset L(u)$ which means they are consistent. Otherwise, u must belong to the subtree of a child of lca. Let us call this child w_0.

By the condition of this lemma, either $L(w_0) \cap c_0 = \emptyset$ or $L(w_0) \subset c_0$. If $L(w_0) \cap c_0 = \emptyset$, since $L(u) \subseteq L(w_0)$, we know $L(u) \cap c_0 = \emptyset$. Otherwise $L(u) \subseteq L(w_0) \subset c_0$. In either case, $L(u)$ is consistent with c_0.

Then we prove the necessity. For every child w of lca, since $L(w)$ has to be consistent with c_0, either one of two conditions in this lemma holds, or $c_0 \subset L(w)$. If $c_0 \subset L(w)$, then w is either lca or a proper ancestor of lca, which contradicts the fact that w is a child of lca. ◀

3.2 Our Algorithm

Algorithm

We begin with a corollary of the characterization to replace $LCA_{\mathcal{T}}(c_0)$.

▶ **Corollary 10.** *c_0 is consistent with the signature S of $\mathcal{T}$ if and only if, there exists a node $u_0 \in T$, such that $c_0 \subseteq L(u_0)$, and every child w of u_0 satisfies one of the following:*

- $L(w) \subseteq c_0$
- $L(w) \cap c_0 = \emptyset$

Equivalently, $S \cup \{c_0\}$ is consistent if and only if $\exists u_0 \in T$ such that

$$\sum_{\substack{child\ w\ of\ u_0 \\ L(w) \subseteq c_0}} |L(w)| = |c_0|$$

Proof. For necessity, let $u_0 = LCA_{\mathcal{T}}(c)$.

For sufficiency, if $u_0 = LCA_{\mathcal{T}}(c_0)$, it follows from Lemma 9. If $u_0 \neq LCA_{\mathcal{T}}(c_0)$, since $c_0 \subseteq L(u_0)$, there must be a child w_0 of u_0 such that $c_0 \subseteq L(w_0)$. Otherwise, $LCA_{\mathcal{T}}(c_0)$ should have been u_0. On the other hand, because $L(w_0) \cap c_0 \neq \emptyset$, $L(w_0) \subseteq c_0$. Thus $c_0 = L(w_0)$. Namely, if $u_0 \neq LCA_{\mathcal{T}}(c_0)$, it must be the parent of $w_0 = LCA_{\mathcal{T}}(c_0)$, and $c_0 = L(w_0)$. c_0 is then consistent with S since $L(w_0) \in S$. ◀

We have the following lemma to help us find such u_0.

▶ **Lemma 11.** *Suppose x is an arbitrary leaf of $\mathcal{T}$ such that $x \in c_0$. Let u_0 be the lowest ancestor of x that $L(u_0) \nsubseteq c_0$. c_0 is consistent with the signature S of $\mathcal{T}$ if and only if u_0 satisfies the conditions in Corollary 10.*

Besides, let p be the path from the root to u_0. If c_0 is consistent with signature S, for each proper ancestors u of u_0, $L(u) \nsubseteq c_0$, and for each proper descendant u of u_0 on path p, $L(u) \subseteq c_0$.

Proof. If c_0 is consistent with S, by Corollary 10, there is a node $u_0 \in T$ satisfying the conditions. Then by $c_0 \subseteq L(u_0)$, we know x is in the subtree of u_0. In other words, u_0 is on path p.

For each proper ancestor u of u_0, $c_0 \subseteq L(u_0) \subset L(u)$. Thus $L(u) \nsubseteq c_0$. Suppose the child of u_0 on path p is w_0. Since leaf $x \in c_0 \cap L(w_0)$, we know $L(w_0) \subseteq c_0$ from the conditions in Corollary 10. For all proper descendants u of u_0 on path p, $L(u) \subseteq L(w_0) \subseteq c_0$. Thus u_0 is the lowest ancestor of x that $L(u_0) \nsubseteq c_0$. ◀

By Lemma 11, we can perform a binary search on path p to find the lowest ancestor u_0 of x that $L(u_0) \nsubseteq c_0$. We will show how to check whether $L(u) \nsubseteq c_0$ for an arbitrary node $u \in \mathcal{T}$ efficiently in Section 4.

Finally, we discuss how $\mathcal{T}$ should change when we add c_0.

▶ **Lemma 12.** *Suppose S is consistent with c_0 and $c_0 \notin S$. When adding c_0 to S and adding the new node (corresponding to c_0) to $\mathcal{T}$, the u_0 in Lemma 11 should be the parent of the new node on $\mathcal{T}$. The children of the new node should be all children w of u_0 such that $L(w) \subset c_0$.*

Proof. Since the leaf $x \in c_0$ (in Lemma 11), all inner nodes u with $c_0 \subset L(u)$ are on path p, the path from the root to x. Thus by Lemma 11, $L(u_0)$ is the smallest set in S that contains c_0. Then the first claim follows from Observation 2. For those children w of u_0 such that $L(w) \subset c_0$, c_0 becomes the smaller set containing them. Thus they must change their parent. For other nodes u, such that $L(u) \subset c_0$, by the conditions of Corollary 10, u must be within the subtree of a child of u_0. Then that child is a smaller set containing it. Thus their parents stay unchanged. ◀

Algorithm 2 Our algorithm to check consistency and update $\mathcal{T}$.

```
1:  Pick an arbitrary species in $c_0$, and $x \leftarrow$ the corresponding leaf of $\mathcal{T}$
2:  Binary search the path from root to $x$.
3:  $u_0 \leftarrow$ the lowest node $u$ on the path that $L(u) \nsubseteq c_0$
4:  $sum \leftarrow \sum_{\text{child } w \text{ of } u_0, L(w) \subseteq c_0} |L(w)|$
5:  if $sum = |c_0|$ then
6:      $c_0$ is consistent with $S$, and we add it to consensus.
7:      We add a new node $w'$ (corresponding to $c_0$) to $\mathcal{T}$
8:      for child $w$ of $u_0$ do
9:          if $L(w) \subset c_0$ then
10:             Move $w$ to be a child of $w'$
11:     Let $w'$ be a child of $u_0$
```

Details of our algorithm are presented in Algorithm 2.

▶ **Lemma 13.** *$sum = |c_0|$ at Line 5, Algorithm 2 if and only if c_0 is consistent with S.*

Proof. If c_0 is consistent with S, by Lemma 11 we must find such a node u_0. On the other hand, if c_0 is not consistent with S, by Corollary 10, there is no such u_0. [1] ◀

To test whether $L(u) \subseteq c_0$ at Line 3, we need the following lemma. Recall $LCA_i(c)$ is the least common ancestor on tree T_i for cluster c.

▶ **Lemma 14.** *Suppose $c_0 = L(v)$ where v is a node in T_i. For a node u in $\mathcal{T}$, $L(u) \subseteq c_0$ if and only if $LCA_i(L(u))$ is in the subtree of v.*

Proof. If $L(u) \subseteq c_0 = L(v)$, every leaf in $L(u)$ is a descendant of v. So $LCA_{T_i}(L(u))$ is in the subtree of v. (note when $c_0 = L(u)$, it is still true since $LCA_i(L(u)) = v$)

Conversely, if $LCA_i(L(u))$ is a descendant of v, since leaves in $L(u)$ are in the subtree of $LCA_i(L(u))$ on T_i, they are also in the subtree of $L(v)$. Thus $L(u) \subseteq L(v) = c_0$. ◀

Thus the subset queries at Line 11 of Algorithm 2 are turned into LCA queries on phylogenetic tree T_i. Also for the summation query at Line 4, $L(w) \subseteq c_0$ if and only if $LCA_i(L(w))$ is in the subtree of v.

Since T_i is a fixed static tree, finding $LCA_i(L(w))$ is tractable in polylogarithmic time. Details are presented in the next section.

[1] Note if c_0 is not consistent with S, u_0 may not equal to $LCA_{\mathcal{T}}(c)$. In this case, we have to refer back to Corollary 10 instead of Lemma 9.

4 Efficiency

In this section, we will show that our algorithm can be implemented efficiently. There are two kinds of queries in Algorithm 2:

1. $\boldsymbol{lca(u, i)}$: Given $u \in \mathcal{T}$, return $LCA_i(L(u))$ on tree T_i. (At Line 3, to see whether $L(u) \subseteq c_0$, by Lemma 14, we need to find out $LCA_i(L(u))$)
2. $\boldsymbol{sum(u, v, i)}$: Given $u \in \mathcal{T}$ and $v \in T_i$, evaluate the summation

$$\sum_{\substack{\text{child } w \text{ of } u \\ LCA_i(L(w)) \in subtree(v) \text{ on } T_i}} |L(w)|$$

   (By Lemma 14, it equals the summation sum at Line 4)

After checking consistency, in Line 7 $\sim$ 11, Algorithm 2, the consensus tree is dynamically updated. The following update operation is needed:

1. $\boldsymbol{add(u, v, i)}$: Given $u \in \mathcal{T}$ and $v \in T_i$, add a new node w' to be a child of u. For all children w of u such that $LCA_i(L(w))$ is in the subtree of v, move them to be the children of the new node w'.

4.1 Data Structures

Recall $E(T)$ is the Euler tour of T where we only keep the first occurrence of each node. For node $v \in T_i$, the subtree of $L(v)$ corresponds to a continuous interval $[l_{T_i}(v), r_{T_i}(v)]$ in $E(T_i)$. Let $l_i(v)$ and $r_i(v)$ be the shorthands for $l_{T_i}(v)$ and $r_{T_i}(v)$.

The data structures we use have three components:

1. For each phylogenetic tree T_i, we maintain a top tree T_i'. Thus by Lemma 6, we can answer the least common ancestor of two nodes in $\tilde{O}(1)$ time.
2. For the consensus tree $\mathcal{T}$, we maintain its structure with a top tree $\mathcal{T}'$. By Lemma 6, we can answer k-th ancestor query in $\tilde{O}(1)$ time. This is for the binary search at Line 3, Algorithm 2.
3. For each node $u \in \mathcal{T}$, we use k splay trees $S_1, \ldots, S_k$ to maintain all its children w. The key of child w in the i-th tree is $l_i(LCA_i(L(w)))$, the position of $LCA_i(L(w))$ in $E(T_i)$. We maintain the following two values for each child w:
   - $|L(w)|$: The corresponding operation is integer additions.
   - $LCA_i(L(w))$: The corresponding operation is the least common ancestor of two nodes on T_i.

   Thus the splay trees support the following two kinds of queries:
   - $Sum\_Size(k_1, k_2)$: return the summation of the first value of each w whose key is in range $[k_1, k_2]$.
   - $Sum\_LCA(k_1, k_2)$: return the LCA of the second value of each w whose key is in range $[k_1, k_2]$.

Here the splay trees can be replaced with any balanced search trees with merge and split operations.

For each node $u \in \mathcal{T}$, we compute $LCA_i(L(u))$ for all i once we add u to our consensus tree and store these k numbers at node u. Namely, we compute them during the updates, not the queries.

4.2 Handle Update

Recall $add(u, v, i)$ requires us to do the following:

- add a new node w' to be a child of u on the consensus tree $\mathcal{T}$
- move some children of u to be children of w'

Let the set of all children of u be C and those children need to move be W. $par(u)$ denotes the parent of u.

Here we use the following idea from [16]:

- If $|W| \leq |C|/2$, we add a new node w' to be a child of u. Then we move nodes in W to be the children of w' one by one.
- If $|W| > |C|/2$, instead of moving nodes in W, we move nodes in $C \backslash W$. We disconnect u from $par(u)$ and connect w' with $par(u)$. Namely, replace u with w'. Then we make u a child of w'. For all children of u in $C \backslash W$, we move them to be the children of w'.

In this way, we need to move at most $\min\{|W|, |C| - |W|\}$ nodes. The details are presented in Algorithm 3.

Algorithm 3 Handle update $add(u, v, i)$.

```
1:  $C \leftarrow$ the set of all children of $u$
2:  $W \leftarrow \{w \in C | LCA_i(L(w)) \in subtree(v)$ on $T_i\}$
3:  if $|W| \leq |C|/2$ then
4:      Add a new node $w'$ to be a child of $u$ on Top tree $\mathcal{T}'$
5:      for $w \in W$ do
6:          Cut the edge between $w$ and $u$ on $\mathcal{T}'$ and connect $w$ with $w'$
7:          Remove $w$ from the splay trees at $u$. Insert $w$ into splay trees at $w'$
8:      for $i \in k$ do
9:          Compute $LCA_i(L(w')) \leftarrow Sum\_LCA(1, n)$ by query the splay tree $S_i$ at node $w'$
10: else
11:     Add a new node $w'$ to replace $u$, and make $u$ a child of $w'$ on Top tree $\mathcal{T}'$
12:     for $w \in C - W$ do
13:         Cut the edge between $w$ and $u$ on $\mathcal{T}'$ and connect $w$ with $w'$
14:         Remove $w$ from the splay trees at $u$. Insert $w$ into splay trees at $w'$
15:     for $i \in k$ do
16:         $LCA_i(L(w))$ gets the answer we stored at node $u$ before
17:         Compute $LCA_i(L(u)) \leftarrow Sum\_LCA(1, n)$ by query the splay tree $S_i$ at node $u$
```

▶ **Lemma 15.** *Each node is moved at most $O(\log n)$ times.*

Proof. No matter we move children in W or $C \backslash W$, the set C is eventually divided into two sets $C \backslash W$ and W after the procedure. Since we always move the nodes in the smaller set, each time we move a node, the size of the set containing it is at least halved. Or equivalently, the number of its siblings is at least halved. Since initially the root has n children, every node is in a set of size n. Each node can be moved at most $O(\log n)$ times. ◀

▶ **Lemma 16.** *All updates $add(u, v, i)$ take $\tilde{O}(kn)$ time in total.*

Proof. See Algorithm 3.

At Line 2, we need to implicitly find out the set W and get its size. By Observation 5, the subtree of v forms a continuous interval in $E(T_i)$. Then the size of W is just the number of nodes with keys within $[l_i(v), r_i(v)]$ in splay tree S_i. We can split this part out from S_i into a splay tree S'. The size of S' is just the size of W.

Then at Line 6, 7, 13, 14, each time we move a node, by Lemma 6 and Lemma 7, it takes $\tilde{O}(k)$ time. (The bottleneck is to delete and insert nodes at all k splay trees) By Lemma 15, each node is moved at most $O(\log n)$ times. Since the consensus tree has at most n nodes in the end, in total this part takes $\tilde{O}(kn)$ time.

At Line 9, and 17, we need to query k splay trees for each node inserted to the consensus tree. By Lemma 7, this takes $\tilde{O}(k)$ time for each node inserted. Thus in total, we need $\tilde{O}(kn)$ time. ◀

4.3 Handle Queries

▶ **Lemma 17.** *The queries $lca(u, i)$ and $sum(u, v, i)$ can be answered in $\tilde{O}(1)$ time.*

Proof. For query $lca(u, i)$, we return the $LCA_i(L(w))$ we computed at Line 17, Algorithm 3 when adding u.

For $sum(u, v, i)$, by Observation 5, all children w that $LCA_i(L(w)) \in subtree(v)$ on T_i are in a continuous interval of $E(T_i)$, namely $[l_i(v), r_i(v)]$. To answer $sum(u, v, i)$, we perform $Sum(l_i(v), r_i(v))$ on splay tree S_i for the second value, namely $|L(w)|$, and we return the summation to be the answer. ◀

4.4 Time complexity

▶ **Theorem 18.** *Greedy consensus tree of k phylogenetic trees for n species can be constructed in $\tilde{O}(kn)$ time.*

Proof. Recall the construction of greedy consensus tree contains two phases:

1. Count the frequency $f(c)$ of clusters and sort them. (Line 1 $\sim$ 3 of Algorithm 1)
2. Repeatedly run the greedy procedure. (Line 4 $\sim$ 7 of Algorithm 1)

By Lemma 4, we can get the identifier of each cluster and handle the first phase in $\tilde{O}(kn)$ time.

To handle the second phase, we run our Algorithm 2.

- For the binary search at Line 3, by Lemma 6, we can randomly access the path by k-th ancestor query in $\tilde{O}(1)$ time. By Lemma 14 and Lemma 17, we can check whether $L(u) \subseteq c_0$ in $\tilde{O}(1)$ time.
- For the summation at Line 4, by Lemma 17, can also be evaluated in $\tilde{O}(1)$ time.

Thus for each of the kn clusters, checking consistency takes $\tilde{O}(1)$ time. Then it takes $\tilde{O}(kn)$ time in total.

For Line 7 $\sim$ 11 of Algorithm 2, we run Algorithm 3. By Lemma 3, this part takes $\tilde{O}(kn)$ time in total.

Thus our algorithm takes $\tilde{O}(kn)$ time. ◀

References

1 Edward N Adams III. Consensus techniques and the comparison of taxonomic trees. *Systematic Biology*, 21(4):390–397, 1972.

2 Alfred V Aho, John E Hopcroft, and Jeffrey D Ullman. On finding lowest common ancestors in trees. *SIAM Journal on computing*, 5(1):115–132, 1976.

3 Stephen Alstrup, Jacob Holm, Kristian De Lichtenberg, and Mikkel Thorup. Maintaining information in fully dynamic trees with top trees. *Acm Transactions on Algorithms (talg)*, 1(2):243–264, 2005.

4 Nina Amenta, Frederick Clarke, and Katherine St John. A linear-time majority tree algorithm. In *International Workshop on Algorithms in Bioinformatics*, pages 216–227. Springer, 2003.

5 Amihood Amir and Dmitry Keselman. Maximum agreement subtree in a set of evolutionary trees: Metrics and efficient algorithms. *SIAM Journal on Computing*, 26(6):1656–1669, 1997.

6 Md Shamsuzzoha Bayzid, Siavash Mirarab, Bastien Boussau, and Tandy Warnow. Weighted statistical binning: enabling statistically consistent genome-scale phylogenetic analyses. *PLoS One*, 10(6):e0129183, 2015.

7 Md Shamsuzzoha Bayzid and Tandy Warnow. Naive binning improves phylogenomic analyses. *Bioinformatics*, 29(18):2277–2284, 2013.

8 Kåre Bremer. Combinable component consensus. *Cladistics*, 6(4):369–372, 1990.

9 David Bryant. A classification of consensus methods for phylogenetics. *DIMACS series in discrete mathematics and theoretical computer science*, 61:163–184, 2003.

10 William HE Day. Optimal algorithms for comparing trees with labeled leaves. *Journal of classification*, 2(1):7–28, 1985.

11 James H Degnan, Michael DeGiorgio, David Bryant, and Noah A Rosenberg. Properties of consensus methods for inferring species trees from gene trees. *Systematic Biology*, 58(1):35–54, 2009.

12 Martin Farach, Teresa M Przytycka, and Mikkel Thorup. Computing the agreement of trees with bounded degrees. In *European Symposium on Algorithms*, pages 381–393. Springer, 1995.

13 Martin Farach and Mikkel Thorup. Optimal evolutionary tree comparison by sparse dynamic programming. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 770–779. IEEE, 1994.

14 James S Farris. On comparing the shapes of taxonomic trees. *Systematic Zoology*, 22(1):50–54, 1973.

15 J Felsenstein. Phylip version 3.6. *Software package, Department of Genome Sciences, University of Washington, Seattle, USA*, 2005.

16 Pawel Gawrychowski, Gad M. Landau, Wing-Kin Sung, and Oren Weimann. A faster construction of greedy consensus trees. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 63:1–63:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. `doi:10.4230/LIPIcs.ICALP.2018.63`.

17 Jotun Hein, Tao Jiang, Lusheng Wang, and Kaizhong Zhang. On the complexity of comparing evolutionary trees. In *Annual Symposium on Combinatorial Pattern Matching*, pages 177–190. Springer, 1995.

18 Monika Rauch Henzinger, Valerie King, and Tandy Warnow. Constructing a tree from homeomorphic subtrees, with applications to computational evolutionary biology. *Algorithmica*, 24(1):1–13, 1999.

19 Jesper Jansson, Zhaoxian Li, and Wing-Kin Sung. On finding the adams consensus tree. *Information and Computation*, 256:334–347, 2017.

20 Jesper Jansson, Ramesh Rajaby, and Wing-Kin Sung. Minimal phylogenetic supertrees and local consensus trees. *AIMS Medical Science*, 5(2):181, 2018.

21 Jesper Jansson, Chuanqi Shen, and Wing-Kin Sung. Algorithms for the majority rule (+) consensus tree and the frequency difference consensus tree. In *International Workshop on Algorithms in Bioinformatics*, pages 141–155. Springer, 2013.

22 Jesper Jansson, Chuanqi Shen, and Wing-Kin Sung. Improved algorithms for constructing consensus trees. *Journal of the ACM (JACM)*, 63(3):28, 2016.

23 Jesper Jansson, Wing-Kin Sung, Hoa Vu, and Siu-Ming Yiu. Faster algorithms for computing the r* consensus tree. *Algorithmica*, 76(4):1224–1244, 2016.

24 Erich D Jarvis, Siavash Mirarab, Andre J Aberer, Bo Li, Peter Houde, Cai Li, Simon YW Ho, Brant C Faircloth, Benoit Nabholz, Jason T Howard, et al. Whole-genome analyses resolve early branches in the tree of life of modern birds. *Science*, 346(6215):1320–1331, 2014.

25 Sampath Kannan, Tandy Warnow, and Shibu Yooseph. Computing the local consensus of trees. *SIAM Journal on Computing*, 27(6):1695–1724, 1998.

26 Liang Liu, Lili Yu, and Scott V Edwards. A maximum pseudo-likelihood approach for estimating species trees under the coalescent model. *BMC evolutionary biology*, 10(1):302, 2010.

27 Liang Liu, Lili Yu, Laura Kubatko, Dennis K Pearl, and Scott V Edwards. Coalescent methods for estimating phylogenetic trees. *Molecular Phylogenetics and Evolution*, 53(1):320–328, 2009.

28 Timothy Margush and Fred R McMorris. Consensus n-trees. *Bulletin of Mathematical Biology*, 43(2):239–244, 1981.

29 Kurt Mehlhorn, Rajamani Sundar, and Christian Uhrig. Maintaining dynamic sequences under equality tests in polylogarithmic time. *Algorithmica*, 17(2):183–198, 1997.

30 Siavash Mirarab, Md Shamsuzzoha Bayzid, and Tandy Warnow. Evaluating summary methods for multilocus species tree estimation in the presence of incomplete lineage sorting. *Systematic Biology*, 65(3):366–380, 2014.

31 James B Pease, David C Haak, Matthew W Hahn, and Leonie C Moyle. Phylogenomics reveals three sources of adaptive variation during a rapid radiation. *PLoS Biology*, 14(2):e1002379, 2016.

32 Fredrik Ronquist and John P Huelsenbeck. Mrbayes 3: Bayesian phylogenetic inference under mixed models. *Bioinformatics*, 19(12):1572–1574, 2003.

33 Leonidas Salichos and Antonis Rokas. Inferring ancient divergences requires genes with strong phylogenetic signals. *Nature*, 497(7449):327, 2013.

34 Leonidas Salichos, Alexandros Stamatakis, and Antonis Rokas. Novel information theory-based measures for quantifying incongruence among phylogenetic trees. *Molecular Biology and Evolution*, 31(5):1261–1271, 2014.

35 Daniel Dominic Sleator and Robert Endre Tarjan. Self-adjusting binary search trees. *Journal of the ACM (JACM)*, 32(3):652–686, 1985.

36 Jordan V Smith, Edward L Braun, and Rebecca T Kimball. Ratite nonmonophyly: independent evidence from 40 novel loci. *Systematic Biology*, 62(1):35–49, 2012.

37 Alexandros Stamatakis. Raxml version 8: a tool for phylogenetic analysis and post-analysis of large phylogenies. *Bioinformatics*, 30(9):1312–1313, 2014.

38 Alexandros Stamatakis, Paul Hoover, and Jacques Rougemont. A rapid bootstrap algorithm for the raxml web servers. *Systematic biology*, 57(5):758–771, 2008.

39 Wing-Kin Sung. Greedy consensus tree and maximum greedy consensus tree problems. In *International Workshop on Algorithms and Computation*, pages 305–316. Springer, 2019.

40 DL Swofford. Paup*, version 4.0. software package, 2003.

Decision Problems in Information Theory

Mahmoud Abo Khamis
relationalAI, Berkeley, CA, USA

Phokion G. Kolaitis
University of California, Santa Cruz, CA, USA
IBM Research – Almaden, CA, USA

Hung Q. Ngo
relationalAI, Berkeley, CA, USA

Dan Suciu
University of Washington, Seattle, WA, USA

Abstract

Constraints on entropies are considered to be the laws of information theory. Even though the pursuit of their discovery has been a central theme of research in information theory, the algorithmic aspects of constraints on entropies remain largely unexplored. Here, we initiate an investigation of decision problems about constraints on entropies by placing several different such problems into levels of the arithmetical hierarchy. We establish the following results on checking the validity over all almost-entropic functions: first, validity of a Boolean information constraint arising from a monotone Boolean formula is co-recursively enumerable; second, validity of "tight" conditional information constraints is in Π^0_3. Furthermore, under some restrictions, validity of conditional information constraints "with slack" is in Σ^0_2, and validity of information inequality constraints involving max is Turing equivalent to validity of information inequality constraints (with no max involved). We also prove that the classical implication problem for conditional independence statements is co-recursively enumerable.

2012 ACM Subject Classification Mathematics of computing → Information theory; Theory of computation → Computability; Theory of computation → Complexity classes

Keywords and phrases Information theory, decision problems, arithmetical hierarchy, entropic functions

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.106

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

Related Version A full version of the paper is available at `https://arxiv.org/abs/2004.08783`.

Funding *Phokion G. Kolaitis*: Research partially supported by NSF Grant IIS-1814152.
Dan Suciu: Research partially supported by NSF IIS-1907997, NSF III-1703281, NSF III-1614738, and NSF AiTF-1535565.

Acknowledgements We thank Miika Hannula for several useful pointers to earlier work on the implication problem for conditional independence.

1 Introduction

The study of constraints on entropies is a central topic of research in information theory. In fact, more than 30 years ago, Pippenger [40] asserted that constraints on entropies are the "*laws of information theory*" and asked whether the *polymatroidal axioms* form the complete laws of information theory, i.e., whether every constraint on entropies can be derived from the polymatroidal axioms. These axioms consist of the following three types of constraints: (1) $H(\varnothing) = 0$, (2) $H(X) \leq H(X \cup Y)$ (monotonicity), and (3) $H(X) + H(Y) \geq H(X \cap Y) + H(X \cup Y)$ (submodularity). It is known that the polymatroidal axioms are

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 106; pp. 106:1–106:20
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

equivalent to Shannon's basic inequalities, that is, to the non-negativity of the entropy, conditional entropy, mutual information, and conditional mutual information [46]. In a celebrated result published in 1998, Zhang and Yeung [51] answered Pippenger's question negatively by finding a linear inequality that is satisfied by all entropic functions, but cannot be derived from the polymatroidal axioms.

Zhang and Yeung's result became the catalyst for the discovery of other information laws that are not captured by the polymatroidal axioms (e.g., [25, 34]). In particular, we now know that there are more elaborate laws, such as conditional inequalities, or inequalities expressed using max, which find equally important applications in a variety of areas. For example, implications between conditional independence statements of discrete random variables can be expressed as conditional information inequalities. In another example, we have recently shown that conjunctive query containment under bag semantics is at least as hard as checking information inequalities using max [1]. Despite the extensive research on various kinds of information inequalities, to the best of our knowledge nothing is known about the algorithmic aspects of the associated decision problem: check whether a given information law is valid.

In this paper, we initiate a study of algorithmic problems that arise naturally in information theory, and establish several results. To this effect, we introduce a generalized form of information inequalities, which we call *Boolean information constraints*, consisting of Boolean combinations of linear information inequalities, and define their associated decision problems. Since it is still an open problem whether linear information inequalities, which are the simplest kind of information laws, are decidable, we focus on placing these decision problems in the arithmetical hierarchy, also known as the Kleene-Mostowski hierarchy [41]. The arithmetical hierarchy has been studied by mathematical logicians since the late 1940s; moreover, it directly influenced the introduction and study of the polynomial-time hierarchy by Stockmeyer [43]. The first level of the arithmetical hierarchy consists of the collection Σ_1^0 of all recursively enumerable sets and the collection Π_1^0 of the complements of all recursively enumerable sets. The higher levels Σ_n^0 and Π_n^0, $n \geq 2$, are defined using existential and universal quantification over lower levels. We prove a number of results, including the following.

(1) Checking the validity of a Boolean information constraint arising from a monotone Boolean formula (in particular, a max information inequality) is in Π_1^0 (Theorem 7).
(2) Checking the validity of a conditional information inequality whose antecedents are "tight" is in Π_3^0 (Corollary 11). "Tight" inequalities are defined in Section 4.2.2, and include conditional independence assertions between random variables.
(3) Checking the validity of a conditional information inequality whose antecedents have "slack" and are group-balanced is in Σ_2^0 (Corollary 14).
(4) Checking the validity of a group-balanced, max information inequality is Turing equivalent to checking the validity of an information inequality (Corollary 17).

While the decidability of linear information inequalities (the simplest kind considered in this paper) remains open, a separate important question is whether more complex Boolean information constraints are any harder. For example, some conditional inequalities, or some max-inequalities can be proven from a simple linear inequality, hence they do not appear to be any harder. However, Kaced and Romashchenko [25] proved that there exist conditional inequalities that are *essentially conditional*, which means that they do not follow from a linear inequality. (We give an example in Equation (9).) We prove here that any conditional information inequality with slack is essentially *unconditioned* (Corollary 10; see also Equation(19)), and that any max-inequality also follows from a single linear inequality (Theorem 16).

A subtle complication involving these results is whether by “validity” it is meant that the given Boolean information constraint holds for the set of all entropic vectors over n variables, denoted by Γ_n^*, or for its topological closure, denoted by $\overline{\Gamma}_n^*$. It is well known that these two spaces differ for all $n \geq 3$. With the exception of (1) above, which holds for both Γ_n^* and $\overline{\Gamma}_n^*$, our results are only for $\overline{\Gamma}_n^*$. A problem of special interest is the implication between conditional independence statements of discrete random variables, and this amounts to checking the Γ_n^*-validity of a tight conditional information inequality; it is known that this problem is not finitely axiomatizable [44], and its decidability remains open. Our result (2) above does not apply here because it is a statement about $\overline{\Gamma}_n^*$-validity. However, we prove that the implication problem for conditional independence statements is in Π_1^0 (Theorem 8).

2 Background and Notations

Throughout this paper, vectors and tuples are denoted by bold-faced letters, and random variables are capitalized. We write $\boldsymbol{x} \cdot \boldsymbol{y} \stackrel{\text{def}}{=} \sum_i x_i y_i$ for the dot product of $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^m$. For a given set $S \subseteq \mathbb{R}^m$, S is *convex* if $\boldsymbol{x}, \boldsymbol{y} \in S$ and $\theta \in [0,1]$ implies $\theta\boldsymbol{x} + (1-\theta)\boldsymbol{y} \in S$; S is called a *cone* if $\boldsymbol{x} \in S$ and $\theta \geq 0$ implies $\theta\boldsymbol{x} \in S$; the topological closure of S is denoted by $\overline{S}$; and, finally, $S^* \stackrel{\text{def}}{=} \{\boldsymbol{y} \mid \forall \boldsymbol{x} \in S, \boldsymbol{x} \cdot \boldsymbol{y} \geq 0\}$ denotes the *dual cone* of S. It is known that S^* is always a closed, convex cone. We provide more background in the full version [2].

For a random variable X with a fixed finite domain D and a probability mass function (pmf) p, its (binary) *entropy* is defined by

$$H(X) \stackrel{\text{def}}{=} -\sum_{x \in D} p(x) \cdot \log p(x) \tag{1}$$

In this paper all logarithms are in base 2.

Fix a joint distribution over n finite random variables $\boldsymbol{V} \stackrel{\text{def}}{=} \{X_1, \ldots, X_n\}$. For each $\alpha \subseteq [n]$, let $\boldsymbol{X}_\alpha$ denote the random (vector-valued) variable $(X_i : i \in \alpha)$. Define the set function $h : 2^{[n]} \to \mathbb{R}_+$ by setting $h(\alpha) \stackrel{\text{def}}{=} H(\boldsymbol{X}_\alpha)$, for all $\alpha \subseteq [n]$. With some abuse, we blur the distinction between the set $[n]$ and the set of variables $\boldsymbol{V} = \{X_1, \ldots, X_n\}$, and write $H(\boldsymbol{X}_\alpha)$, $h(\boldsymbol{X}_\alpha)$, or $h(\alpha)$ interchangeably. We call the function h an *entropic function*, and also identify it with a vector $\boldsymbol{h} \stackrel{\text{def}}{=} (h(\alpha))_{\alpha \subseteq [n]} \in \mathbb{R}_+^{2^n}$, which is called an *entropic vector*. Note that most texts and papers on this topic drop the component $h(\emptyset)$, which is always 0, leading to entropic vectors in $\mathbb{R}^{2^n - 1}$. We prefer to keep the $\emptyset$-coordinate to simplify notations. The implicit assumption $h(\emptyset) = 0$ is used through the rest of the paper.

The set of entropic functions/vectors is denoted by $\Gamma_n^* \subseteq \mathbb{R}_+^{2^n}$. Its topological closure, denoted by $\overline{\Gamma}_n^*$, is the set of *almost entropic* vectors (or functions). It is known [46] that $\Gamma_n^* \subsetneq \overline{\Gamma}_n^*$ for $n \geq 3$. In general, Γ_n^* is neither a cone nor convex, but its topological closure $\overline{\Gamma}_n^*$ is a closed convex cone [46].

Every entropic function h satisfies the following *basic Shannon inequalities*:

$$h(\boldsymbol{Y} \cup \boldsymbol{X}) \geq h(\boldsymbol{X}) \qquad\qquad h(\boldsymbol{X}) + h(\boldsymbol{Y}) \geq h(\boldsymbol{X} \cup \boldsymbol{Y}) + h(\boldsymbol{X} \cap \boldsymbol{Y})$$

called *monotonicity* and *submodularity* respectively. Any inequality obtained by taking a positive linear combination of Shannon inequalities is called a *Shannon-type inequality.*

Throughout this paper we will abbreviate the union $\boldsymbol{X} \cup \boldsymbol{Y}$ of two sets of variables as $\boldsymbol{XY}$. The quantities $h(\boldsymbol{Y}|\boldsymbol{X}) \stackrel{\text{def}}{=} h(\boldsymbol{XY}) - h(\boldsymbol{X})$ and $I_h(\boldsymbol{Y};\boldsymbol{Z}|\boldsymbol{X}) \stackrel{\text{def}}{=} h(\boldsymbol{XY}) + h(\boldsymbol{XZ}) - h(\boldsymbol{XYZ}) - h(\boldsymbol{X})$ are called the *conditional entropy* and the *conditional mutual information* respectively. It can be easily checked that $h(\boldsymbol{Y}|\boldsymbol{X}) \geq 0$ and $I_h(\boldsymbol{Y};\boldsymbol{Z}|\boldsymbol{X}) \geq 0$ are Shannon-type inequalities.

▶ Remark 1. The established notation Γ_n^* [47, 50, 11] for the set of entropic vectors is unfortunate, because the star in this context does **not** represent the dual cone. We will continue to denote by Γ_n^* the set of entropic vectors (which is not a cone!), and use explicit parentheses, as in $(\Gamma_n^*)^*$, to represent the dual cone.

3 Boolean information Constraints

Most of this paper considers the following problem: given a Boolean combination of information inequalities, check whether it is valid. However in Section 5 we briefly discuss the dual problem, namely, recognizing whether a given vector $\boldsymbol{h}$ is an entropic vector (or an almost entropic vector).

A *Boolean function* is a function $F : \{0,1\}^m \to \{0,1\}$. We often denote its inputs with variables $Z_1, \dots, Z_m \in \{0,1\}$, and write $F(Z_1, \dots, Z_m)$ for the value of the Boolean function.

3.1 Problem Definition

A vector $\boldsymbol{c} \in \mathbb{R}^{2^n}$ defines the following (linear) *information inequality*:

$$\boldsymbol{c} \cdot \boldsymbol{h} = \sum_{\alpha \subseteq [n]} c_\alpha h(X_\alpha) \geq 0. \tag{2}$$

The information inequality is said to be *valid* if it holds for all vectors $\boldsymbol{h} \in \Gamma_n^*$; equivalently, $\boldsymbol{c}$ is in the dual cone, $\boldsymbol{c} \in (\Gamma_n^*)^*$. By continuity, an information inequality holds $\forall \boldsymbol{h} \in \Gamma_n^*$ iff it holds $\forall \boldsymbol{h} \in \overline{\Gamma}_n^*$. In 1986, Pippenger [40] defined the "*laws of information theory*" as the set of all information inequalities, and asked whether all of them are Shannon-type inequalities. This was answered negatively by Zhang and Yeung in 1998 [51]. We know today that several applications require more elaborate laws, such as max-inequalities and conditional inequalities. Inspired by these new laws, we define the following generalization.

▶ **Definition 2.** *To each Boolean function F with m inputs, and every m vectors $\boldsymbol{c}_j \in \mathbb{R}^{2^n}, j \in [m]$, we associate the following* Boolean information constraint*:*

$$F(\boldsymbol{c}_1 \cdot \boldsymbol{h} \geq 0, \dots, \boldsymbol{c}_m \cdot \boldsymbol{h} \geq 0). \tag{3}$$

For a set $S \subseteq \mathbb{R}^{2^n}$, a Boolean information constraint is said to be *S-valid* if it holds for all $\boldsymbol{h} \in S$. Thus, we will distinguish between Γ_n^*-validity and $\overline{\Gamma}_n^*$-validity. Unlike in the case of information inequalities, these two notions of validity no longer coincide for arbitrary Boolean information constraints in general, as we explain in what follows.

▶ **Definition 3.** *Let F be a Boolean function. The* entropic Boolean information constraint *problem parameterized by F, denoted by $\mathsf{EBIC}(F)$, is the following: given m integer vectors $\boldsymbol{c}_j \in \mathbb{Z}^{2^n}$, where $j \in [m]$, check whether the constraint (3) holds for all entropic functions $\boldsymbol{h} \in \Gamma_n^*$. In the* almost-entropic *version, denoted by $\mathsf{AEBIC}(F)$, we replace Γ_n^* by $\overline{\Gamma}_n^*$.*

The inputs $\boldsymbol{c}_j, j \in [m]$, to these problems are required to be integer vectors in order for $\mathsf{EBIC}(F)$ and $\mathsf{AEBIC}(F)$ to be meaningful computational problems. Equivalently, one can require the inputs to be rational vectors $\boldsymbol{c}_j \in \mathbb{Q}^{2^n}, j \in [m]$.

Let F be a Boolean function. F can be written as a conjunction of clauses $F = C_1 \wedge C_2 \wedge \cdots$, where each clause is a disjunction of literals. Equivalently, a clause C has this form:

$$(Z_1' \wedge \cdots \wedge Z_k') \Rightarrow (Z_1 \vee \cdots \vee Z_\ell) \tag{4}$$

Problem	Abbreviation		Simple Example
	Entropic	Almost-entropic	
Boolean information constraint	EBIC(F)	AEBIC(F)	$h(XY) \le \frac{2}{3}h(XYZ) \Rightarrow$ $\max(h(YZ), h(XZ)) \ge \frac{2}{3}h(XYZ)$
Information Inequality	IIP		$h(XY) + h(YZ) + h(XZ) \ge 2h(XYZ)$
Max-Information Inequality	MaxIIP		$\max(h(XY), h(YZ), h(XZ)) \ge \frac{2}{3}h(XYZ)$
Conditional Information Inequality	ECIIP	AECIIP	$((h(XY) \le \frac{2}{3}h(XYZ)) \wedge (h(YZ) \le \frac{2}{3}h(XYZ)))$ $\Rightarrow h(XZ) \ge \frac{2}{3}h(XYZ)$
Conditional Independence	CI	(no name)	$(I(X;Y) = 0 \wedge I(X;Z\vert Y) = 0) \Rightarrow I(X;Z) = 0$

Figure 1 Notations for various Boolean Information Constraint Problems.

where $Z_1', \dots, Z_k', Z_1, \dots, Z_\ell$ are distinct Boolean variables. Checking EBIC(F) is equivalent to checking EBIC(C), for each clause of F (and similarly for AEBIC(F)); therefore and without loss of generality, we will assume in the rest of the paper that F consists of a single clause (4) and study the problem along these dimensions:

Conditional and Unconditional Constraints. When $k = 0$ (i.e., when the antecedent is empty), the formula F is *monotone*, and we call the corresponding Boolean information constraint *unconditional*. When $k > 0$, the formula F is *non-monotone*, and we call the corresponding constraint *conditional*.

Simple and Max Constraints. When $k = 0$ and $\ell = 1$, then we say that F defines a *simple* inequality; when $k = 0$ and $\ell > 1$, then we say that F defines a max-*inequality*. The case when $\ell = 0$ and $k > 0$ is not interesting because F is not valid, since the zero-vector $\boldsymbol{h} = \boldsymbol{0}$ violates the constraint.

3.2 Examples and Applications

This section presents examples and applications of Boolean Function Information Constraints and their associated decision problems. A summary of the notations is in Fig. 1.

3.2.1 Information Inequalities

We start with the simplest form of a Boolean information constraint, namely, the linear information inequality in Eq. (2), which arises from the single-variable Boolean formula Z_1. We will call the corresponding decision problem the *information-inequality problem*, denoted by IIP: given a vector of integers $\boldsymbol{c}$, check whether Eq. (2) is Γ_n^*-valid or, equivalently, $\overline{\Gamma}_n^*$-valid. Pippenger's question from 1986 was essentially a question about decidability. Shannon-type inequalities are decidable in exponential time using linear programming methods, and software packages have been developed for this purpose [46, Chapter 13] (it is not known, however, if there is a matching lower bound in the complexity of this problem). Thus, if every information inequality were a Shannon-type inequality, then information inequalities would be decidable. However, Zhang and Yeung's gave the first example of a non-Shannon-type information inequality [51]. Later, Matúš [34] proved that, when $n \ge 4$ variables, there exists infinitely many inequivalent non-Shannon entropic inequalities. More precisely, he proved that the following is a non-Shannon inequality, for every $k \ge 1$:

$$I_h(C;D|A) + \frac{k+3}{2}I_h(C;D|B) + I_h(A;B) + \frac{k-1}{2}I_h(B;C|D) + \frac{1}{k}I_h(B;D|C) \ge I_h(C;D) \tag{5}$$

This ruined any hope of proving decidability of information inequalities by listing a finite set of axioms. To date, the study of non-Shannon-type inequalities is an active area of research [49, 31, 48], and the question whether IIP is decidable remains open.

Hammer et al. [23], showed that, up to logarithmic precision, information inequalities are equivalent to linear inequalities in Kolmogorov complexity (see also [20, Theorem 3.5]).

3.2.2 Max Information Inequalities

Next, we consider constraints defined by a disjunction of linear inequalities, in other words $(\boldsymbol{c}_1 \cdot \boldsymbol{h} \geq 0) \vee \cdots \vee (\boldsymbol{c}_m \cdot \boldsymbol{h} \geq 0)$, where $\boldsymbol{c}_j \in \mathbb{R}^{2^n}$. This is equivalent to:

$$\max(\boldsymbol{c}_1 \cdot \boldsymbol{h}, \boldsymbol{c}_2 \cdot \boldsymbol{h}, \dots, \boldsymbol{c}_m \cdot \boldsymbol{h}) \geq 0 \tag{6}$$

and, for that reason, we call them *Max information inequalities* and denote the corresponding decision problem by MaxIIP. As before, Γ_n^*-validity and $\overline{\Gamma}_n^*$-validity coincide.

Application to Constraint Satisfaction and Database Theory. Given two finite structures $\boldsymbol{A}$ and $\boldsymbol{B}$, we write $\mathsf{HOM}(\boldsymbol{A}, \boldsymbol{B})$ for the set of homomorphisms from $\boldsymbol{A}$ to $\boldsymbol{B}$. We say that $\boldsymbol{B}$ *dominates* structure $\boldsymbol{A}$, denote by $\boldsymbol{A} \preceq \boldsymbol{B}$, if for every finite structure $\boldsymbol{C}$, we have that $|\mathsf{HOM}(\boldsymbol{A}, \boldsymbol{C})| \leq |\mathsf{HOM}(\boldsymbol{B}, \boldsymbol{C})|$. The *homomorphism domination problem* asks whether $\boldsymbol{A} \preceq \boldsymbol{B}$, given $\boldsymbol{A}$ and $\boldsymbol{B}$. In database theory this problem is known as the *query containment problem under bag semantics* [13]. In that setting we are given two Boolean conjunctive queries Q_1, Q_2, which we interpret using bag semantics, i.e., given a database D, the answer $Q_1(D)$ is the number of homomorphisms $Q_1 \to D$ [28]. Q_1 *is contained in* Q_2 *under bag semantics* if $Q_1(D) \leq Q_2(D)$ for every database D. It is open whether the homomorphism domination problem is decidable.

Kopparty and Rossman [29] described a MaxIIP problem that yields a sufficient condition for homomorphism domination. In recent work [1] we proved that, when $\boldsymbol{B}$ is acyclic, then that condition is also necessary, and, moreover, the domination problem for acyclic $\boldsymbol{B}$ is Turing-equivalent to MaxIIP. Hence, any result on the complexity of MaxIIP immediately carries over to the homomorphism domination problem for acyclic $\boldsymbol{B}$, and vice versa.

We illustrate here Kopparty and Rossman's MaxIIP condition on a simple example. Consider the following two Boolean conjunctive queries: $Q_1() = R(u,v) \wedge R(v,w) \wedge R(w,u)$, $Q_2() = R(x,y) \wedge R(x,z)$; interpreted using bag semantics, Q_1 returns the number of triangles and Q_2 the number of V-shaped subgraphs. Kopparty and Rossman proved that $Q_1 \preceq Q_2$ follows from the following max-inequality:

$$\max\{2h(XY) - h(X) - h(XYZ), 2h(YZ) - h(Y) - h(XYZ), 2h(XZ) - h(Z) - h(XYZ)\} \geq 0 \tag{7}$$

3.2.3 Conditional Information Inequalities

A *conditional information inequality* has the form:

$$(\boldsymbol{c}_1 \cdot \boldsymbol{h} \geq 0 \wedge \cdots \wedge \boldsymbol{c}_k \cdot \boldsymbol{h} \geq 0) \Rightarrow \boldsymbol{c}_0 \cdot \boldsymbol{h} \geq 0 \tag{8}$$

Here we need to distinguish between Γ_n^*-validity and $\overline{\Gamma}_n^*$-validity, and denote by ECIIP and AECIIP the corresponding decision problems. Notice that, without loss of generality, we can allow equality in the antecedent, because $\boldsymbol{c}_i \cdot \boldsymbol{h} = 0$ is equivalent to $\boldsymbol{c}_i \cdot \boldsymbol{h} \geq 0 \wedge -\boldsymbol{c}_i \cdot \boldsymbol{h} \geq 0$.

Suppose that there exist $\lambda_1 \geq 0, \dots, \lambda_m \geq 0$ such that the inequality $\boldsymbol{c}_0 \cdot \boldsymbol{h} - (\sum_i \lambda_i \boldsymbol{c}_i \cdot \boldsymbol{h}) \geq 0$ is valid; then Eq. (8) is, obviously, also valid. Kaced and Romashchenko [25] called Eq. (8) an *essentially conditioned inequality* if no such λ_i's exist, and discovered several valid conditional inequalities that are essentially conditioned.

Application to Conditional Independence. Fix three set of random variables $\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}$. A *conditional independence* (CI) statement is a statement of the form $\phi = (\boldsymbol{Y} \perp\!\!\!\perp \boldsymbol{Z} \mid \boldsymbol{X})$, and it asserts that $\boldsymbol{Y}$ and $\boldsymbol{Z}$ are independent conditioned on $\boldsymbol{X}$. A *CI implication* is a statement $\varphi_1 \wedge \cdots \wedge \varphi_k \Rightarrow \varphi_0$, where $\varphi_i, i \in \{0, \dots, k\}$ are CI statements. The *CI implication problem* is: given an implication, check if it is valid for all discrete probability distributions. Since $(\boldsymbol{Y} \perp\!\!\!\perp \boldsymbol{Z} \mid \boldsymbol{X}) \Leftrightarrow I_h(\boldsymbol{Y}; \boldsymbol{Z}|\boldsymbol{X}) = 0 \Leftrightarrow -I_h(\boldsymbol{Y}; \boldsymbol{Z}|\boldsymbol{X}) \geq 0$, the CI implication problem is a special case of ECIIP.

The CI implication problem has been studied extensively in the literature [30, 44, 18, 27]. Pearl and Paz [39] gave a sound, but incomplete, set of *graphoid axioms*, Studený [44] proved that no finite axiomatization exists, while Geiger and Pearl [18] gave a complete axiomatization for two restricted classes, called saturated, and marginal CIs. See [16, 21, 38] for some recent work on the CI implication problem. The decidability of the CI implication problem remains open to date.

Results in [25] imply that the following CI implication is essentially conditioned (see [27]):

$$I_h(C; D|A) = I_h(C; D|B) = I_h(A; B) = I_h(B; C|D) = 0 \Longrightarrow I_h(C; D) = 0 \tag{9}$$

While a CI implication problem is an instance of an *entropic* conditional inequality, one can also consider the question whether a CI implication statement holds for all *almost entropic* functions; for example the implication (9) holds for all almost entropic functions. Kaced and Romashchenko [25] proved that these two problems differ, by giving examples of CI implications that hold for all entropic functions but fail for almost entropic functions.

3.2.4 Group-Theoretic Inequalities

There turns out to be a way to "rephrase" IIP as a decision problem in group theory; This was a wonderful result by Chan and Yeung [12] (see also [11]). A tuple $(G; G_1, \dots, G_n)$ is called a *group system* if G is a finite group and $G_1, \dots, G_n \subseteq G$ are n subgroups. For any $\alpha \subseteq [n]$, define $G_\alpha := \bigcap_{i \in \alpha} G_i$; implicitly, we set $G_\varnothing := G$. A vector $\boldsymbol{c} \subseteq \mathbb{R}^{2^n}$ defines the following *group-theoretic inequality*:

$$\sum_{\alpha \subseteq [n]} c_\alpha \log \frac{|G|}{|G_\alpha|} \geq 0 \tag{10}$$

▶ **Theorem 4** ([12]). *An information inequality* (2) *is* Γ_n^**-valid if and only if the corresponding group-theoretic inequality* (10) *holds for all group systems* $(G, G_1, \dots, G_n)$*,*

In particular, a positive or negative answer to the decidability problem for IIP immediately carries over to the validity problem of group-theoretic inequalities of the form (10). We note that the group-theoretic inequalities considered here are different from the word problems in group, see e.g. the survey [35]; the undecidability results for word problems in groups do not carry over to the group-theoretic inequalities and, thus, to information inequalities.

3.2.5 Application to Relational Query Evaluation

The problem of bounding the number of copies of a graph inside of another graph has a long and interesting history [17, 5, 14, 36]. The subgraph homomorphism problem is a special case of the relational query evaluation problem, in which case we want to find an upper bound on the output size of a full conjunctive query. Using the entropy argument from [14], *Shearer's lemma* in particular, Atserias, Grohe, and Marx [6] established a tight upper bound on the answer to a full conjunctive query over a database. Note that Shearer's lemma is a Shannon-type inequality. Their result was extended to include functional dependencies and more generally degree constraints in a series of recent work in database theory [19, 3, 4]. All these results can be cast as applications of Shannon-type inequalities. For a simple example, let $R(X,Y), S(Y,Z), T(Z,U)$ be three binary relations (tables), each with N tuples, then their join $R(X,Y) \bowtie S(Y,Z) \bowtie T(Z,U)$ can be as large as N^2 tuples. However, if we further know that the functional dependencies $XZ \to U$ and $YU \to X$ hold in the output, then one can prove that the output size is $\leq N^{3/2}$, by using the following Shannon-type information inequality:

$$h(XY) + h(YZ) + h(ZU) + h(X|YU) + h(U|XZ) \geq 2h(XYZU) \tag{11}$$

While the tight upper bound of any conjunctive query can be proven using only Shannon-type inequalities, this no longer holds when the relations used in the query are constrained to satisfy functional dependencies. In that case, the tight upper bound can always be obtained from an information inequality, but Abo Khamis et al. [4] gave an example of a conjunctive query for which the tight upper bound requires a non-Shannon inequality.

3.2.6 Application to Secret Sharing

An interesting application of conditional information inequalities is secret sharing, which is a classic problem in cryptography, independently introduced by Shamir [42] and Blakley [8]. The setup is as follows. There is a set P of *participants*, a *dealer* $d \notin P$, and an *access structure* $\mathcal{F} \subset 2^P$. The access structure is closed under taking superset: $A \in \mathcal{F}$ and $A \subseteq B$ implies $B \in \mathcal{F}$. The dealer has a secret s, from some finite set K, which she would like to share in such a way that every set $F \in \mathcal{F}$ of participants can recover the secret s, but every set $F \notin \mathcal{F}$ knows *nothing* about s. The dealer shares her secret by using a *secret sharing scheme*, in which she gives each participant $p \in P$ a *share* $s_p \in K_p$, where K_p is some finite domain. The scheme is designed in such a way that from the tuple $(s_p)_{p \in F}$ one can recover s if $F \in \mathcal{F}$, and conversely one cannot infer any information about s if $F \notin \mathcal{F}$.

One way to formalize secret sharing uses information theory (for other formalisms, see [7]). We identify the participants P with the set $[n-1]$, and the dealer with the number n. A secret sharing scheme on P with access structure $\mathcal{F} \subseteq 2^P$ is a joint distribution on n discrete random variables $(X_1, \ldots, X_n)$ satisfying:

(i) $H(X_n) > 0$
(ii) $H(X_n \mid \boldsymbol{X}_F) = 0$ if $F \in \mathcal{F}$
(iii) $H(X_n \mid \boldsymbol{X}_F) = H(X_n)$ if $F \notin \mathcal{F}$; equivalently, $I_H(X_n; \boldsymbol{X}_F) = 0$.

Intuitively, X_i denotes the share given to the ith participant, and X_n is the unknown secret. It can be shown, without loss of generality, that (i) can be replaced by the assumption that the marginal distribution on X_n is uniform [9], which encodes the fact that the scheme does not reveal any information about the secret X_n. Condition (ii) means one can recover the secret from the shares of qualified participants, while condition (iii) guarantees the complete opposite. A key challenge in designing a good secret sharing scheme is to reduce the total size of the shares. The only known [15, 10, 26] way to prove a *lower bound* on share sizes is to lower bound the *information ratio* $\frac{\max_{p \in P} H(X_p)}{H(X_n)}$. In order to prove that some number ℓ is a lower bound on the information ratio, we need to check that $\max_{i \in [n-1]} \{ h(X_i) - \ell \cdot h(X_n) \} \geq 0$ holds for all entropic functions $\boldsymbol{h} \in \Gamma_n^*$ satisfying the extra conditions (i), (ii), and (iii) above. Equivalently, ℓ is a lower bound on the information ratio if and only if the following Boolean information constraint is Γ_n^*-valid:

$$\bigwedge_{F \in \mathcal{F}} (h(X_n \mid \boldsymbol{X}_F) = 0) \wedge \bigwedge_{F \notin \mathcal{F}} (I_h(X_n; \boldsymbol{X}_F) = 0) \Longrightarrow (h(X_n) = 0) \vee \Big[\bigvee_{i \in [n-1]} (h(X_i) \geq \ell \cdot h(X_n)) \Big]$$

4 Placing EBIC and AEBIC in the Arithmetical Hierarchy

What is the complexity of EBIC(F) / AEBIC(F)? Is it even decidable? As we have seen there are numerous applications of the Boolean Information Constraint problem, hence any positive or negative answer, even for special cases, would shed light on these applications. While their (un)decidability is currently open, in this paper we provide several upper bounds on their complexity, by placing them in the arithmetical hierarchy.

We briefly review some concepts from computability theory. In this setting it is standard to assume objects are encoded as natural numbers. A set $A \subseteq \mathbb{N}^k$, for $k \geq 1$, is *Turing computable*, or *decidable*, if there exists a Turing machine that, given $x \in \mathbb{N}^k$ decides whether $x \in A$. A set A is *Turing reducible* to B if there exists a Turing machine with an oracle for B that can decide membership in A. The *arithmetical hierarchy* consists of the classes of sets Σ_n^0 and Π_n^0 defined as follows. The class Σ_n^0 consists of all sets of the form $\{x \mid \exists y_1 \forall y_2 \exists y_3 \cdots \mathsf{Q} y_n R(x, y_1, \ldots, y_n)\}$, where R is an $(n+1)$-ary decidable predicate, $\mathsf{Q} = \exists$ if n is odd, and $Q = \forall$ if n is even. In a dual manner, the class Π_n^0 consists of sets of the form $\{x \mid \forall y_1 \exists y_2 \forall y_3 \cdots \mathsf{Q} y_n R(x, y_1, \ldots, y_n)\}$. Then $\Sigma_0^0 = \Pi_0^0$ are the decidable sets, while Σ_1^0 consists of the *recursively enumerable* sets, and Π_1^0 consists of the *co-recursively enumerable* sets. It is known that these classes are closed under union and intersection, but not under complements, and that they form a strict hierarchy, $\Sigma_n^0, \Pi_n^0 \subsetneq (\Sigma_{n+1}^0 \cap \Pi_{n+1}^0)$. For more background, we refer to [41]. Our goal is to place the problems EBIC(F), AEBIC(F), and their variants in concrete levels of the arithmetical hierarchy.

4.1 Unconditional Boolean Information Constraints

We start by discussing unconditional Boolean information constraints, or, equivalently, a Boolean information constraint defined by a monotone Boolean formula F. The results here are rather simple; we include them only as a warmup for the less obvious results in later sections. Based on our discussion in Sections 3.2.1 and 3.2.2, we have the following result.

▶ **Theorem 5.** *If F is monotone, then EBIC(F) and AEBIC(F) are equivalent problems.*

Next, we prove that these problems are co-recursively enumerable, by using the following folklore fact. A *representable set of n random variables* is a finite relation Ω with N rows and $n+1$ columns $X_1, \ldots, X_n, p$, where column p contains rational probabilities in $[0,1] \cap \mathbb{Q}$ that sum to 1. Thus, Ω defines n random variables with finite domain and probability mass given by rational numbers. We denote $\boldsymbol{h}^\Omega$ its entropic vector. By continuity of Eq.(1), we obtain:

▶ **Proposition 6.** *For every entropic vector $\boldsymbol{h} \in \Gamma_n^*$ and every $\varepsilon > 0$, there exists a representable space Ω such that $\|\boldsymbol{h} - \boldsymbol{h}^\Omega\| < \varepsilon$.*

The group-characterization proven by Chan and Yeung [12] implies a much stronger version of the proposition; we do not need that stronger version in this paper.

▶ **Theorem 7.** *Let F be a monotone Boolean formula. Then* $\mathsf{EBIC}(F)$ *(and, hence,* $\mathsf{AEBIC}(F)$*) is in Π^0_1, i.e., it is co-recursively enumerable.*

Proof. Fix $F = Z_1 \vee \cdots \vee Z_m$ and $\boldsymbol{c}_i \in \mathbb{Z}^{2^n}$, $i \in [m]$. We need to check:

$$\forall \boldsymbol{h} \in \Gamma^*_n : \qquad \boldsymbol{c}_1 \cdot \boldsymbol{h} \geq 0 \vee \cdots \vee \boldsymbol{c}_m \cdot \boldsymbol{h} \geq 0 \tag{12}$$

We claim that (12) is equivalent to:

$$\forall \Omega \qquad \boldsymbol{c}_1 \cdot \boldsymbol{h}^{\Omega} \geq 0 \vee \cdots \vee \boldsymbol{c}_m \cdot \boldsymbol{h}^{\Omega} \geq 0 \tag{13}$$

Obviously (12) implies (13), and the opposite follows from Prop. 6: if (12) fails on some entropic vector $\boldsymbol{h}$, then it also fails on some representable $\boldsymbol{h}^{\Omega}$ close enough to $\boldsymbol{h}$. Finally, (13) is in Π^0_1 because, the property after $\forall\Omega$ is decidable, by expanding the definition of entropy (1) in each condition $\boldsymbol{c}_i \cdot \boldsymbol{h}^{\Omega} \geq 0$, and writing the latter as $\sum_j a_j \log b_j \geq 0$, or, equivalently, $\prod_j (b_j)^{a_j} \geq 1$, where a_j, b_j are rational numbers, which is decidable. ◀

4.2 Conditional Boolean Information Constraints

We now consider non-monotone Boolean functions, in other words, conditional information constraints (8). Since Γ^*_n- and $\overline{\Gamma}^*_n$-validity no longer coincide, we study $\mathsf{EBIC}(F)$ and $\mathsf{AEBIC}(F)$ separately. The results here are non-trivial, and some proofs are deferred to [2].

4.2.1 The Entropic Case

Our result for $\mathsf{EBIC}(F)$ is restricted to the CI implication problem. Recall from Sec. 3.2.3 that this problem consists of checking whether an implication between statements of the form $(\boldsymbol{Y} \perp\!\!\!\perp \boldsymbol{Z} \mid \boldsymbol{X})$ holds for all random variables with finite domain, and this is equivalent to checking whether a certain conditional inequality holds for all entropic functions. We prove that this problem is in Π^0_1 by using Tarski's theorem of the decidability of the theory of reals with $+, *$ [45].

▶ **Theorem 8.** *The CI implication problem (Section 3.2.3) is in Π^0_1.*

Proof. Tarski has proven that the theory of reals with $+, *$ is decidable. More precisely, given a formula Φ in FO with symbols $+$ and $*$, it is decidable whether that formula is true in the model of real numbers $(\mathbb{R}, +, *)$; for example, it is decidable whether[1] $\Phi \equiv \forall x \exists y \forall z (x^2 + 3y \geq z \wedge (y^3 + yz \leq xy^2))$ is true. We will write $(\mathbb{R}, +, *) \models \Phi$ to denote the fact that Φ is true in the model of reals.

Consider a conditional inequality over a set of n joint random variables:

$$I_h(Y_1; Z_1|X_1) = 0 \wedge \cdots \wedge I_h(Y_k; Z_k|X_k) = 0 \Rightarrow I_h(Y; Z|X) = 0$$

The following algorithm returns *false* if the inequality fails on some entropic function h, and runs forever if the inequality holds for all h, proving that the problem is in Π^0_1:

- Iterate over all $N \geq 0$. For each N, do the following steps.

[1] $3y$ is a shorthand for $y + y + y$ and $x \geq y$ is a shorthand for $\exists u(x = y + u^2)$.

- Consider n joint random variables $X_1, \dots, X_n$ where each has outcomes in the domain $[N]$; thus there are N^n possible outcomes. Let $p_1, \dots, p_{N^n}$ be real variables representing the probabilities of these outcomes.
- Construct a formula Δ stating "there exist probabilities $p_1, \dots, p_{N^n}$ for these outcomes, whose entropy fails the conditional inequality". More precisely, the formula consists of the following:
  - Convert each conditional independence statement in the antecedent $I_h(Y_i; Z_i|X_i) = 0$ into its equivalent statement on probabilities: $p(X_iY_iZ_i)p(X_i) = p(X_iY_i)p(X_iZ_i)$.
  - Replace each such statement with a conjunction of statements of the form $p(X_i = x, Y_i = y, Z_i = z) \cdot p(X_i = x) = p(X_i = x, Y_i = y) \cdot p(X_i = x, Z_i = z)$, for all combinations of values x, y, z. If X_i, Y_i, Z_i have in total k random variables, then there are N^k combinations of values x, y, z, thus we create a conjunction of N^k equality statements.
  - Each marginal probability is a sum of atomic probabilities, for example $p(X_i = x, Y_i = y) = p_{k_1} + p_{k_2} + \cdots$ where $p_{k_1}, p_{k_2}, \dots$ are the probabilities of all outcomes that have $X_i = x$ and $Y_i = y$. Thus, the equality statement in the previous step becomes the following formula: $(p_{i_1} + p_{i_2} + \cdots)(p_{j_1} + p_{j_2} + \cdots) = (p_{k_1} + p_{k_2} + \cdots)(p_{\ell_1} + p_{\ell_2} + \cdots)$. There is one such formula for every combination of values x, y, z; denote Φ_i the conjunction of all these formulas. Thus, Φ_i asserts $I_h(Y_i; Z_i|X_i) = 0$.
  - Let $\Phi = \Phi_1 \wedge \cdots \wedge \Phi_k$. Let Ψ be the similar formula for the consequent: thus, Ψ asserts $I_h(Y; Z|X) = 0$.
  - Finally, construct the formula $\Delta \overset{\text{def}}{=} \exists p_1, \dots, \exists p_{N^n}, (\Phi \wedge \neg\Psi)$.
- Check whether $(\mathbb{R}, +, *) \vDash \Delta$. By Tarski's theorem this step is decidable.
- If Δ is true, then return *false*; otherwise, continue with $N + 1$. ◀

Tarski's exponential function problem

One may attempt to extend the proof above from the CI implication problem to arbitrary conditional inequalities (8). To check if a conditional inequality is valid for all entropic functions, we can repeat the argument above: iterate over all domain sizes $N = 1, 2, 3, \dots$, and check if there exists probabilities $p_1, \dots, p_{N^n}$ that falsify the implication $(\boldsymbol{c}_1 \cdot \boldsymbol{h} \geq 0 \wedge \cdots \wedge \boldsymbol{c}_k \cdot \boldsymbol{h} \geq 0) \Rightarrow \boldsymbol{c}_0 \cdot \boldsymbol{h} \geq 0$. The problem is that in order to express $\boldsymbol{c}_i \cdot \boldsymbol{h} \geq 0$ we need to express the vector $\boldsymbol{h}$ in terms of the probabilities $p_1, \dots, p_{N^n}$. To apply directly the definition of entropy in (1) we need to use the log function, or, alternatively, the exponential function, and this takes us outside the scope of Tarski's theorem. A major open problem in model theory, originally formulated also by Tarski, is whether decidability continues to hold if we augment the structure of the real numbers with the exponential function (see, e.g., [32] for a discussion). Decidability of the first-order theory of the reals with exponentiation would easily imply that the entropic conditional information inequality problem ECIIP (not just the entropic conditional independence (CI) implication problem) is in Π^0_1, because every condition $\boldsymbol{c} \cdot \boldsymbol{h} \geq 0$ can be expressed using $+, *$ and the exponential function, by simply expanding the definition of entropy in Equation (1).

4.2.2 The Almost-Entropic Case

Suppose the antecedent of (8) includes the condition $\boldsymbol{c} \cdot \boldsymbol{h} \geq 0$. Call $\boldsymbol{c} \in \mathbb{R}^{2^n}$ *tight* if $\boldsymbol{c} \cdot \boldsymbol{h} \leq 0$ is $\overline{\Gamma}^*_n$-valid. When $\boldsymbol{c}$ is tight, we can rewrite $\boldsymbol{c} \cdot \boldsymbol{h} \geq 0$ as $\boldsymbol{c} \cdot \boldsymbol{h} = 0$. If $\boldsymbol{c}$ is not tight, then there exists $\boldsymbol{h} \in \overline{\Gamma}^*_n$ such that $\boldsymbol{c} \cdot \boldsymbol{h} > 0$; in that case we say that $\boldsymbol{c}$ *has slack*. For example, all conditions occurring in CI implications are tight, because they are of the form $-I_h(Y; Z|X) \geq 0$, and more conveniently written $I_h(Y; Z|X) = 0$, while a condition like $3h(X) - 4h(YZ) \geq 0$ has slack. We extend the definition of slack to a set. We say that the set $\{\boldsymbol{c}_1, \dots, \boldsymbol{c}_k\} \subset \mathbb{R}^{2^n}$ has slack if there exists $\boldsymbol{h} \in \overline{\Gamma}^*_n$ such that $\boldsymbol{c}_i \cdot \boldsymbol{h} > 0$ for all $i = 1, k$; notice that this is more restricted than requiring each of $\boldsymbol{c}_i$ to have slack. We present below results on the complexity of $\mathsf{AEBIC}(F)$ in two special cases: when all antecedents are tight, and when the set of antecedents has slack. Both results use the following theorem, which allows us to move one condition $\boldsymbol{c}_k \cdot \boldsymbol{h} \geq 0$ from the antecedent to the consequent:

▶ **Theorem 9.** *The following statements are equivalent:*

$$\forall \boldsymbol{h} \in \overline{\Gamma}^*_n : \qquad \bigwedge_{i \in [k]} \boldsymbol{c}_i \cdot \boldsymbol{h} \geq 0 \Rightarrow \boldsymbol{c} \cdot \boldsymbol{h} \geq 0 \tag{14}$$

$$\forall \varepsilon > 0, \exists \lambda \geq 0, \forall \boldsymbol{h} \in \overline{\Gamma}^*_n : \qquad \bigwedge_{i \in [k-1]} \boldsymbol{c}_i \cdot \boldsymbol{h} \geq 0 \Rightarrow \boldsymbol{c} \cdot \boldsymbol{h} + \varepsilon h([n]) \geq \lambda \boldsymbol{c}_k \cdot \boldsymbol{h} \tag{15}$$

Moreover, if the set $\{\boldsymbol{c}_1, \dots, \boldsymbol{c}_k\}$ has slack, then one can set $\varepsilon = 0$ in Eq.(15)*.*

Proof. We prove here only the implication from (15) to (14); the other direction is non-trivial and is proven in the full version [2] using only the properties of closed convex cones. Assume condition (15) holds, and consider any $\boldsymbol{h} \in \overline{\Gamma}^*_n$ s.t. $\bigwedge_{i \in [k]} \boldsymbol{c}_i \cdot \boldsymbol{h} \geq 0$. We prove that $\boldsymbol{c} \cdot \boldsymbol{h} \geq 0$. For any $\varepsilon > 0$, condition (15) states that there exists $\lambda > 0$ such that $\boldsymbol{c} \cdot \boldsymbol{h} + \varepsilon h([n]) \geq \lambda \boldsymbol{c}_k \cdot \boldsymbol{h}$ and therefore $\boldsymbol{c} \cdot \boldsymbol{h} + \varepsilon h([n]) \geq 0$. Since $\varepsilon > 0$ is arbitrary, we conclude that $\boldsymbol{c} \cdot \boldsymbol{h} \geq 0$, as required. ◀

By applying the theorem repeatedly, we can move all antecedents to the consequent:

▶ **Corollary 10.** *Condition* (14) *is equivalent to:*

$$\forall \varepsilon > 0, \exists \lambda_1 \geq 0, \cdots, \exists \lambda_k \geq 0, \forall \boldsymbol{h} \in \overline{\Gamma}^*_n : \qquad \boldsymbol{c} \cdot \boldsymbol{h} + \varepsilon h([n]) \geq \sum_{i \in [k]} \lambda_i \boldsymbol{c}_i \cdot \boldsymbol{h} \tag{16}$$

Moreover, if the set $\{\boldsymbol{c}_1, \dots, \boldsymbol{c}_k\}$ has slack, then one can set $\varepsilon = 0$ in Eq.(16)*.*

Antecedents Are Tight. We consider now the case when all antecedents are tight, a condition that can be verified in Π^0_1, by Th.7. In that case, condition (14) is equivalent to:

$$\forall p \in \mathbb{N}, \exists q \in \mathbb{N}, \forall \boldsymbol{h} \in \overline{\Gamma}^*_n : \qquad \boldsymbol{c} \cdot \boldsymbol{h} + \frac{1}{p} h([n]) \geq q \sum_{i \in [k]} \boldsymbol{c}_i \cdot \boldsymbol{h} \tag{17}$$

Indeed, the non-trivial direction (16)⇒ (17) follows by setting $q \overset{\text{def}}{=} \lceil \max(\lambda_1, \dots, \lambda_k) \rceil \in \mathbb{N}$ and noting that $\boldsymbol{c}_i$ is tight, hence $\boldsymbol{c}_i \cdot \boldsymbol{h} \leq 0$ and therefore $\lambda_i \boldsymbol{c}_i \cdot \boldsymbol{h} \geq q \boldsymbol{c}_i \cdot \boldsymbol{h}$.

▶ **Corollary 11.** *Consider a conditional inequality* (8)*. If all antecedents are tight, then the corresponding decision problem* AECIIP *is in* Π^0_3

Proof. Based on our discussion, the inequality (8) is equivalent to condition (17), which is of the form $\forall p \exists q \forall \boldsymbol{h}$. Replace $\boldsymbol{h}$ with a representable entropic vector $\boldsymbol{h}^\Omega$, as in the proof of Theorem 7, and it becomes $\forall p \exists q \forall \boldsymbol{h}^\Omega$, placing it in Π^0_3. ◀

Recall that the implication problem for CI is a special case of a conditional inequality with tight antecedents. We have seen in Theorem 8 that the *entropic* version of the CI implication problem is in Π^0_1; Corollary 11 proves that the *almost entropic* version is in Π^0_3.

Consider any conditional inequality (8) where the antecedents are tight. If this inequality holds for all almost entropic functions, then it can be proven by proving a family of (unconditional) inequalities (17). In fact, some conditional inequalities in the literature have been proven precisely in this way. For example, consider the CI implication (9) (Sec. 3.2.3), and replace each antecedent $I_h(\boldsymbol{Y};\boldsymbol{Z}|\boldsymbol{X}) = 0$ with $-I_h(\boldsymbol{Y};\boldsymbol{Z}|\boldsymbol{X}) \geq 0$. By Eq. (17), the following condition holds: $\forall p \in \mathbb{N}, \exists q \in \mathbb{N}$ such that

$$q(I_h(C;D \mid A) + I_h(C;D \mid B) + I_h(A;B) + I_h(B;C \mid D)) + \frac{1}{p}h(ABCD) \geq I_h(C;D) \tag{18}$$

Thus, in order to prove (9), it suffices to prove (18). Matúš's inequality (5) provides precisely the proof of (18) (by setting $k \stackrel{\text{def}}{=} p$, $q \stackrel{\text{def}}{=} \max(\lceil \frac{k+3}{2} \rceil, 1)$, and observing that $I_h(B;D \mid C) \leq h(ABCD)$).

Antecedents Have Slack. Next, we consider the case when the antecedents have slack, which is a recursively enumerable condition. In that case, condition (16) is equivalent to:

$$\exists\lambda_1 \geq 0, \cdots, \exists\lambda_k \geq 0, \forall \boldsymbol{h} \in \overline{\Gamma}^*_n : \qquad \boldsymbol{c}\cdot\boldsymbol{h} \geq \sum_{i\in[k]} \lambda_i \boldsymbol{c}_i \cdot \boldsymbol{h} \tag{19}$$

In other words, we have proven the following result of independent interest: any conditional implication with slack is essentially unconditioned. However, we cannot immediately use (19) to prove complexity bounds for $\mathsf{AEBIC}(F)$, because the λ_i's in (19) are not necessarily rational numbers. When we derived Eq. (17) we used the fact that the antecedents are tight, hence $\boldsymbol{c}_i \cdot \boldsymbol{h} \leq 0$, hence we could replace the λ_i's with some natural number q larger than all of them. But now, the sign of $\boldsymbol{c}_i \cdot \boldsymbol{h}$ is unknown. We prove below that, under a restriction called *group balance*, the λ_i's can be chosen in $\mathbb{Q}$, placing the decision problem in Σ^0_2. Group balance generalizes Chan's notion of a *balanced inequality*, which we review below. In the full version [2] we give evidence that some restriction is necessary to ensure the λ_i's are rationals, and also show that every conditional inequality can be strengthened to be group balanced.

A vector $\boldsymbol{h} \in \mathbb{R}^{2^n}$ is called *modular* if $h(\boldsymbol{X}) + h(\boldsymbol{Y}) = h(\boldsymbol{X}\cup\boldsymbol{Y}) + h(\boldsymbol{X}\cap\boldsymbol{Y})$ forall sets of variables $\boldsymbol{X},\boldsymbol{Y} \subseteq \boldsymbol{V}$. Every non-negative modular function is entropic [46], and is a non-negative linear combination of the *basic modular functions* $\boldsymbol{h}^{(1)},\dots,\boldsymbol{h}^{(n)}$, where $h^{(j)}(\alpha) \stackrel{\text{def}}{=} 1$ when $j \in \alpha$ and is $h^{(j)}(\alpha) \stackrel{\text{def}}{=} 0$ otherwise. Chan [22] called an inequality $\boldsymbol{c}\cdot\boldsymbol{h} \geq 0$ *balanced* if $\boldsymbol{c}\cdot\boldsymbol{h}^{(j)} = 0$ for every $j \in [n]$. He proved that *any* valid inequality can be strengthened to a balanced one. More precisely: $\boldsymbol{c}\cdot\boldsymbol{h} \geq 0$ is valid iff $\boldsymbol{c}\cdot\boldsymbol{h}^{(i)} \geq 0$ for all $i \in [n]$ and $\boldsymbol{c}\cdot\boldsymbol{h} - \sum_i(\boldsymbol{c}\cdot\boldsymbol{h}^{(i)})h(X_i \mid X_{[n]-\{i\}}) \geq 0$ is valid; notice that the latter inequality is balanced. For example, $h(XY) + h(XZ) - h(X) - h(XYZ) \geq 0$ is balanced, while $h(XY) - h(X) \geq 0$ is not balanced, and can be strengthened to $h(XY) - h(X) - h(Y|X) \geq 0$. We generalize Chan's definition:

▶ **Definition 12.** *Call a set* $\{\boldsymbol{d}_1,\dots,\boldsymbol{d}_k\} \subseteq \mathbb{R}^{2^n}$ group balanced *if (a)* $\text{rank}(\boldsymbol{A}) = k-1$ *where* $\boldsymbol{A}$ *is the* $k \times n$ *matrix* $A_{ij} = \boldsymbol{d}_i \cdot \boldsymbol{h}^{(j)}$*, and (b) there exists a non-negative modular function* $\boldsymbol{h}^{(*)} \neq 0$ *such that* $\boldsymbol{d}_i \cdot \boldsymbol{h}^{(*)} = 0$ *for all* i*.*

If $k = 1$ then $\{\boldsymbol{d}_1\}$ is group balanced iff $\boldsymbol{d}_1$ is balanced, because the matrix $\boldsymbol{A}$ has a single row $\left(\boldsymbol{d}\cdot\boldsymbol{h}^{(1)} \cdots \boldsymbol{d}\cdot\boldsymbol{h}^{(n)}\right)$, and its rank is 0 iff all entries are 0. We prove in [2]:

▶ **Theorem 13.** *Consider a group balanced set of* n *vectors with rational coefficients,* $D = \{\boldsymbol{d}_1,\dots,\boldsymbol{d}_n\} \subseteq \mathbb{Q}^{2^n}$*. Suppose the following condition holds:*

$$\exists\lambda_1 \geq 0, \cdots, \exists\lambda_n \geq 0, \sum_{i\in[n]} \lambda_i = 1, \forall \boldsymbol{h} \in \overline{\Gamma}^*_n : \qquad \sum_{i\in[n]} \lambda_i \boldsymbol{d}_i \cdot \boldsymbol{h} \geq 0 \tag{20}$$

Then there exists rational $\lambda_1,\dots,\lambda_k \geq 0$ *with this property.*

This implies that, if $\boldsymbol{c}_1,\dots,\boldsymbol{c}_k$ have slack and $\{\boldsymbol{c},-\boldsymbol{c}_1,\dots,-\boldsymbol{c}_k\}$ is group balanced, then there exist rational λ_i's for inequality (19). In particular:

▶ **Corollary 14.** *Consider a conditional inequality* (8)*. If the antecedents have slack and* $\{\boldsymbol{c},-\boldsymbol{c}_1,\dots,-\boldsymbol{c}_k\}$ *is group balanced, then the corresponding decision problem is in* Σ_2^0*.*

We end this section by illustrating with an example:

▶ **Example 15.** Consider the following conditional inequality:

$$h(XYZ)+h(X)\geq 2h(XY)\wedge h(XYZ)+h(Y)\geq 2h(YZ) \quad\Rightarrow\quad 2h(XZ)\geq h(XYZ)+h(Z) \tag{21}$$

The antecedents have slack, because, by setting[2] $\boldsymbol{h}\overset{\text{def}}{=}2\boldsymbol{h}^{(X)}+\boldsymbol{h}^{(Z)}$, both antecedents become strict inequalities: $h(XYZ)+h(X)-2h(XY)=3+2-4>0$ and $h(XYZ)+h(Y)-2h(YZ)=3+0-2>0$. To check validity, we prove in Example 18 the following inequality:

$$(2h(XY)-h(XYZ)-h(X))+(2h(YZ)-h(XYZ)-h(Y))+(2h(XZ)-h(XYZ)-h(Z))\geq 0$$

and this immediately implies (21).

Consider now the following set $D=\{\boldsymbol{d}_1,\boldsymbol{d}_2,\boldsymbol{d}_3\}$, where the vectors $\boldsymbol{d}_1,\boldsymbol{d}_2,\boldsymbol{d}_3$ represent the expressions $2h(XY)-h(XYZ)-h(X)$, $2h(YZ)-h(XYZ)-h(Y)$, and $2h(XZ)-h(XYZ)-h(Z)$ respectively. We prove that D is group balanced. To check condition (a) of Def. 12 we verify that the matrix $\boldsymbol{A}$ has rank 2; in our example the matrix is $\boldsymbol{A}=\begin{pmatrix}0&1&-1\\-1&0&1\\-1&1&0\end{pmatrix}$ and its rank is 2 as required. To check condition (b), we define $\boldsymbol{h}^{(*)}=h^{(X)}+h^{(Y)}+h^{(Z)}$ and verify that $\boldsymbol{d}_1\cdot\boldsymbol{h}^{(*)}=\boldsymbol{d}_2\cdot\boldsymbol{h}^{(*)}=\boldsymbol{d}_3\cdot\boldsymbol{h}^{(*)}=4-3-1=0$. Thus, D is group balanced.

4.3 Discussion on the Decidability of MaxIIP

A proof of the decidability of MaxIIP would immediately imply that the domination problem $\boldsymbol{A}\leq\boldsymbol{B}$ for acyclic structures $\boldsymbol{B}$ is also decidable [1]. It is currently open whether MaxIIP is decidable, or even if the special case IIP is decidable. But what can we say about the domination problem if IIP were decidable? Theorem 7 only says that both problems are in Π_1^0, and does not tell us anything about MaxIIP if IIP were decidable. We prove here that, the decidability of IIP implies the decidability of group-balanced MaxIIP. We start with a result of general interest, which holds even for conditional Max-Information constraints.

▶ **Theorem 16.** *The following two statements are equivalent:*

$$\forall\boldsymbol{h}\in\overline{\Gamma}_n^*: \qquad \bigwedge_{i\in[k]}\boldsymbol{c}_i\cdot\boldsymbol{h}\geq 0\Rightarrow\bigvee_{j\in[m]}\boldsymbol{d}_j\cdot\boldsymbol{h}\geq 0 \tag{22}$$

$$\exists\lambda_1,\dots,\lambda_m\geq 0,\sum_j\lambda_j=1,\forall\boldsymbol{h}\in\overline{\Gamma}_n^*: \qquad \bigwedge_{i\in[k]}\boldsymbol{c}_i\cdot\boldsymbol{h}\geq 0\Rightarrow\sum_{j\in[m]}\lambda_j\boldsymbol{d}_j\cdot\boldsymbol{h}\geq 0 \tag{23}$$

The theorem says that every max-inequality is essentially a linear inequality. The proof of (23) ⇒ (22) is immediate; we prove the reverse in [2]. As before, we don't know whether these coefficients λ_i can be chosen to be rational numbers in general, but by Theorem 13 this is the case when $\{\boldsymbol{c}_1,\dots,\boldsymbol{c}_k\}$ is group-balanced, and this implies:

▶ **Corollary 17.** *The MaxIIP problem where the inequalities* $\boldsymbol{c}_1,\dots,\boldsymbol{c}_n$ *are group balanced is Turing equivalent to the IIP problem.*

[2] Where $\boldsymbol{h}^{(X)}$ denotes the basic modular function at X, i.e. $h^{(X)}(X)=1$, $h^{(X)}(Y)=h^{(X)}(Z)=0$.

Proof. We describe a Turing reduction from MaxIIP to IIP. Consider a MaxIIP problem, $\bigvee_{j\in[m]}(\boldsymbol{c}_j \cdot \boldsymbol{h} \geq 0)$. We run two computations in parallel. The first computation iterates over all representable spaces Ω, and checks whether $\bigwedge_j(\boldsymbol{c}_j \cdot \boldsymbol{h}^\Omega < 0)$; if we find such a space then we stop and we return *false*. If the inequality is invalid then this computation will eventually terminate because in that case there exists a representable counterexample Ω. The second computation iterates over all m-tuples of natural numbers $(\lambda_1,\dots,\lambda_m) \in \mathbb{N}^m$ and checks $\forall \boldsymbol{h} \in \Gamma_n^*, \sum_j \lambda_j \boldsymbol{c}_j \cdot \boldsymbol{h} \geq 0$ by using the oracle for IIP: if it finds such λ_j's, then it stops and returns *true*. If the inequality is valid then this computation will eventually terminate, by Theorems 16 and 13. ◀

We illustrate with an example.

▶ **Example 18.** Consider Kopparty and Rossman's inequality (7), which can be stated as $\max(\boldsymbol{c}_1,\boldsymbol{c}_2,\boldsymbol{c}_3) \geq 0$, where $\boldsymbol{c}_1,\boldsymbol{c}_2,\boldsymbol{c}_3$ define the three expressions in (7). To prove that it is valid, it suffices to prove that their sum is ≥ 0; we show this briefly here[3]:

$$\begin{aligned}
&(2h(XY)-h(X))+(2h(YZ)-h(Y))+(2h(XZ)-h(Z))-3h(XYZ)\\
&=(h(XY)+h(YZ)+h(XZ))+(h(XY)-h(X))+(h(YZ)-h(Y))+(h(XZ)-h(Z))\\
&\quad -3h(XYZ)\\
&\geq (h(XY)+h(YZ)+h(XZ))+(h(XYZ)-h(XZ))+(h(XYZ)-h(XY))\\
&\qquad +(h(XYZ)-h(YZ))-3h(XYZ)=0
\end{aligned}$$

Theorem 16 proves that *any* max-inequality necessarily follows from such a linear inequality; we just have to find the right λ_i's. In this example, the set $\boldsymbol{c}_1,\boldsymbol{c}_2,\boldsymbol{c}_3$ is group balanced (as we showed in Example 15), therefore there exists rational λ_i's; indeed, our choice here is $\lambda_1=\lambda_2=\lambda_3=1$.

5 The Recognizability Problems

We study here two problems that are the dual of the Boolean information constraint problem. The *entropic-recognizability problem* takes as input a vector $\boldsymbol{h}$ and checks if $\boldsymbol{h} \in \Gamma_n^*$. The *almost-entropic-recognizability problem* checks if $\boldsymbol{h} \in \overline{\Gamma}_n^*$. We will prove that the latter is in Π_2^0, and leave open the complexity of the former.

Before we define these problems formally, we must first address the question of how to represent the input $\boldsymbol{h}$. One possibility is to represent $\boldsymbol{h}$ as a vector of rational numbers, but this is unsatisfactory, because usually entropies are not rational numbers. Instead, we will allow a more general representation. To justify it, assume first that $\boldsymbol{h}$ were given by some representable space Ω (Sec. 4.1), where all probabilities are rational numbers. In that case, every term $p_i \log p_i$ in the definition of the entropy can be written as $\log(p_i^{p_i})$, hence the quantity $h(\boldsymbol{X})$ has the form $h(\boldsymbol{X}) = \log\prod_i p_i^{p_i}$. In general, any product $\prod_i m_i^{n_i}$ where $m_i, n_i \in Q$, for $i = 1, n$, can be rewritten as $\left(\frac{a}{b}\right)^{\frac{1}{c}}$, where $a,b,c \in \mathbb{N}$. Indeed, writing $m_i = u_i/v_i$ and $n_i = s_i/t_i$ where $u_i, v_i, s_i, t_i \in \mathbb{N}$, we have:

$$\prod_i\left(\frac{u_i}{v_i}\right)^{\frac{s_i}{t_i}} = \prod_i\left(\frac{u_i^{s_i}}{v_i^{s_i}}\right)^{\frac{1}{t_i}} = \left(\prod_i \frac{u_i^{s_i\cdot\prod_{j\neq i}t_j}}{v_i^{s_i\cdot\prod_{j\neq i}t_j}}\right)^{\frac{1}{\prod_i t_i}} = \left(\frac{a}{b}\right)^{\frac{1}{c}} \qquad a,b,c\in\mathbb{N}$$

[3] We apply submodularity: $h(XY) - h(X) \geq h(XYZ) - h(XZ)$ etc.

Justified by this observation, we assume that the input to our problem consists of three vectors $(a_{\boldsymbol{X}})_{\boldsymbol{X}\subseteq\boldsymbol{V}}$, $(b_{\boldsymbol{X}})_{\boldsymbol{X}\subseteq\boldsymbol{V}}$, and $(c_{\boldsymbol{X}})_{\boldsymbol{X}\subseteq\boldsymbol{V}}$ in $\mathbb{N}^{2^n}$, with the convention that $h(\boldsymbol{X}) \overset{\text{def}}{=} \frac{1}{c_{\boldsymbol{X}}} \log \frac{a_{\boldsymbol{X}}}{b_{\boldsymbol{X}}}$. Thus, we do not assume that these vectors come from a representable space Ω, we only assume their entropies can be represented in this form.

▶ **Definition 19** ((Almost-)Entropic Recognizability Problem)**.** *Given natural numbers $(a_{\boldsymbol{X}})_{\boldsymbol{X}\subseteq\boldsymbol{V}}$, $(b_{\boldsymbol{X}})_{\boldsymbol{X}\subseteq\boldsymbol{V}}$ and $(c_{\boldsymbol{X}})_{\boldsymbol{X}\subseteq\boldsymbol{V}}$, check whether the vector $h(\boldsymbol{X}) \overset{\text{def}}{=} \frac{1}{c_{\boldsymbol{X}}} \log \frac{a_{\boldsymbol{X}}}{b_{\boldsymbol{X}}}$, $\boldsymbol{X} \subseteq \boldsymbol{V}$, represents an entropic vector, or an almost-entropic vector.*

Our result in this section is (see [2] for a proof):

▶ **Theorem 20.** *The almost entropic recognizability problem is in Π_2^0.*

We end with a brief comment on the complexity of the entropic-recognizability problem: given $\boldsymbol{h}$ (represented as in Def. 19) check if $\boldsymbol{h} \in \Gamma_n^*$. Consider the following restricted form of the problem: check if $\boldsymbol{h}$ is the entropic vector of a representable space Ω (i.e. finite space with rational probabilities). This problem is in Σ_1^0, because one can iterate over all representable spaces Ω and check that their entropies are those required. However, in the general setting we ask whether *any* finite probability space has these entropies, not necessarily one with rational probabilities. This problem would remain in Σ_1^0 if the theory of reals with exponentiation were decidable. Recall that Tarski's theorem states that the theory of reals $\text{FO}(\mathbb{R}, 0, 1, +, *)$ is decidable. A major open problem in model theory is whether the theory remains decidable if we add exponentiation. If that were decidable, then the entropic-recognizability problem would be in Σ_1^0. To see this, consider the following semi-decision problem. Iterate over $N = 1, 2, 3, \ldots$ and for each N check if there exists a probability space whose active domain has size N (thus, there are N^n outcomes, where $n = |\boldsymbol{V}|$ is the number of variables) and whose entropies are precisely those given. This statement that can be expressed using the exponential function (which we need in order to express the entropy as $\sum_i p_i \log p_i$). If there exists any finite probability space with the required entropies, then this procedure will find it; otherwise it will run forever, placing the problem in Σ_1^0.

6 Discussion

CI Implication Problem. The implication problem for Conditional Independence statements has been extensively studied in the literature, but its complexity remains an open problem. It is not even known whether this problem is decidable [18, 37, 38]. Our Theorem 8 appears to be the first upper bound on the complexity of the CI implication problem, placing it in Π_1^0. Hannula et al. [24] prove that, if all random variables are restricted to be binary random variables, then the CI implication problem is in EXPSPACE; the implication problem for binary random variables differs from that for general discrete random variables; see the discussion in [18].

Finite, infinite, continuous random variables. In this paper, all random variables have a finite domain. There are two alternative choices: discrete random variables (possibly infinite), and continuous random variables. The literature on entropic functions has mostly alternated between defining entropic functions over finite random variables, or over discrete infinite random variables with finite entropy. For example discrete (possibly infinite) random variables are considered by Zhang and Yeung, [50], by Chan and Yeung [12], and by Chan [22], while random variables with finite domains are considered by Matúš [33, 34] and by Kaced and Romashchenko [25]. The reason for this inconsistency is that for information inequalities the distinction doesn't matter: every entropy of a set of discrete random variables can be approximated arbitrarily well by the entropy of a set of random variables with finite domain, and Prop. 6 extends immediately to discrete random variables[4]. However, the distinction is significant for conditional inequalities, and here the choice in the literature is always for finite domains. For example, the implication problem for conditional independence, i.e. the graphoid axioms, is stated for finite probability spaces by Geiger and Pearl [18], while Kaced and Romashchenko [25] also use finite distributions to prove the existence of conditional inequalities that hold over entropic but fail for almost-entropic functions. One could also consider continuous distributions, whose entropy is $\int p(x)\log(1/p(x))dx$, where p is the probability density function. Chan [22] showed that an information inequality holds for all continuous distributions iff it is balanced and it holds for all discrete distributions. For example, $h(X) \geq 0$ is not balanced, hence it fails in the continuous, because the entropy of the uniform distribution in the interval $[0, c]$ is $\log c$, which is < 0 when $c < 1$.

Strict vs. non-strict inequalities. The literature on information inequalities always defines inequalities using ≥ 0, in which case validity for entropic functions is the same as validity for almost entropic functions. One may wonder what happens if one examines strict inequalities $\boldsymbol{c} \cdot \boldsymbol{h} > 0$ instead. Obviously, each such inequality fails on the zero-entropic vector, but we can consider the conditional version $\boldsymbol{h} \neq 0 \Rightarrow \boldsymbol{c} \cdot \boldsymbol{h} > 0$, which we can write formally as $\boldsymbol{c} \cdot \boldsymbol{h} \leq 0 \Rightarrow h(\boldsymbol{V}) \leq 0$. This a special case of a conditional inequality as discussed in this paper. An interesting question is whether for this special case Γ_n^*-validity and $\overline{\Gamma}_n^*$-validity coincide; a negative answer would represent a significant extension of Kaced and Romashchenko's result [25].

References

1 Mahmoud Abo Khamis, Phokion G. Kolaitis, Hung Q. Ngo, and Dan Suciu. Bag query containment and information theory. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2020, Portland, CA, USA*, 2020. to appear.

2 Mahmoud Abo Khamis, Phokion G. Kolaitis, Hung Q. Ngo, and Dan Suciu. Decision problems in information theory. *CoRR*, abs/2004.08783, 2020. arXiv:2004.08783.

3 Mahmoud Abo Khamis, Hung Q. Ngo, and Dan Suciu. Computing join queries with functional dependencies. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, pages 327–342, 2016.

4 Mahmoud Abo Khamis, Hung Q. Ngo, and Dan Suciu. What do Shannon-type inequalities, submodular width, and disjunctive Datalog have to do with one another? In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2017*, pages 429–444, 2017.

5 Noga Alon. On the number of subgraphs of prescribed type of graphs with a given number of edges. *Israel J. Math.*, 38(1-2):116–130, 1981. doi:10.1007/BF02761855.

6 Albert Atserias, Martin Grohe, and Dániel Marx. Size bounds and query plans for relational joins. *SIAM J. Comput.*, 42(4):1737–1767, 2013. doi:10.1137/110859440.

[4] The idea of the proof relies on the fact that every entropy is required to converge, i.e. $h(X_\alpha) = \sum_i p_i \log 1/p_i$, hence there exists a finite subspace of outcomes $\{1, 2, \ldots, N\}$ for which the sum is ε-close to $h(X_\alpha)$. The union of these spaces over all $\alpha \subseteq [n]$ suffices to approximate h well enough.

7 Amos Beimel. Secret-sharing schemes: A survey. In Yeow Meng Chee, Zhenbo Guo, San Ling, Fengjing Shao, Yuansheng Tang, Huaxiong Wang, and Chaoping Xing, editors, *Coding and Cryptology - Third International Workshop, IWCC 2011, Qingdao, China, May 30-June 3, 2011. Proceedings*, volume 6639 of *Lecture Notes in Computer Science*, pages 11–46. Springer, 2011. doi:10.1007/978-3-642-20901-7_2.

8 G. R. Blakley. Safeguarding cryptographic keys. In *Managing Requirements Knowledge, International Workshop on*, page 313, Los Alamitos, CA, USA, June 1979. IEEE Computer Society. doi:10.1109/AFIPS.1979.98.

9 Carlo Blundo, Alfredo De Santis, and Ugo Vaccaro. On secret sharing schemes. *Inf. Process. Lett.*, 65(1):25–32, 1998. doi:10.1016/S0020-0190(97)00194-4.

10 Renato M. Capocelli, Alfredo De Santis, Luisa Gargano, and Ugo Vaccaro. On the size of shares for secret sharing schemes. *J. Cryptology*, 6(3):157–167, 1993. doi:10.1007/BF00198463.

11 Terence H. Chan. Group characterizable entropy functions. In *IEEE International Symposium on Information Theory, ISIT 2007, Nice, France, June 24-29, 2007*, pages 506–510. IEEE, 2007.

12 Terence H. Chan and Raymond W. Yeung. On a relation between information inequalities and group theory. *IEEE Transactions on Information Theory*, 48(7):1992–1995, 2002.

13 Surajit Chaudhuri and Moshe Y. Vardi. Optimization of *Real* conjunctive queries. In Catriel Beeri, editor, *Proceedings of the Twelfth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 25-28, 1993, Washington, DC, USA*, pages 59–70. ACM Press, 1993. URL: http://dl.acm.org/citation.cfm?id=153850.

14 F. R. K. Chung, R. L. Graham, P. Frankl, and J. B. Shearer. Some intersection theorems for ordered sets and graphs. *J. Combin. Theory Ser. A*, 43(1):23–37, 1986. doi:10.1016/0097-3165(86)90019-1.

15 László Csirmaz. The size of a share must be large. *J. Cryptology*, 10(4):223–231, 1997. doi:10.1007/s001459900029.

16 Peter R. de Waal and Linda C. van der Gaag. Stable independance and complexity of representation. In David Maxwell Chickering and Joseph Y. Halpern, editors, *UAI '04, Proceedings of the 20th Conference in Uncertainty in Artificial Intelligence, Banff, Canada, July 7-11, 2004*, pages 112–119. AUAI Press, 2004. URL: https://dslpitt.org/uai/displayArticleDetails.jsp?mmnu=1&smnu=2&article_id=1165&proceeding_id=20.

17 Ehud Friedgut and Jeff Kahn. On the number of copies of one hypergraph in another. *Israel J. Math.*, 105:251–256, 1998. doi:10.1007/BF02780332.

18 Dan Geiger and Judea Pearl. Logical and algorithmic properties of conditional independence and graphical models. *The Annals of Statistics*, 21(4):2001–2021, 1993.

19 Georg Gottlob, Stephanie Tien Lee, Gregory Valiant, and Paul Valiant. Size and treewidth bounds for conjunctive queries. *J. ACM*, 59(3):16:1–16:35, 2012. doi:10.1145/2220357.2220363.

20 Peter Grunwald and Paul Vitányi. Shannon information and Kolmogorov complexity. *arXiv preprint*, 2004. arXiv:cs/0410002.

21 Marc Gyssens, Mathias Niepert, and Dirk Van Gucht. On the completeness of the semigraphoid axioms for deriving arbitrary from saturated conditional independence statements. *Inf. Process. Lett.*, 114(11):628–633, 2014. doi:10.1016/j.ipl.2014.05.010.

22 Terence H. Chan. Balanced information inequalities. *Information Theory, IEEE Transactions on*, 49:3261–3267, January 2004. doi:10.1109/TIT.2003.820037.

23 Daniel Hammer, Andrei Romashchenko, Alexander Shen, and Nikolai Vereshchagin. Inequalities for Shannon entropy and Kolmogorov complexity. *Journal of Computer and System Sciences*, 60(2):442–464, 2000.

24 Miika Hannula, Åsa Hirvonen, Juha Kontinen, Vadim Kulikov, and Jonni Virtema. Facets of distribution identities in probabilistic team semantics. *CoRR*, abs/1812.05873, 2018. arXiv:1812.05873.

25 Tarik Kaced and Andrei E. Romashchenko. Conditional information inequalities for entropic and almost entropic points. *IEEE Trans. Information Theory*, 59(11):7149–7167, 2013. doi:10.1109/TIT.2013.2274614.

26 Ehud D. Karnin, J. W. Greene, and Martin E. Hellman. On secret sharing systems. *IEEE Trans. Information Theory*, 29(1):35–41, 1983. doi:10.1109/TIT.1983.1056621.

27 Batya Kenig and Dan Suciu. Integrity constraints revisited: From exact to approximate implication. *CoRR*, abs/1812.09987, 2018. arXiv:1812.09987.

28 Phokion G. Kolaitis and Moshe Y. Vardi. Conjunctive-query containment and constraint satisfaction. In *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 1-3, 1998, Seattle, Washington, USA*, pages 205–213, 1998. doi:10.1145/275487.275511.

29 Swastik Kopparty and Benjamin Rossman. The homomorphism domination exponent. *European Journal of Combinatorics*, 32(7):1097–1114, 2011. Homomorphisms and Limits.

30 Tony T. Lee. An information-theoretic analysis of relational databases - part I: data dependencies and information metric. *IEEE Trans. Software Eng.*, 13(10):1049–1061, 1987. doi:10.1109/TSE.1987.232847.

31 Konstantin Makarychev, Yury Makarychev, Andrei Romashchenko, and Nikolai Vereshchagin. A new class of non-Shannon-type inequalities for entropies. *Commun. Inf. Syst.*, 2(2):147–165, 2002. doi:10.4310/CIS.2002.v2.n2.a3.

32 David Marker. Model theory and exponentiation, 1996.

33 Frantisek Matúš. Probabilistic conditional independence structures and matroid theory: Background, 1994.

34 Frantisek Matúš. Infinitely many information inequalities. In *IEEE International Symposium on Information Theory, ISIT 2007, Nice, France, June 24-29, 2007*, pages 41–44, 2007. doi:10.1109/ISIT.2007.4557201.

35 C.F. Miller. Decision problems for groups — survey and reflections. In G. Baumslag and C.F. Miller, editors, *Algorithms and Classification in Combinatorial Group Theory. Mathematical Sciences Research Institute Publications*, volume 23. Springer, NY, 1992.

36 Hung Q. Ngo. Worst-case optimal join algorithms: Techniques, results, and open problems. In Jan Van den Bussche and Marcelo Arenas, editors, *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, Houston, TX, USA, June 10-15, 2018*, pages 111–124. ACM, 2018. doi:10.1145/3196959.3196990.

37 Mathias Niepert, Dirk Van Gucht, and Marc Gyssens. Logical and algorithmic properties of stable conditional independence. *Int. J. Approx. Reason.*, 51(5):531–543, 2010. doi:10.1016/j.ijar.2010.01.011.

38 Mathias Niepert, Marc Gyssens, Bassem Sayrafi, and Dirk Van Gucht. On the conditional independence implication problem: A lattice-theoretic approach. *Artif. Intell.*, 202:29–51, 2013. doi:10.1016/j.artint.2013.06.005.

39 Judea Pearl and Azaria Paz. Graphoids: Graph-based logic for reasoning about relevance relations or when would x tell you more about y if you already know z? In *ECAI*, pages 357–363, 1986.

40 Nicholas Pippenger. What are the laws of information theory. In *1986 Special Problems on Communication and Computation Conference*, pages 3–5, 1986.

41 Hartley Rogers and H Rogers. *Theory of recursive functions and effective computability*, volume 5. McGraw-Hill New York, 1967.

42 Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979. doi:10.1145/359168.359176.

43 Larry J Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1–22, 1976.

44 Milan Studený. Conditional independence relations have no finite complete characterization. In *11th Prague Conf. Information Theory, Statistical Decision Foundation and Random Processes*, pages 377–396. Norwell, MA, 1990.

45 Alfred Tarski. A decision method for elementary algebra and geometry. In *Quantifier elimination and cylindrical algebraic decomposition*, pages 24–84. Springer, 1998.

46 Raymond W. Yeung. *A first course in information theory*. Information Technology: Transmission, Processing and Storage. Kluwer Academic/Plenum Publishers, New York, 2002. With a foreword by Toby Berger, With 1 CD-ROM. `doi:10.1007/978-1-4419-8608-5`.

47 Raymond W. Yeung. *Information Theory and Network Coding*. Springer Publishing Company, Incorporated, 1 edition, 2008.

48 Raymond W. Yeung and Zhen Zhang. A class of non-Shannon-type information inequalities and their applications. *Commun. Inf. Syst.*, 1(1):87–100, 2001. `doi:10.4310/CIS.2001.v1.n1.a6`.

49 Zhen Zhang. On a new non-Shannon type information inequality. *Commun. Inf. Syst.*, 3(1):47–60, 2003. `doi:10.4310/CIS.2003.v3.n1.a4`.

50 Zhen Zhang and Raymond W. Yeung. A non-Shannon-type conditional inequality of information quantities. *IEEE Trans. Information Theory*, 43(6):1982–1986, 1997.

51 Zhen Zhang and Raymond W Yeung. On characterization of entropy function via information inequalities. *IEEE Transactions on Information Theory*, 44(4):1440–1452, 1998.

Invariants for Continuous Linear Dynamical Systems

Shaull Almagor
Department of Computer Science, Technion, Haifa, Israel
shaull@cs.technion.ac.il

Edon Kelmendi
Department of Computer Science, Oxford University, UK
edon.kelmendi@cs.ox.ac.uk

Joël Ouaknine
Max Planck Institute for Software Systems, Saarland Informatics Campus, Saarbrücken, Germany
Department of Computer Science, Oxford University, UK
joel@mpi-sws.org

James Worrell
Department of Computer Science, Oxford University, UK
jbw@cs.ox.ac.uk

Abstract

Continuous linear dynamical systems are used extensively in mathematics, computer science, physics, and engineering to model the evolution of a system over time. A central technique for certifying safety properties of such systems is by synthesising inductive invariants. This is the task of finding a set of states that is closed under the dynamics of the system and is disjoint from a given set of error states. In this paper we study the problem of synthesising inductive invariants that are definable in o-minimal expansions of the ordered field of real numbers. In particular, assuming Schanuel's conjecture in transcendental number theory, we establish effective synthesis of o-minimal invariants in the case of semi-algebraic error sets. Without using Schanuel's conjecture, we give a procedure for synthesizing o-minimal invariants that contain all but a bounded initial segment of the orbit and are disjoint from a given semi-algebraic error set. We further prove that effective synthesis of semi-algebraic invariants that contain the whole orbit, is at least as hard as a certain open problem in transcendental number theory.

2012 ACM Subject Classification Theory of computation → Logic and verification; Computing methodologies → Algebraic algorithms; Mathematics of computing → Continuous mathematics; Mathematics of computing → Continuous functions; Theory of computation → Finite Model Theory; Software and its engineering → Formal software verification

Keywords and phrases Invariants, continuous linear dynamical systems, continuous Skolem problem, safety, o-minimality

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.107

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

Related Version A full version of the paper is available at [3] `https://arxiv.org/abs/2004.11661`.

Funding *Shaull Almagor*: Supported by a European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 837327.
Joël Ouaknine: Supported by ERC grant AVS-ISS (648701) and DFG grant 389792660 as part of TRR 248 (see https://perspicuous-computing.science).
James Worrell: Supported by EPSRC Fellowship EP/N008197/1.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 107; pp. 107:1–107:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

A *continuous linear dynamical system* (CDS) is a system whose evolution is governed by a differential equation of the form $\dot{\mathbf{x}}(t) = A\mathbf{x}(t)$, where A is a matrix with real entries. CDSs are ubiquitous in mathematics, physics, and engineering; they have been extensively studied as they describe the evolution of many types of systems (or abstractions thereof) over time. More recently, CDSs have become central in the study of cyber-physical systems (see, e.g., the textbook [4]).

In the study of CDSs, particularly from the perspective of control theory, a fundamental problem is *reachability* – namely whether the orbit $\{\mathbf{x}(t) \ : \ t \geq 0\}$ intersects a given target set $Y \subseteq \mathbb{R}^d$. For example, when $\mathbf{x}(t)$ describes the state of an autonomous car (i.e., its location, velocity, etc.). Y may describe situations where the car is not able to stop in time to respond to a hazard.

When Y is a singleton set, reachability is decidable [14, Theorem 2]. However, already when Y is a half-space it is open whether or not reachability is decidable. The latter decision problem is known in the literature as the *continuous Skolem problem.* Some partial positive results were given in [6] and [10]. The continuous Skolem problem is related to notoriously difficult problems in the theory of Diophantine approximation: specifically a procedure for the continuous Skolem problem would yield one for computing to arbitrary precision the *Diophantine-approximation types* of all real algebraic numbers [10].

In lieu of an algorithm to decide reachability, a popular approach to certifying non-reachability is by finding an *inductive invariant*, that is, a set X that is closed under the dynamics of the system and disjoint from the target Y. For such a set, it is immediate that the orbit of any point in X does not reach Y. In order to make this approach effective it is natural to seek invariants among suitably tame classes of sets (e.g., polyhedra, semi-algebraic sets, *etc*). In this work we take a very general approach – we consider *o-minimal invariants*, that is, invariants definable in *some* o-minimal expansion of the ordered set of real numbers. The class of o-minimal invariants includes those defined by Boolean combinations of inequalities involving polynomials, the real exponential function, and bounded versions of the trigonometric functions. One can potentially use many other functions to define o-minimal invariants, but the key to our approach is to show that the ingredients just listed suffice in the case of a CDS and semi-algebraic target Y.

The papers [2, 1] study o-minimal invariants for *discrete* linear dynamical systems. There it is proved that when the target Y is a semi-algebraic set, the question of whether there exists an o-minimal invariant disjoint from Y is decidable. Furthermore, if there is an o-minimal invariant then there is in fact a semi-algebraic invariant which can moreover be constructed effectively. The present paper uses similar ideas, although the case of continuous linear dynamical systems differs in several important ways.

Main Contributions. We consider the following problem: given a CDS by means of a matrix A with rational entries, an initial point $\mathbf{x}_0 = \mathbf{x}(0)$, and a semi-algebraic set Y of error states, decide whether there exists a set that is definable in some o-minimal expansion of the ordered real field and is (1) disjoint from Y, (2) invariant under the dynamics of the system, and (3) contains the initial point $\mathbf{x}_0$. We show that in searching for such invariants it suffices to look among sets definable in the expansion of the reals with the real exponential function and trigonometric functions restricted to bounded domains. Moroever, assuming Schanuel's conjecture (a unifying conjecture in transcendental number theory), we prove that the existence of such an invariant is decidable, and that invariants can effectively be constructed when they exist.

Without assuming Schanuel's conjecture we can decide a related problem, namely the question of whether there exists a set that is definable in an o-minimal expansion of the real field and is (1) disjoint from Y, (2) invariant under the dynamics of the system, and (3) meets the orbit of the initial point $\mathbf{x}_0$. Notice that such a set – which could be called an *eventual invariant* – must contain all but a bounded initial segment of the orbit. We show that when such a set exists, it can be effectively constructed and moreover that it can be chosen to be a semi-algebraic set. Such an invariant can serve as a certificate that the orbit does not enter the error set Y infinitely often. The latter is a very difficult problem to decide, even when the target set is a half-space [9].

As mentioned earlier, for discrete linear dynamical systems the question of whether there exists a semi-algebraic invariant that contains the *whole* orbit is decidable [2, 1]. We provide an explanation of why the analogous result for continuous systems is not easy to prove; this is by way of a reduction from a difficult problem that highlights the complications of continuous systems. The problem asks whether a given exponential polynomial of the form

$$f(t) = a_1 e^{b_1 t} + \cdots + a_n e^{b_n t}$$

has zeros in a bounded interval, where a_i, b_i are real algebraic numbers. Deciding whether f has zeros in a bounded region seems to be difficult because all the zeros have to be transcendental (a consequence of Hermite-Lindemann Theorem), and they can be tangential, i.e., f never changes its sign, yet it has a zero.

Related Work. Invariant synthesis is a central technique for establishing safety properties of hybrid systems. It has long been known how to compute a strongest *algebraic* invariant [20] (i.e., a smallest algebraic set that contains the collection of reachable states) for an arbitrary CDS. Here an algebraic invariant is one that is specified by a conjunction of polynomial equalities. If one moves to the more expressive setting of semi-algebraic invariants, which allow inequalities, then there is typically no longer a strongest (or smallest) invariant, but one can still ask to decide the existence of an invariant that avoids a given target set of configurations. This is the problem that is addressed in the present paper.

Partial positive results are known, for example when strong restrictions on the matrix A are imposed, such as when all the eigenvalues are real and rational, or purely imaginary with rational imaginary part [15].

A popular approach in previous work has been to seek invariants that match a given syntactic *template*, which allows to reduce invariant synthesis to constraint solving [13, 23, 16]. While this technique can be applied to much richer classes of systems than those considered here (e.g., with discrete control modes and non-linear differential equations), it does not appear to offer a way to decide the existence of arbitrary semi-algebraic invariants. An alternative to the template approach for invariant generation involves obtaining candidate invariants from semi-algebraic abstractions of a system [21]. Another active area of current research lies in developing powerful techniques to check whether a given semi-algebraic set is actually an invariant [12, 16].

Other avenues for analysing dynamical systems in the literature include bisimulations [7], forward/backward reach-set computation [5], and methods for directly proving liveness properties [22]. The latter depends on constructing *staging sets*, which are essentially semi-algebraic invariants.

Often, questions about dynamical systems can be reduced to deciding whether a sentence belongs to the elementary theory of an appropriate expansion of the ordered field of real numbers. While the latter is typically undecidable, there are partial positive results, namely quasi-decidability in bounded domains, see [11] and the references therein. This can be used to reason about the dynamics of a system in a bounded time interval, under the assumption that it does not tangentially approach the set that we want to avoid. However, it seems unlikely that such results can be easily applied to the problems considered here.

The rest of the paper is organised as follows. In Section 2, we give the necessary definitions and terminology. In Section 3, we define *cones*, which are over-approximations of the orbit, and prove that they are in a certain sense canonical. The positive results assuming Schanuel's conjecture are subsequently given in this section. Section 4 is devoted to the effective construction of the semi-algebraic invariants which allows us to state and prove the unconditional positive results. In Section 5, we give the aforementioned reduction, from finding zeros of exponential polynomials.

See [3] for the complete proofs.

2 Preliminaries

A *continuous-time linear dynamical system* is a pair

$$\langle A, \mathbf{x}_0 \rangle$$

where $A \in \mathbb{Q}^{d \times d}$ and $\mathbf{x}_0 \in \mathbb{Q}^d$. The system evolves in time according the function $x(t)$ which is the unique solution to the differential equation $\dot{\mathbf{x}}(t) = A\mathbf{x}(t)$ with $\mathbf{x}(0) = \mathbf{x}_0$. Explicitly this solution can be written as:

$$\mathbf{x}(t) = e^{At}\mathbf{x}_0.$$

The *orbit of* $\langle A, \mathbf{x}_0 \rangle$ *from time* t_0 is the set $\mathcal{O}(t_0) = \{e^{At}\mathbf{x}_0 \ : \ t \geq t_0\}$. An *invariant for* $\langle A, \mathbf{x}_0 \rangle$ *from time* t_0 is a set $\mathcal{I} \subseteq \mathbb{R}^d$ that contains $e^{At_0}\mathbf{x}_0$ and is stable under applications of e^{At}, i.e., $e^{At}\mathcal{I} \subseteq \mathcal{I}$ for every $t \geq 0$. Note that an invariant from time t_0 contains $\mathcal{O}(t_0)$. Given a set $Y \subseteq \mathbb{R}^d$ (referred to henceforth as an *error set*), we say that the invariant $\mathcal{I}$ *avoids* Y if the two sets are disjoint.

We denote by $\mathfrak{R}_0$ the structure $\langle \mathbb{R}, 0, 1, +, \cdot, < \rangle$. This is the ordered field of real numbers with constants 0 and 1. A sentence in the corresponding first-order language is a quantified Boolean combination of atomic propositions of the form $P(x_1, \dots, x_n) > 0$, where P is a polynomial with integer coefficients and $x_1, \dots, x_n$ are variables. In addition to $\mathfrak{R}_0$, we also consider its following expansions:

- $\mathfrak{R}_{\exp}$, obtained by expanding $\mathfrak{R}_0$ with the real exponentiation function $x \mapsto e^x$.
- $\mathfrak{R}^{\mathrm{RE}}$, obtained by expanding $\mathfrak{R}_0$ with the *restricted elementary functions*, namely $x \mapsto e^x|_{[0,1]}$, $x \mapsto \sin x|_{[0,1]}$, and $x \mapsto \cos x|_{[0,1]}$.
- $\mathfrak{R}^{\mathrm{RE}}_{\exp}$, obtained by expanding $\mathfrak{R}_{\exp}$ with the restricted elementary functions.

Tarski famously showed that the first-order theory of $\mathfrak{R}_0$ admits quantifier elimination, moreover the elimination is effective and therefore the theory is decidable [24, Theorem 37].

It is an open question whether the theory of the reals with exponentiation ($\mathfrak{R}_{\exp}$) is decidable; however decidability was established subject to Schanuel's conjecture by MacIntyre and Wilkie [18, Theorem 1.1]. MacIntyre and Wilkie further showed in [18, Section 5] that decidability of the theory of $\mathfrak{R}_{\exp}$ implies a weak form of Schanuel's conjecture.

Similarly, it is an open question whether $\mathfrak{R}^{\mathrm{RE}}$ and $\mathfrak{R}^{\mathrm{RE}}_{\exp}$ are decidable, but they are also known to be decidable subject to Schanuel's conjecture [17, Theorem 3.1][1].

[1] More precisely, the decidability of $\mathfrak{R}_{\exp}$ requires Schanuel's conjecture over $\mathbb{R}$, whereas that of $\mathfrak{R}^{\mathrm{RE}}_{\exp}$ requires it over $\mathbb{C}$.

Let $\mathfrak{R}$ be an expansion of the structure $\mathfrak{R}_0$. A set $S \subseteq \mathbb{R}^d$ is *definable* in $\mathfrak{R}$ if there exists a formula $\phi(x_1, \ldots, x_d)$ in $\mathfrak{R}$ with free variables $x_1, \ldots, x_d$ such that $S = \{(c_1, \ldots, c_d) \in \mathbb{R}^d \mid \mathfrak{R} \models \phi(c_1, \ldots, c_d)\}$. For $\mathfrak{R} = \mathfrak{R}_0$, the ordered field of real numbers, $\mathfrak{R}_0$-definable sets are known as *semi-algebraic* sets.

▶ Remark 2.1. There is a natural first-order interpretation of the field of complex numbers $\mathbb{C}$ in the field of real numbers $\mathbb{R}$. We shall say that a set $S \subseteq \mathbb{C}^d$ is $\mathfrak{R}$-*definable* if the image $\{(x, y) \in \mathbb{R}^d \times \mathbb{R}^d \mid x + iy \in S\}$ of S under this interpretation is $\mathfrak{R}$-definable.

A totally ordered structure $\langle M, <, \ldots \rangle$ is said to be *o-minimal* if every definable subset of M is a finite union of intervals. Tarski's result on quantifier elimination implies that $\mathfrak{R}_0$ is o-minimal. The o-minimality of $\mathfrak{R}_{\exp}$ and $\mathfrak{R}^{\text{RE}}$ is shown in [27], and the o-minimality of $\mathfrak{R}^{\text{RE}}$ and $\mathfrak{R}^{\text{RE}}_{\exp}$ is due to [25, 26].

A *semi-algebraic invariant* is one that is definable in $\mathfrak{R}_0$. An *o-minimal invariant* is one that is definable in an o-minimal expansion of $\mathfrak{R}_{\exp}$.

3 Orbit Cones

In this section we define orbit cones, an object that plays a central role in the subsequent results. They can be thought of as over-approximations of the orbit that has certain desirable properties, and moreover it is canonical in the sense that any other invariant must contain a cone.

3.1 Jordan Normal Form

Let $\langle A, \mathbf{x}_0 \rangle$ be a continuous linear dynamical system. The exponential of a square matrix A is defined by its formal power series as

$$e^A \stackrel{\text{def}}{=} \sum_{n=0}^{\infty} \frac{A^n}{n!}.$$

Let $\lambda_1, \ldots, \lambda_k$ be the eigenvalues of A, and recall that when $A \in \mathbb{Q}^{d \times d}$, all the eigenvalues are algebraic. We can write A in *Jordan Normal Form* as $A = PJP^{-1}$ where $P \in \mathbb{C}^{d \times d}$ is an invertible matrix with algebraic entries, and $J = \text{diag}(B_1, \ldots, B_k)$ is a block-diagonal matrix where each block B_l is a Jordan block that corresponds to eigenvalue λ_l, and it has the form

$$B_l = \begin{pmatrix} \lambda_l & 1 & 0 & \cdots & 0 \\ 0 & \lambda_l & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & \lambda_l \end{pmatrix} \in \mathbb{C}^{d_l \times d_l}$$

with $\sum_{l=1}^{k} d_l = d$.

From the power series, we can write $e^{At} = Pe^{Jt}P^{-1}$. Further, $e^{Jt} = \text{diag}(e^{B_1}, \ldots, e^{B_k})$. For each $1 \leq l \leq k$, write $B_l = \Lambda_l + N_l$, where Λ_l is the $d_l \times d_l$ diagonal matrix $\text{diag}(\lambda_l, \ldots, \lambda_l)$ and N_l is the $d_l \times d_l$ matrix $\text{diag}_2(1, \ldots, 1)$; where $\text{diag}_j(\cdot)$ is the j-th diagonal matrix, with other entries zero.

The matrices Λ_l and N_l commute, since the former is a diagonal matrix. A fundamental property of matrix exponentiation is that if matrices A, B commute, then $e^{A+B} = e^A e^B$. Thus, we have

$$e^{Jt} = e^{\text{diag}(\Lambda_1 t + N_1 t, \ldots, \Lambda_k t + N_k t)} = \text{diag}(e^{\lambda_1 t}, \ldots, e^{\lambda_k t}) e^{\text{diag}(N_1 t, \ldots, N_k t)},$$

where by $\mathrm{diag}(e^{\lambda_1 t}, \ldots, e^{\lambda_k t})$ we mean the $d \times d$ diagonal matrix that has the entry $e^{\lambda_1 t}$ written d_1 times, the entry $e^{\lambda_2 t}$ written d_2 times and so on. It will always be clear from the context whether we repeat the entries because of their multiplicity or not.

Matrices N_l are nilpotent, so its power series expansion is a finite sum, i.e. a polynomial in $N_l t$. More precisely, one can verify that:

$$e^{N_l t} = I + \mathrm{diag}_2(t, \ldots, t) + \mathrm{diag}_3(\frac{t^2}{2}, \ldots, \frac{t^2}{2}) + \ldots + \mathrm{diag}_{d_l}\left(\frac{t^{(d_l-1)}}{(d_l-1)!}\right).$$

Set $Q(t) \overset{\text{def}}{=} \mathrm{diag}(e^{N_1 t}, \ldots, e^{N_k t})$. From the equation above, the entries of $Q(t)$ are polynomials in t with rational coefficients.

Write the eigenvalues as $\lambda_l = \rho_l + \mathrm{i}\omega_l$, so that

$$\mathrm{diag}(e^{\lambda_1 t}, \ldots, e^{\lambda_k t}) = \underbrace{\mathrm{diag}(e^{\rho_1 t}, \ldots, e^{\rho_k t})}_{E(t)} \cdot \underbrace{\mathrm{diag}(e^{\omega_1 \mathrm{i} t}, \ldots, e^{\omega_k \mathrm{i} t})}_{R(t)}$$

We have in this manner decomposed the orbit

$$\mathcal{O}(t_0) = \{P \quad E(t)\ R(t)\ Q(t) \quad P^{-1}\mathbf{x}_0 \ : \ t \geq t_0\},$$

into an exponential $E(t)$, a rotation $R(t)$, and a simple polynomial $Q(t)$ matrices that commute with one another. Having the orbit in such a form will facilitate the analysis done in the sequel.

3.2 Cones as Canonical Invariants

In a certain sense, the rotation matrix $R(t)$ is the most complicated, because of it, the orbit is not even definable in $\mathfrak{R}_{\exp}$. The purpose of cones is to abstract away this matrix by a much simpler subgroup of the complex torus

$$\mathbb{T} \overset{\text{def}}{=} \{\mathbf{z} \in \mathbb{C}^k \ : \ |z_i| = 1, 1 \leq i \leq k\}.$$

To this end, consider the group of additive relations among the frequencies $\omega_1, \ldots, \omega_k$:

$$S \overset{\text{def}}{=} \{\mathbf{a} \in \mathbb{Z}^k \ : \ a_1\omega_1 + \cdots + a_k\omega_k = 0\}.$$

The subgroup of the torus of interest, respects the additive relations as follows:

$$\mathbb{T}_\omega \overset{\text{def}}{=} \{(\tau_1, \ldots, \tau_k) \in \mathbb{T} \ : \ \text{for all } \mathbf{a} \in S,\ \tau_1^{a_1} \cdots \tau_k^{a_k} = 1\}.$$

Its desirable properties are summarised in the following proposition:

▶ **Proposition 3.1.** *For algebraic numbers $\omega_1, \ldots, \omega_k$,*

1. *$\mathbb{T}_\omega$ is semi-algebraic,*
2. *diagonals of $\{R(t) \ : \ t \geq 0\}$ form a dense subset of $\mathbb{T}_\omega$.*

Proof. Being an Abelian subgroup of $\mathbb{Z}^k$, S has a finite basis, moreover this basis can be computed because of effective bounds, [19, Section 3]. To check that $(\tau_1, \ldots, \tau_k)$ belongs to $\mathbb{T}_\omega$, it suffices to check that $\tau_1^{a_1} \cdots \tau_k^{a_k} = 1$ for $(a_1, \ldots, a_k)$ in the finite basis. This forms a finite number of equations, therefore $\mathbb{T}_\omega$ is semi-algebraic. The fact that this is a subset of vectors of complex numbers is not problematic in this case because of the simple first-order interpretation in the theory of reals, see Remark 2.1.

The second statement of the proposition is a consequence of Kronecker's theorem on inhomogeneous simultaneous Diophantine approximations, see [8, Page 53, Theorem 4]. The proof of a slightly stronger statement can also be found in [9, Lemma 4]. Examples can be found where the set of diagonals of $\{R(t) \ : \ t \geq 0\}$ is a strict subset of $\mathbb{T}_\omega$. ◀

The orbit cone can now be defined by replacing the rotations with the subgroup of the torus. As it turns out, for our purposes this approximation is not too rough.

▶ **Definition 3.2.** *The* orbit cone *from* $t_0 \geq 0$ *is*

$$\mathcal{C}_{t_0} \overset{\text{def}}{=} \left\{ P\ E(t)\ \mathrm{diag}(\boldsymbol{\tau})\ Q(t)\ P^{-1}\mathbf{x}_0\ :\ \boldsymbol{\tau} \in \mathbb{T}_\omega, t \geq t_0 \right\}.$$

We prove that the cone is an inductive invariant and also a subset of $\mathbb{R}^d$.

▶ **Lemma 3.3.** *For all* $\delta, t_0 \geq 0$*,* $e^{A\delta}\mathcal{C}_{t_0} \subseteq \mathcal{C}_{t_0}$*.*

Proof. Fix $t \geq t_0$ and $\boldsymbol{\tau} \in \mathbb{T}_\omega$, and consider the point

$$\mathbf{v} = P\ E(t)\ \mathrm{diag}(\boldsymbol{\tau})\ Q(t)\ P^{-1}\mathbf{x}_0 \in \mathcal{C}_{t_0},$$

then we can write $e^{A\delta}\mathbf{v}$ as

$$\begin{aligned} e^{A\delta}\mathbf{v} &= P\ E(\delta)R(\delta)Q(\delta) \cdot E(t)\mathrm{diag}(\boldsymbol{\tau})Q(t)\ P^{-1}\mathbf{x}_0 \\ &= P\ E(\delta+t)\ R(\delta)\mathrm{diag}(\boldsymbol{\tau})\ Q(\delta)Q(t)\ P^{-1}\mathbf{x}_0. \end{aligned}$$

The matrix $R(\delta)\mathrm{diag}(\boldsymbol{\tau})$ is equal to $\mathrm{diag}(\boldsymbol{\tau}')$ for some $\boldsymbol{\tau}' \in \mathbb{T}_\omega$. Otherwise said, the vector $(e^{\delta\omega_1 \mathrm{i}}\tau_1, \ldots, e^{\delta\omega_k \mathrm{i}}\tau_k)$ belongs to $\mathbb{T}_\omega$. Indeed this is the case because for any $\mathbf{a} \in S$ we have

$$e^{a_1\delta\omega_1\mathrm{i}}\tau_1^{a_1} \cdots e^{a_k\delta\omega_k\mathrm{i}}\tau_k^{a_k} = e^{\delta\mathrm{i}\ (a_1\omega_1+\cdots+a_k\omega_k)} \cdot \tau_1^{a_1} \cdots \tau_k^{a_k} = 1.$$

Finally, by induction on the dimension d one can verify that $Q(\delta)Q(t) = Q(\delta + t)$. ◀

The fact that cones are subsets of $\mathbb{R}^d$ comes as a corollary of the following proposition whose proof can be found in Appendix A of the full version of the paper [3].

▶ **Proposition 3.4.** *Let* $A = PJP^{-1}$ *as above, and let* $C_i \in \mathbb{C}^{d_i \times d_i}$ *for* $i = 1, \ldots, k$*, with dimensions compatible to the Jordan blocks of* A*, and such that for every* i_1, i_2*, if* $B_{i_1} = \overline{B_{i_2}}$*, then* $C_{i_1} = \overline{C_{i_2}}$*. Then* $P\mathrm{diag}(C_1, \ldots, C_k)P^{-1}$ *has real entries.*

The matrix $E(t)\mathrm{diag}(\tau)Q(t)$ can be written as $\mathrm{diag}(C_1, \ldots, C_k)$ where the C_i matrices satisfy the conditions of Proposition 3.4, hence the following corollary.

▶ **Corollary 3.5.** *For all* $t_0 \geq 0$ *we have* $\mathcal{C}_{t_0} \subseteq \mathbb{R}^d$*.*

It is surprising that, already, the cones are a complete characterisation of o-minimal inductive invariants in the following sense.

▶ **Theorem 3.6.** *Let* $\mathcal{I}$ *be an o-minimal invariant that contains the orbit* $\mathcal{O}(u)$ *from some time* $u \geq 0$*, then there exists* $t_0 \geq u$ *such that:*

$$\mathcal{C}_{t_0} \subseteq \mathcal{I}.$$

Proof sketch. Conceptually, the proof follows along the lines of its analogue in [1]. There are a few differences, namely that the entries of the matrix A in [1] are assumed to be algebraic, while this is not true for the entries of e^A.

We define rays of the cone, which are subsets where $\boldsymbol{\tau} \in \mathbb{T}_\omega$ is fixed. Then we prove that for every ray, all but a finite part of it, is contained in the invariant. This is done by contradiction: if a ray is not contained in the invariant, a whole dense subset of the cone can be shown not to be contained in the invariant, leading to a contradiction, since the invariant is assumed to contain the orbit. We achieve this using some results on the topology of o-minimal sets.

The complete proof can be found in Appendix B of [3]. ◀

Another desirable property of cones is that they are $\mathfrak{R}_{\exp}$-definable. Also, one can observe that for every t_0, the set $\{e^{At}\mathbf{x}_0 : 0 \le t \le t_0\}$ is definable in $\mathfrak{R}_{\exp}^{\mathrm{RE}}$ (as we only need bounded restrictions of sin and cos to capture e.g. $e^{\mathrm{i}\omega_i}$ up to time t_0). As an immediate corollary of Theorem 3.6, we have the following theorems.

▶ **Theorem 3.7.** *Let $\langle A, \mathbf{x}_0\rangle$ be a CDS. For every $t_0 \ge 0$, the set $\mathcal{C}_{t_0} \cup \{e^{At}\mathbf{x}_0 : 0 \le t \le t_0\}$ is an invariant that contains the whole orbit of $\langle A, \mathbf{x}_0\rangle$. Moreover, this invariant is definable in $\mathfrak{R}_{\exp}^{\mathrm{RE}}$ (and in particular is o-minimal).*

▶ **Theorem 3.8.** *Let $\langle A, \mathbf{x}_0\rangle$ be a CDS and let $Y \subseteq \mathbb{R}^d$ be an error set. There exists an o-minimal invariant $\mathcal{I}$ that contains the orbit and is disjoint from Y if and only if there exists t_0 such that $\mathcal{C}_{t_0} \cup \{e^{At}\mathbf{x}_0 : 0 \le t \le t_0\}$ is such an invariant.*

Theorem 3.8 now allows us to provide an algorithm for deciding the existence of an invariant, subject to Schanuel's conjecture:

▶ **Theorem 3.9.** *Assuming Schanuel's conjecture, given a CDS $\langle A, \mathbf{x}_0\rangle$ and an $\mathfrak{R}_{\exp}^{\mathrm{RE}}$ definable error set Y, it is decidable whether there exists an o-minimal invariant for $\langle A, \mathbf{x}_0\rangle$ that avoids Y. Moreover, if such an invariant exists, we can compute a representation of it.*

Proof. By Theorem 3.8, there exists an o-minimal invariant $\mathcal{I}$ that avoids Y if and only if there exists some $t_0 \in \mathbb{R}$ such that $\mathcal{C}_{t_0} \cup \{e^{At}\mathbf{x}_0 : 0 \le t \le t_0\}$ is such an invariant. Thus, the problem reduces to deciding the truth value of the following $\mathfrak{R}_{\exp}^{\mathrm{RE}}$ sentence:

$$\exists t_0 \;:\; (\mathcal{C}_{t_0} \cup \{e^{At}\mathbf{x}_0 : 0 \le t \le t_0\}) \cap Y = \emptyset$$

The theory of $\mathfrak{R}_{\exp}^{\mathrm{RE}}$ is decidable subject to Schanuel's conjecture, and therefore we can decide the existence of an invariant. Moreover, if an invariant exists, we can compute a representation of it by iterating over increasing values of t_0, until we find a value for which the sentence $\left(\mathcal{C}_{t_0} \cup \{e^{At}\mathbf{x}_0 : 0 \le t \le t_0\}\right) \cap Y = \emptyset$ is true. ◀

4 Semi-algebraic Error Sets and Fat Trajectory Cones

In this section, we restrict attention to semi-algebraic invariants and semi-algebraic error sets, in order to regain unconditional decidability.

Substitute $s = e^t$ in the definition of the cone to get:

$$\mathcal{C}_{t_0} = \left\{P \; E(\log s) \; \mathrm{diag}(\boldsymbol{\tau}) \; Q(\log s) \; P^{-1}\mathbf{x}_0 \;:\; \boldsymbol{\tau} \in \mathbb{T}_\omega, \; s \ge e^{t_0}\right\}.$$

Written this way, observe that $E(\log s) = \mathrm{diag}(s^{\rho_1}, \ldots, s^{\rho_k})$, which is almost semi-algebraic, apart from the fact that the exponents need not be rational.

4.1 Unconditional Decidability

We give the final, yet crucial property of the cones. When the error set is semi-algebraic, it is possible to decide, unconditionally, whether there exists some cone that avoids the error set. Moreover the proof is constructive, it will produce the cone for which this property holds.

▶ **Theorem 4.1.** *For a semi-algebraic error set Y, it is (unconditionally) decidable whether there exists $t_0 \ge 0$ such that $\mathcal{C}_{t_0} \cap Y = \emptyset$. Moreover, such a t_0 can be computed.*

Proof. Define the set

$$U \overset{\text{def}}{=} \left\{ \mathcal{V} \in \mathbb{R}^{d\times d} \ : \ \forall \boldsymbol{\tau} \in \mathbb{T}_\omega, \quad P\ \mathcal{V}\ \text{diag}(\boldsymbol{\tau})\ P^{-1}\mathbf{x}_0 \in \mathbb{R}^d \setminus Y \right\}.$$

The set U can be seen to be semi-algebraic and thus is expressed by a quantifier-free formula that is a finite disjunction of formulas of the form $\bigwedge_{l=1}^m R_l(\mathcal{V}) \sim_l 0$, where each R_l is a polynomial with integer coefficients, over $d \times d$ variables of the entries of the matrix $\mathcal{V}$, and$\sim_l \in \{>, =\}$. Define the matrix

$$\Lambda(s) \overset{\text{def}}{=} \text{diag}(s^{\rho_1}, \ldots, s^{\rho_k}) Q(\log s) \in \mathbb{R}^{d\times d},$$

and notice that $\mathcal{C}_{t_0} \cap Y = \emptyset$ if and only if $\Lambda(s) \in U$ for every $s \geq e^{t_0}$. Thus, it is enough to decide whether there exists $s_0 \geq 1$ such that for every $s \geq s_0$, at least one of the disjuncts $\bigwedge_{l=1}^m R_l(\Lambda(s)) \sim_l 0$ is satisfied.

Since $R_l(\Lambda(s))$ are polynomials in entries of the form s^{ρ_i} and $\log(s)$, there is an effective bound s_0 such that for all $s \geq s_0$, none of the values $R_l(\Lambda(s))$ change sign for any $1 \leq l \leq m$. Hence we only need to decide whether there exists some $s_0' \geq s_0$ such that for all $s \geq s_0'$ we have $R_l(\Lambda(s)) \sim_l 0$ for every $1 \leq l \leq m$.

Fix some l. The polynomial $R_l(v_1, \ldots, v_D)$ has the form $\sum_i a_i v_1^{n_{i,1}} \cdots v_D^{n_{i,D}}$. After identifying the matrix $\Lambda(s)$ with a vector in $\mathbb{R}^D$ for $D = d^2$, we see that $R_l(\Lambda(s))$ is a sum of terms of the form

$$a_i s^{n'_{i,1}\rho_1 + \ldots n'_{i,k}\rho_k} \cdot Q_{i,1}(\log s) \cdots Q_{i,D}(\log s)$$

where the $n'_{i,j}$ are aggregations of the $n_{i,j}$ for identical entries of $\text{diag}(s_1^\rho, \ldots, s_k^\rho)$, and $Q_{i,j}(\log s)$ are polynomials obtained from the entries of $Q(\log s)$ under R_l. We can join the polynomials $Q_1, \ldots, Q_D$ into a single polynomial f_i, which would also absorb a_i. Thus, we rewrite R_l in the form $\sum_i s^{n'_{i,1}\rho_1 + \ldots n'_{i,k}\rho_k} f_i(\log s)$ where each f_i is a polynomial with rational coefficients (as the coefficients in $Q(\log s)$ are rational).

In order to reason about the sign of this expression as $s \to \infty$, we need to find the leading term of $R_l(\Lambda(s))$. This, however, is easy: the exponents $n'_{i,1}\rho_1 + \ldots + n'_{i,k}\rho_k$ are algebraic numbers, and are therefore susceptible to effective comparison. Thus, we can order the terms by magnitude. Then, we can determine the asymptotic sign of each coefficient $f_i(\log s)$ by looking at the leading term in f_i.

We can thus determine the asymptotic behaviour of each $R_l(\Lambda(s))$, to conclude whether $\bigwedge_{l=1}^m R_l(\Lambda(s)) \sim_l 0$ eventually holds. Moreover, for rational s, every quantity above can be computed to arbitrary precision, therefore it is possible to compute a threshold s_0', after which, for all $s \geq s_0'$, $\bigwedge_{l=1}^m R_l(\Lambda(s)) \sim_l 0$ holds. This completes the proof. ◀

▶ **Theorem 4.2.** *For a semi-algebraic set Y, it is decidable whether there exists a o-minimal invariant, disjoint from Y, that contains the orbit $\mathcal{O}(u)$ after some time $u \geq 0$. Moreover in the positive instances an invariant that is $\mathfrak{R}_{\exp}$-definable can be constructed.*

Proof. If there is an invariant $\mathcal{I}$ that contains $\mathcal{O}(u)$, for some $u \geq 0$, then Theorem 3.6 implies that there exists some $t_0 \geq u$ such that $\mathcal{C}_{t_0}$ is contained in $\mathcal{I}$. Consequently, the question that we want to decide is equivalent to the question of whether there exists a t_0, such that $\mathcal{C}_{t_0} \cap Y = \emptyset$. The latter is decidable thanks to Theorem 4.1. The effective construction follows from the fact that such a t_0 is computable and that the cone is $\mathfrak{R}_{\exp}$-definable. ◀

4.2 Effectively Constructing the Semi-algebraic Invariant

We now turn to show that in fact, for semi-algebraic error sets Y, we can approximate $\mathcal{C}_{t_0}$ with a semi-algebraic set such that if $\mathcal{C}_{t_0}$ avoids Y, so does the approximation. Intuitively, this is done by relaxing the "non semi-algebraic" parts of $\mathcal{C}_{t_0}$ in order to obtain a *fat cone.* This relaxation has two parts: one is to "rationalize" the (possibly irrational) exponents $\rho_1, \dots, \rho_k$, and the other is to approximate the polylogs in $Q(\log s)$ by polynomials.

Relaxing the exponents. We start by approximating the exponents $\rho_1, \dots, \rho_k$ with rational numbers. We remark that naively taking rational approximations is not sound, as the approximation must also adhere to the additive relationships of the exponents.

Let $\boldsymbol{\ell} = (\ell_1, \dots, \ell_k)$ and $\mathbf{u} = (u_1, \dots, u_k)$ be tuples of rational numbers such that $\ell_i \le \rho_i \le u_i$ for $i = 1, \dots, k$. Define $\mathbb{S} \subseteq \mathbb{R}^k$ as:

$$\mathbb{S} \overset{\text{def}}{=} \left\{ (q_1, \dots, q_k) \in \mathbb{R}^k \ : \ \forall n_1, \dots, n_k \in \mathbb{Z}, \quad \left(\sum_{i=1}^{k} n_i \rho_i = 0 \Rightarrow \sum_{i=1}^{k} n_i q_i = 0 \right) \right\}$$

Thus, $\mathbb{S}$ captures the integer additive relationships among the ρ_i. Define

$$\mathrm{Box}(\boldsymbol{\ell}, \mathbf{u}) \overset{\text{def}}{=} \{ \mathrm{diag}(\mathbf{q}) \ : \ \boldsymbol{\ell} \le \mathbf{q} \le \mathbf{u}, \mathbf{q} \in \mathbb{S} \}.$$

Approximating polylogs. Let $\epsilon, \delta > 0$. We simply replace $\log s$ by r such that $\delta \le r \le s^\epsilon$. Note that it is not necessarily the case that $\delta \le \log s \le s^\epsilon$, so this replacement is a-priori not sound. However, for large enough s the inequalities do hold, which will suffice for our purposes.

We can now define the fat cone. Let $\epsilon, \delta > 0$ and $\boldsymbol{\ell} = (\ell_1, \dots, \ell_k)$ and $\mathbf{u} = (u_1, \dots, u_k)$ as above, the fat orbit cone $\mathcal{F}_{s_0, \epsilon, \delta, \boldsymbol{\ell}, \mathbf{u}}$ is the set:

$$\left\{ P \ \mathrm{diag}(s^{q_1}, \dots, s^{q_k}) \mathrm{diag}(\boldsymbol{\tau}) \ Q(r) P^{-1} \mathbf{x}_0 \ : \ \boldsymbol{\tau} \in \mathbb{T}_\omega, \ s \ge s_0, \ \delta \le r \le s^\epsilon, \ \mathbf{q} \in \mathrm{Box}(\boldsymbol{\ell}, \mathbf{u}) \right\}.$$

That is, the fat cone is obtained from $\mathcal{C}_{t_0}$ with the following changes:

- $R(\log s) = \mathrm{diag}(s^{\rho_1}, \dots, s^{\rho_k})$ is replaced with $\mathrm{diag}(s^{q_1}, \dots, s^{q_k})$, where the q_i are rational approximations of the ρ_i, and maintain the additive relationships.
- $Q(\log s)$ is replaced with $Q(r)$ where $\delta \le r \le s^\epsilon$.
- The variable s starts from s_0 (as opposed to e^{t_0}).

We first show that the fat cone is semi-algebraic (the proof is in [3] Appendix C), then proceed to prove that if there is a cone that avoids the error set, then there is a fat one that avoids it as well.

▶ **Lemma 4.3.** *$\mathcal{F}_{s_0, \epsilon, \delta, \boldsymbol{\ell}, \mathbf{u}}$ is definable in $\mathfrak{R}_0$, and we can compute a representation of it.*

▶ **Lemma 4.4.** *Let $Y \subseteq \mathbb{R}^d$ be a a semi-algebraic error set such that $\mathcal{C}_{t_0} \cap Y = \emptyset$ for some $t_0 \in \mathbb{R}$, then there exists $\delta, \epsilon, s_0, \boldsymbol{\ell}, \boldsymbol{u}$ as above such that*

1. *$\mathcal{F}_{s_0, \epsilon, \delta, \boldsymbol{\ell}, \boldsymbol{u}} \cap Y = \emptyset$, and*
2. *for every $t \ge 0$ it holds that $e^{At} \cdot \mathcal{F}_{s_0, \epsilon, \delta, \boldsymbol{\ell}, \boldsymbol{u}} \subseteq \mathcal{F}_{s_0, \epsilon, \delta, \boldsymbol{\ell}, \boldsymbol{u}}$.*

The result is constructive, so when t_0 is given, the constants $s_0, \epsilon, \delta, \boldsymbol{\ell}, \mathbf{u}$ can be computed. It follows that a corollary of this lemma, and Lemma 4.3, is a stronger statement than that of Theorem 4.2, namely one where $\mathfrak{R}_{\exp}$ is replaced by $\mathfrak{R}_0$. We state it here before moving on with the proof of Lemma 4.4.

▶ **Theorem 4.5.** *For a semi-algebraic set Y, it is decidable whether there exists a o-minimal invariant, disjoint from Y, that contains the orbit $\mathcal{O}(u)$ after some time $u \geq 0$. Moreover in the positive instances an invariant that is $\mathfrak{R}_0$-definable can be constructed.*

The proof of Lemma 4.4 is given by the two corresponding steps. The second step, proving the invariance of the fat cone, can be found in [3]. We turn our attention to the first step.

▶ **Lemma 4.6.** *Let $Y \subseteq \mathbb{R}^d$ be a semi-algebraic error set, and let $t_0 \in \mathbb{R}$ be such that $\mathcal{C}_{t_0} \cap Y = \emptyset$, then there exists $\delta, \epsilon, s_0, \boldsymbol{\ell}, \mathbf{u}$ as above such that $\mathcal{F}_{s_0,\epsilon,\delta,\boldsymbol{\ell},\mathbf{u}} \cap Y = \emptyset$.*

Proof. We use the same analysis and definitions of U, R_l, $\sim_l$, $\Lambda(s)$ as in the proof of Theorem 4.1 and focus on a single polynomial R_l. Recall that we had

$$R_l(\Lambda(s)) = \sum_i s^{n_{i,1}\rho_1 + \dots n_{i,k}\rho_k} f_i(\log s) \tag{1}$$

where each f_i is a polynomial with rational coefficients.

Denote $\boldsymbol{\rho} = (\rho_1, \dots, \rho_k)$. We show, first, how to replace the exponents vector $\boldsymbol{\rho}$ by any exponents vector in $\mathrm{Box}(\boldsymbol{\ell}, \mathbf{u})$ for appropriate $\boldsymbol{\ell}, \mathbf{u}$, and second, how to replace $\log s$ by r where $\delta \leq r \leq s^\epsilon$ for some appropriate δ and ϵ, while maintaining the inequality or equality prescribed by $\sim_l$.

Denote by N the set of vectors $\mathbf{n_i} = (n_{i,1}, \dots, n_{i,k})$ of exponents in (1). Let $\mu > 0$, such that for every $\mathbf{n}, \mathbf{n'} \in N$, if $\boldsymbol{\rho} \cdot (\mathbf{n} - \mathbf{n'}) \neq 0$ then $|\boldsymbol{\rho} \cdot (\mathbf{n} - \mathbf{n'})| > \mu$. That is, μ is a lower bound on the minimal difference between distinct exponents in (1). Observe that we can compute a description of μ, as the exponents are algebraic numbers.

Let $M = \max_{\mathbf{n},\mathbf{n'} \in N} \|\mathbf{n} - \mathbf{n'}\|$ (where $\|\cdot\|$ is the Euclidean norm in $\mathbb{R}^k$).

▷ Claim 4.7. Let $\mathbf{c} \in \mathbb{R}^k$ be such that $\|\boldsymbol{\rho} - \mathbf{c}\| \leq \frac{\mu}{2M}$, then, for all $\mathbf{n}, \mathbf{n'} \in N$, if $\boldsymbol{\rho} \cdot (\mathbf{n} - \mathbf{n'}) > 0$ then $\mathbf{c} \cdot (\mathbf{n} - \mathbf{n'}) > \frac{\mu}{2}$.

Proof of Claim 4.7. Suppose that $\boldsymbol{\rho} \cdot (\mathbf{n} - \mathbf{n'}) > 0$, then by the above we have $\boldsymbol{\rho} \cdot (\mathbf{n} - \mathbf{n'}) > \mu$, and hence

$$\mathbf{c} \cdot (\mathbf{n} - \mathbf{n'}) = \boldsymbol{\rho} \cdot (\mathbf{n} - \mathbf{n'}) + (\mathbf{c} - \boldsymbol{\rho}) \cdot (\mathbf{n} - \mathbf{n'}) \geq \mu - \|\mathbf{c} - \boldsymbol{\rho}\| \cdot \|\mathbf{n} - \mathbf{n'}\| \geq \mu - \frac{\mu}{2M} M = \frac{\mu}{2}. \blacktriangleleft$$

We can now choose $\boldsymbol{\ell}$ and $\mathbf{u}$ such that $u_i - \ell_i \leq \frac{\mu}{2M\sqrt{k}}$ and for all $\mathbf{c} \in \mathrm{Box}(\boldsymbol{\ell}, \mathbf{u})$ we have

$$\|\boldsymbol{\rho} - \mathbf{c}\| \leq \sqrt{\sum_{i=1}^{k} (u_i - \ell_i)^2} \leq \sqrt{\frac{\mu^2}{(2M)^2}} = \frac{\mu}{2M}.$$

It follows from Claim 4.7 and from the definition of $\mathrm{Box}(\boldsymbol{\ell}, \mathbf{u})$ that, intuitively, every $\mathbf{c} \in \mathrm{Box}(\boldsymbol{\ell}, \mathbf{u})$ maintains the order of magnitude of the monomials $s^{n_{i,1}\cdot\rho_1 + \dots + n_{i,k}\cdot\rho_k}$ in $R_l(\Lambda(s))$.

More precisely, let $\Lambda'(s) = \mathrm{diag}(s^{c_1}, \dots, s^{c_k})Q(\log s)$ for some $\mathbf{c} \in \mathrm{Box}(\boldsymbol{\ell}, \mathbf{u})$, then the exponent of the ratio of every two monomials in $R_l(\Lambda'(s))$ has the same (constant) sign as the corresponding exponent in $R_l(\Lambda(s))$. Moreover, the exponents of distinct monomials in $R_l(\Lambda(s))$ differ by at least $\frac{\mu}{2}$ in $R_l(\Lambda'(s))$.

We now turn our attention to the $\log s$ factor. First, let s_0 be large enough that $f_i(\log s)$ has constant sign for every $s \geq s_0$. We can now let δ be large enough such that for every $r \geq \delta$, the sign of $f_i(\log s)$ coincides with the sign of $f_i(r)$ for every $s \geq s_0$. It remains to give an upper bound on r of the form s^ϵ such that plugging $f_i(r)$ instead of $f_i(\log s)$ does not change the ordering of the terms (by their magnitude) in $R_l(\Lambda'(s))$.

Let B be the maximum degree of all polynomials f_i in (1), and define $\epsilon = \frac{\mu}{3B}$ (in fact, any $\epsilon < \frac{\mu}{2B}$ would suffice), then we have that, for $s \geq s_0$, $f_i(r)$ has the same sign as $f_i(\log s)$ for every $\delta \leq r \leq s^\epsilon$ (by our choice of δ), and guarantees that plugging s^ϵ instead of s does not change the ordering of the terms (by their magnitude) in R_l. Since the exponents of the monomials in $R_l(\Lambda'(s))$ differ by at least $\frac{\mu}{2}$, it follows that their order is maintained when replacing $\log s$ by $\delta \leq r \leq s^\epsilon$.

Let $\Lambda''(s) = \mathrm{diag}(s^{c_1}, \dots, s^{c_k})Q(r)$ for some $\mathbf{c} \in \mathrm{Box}(\boldsymbol{\ell}, \mathbf{u})$ and $\delta \leq r \leq s^\epsilon$, then by our choice of ϵ, the dominant term in $R_l(\Lambda''(s))$ is the same as that in $R_l(\Lambda(s))$. Therefore, for large enough s, the signs of $R_l(\Lambda''(s))$ and $R_l(\Lambda(s))$ are the same.

Note that since $\mathcal{C}_{t_0} \cap Y = \emptyset$, then w.l.o.g. $R_l(\Lambda(s)) \sim_l 0$ for every l. Thus, by repeating the above argument for each R_l, we can compute $s_0 \in \mathbb{R}$, $\epsilon > 0$, $\delta \in \mathbb{R}$, and $\boldsymbol{\ell}, \mathbf{u} \in \mathbb{Q}^k$ such that $\mathcal{F}_{s_0,\epsilon,\delta,\boldsymbol{\ell},\mathbf{u}} \cap Y = \emptyset$, and we are done. ◀

5 A Reduction from Zeros of an Exponential Polynomial

In Theorem 4.5, we showed unconditional decidability for the question of whether there exists an invariant containing the orbit $\mathcal{O}(u)$, for some $u \geq 0$. Even though we construct such an invariant, it cannot be used as a certificate proving that the orbit never enters the error set; however it is a certificate that the orbit of the system does not enter Y *after* time u.

In this section we give indications that deciding whether there exists an invariant that takes into account the orbit $\leq u$ is difficult. More precisely, we will reduce a problem about zeros of a certain exponential polynomial to the question of whether there exists a semi-algebraic invariant disjoint from Y containing $\mathcal{O}(0)$.

▶ Remark 5.1. In the setting of discrete linear dynamical systems, the existence of a semi-algebraic invariant from time t_0 immediately implies the existence of one from time 0. This is because the system goes through finitely many points from 0 to t_0, which can be added one by one to the semi-algebraic set. In this respect CDSs are more complicated to analyse.

The problem that we reduce from, can be stated as follows. We are given as input real algebraic numbers $a_1, \dots, a_n, \rho_1, \dots, \rho_n$, and $t_0 \in \mathbb{Q}$, and asked to decide whether the exponential function:

$$f(t) \stackrel{\text{def}}{=} a_1 e^{\rho_1 t} + \cdots + a_n e^{\rho_n t},$$

has any zeros in the interval $[0, t_0]$. This is a special case of the so-called Continuous Skolem Problem [6, 10].

While there has been progress on characterising the asymptotic distribution of complex zeros of such functions, less is known about the real zeros, and we lack any effective characterisation, see [6, 10] and the references therein. The difficulty of knowing whether f has a zero in the specified region is because (a) all the zeros have to be transcendental (a consequence of Hermite-Lindemann Theorem) and (b) there can be tangential zeros, that is f has a zero but it never changes its sign. See the discussion in [6, Section 6]. Finding the zeros of such a polynomial is a special case of the *bounded* continuous Skolem problem. We note that when ρ_i are all rational the problem is equivalent to a sentence of $\mathfrak{R}_0$ (and hence decidable) by replacing $t = \log s$.

The rest of this section is devoted to the proof of the following theorem.

▶ **Theorem 5.2.** *For every exponential polynomial f we can construct a CDS $\langle A, \mathbf{x}_0 \rangle$ and semi-algebraic set Y such that the following two statements are equivalent:*

- *there exists a semi-algebraic invariant disjoint from Y that contains $\mathcal{O}(0)$,*
- *f does not have a zero in $[0, t_0]$.*

Fix the function f, *i.e.* real algebraic numbers $a_1, \ldots, a_n, \rho_1, \ldots, \rho_n$ and $t_0 \in \mathbb{Q}$. Without loss of generality we can assume that $\rho_1, \ldots, \rho_n$ are all nonnegative, since $e^{\rho t} f(t) = 0$ if and only if $f(t) = 0$ where ρ is larger than all $\rho_1, \ldots, \rho_n$.

Since every ρ_i is algebraic, there is a minimal polynomial p_i, that has ρ_i as a simple root. Let A be the $d \times d$ companion matrix of the polynomial $p_1(x) \cdots p_n(x) x^2$. The numbers ρ_i are eigenvalues A of multiplicity one, and the latter also has zero as an eigenvalue of multiplicity two. In addition to those, the matrix A generally has other (complex) eigenvalues as well. We put A in Jordan normal form, $P^{-1}AP = J$ where J is made of two block diagonals: $\tilde{A}$ and B, where

$$\tilde{A} \stackrel{\text{def}}{=} \begin{pmatrix} \operatorname{diag}(\rho_1, \ldots, \rho_n) & & \\ & 0 & 1 \\ & 0 & 0 \end{pmatrix},$$

and B is some $(d-n-2) \times (d-n-2)$ matrix. Define:

$$\tilde{\mathbf{x}}_0 \stackrel{\text{def}}{=} (\underbrace{1, \ldots, 1}_{n+2}, 0, \ldots, 0),$$

the vector that has $n+2$ ones and the rest, $d-(n+2)$ zeros, whose purpose is to ignore the contribution of the eigenvalues in matrix B in the system. To simplify notation, since $\tilde{\mathbf{x}}_0$ is ignoring the contribution of the matrix B, the dynamics of the system $\langle J, \tilde{\mathbf{x}}_0 \rangle$ can be assume to be the same as:

$$e^{\tilde{A}t}(1, \ldots, 1) = (e^{\rho_1 t}, \ldots, e^{\rho_n t}, t).$$

Focus on a single eigenvalue, *i.e.* on the graph $\{(e^{\rho t}, t) \ : \ t \geq 0\}$, as the analysis will easily generalise to the CDS in question. This is itself a CDS, so terminology such as orbits *etc.* make sense. The challenge is to find a family of *tubes* around this exponential curve such that (a) all the tubes together with $\{(y, t) \ : \ t \geq t_0\}$ are invariants and (b) the tubes are arbitrarily close approximations of the curve.

We achieve this by the following families of polynomials:

- under-approximations are given by the family indexed by $n \in \mathbb{N}$:

$$P_n(t) \stackrel{\text{def}}{=} \sum_{k=0}^{n} \frac{(\rho t)^k}{k!}.$$

- over-approximations are given by a family indexed by $n \in \mathbb{N}$ and $\mu > 1$:

$$Q_{n,\mu}(t) \stackrel{\text{def}}{=} P_n(\mu t).$$

Define:

$$\mathcal{I}_{n,\mu} \stackrel{\text{def}}{=} \{(y, t) \ : \ P_n(t) \leq y \leq Q_{n,\mu}(t) \text{ and } 0 \leq t \leq t_0\}.$$

It is clear from Taylor's theorem and the assumption that $\rho > 0$, that by taking $n \to \infty$, and $\mu \to 1^+$ the sets $\mathcal{I}_{n,\mu}$ are arbitrary precise approximations of the graph $\{(e^{\rho t}, t) \ : \ t \geq 0\}$, what remains to show is that they are invariant.

▶ **Lemma 5.3.** *For every $\mu > 1$ there exists $n_0 \in \mathbb{N}$ such that for all $n \geq n_0$ the set*

$$\mathcal{I}_{n,\mu} \cup \{(y, t) \ : \ t > t_0\}$$

is an invariant containing the whole orbit, i.e. $\{(e^{\rho t}, t) \ : \ t \geq 0\}$.

The proof is in Appendix D of [3].

We can construct such invariants for every curve $e^{\rho_i t}$, and thus build $\tilde{\mathcal{I}}_{n,\mu}$ for

$$\left\{(e^{\rho_1 t}, \dots, e^{\rho_n t}, t) \ : \ t \geq 0\right\}.$$

To prove Theorem 5.2 we define $\tilde{Y}$ by the formula

$$\Phi(x_1, \dots, x_n, x_{n+1}) \overset{\text{def}}{=} a_1 x_1 + \cdots + a_n x_n = 0 \text{ and } 0 \leq x_{n+1} \leq t_0.$$

Since the analysis was done on the CDS $\langle J, \tilde{\mathbf{x}}_0 \rangle$, whose entries are not rational in general, before proceeding with the proof of Theorem 5.2, we need the following lemma to say that changing basis does not have an effect in the decision problem at hand:

▶ **Lemma 5.4.** *For every $\tilde{Y}$ semi-algebraic, there exists another semi-algebraic set Y and $\mathbf{x}_0$ with rational entries such that the following two statements are equivalent:*

- *$\langle J, \tilde{\mathbf{x}}_0 \rangle$ has a semi-algebraic invariant disjoint from $\tilde{Y}$, containing the whole orbit,*
- *$\langle PJP^{-1}, \mathbf{x}_0 \rangle$ has a semi-algebraic invariant disjoint from Y, containing the whole orbit.*

The proof can be found in [3]. Thanks to this lemma, we can prove Theorem 5.2 for the CDS $\langle J, \tilde{\mathbf{x}}_0 \rangle$ and the set $\tilde{Y}$ instead. This is done as follows. The direct implication is trivial. For the converse, observe that $f(t)$ does not have a zero in $[0, t_0]$ if and only if the $\mathcal{O}(0)$ and $\tilde{Y}$ are disjoint. Since both $\mathcal{O}(0)$ and $\tilde{Y}$ are closed sets, we can find a tube that contains $\mathcal{O}(0)$ and is disjoint from $\tilde{Y}$, *i.e.* there exists some $\mu > 1$ and $n \in \mathbb{N}$ such that

$$\tilde{\mathcal{I}}_{n,\mu} \cup \{(y, t) \ : \ t > t_0\},$$

is an invariant that is disjoint from $\tilde{Y}$ but contains $\mathcal{O}(0)$.

References

1 Shaull Almagor, Dmitry Chistikov, Joël Ouaknine, and James Worrell. O-minimal invariants for linear loops. In *45th International Colloquium on Automata, Languages, and Programming ICALP*, pages 1–14. Schloss Dagstuhl, 2018.

2 Shaull Almagor, Dmitry Chistikov, Joël Ouaknine, and James Worrell. O-minimal invariants for discrete-time dynamical systems, 2020.

3 Shaull Almagor, Edon Kelmendi, Joël Ouaknine, and James Worrell. Invariants for continuous linear dynamical systems, 2020. `arXiv:2004.11661`.

4 Rajeev Alur. *Principles of cyber-physical systems.* MIT Press, 2015.

5 Hirokazu Anai and Volker Weispfenning. Reach set computations using real quantifier elimination. In *International Workshop on Hybrid Systems: Computation and Control*, pages 63–76. Springer, 2001.

6 Paul C. Bell, Jean-Charles Delvenne, Raphaël M. Jungers, and Vincent D. Blondel. The continuous skolem-pisot problem. *Theor. Comput. Sci.*, 411(40–42):3625–3634, September 2010.

7 Mireille Broucke. Reachability analysis for hybrid systems with linear dynamics. *Mathematical Theory of Networks and Systems (MTNS'02)*, 2002.

8 John W.S. Cassels. *An Introduction to Diophantine Approximation.* Cambridge University Press, 1965.

9 Ventsislav Chonev, Joël Ouaknine, and James Worrell. On recurrent reachability for continuous linear dynamical systems. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 515–524, 2016.

10 Ventsislav Chonev, Joël Ouaknine, and James Worrell. On the skolem problem for continuous linear dynamical systems. In *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016).* Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

11 Peter Franek, Stefan Ratschan, and Piotr Zgliczynski. Quasi-decidability of a fragment of the first-order theory of real numbers. *Journal of Automated Reasoning*, 57(2):157–185, 2016.

12 Khalil Ghorbal, Andrew Sogokon, and André Platzer. A hierarchy of proof rules for checking positive invariance of algebraic and semi-algebraic sets. *Comput. Lang. Syst. Struct.*, 47:19–43, 2017.

13 Sumit Gulwani and Ashish Tiwari. Constraint-based approach for analysis of hybrid systems. In *Computer Aided Verification, 20th International Conference, CAV 2008, Proceedings*, volume 5123 of *Lecture Notes in Computer Science*, pages 190–203. Springer, 2008.

14 Emmanuel Hainry. Reachability in linear dynamical systems. In Arnold Beckmann, Costas Dimitracopoulos, and Benedikt Löwe, editors, *Logic and Theory of Algorithms*, pages 241–250. Springer Berlin Heidelberg, 2008.

15 Gerardo Lafferriere, George J. Pappas, and Sergio Yovine. Symbolic reachability computation for families of linear vector fields. *J. Symb. Comput.*, 32(3):231–253, 2001.

16 Jiang Liu, Naijun Zhan, and Hengjun Zhao. Computing semi-algebraic invariants for polynomial dynamical systems. In *Proceedings of the 11th International Conference on Embedded Software, EMSOFT 2011*, pages 97–106. ACM, 2011.

17 Angus Macintyre. Turing meets schanuel. *Annals of Pure and Applied Logic*, 167(10):901–938, 2016.

18 Angus Macintyre and Alex J. Wilkie. On the decidability of the real exponential field. In Piergiorgio Odifreddi, editor, *Kreiseliana. About and Around Georg Kreisel*, pages 441–467. A K Peters, 1996.

19 David W Masser. Linear relations on algebraic groups. *New Advances in Transcendence Theory*, pages 248–262, 1988.

20 Enric Rodríguez-Carbonell and Ashish Tiwari. Generating polynomial invariants for hybrid systems. In *Hybrid Systems: Computation and Control, 8th International Workshop, HSCC 2005, Proceedings*, volume 3414 of *Lecture Notes in Computer Science*, pages 590–605. Springer, 2005.

21 Andrew Sogokon, Khalil Ghorbal, Paul B. Jackson, and André Platzer. A method for invariant generation for polynomial continuous systems. In *Verification, Model Checking, and Abstract Interpretation - 17th International Conference, VMCAI, Proceedings*, volume 9583 of *Lecture Notes in Computer Science*, pages 268–288. Springer, 2016.

22 Andrew Sogokon and Paul B Jackson. Direct formal verification of liveness properties in continuous and hybrid dynamical systems. In *International Symposium on Formal Methods*, pages 514–531. Springer, 2015.

23 Thomas Sturm and Ashish Tiwari. Verification and synthesis using real quantifier elimination. In *Symbolic and Algebraic Computation, International Symposium, ISSAC 2011, Proceedings*, pages 329–336. ACM, 2011.

24 Alfred Tarski. A decision method for elementary algebra and geometry. *RAND Corporation, R-109*, 1951.

25 Lou van den Dries, Angus Macintyre, and David Marker. The elementary theory of restricted analytic fields with exponentiation. *Annals of Mathematics*, 140(1):183–205, 1994.

26 Lou Van den Dries, Chris Miller, et al. Geometric categories and o-minimal structures. *Duke Math. J*, 84(2):497–540, 1996.

27 A. J. Wilkie. Model completeness results for expansions of the ordered field of real numbers by restricted pfaffian functions and the exponential function. *Journal of the American Mathematical Society*, 9(4):1051–1094, 1996.

On Higher-Order Cryptography

Boaz Barak
Harvard University, Cambridge, MA, USA
b@boazbarak.org

Raphaëlle Crubillé
IMDEA Software Institute, Madrid, Spain
University of Paris, IRIF, France
rcrubille@irif.fr

Ugo Dal Lago
University of Bologna, Italy
INRIA, Sophia Antipolis, France
ugo.dallago@unibo.it

Abstract

Type-two constructions abound in cryptography: adversaries for encryption and authentication schemes, if active, are modeled as algorithms having access to oracles, i.e. as second-order algorithms. But how about making cryptographic schemes *themselves* higher-order? This paper gives an answer to this question, by first describing why higher-order cryptography is interesting as an object of study, then showing how the concept of probabilistic polynomial time algorithm can be generalized so as to encompass algorithms of order strictly higher than two, and finally proving some positive and negative results about the existence of higher-order cryptographic primitives, namely authentication schemes and pseudorandom functions.

2012 ACM Subject Classification Theory of computation → Denotational semantics; Theory of computation → Probabilistic computation; Theory of computation → Cryptographic primitives

Keywords and phrases Higher-order computation, probabilistic computation, game semantics, cryptography

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.108

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

Related Version An extended version [3] of this paper is available at `https://arxiv.org/abs/2002.07218`.

Funding *Boaz Barak*: is supported by NSF awards CCF 1565264 and CNS 1618026 and by a Simons Investigator Fellowship.
Ugo Dal Lago: is supported by the ERC CoG 818616 “DIAPASoN”, and by the ANR grants 19CE480014 “PPS” and 16CE250011 “REPAS”.

Acknowledgements We thank Juspreet Singh Sandhu for helpful discussions during the early stages of this project.

1 Introduction

Higher-order computation generalizes classic first-order one by allowing algorithms and functions to not only take *strings* but also *functions* in input. It is well-known that this way of computing gives rise to an interesting computability and complexity theory [26, 25, 31], and that it also constitutes a conceptual basis for the functional programming paradigm, in which higher-order subroutines allow for a greater degree of modularity and conciseness in programs.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 108; pp. 108:1–108:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In cryptography [19, 20, 24], computation is necessarily randomized, and being able to restrict the time complexity of adversaries is itself crucial: most modern cryptographic schemes are insecure against computationally *unbounded* adversaries. Noticeably, higher-order constructions are often considered in cryptography, in particular when modeling active adversaries, which have access to oracles for the underlying encryption, decryption, or authentication functions, and can thus naturally be seen as second-order algorithms. Another example of useful cryptographic constructions which can be spelled out at different type orders, are *pseudorandom* primitives. Indeed, pseudorandomness can be formulated on (families of) strings, giving rise to so-called pseudorandom *generators* [5], but also on (families of) first-order functions on strings, giving rise to the so-called pseudorandom *functions* [21]. In the former case, again, adversaries (i.e., distinguishers) are ordinary polynomial time algorithms, while in the latter case, they are polytime oracle machines.

Given the above, it is natural to wonder whether standard primitives like encryption, authentication, hash functions, or pseudorandom functions, could be made higher-order. As discussed in Section 2 below, that would represent a way of dealing with code-manipulating programs and their security in a novel, fundamentally interactive, way. Before even looking at the feasibility of this goal, there is one challenge we are bound to face, which is genuinely definitional: how could we even *give* notions of security (e.g. pseudorandomness, unforgeability, and the like) for *second-order* functions, given that those definitions would rely on a notion of *third-order* probabilistic polynomial time adversary, itself absent from the literature? Indeed, although different proposals exist for classes of feasible *deterministic* functionals [9, 25], not much is known if the underlying algorithm has access to a source of randomness. Moreover, the notion of feasibility cryptography relies on is based on the so-called *security parameter*, a global numerical value which controls the complexity of all the involved parties. In Section 3, we give a definition of higher-order probabilistic polynomial time by way of concepts borrowed from game semantics [22, 23, 2], and being inspired by recent work by Ferée [13]. We give evidence to the fact that the provided definition is general enough to capture a broad class of adversaries of order strictly higher than two.

After having introduced the model, we take a look at whether any concrete instance of a secure higher-order cryptographic primitive can be given. The results we provide are about pseudorandom functions and (deterministic) authentication. We prove on the one hand that those constructions are *not* possible if one insists on them having the expected type (see Section 4.2). On the other hand, we prove (in Section 4.3 below) that second-order pseudorandomness *is* possible if the argument function takes as input a string of *logarithmic* length.

2 The Why and How of Authenticating Functions

Encryption and authentication, arguably the two simplest cryptographic primitives, are often applied to *programs* rather than mere data. But when this is done, programs are treated as ordinary data, i.e., as *strings of symbols*. In particular, two different but equivalent programs are seen as different strings, and their encryptions or authentication tags can be completely different objects. It is natural to ask the following: would it be possible to deal with programs as *functions* and not as *strings*, in a cryptographic scenario? Could we, e.g., encrypt or authenticate programs seeing them as *black boxes*, thus without any access to their code?

For the sake of simplicity, suppose that the program `P` we deal with has a very simple IO behaviour, i.e. it takes as input a binary string of length n and returns a boolean. Authenticating `P` could in principle be done by querying `P` on some of its inputs and, based on the outputs to the queries, compute a tag for $\mathtt{P}$. As usual, such an authenticating scheme would be secure if no efficient adversary $\mathcal{A}$ could produce a tag for $\mathtt{P}$ without knowing the underlying secret key k (such that $|k| = n$), unless with negligible probability. Please notice that the adversary, contrarily to the scheme itself, could have access to the code of $\mathtt{P}$, even if that code has not been used during the authenticating process.

But how could security be *defined* in a setting like the above? The three entities at hand have the following types, where $\mathtt{MAC}$ is the authentication algorithm, $\mathbb{S} = \{0,1\}^*$ is the set of binary strings, and $\mathbb{B} = \{0,1\}$ is the set of boolean values:

$$\begin{aligned}\mathtt{P} &: \mathbb{S} \to \mathbb{B}\\ \mathtt{MAC} &: \mathbb{S} \to (\mathbb{S} \to \mathbb{B}) \to \mathbb{S}\\ \mathcal{A} &: ((\mathbb{S} \to \mathbb{B}) \to \mathbb{S}) \to (\mathbb{S} \to \mathbb{B}) \times \mathbb{S}\end{aligned}$$

The first argument of $\mathtt{MAC}$ is the key k, which is of course not passed to the adversary $\mathcal{A}$. The latter can query $\mathtt{MAC}_k$ and produce a function and its tag. Its type, as expected, has order three. The above is not an accurate description of the input-output behaviour of the involved algorithms, and in particular of the fact that the length of the input string to $\mathtt{P}$ might be in a certain relation to the length of k, i.e., the underlying security parameter. Reflecting all this in the types above is however possible by replacing occurrences of the type $\mathbb{S}$ with *refinements* of $\mathbb{S}$, as follows:

$$\begin{aligned}\mathtt{P} &: \mathbb{S}[n] \to \mathbb{B}\\ \mathtt{MAC} &: \mathbb{S}[n] \to (\mathbb{S}[r(n)] \to \mathbb{B}) \to \mathbb{S}[p(n)]\\ \mathcal{A} &: ((\mathbb{S}[r(n)] \to \mathbb{B}) \to \mathbb{S}[p(n)]) \to (\mathbb{S}[r(n)] \to \mathbb{B}) \times \mathbb{S}[p(n)]\end{aligned}$$

But how could the time complexity of the three algorithms above be defined? While polynomial time computability of the function $\mathtt{P}$ and the authenticating algorithm $\mathtt{MAC}$ can be captured in a standard way using, e.g., oracle Turing machines, the same cannot be said about $\mathcal{A}$. How to, e.g., appropriately account for the time $\mathcal{A}$ needs to "cook" a function f in $\mathbb{S}[n] \to \mathbb{B}$ to be passed to its argument functional? Appealing as it is, our objective of studying higher-order forms of cryptography is actually bound to be nontrivial, even from a purely *definitional* perspective.

Given the above discussion, the contributions of this paper can be described in greater detail, as follows:

- On the one hand, we give a *definition* of a polynomial-time higher-order probabilistic algorithm whose time complexity depends on a global security parameter and which is based on games and strategies, in line with game semantics [22, 23, 2]. This allows to discriminate satisfactorily between efficient and non-efficient adversaries, and accounts for the complexity of first-and-second-order algorithms consistently with standard complexity theory.
- On the other hand, we give some *positive* and *negative* results about the possibility of designing second-order cryptographic primitives, and in particular pseudorandom functions and authentication schemes. In particular we prove, by an essentially information-theoretic argument, that secure deterministic second-order authentication schemes of the kind sketched above *cannot* exist. A simple and direct reduction argument shows that a more restricted form of pseudorandom function exists under standard cryptographic assumptions. Noticeably, the adversaries we prove the existence of are of a very peculiar form, while the ones which we prove impossible to build are quite general.

3 Higher-Order Probabilistic Polynomial Time Through Parametrized Games

In this section, we introduce a framework inspired by game semantics, in which one can talk about the efficiency of probabilistic higher-order programs in presence of a global security parameter. While the capability of interpreting higher-order programs is a well-established feature of game semantics, dealing *at the same time* with probabilistic behaviors *and* efficiency constraints has – to the best of the authors' knowledge – not been considered so far. The two aspects have however been tackled *independently.* Several game models of probabilistic languages have been introduced: we can cite here, for instance, the fully abstract model of probabilistic Idealized Algol by Danos and Harmer [11], or the model of probabilistic PCF by Clairambault at al. [8]. About efficiency, we can cite the work by Férée [13] on higher-order complexity and game semantics, in which the cost of evaluating higher-order programs is measured parametrically on the size of *all* their inputs, including functions, thus in line with type-two complexity [9]. We are instead interested in the efficiency of higher-order definitions with respect to the *security parameter.* Unfortunately, existing probabilistic game models do not behave well when restricted to capture feasibility: *polytime computable* probabilistic strategies in the spirit of Danos and Harmer do not *compose* (see the Extended Version of this paper [3] for more details).

Contrary to most works in game semantics, we do not aim at building a model of a *particular* programming language, but we take game semantics as our model of computation. As a consequence, we are not bound by requirements to interpret particular programming features or to reflect their discriminating power, and the resulting notions of games and strategies will be very simple.

We present our game-based model of computation in three steps: first, we define a category of deterministic games and strategies called $\mathcal{PG}$ – for *parametrized games* – which capture computational agents whose behavior is parametrized by the security parameter. This model ensures that computational agents are *total*: they *always* answer any request by the opponent. In a second step, we introduce $\mathcal{PPG}$, as a sub-category of $\mathcal{PG}$ designed to model those agents whose time complexity is *polynomially bounded* with respect to the security parameter. Finally, we deal with *randomized agents* by allowing them to interact with a *probabilistic oracle*, that outputs (a bounded amount of) random bits.

3.1 Parametrized Deterministic Games

Our game model has been designed so as to be able to deal with security properties that – as exemplified by *computational indistinguishability* – are expressed by looking at the behavior of adversaries *at the limit*, i.e., when the security parameter tends towards infinity. The *agents* we consider are actually *families* of functions, indexed by the security parameter. As such, our game model can be seen as a parametrized version of Hyland's simple games [22], where the set of plays is replaced by a *family* of sets of plays, indexed by the natural numbers. Moreover, we require the total length of any interaction between the involved parties to be polynomially bounded in the security parameter.

We need a few preliminary definitions before delving into the definition of a game. Given two sets X and Y, we define $\mathrm{Alt}(X,Y)$ as $\{(a_1,\ldots,a_n) \mid a_i \in X \text{ if } i \text{ is odd}, a_i \in Y \text{ if } i \text{ is even}\}$, i.e., as the set of finite alternating sequences whose first element is in X. Given any set of sequences X, $\mathsf{Odd}(X)$ (respectively, $\mathsf{Even}(X)$) stands for the subset of X containing the sequences of odd (respectively, even) length. From now on, we implicitly assume that any (opponent or player) move m can be faithfully encoded as a string in an appropriate, fixed alphabet. This way, moves and plays implicitly have a length, that we will indicate through the unary operator $|\cdot|$. We fix a set **Pol** of unary polynomially-bounded total functions on the natural numbers, which includes the identity ι, base-2 logarithm $\lfloor \lg \rfloor$, addition, multiplication, and closed under composition. **Pol** can be equipped with the pointwise partial order: $p \leq q$ when $\forall n \in \mathbb{N}, p(n) \leq q(n)$.

▶ **Definition 1** (Parametrized Games). *A* parametrized game $G = (O_G, P_G, L_G)$ *consists of sets* O_G, P_G *of opponent and player moves, respectively, together with a family of non-empty prefix-closed sets* $L_G = \{L^n_G\}_{n\in\mathbb{N}}$, *where* $L^n_G \subseteq \mathit{Alt}(O_G, P_G)$, *such that there is* $p \in \mathbf{Pol}$ *with* $\forall n \in \mathbb{N}.\forall s \in L^n_G.|s| \leq p(n)$. *The union of* O_G *and* P_G *is indicated as* M_G, *and is said to be set of* moves *of* G.

For every $n \in \mathbb{N}$, L^n_G represents the set of legal plays, when n is the current value of the security parameter. Observe that the first move is always played by *the opponent*, and that for any fixed value of the security parameter n, the length of legal plays is bounded by $p(n)$, where $p \in \mathbf{Pol}$. In the following, we often form plays from moves coming from different games or from different copies of the same game. If s is such a play, we indicate, e.g., the sub-play of s consisting of the moves from G as s_G.

▶ **Example 2** (Ground Games). We present here some games designed to model *data-types*. The simplest game is probably the *unit game* $\mathbf{1} = (\{?\}, \{*\}, \{L^n_{\mathbf{1}}\}_{n\in\mathbb{N}})$ with just one opponent move and one player move, where $L^n_{\mathbf{1}} = \{\varepsilon, ?, ?*\}$ for every n. Just slightly more complicated than the unit game is the *boolean game* $\mathbf{B}$ in which the two moves 0 and 1 take the place of $*$. In the two games introduced so far, parametrization is not really relevant, since $L^n_G = L^m_G$ for every $n, m \in \mathbb{N}$. The latter is not true in $\mathbf{S}[p] = (\{?\}, \{0,1\}^*, \{L^n_{\mathbf{S}[p]}\}_{n\in\mathbb{N}})$ with $L^n_{\mathbf{S}[p]} = \{\varepsilon, ?\} \cup \{?s \mid |s| = p(n)\}$, which will be our way of capturing strings. A slight variation of $\mathbf{S}[p]$ is $\mathbf{L}[p]$, in which the returned string can have length *smaller or* equal to $p(n)$.

▶ **Example 3** (Oracle Games). As another example, we describe how to construct *polynomial boolean oracles* as games. For every polynomial $p \in \mathbf{Pol}$ we define a game $\mathbf{O}^p$ as $(\{?\}, \{0,1\}, \{L^n_{\mathbf{O}^p}\}_{n\in\mathbb{N}})$ with

$$\begin{aligned} L^n_{\mathbf{O}^p} = \{?\} &\cup \{?b_1?b_2\ldots?b_m \mid b_i \in \{0,1\} \wedge m \leq p(n)\} \\ &\cup \{?b_1?b_2\ldots?b_m? \mid b_i \in \{0,1\} \wedge m < p(n)\}. \end{aligned}$$

Our oracle games are actually a special case of a more general construction, that amounts to building, from any game G, and any polynomial p, a game which consists in playing G at most $p(n)$ times. That is itself nothing more than a bounded version [18] of the exponential construction from models of linear logic [16].

▶ **Definition 4** (Bounded Exponentials). *Let* $G = (O_G, P_G, L_G)$ *be a parametrized game. For every* $p \in \mathbf{Pol}$, *we define a new parametrized game* $!_pG := (O_{!_pG}, P_{!_pG}, L_{!_pG})$ *as follows:*

- $O_{!_pG} = \mathbb{N}_{>0} \times O_G$, *and* $P_{!_pG} = \mathbb{N}_{>0} \times P_G$;
- *For* $n \in \mathbb{N}$, $L^n_{!_pG}$ *is the set of those plays* $s \in \mathit{Alt}(O_{!_pG}, P_{!_pG})$ *such that:*
  - *for every* i, *the* i*-th projection* s_i *of* s *is in* L^n_G;
  - *if a move* $(i+1, z)$ *appears in* s *for* $i \in \mathbb{N}_{>0}$, *then a move* (i, x) *appears at some earlier point of* s, *and* $i + 1 \leq p(n)$.

We do not need any switching condition as in so-called AJM games [1]: the impossibility for the observer to switch between the various copies of G when playing in $!_pG$ is a byproduct of our very definition of a game. Observe that the game $\mathbf{O}^p$ is isomorphic to the game $!_p\mathbf{B}$ – we can build a bijection between legal plays having the same length.

Games specify *how* agents could play in a certain interactive scenario. As such, they do not represent *one* such agent, this role being the one of *strategies*. Indeed, a strategy on a game is precisely a way of specifying the *deterministic* behavior of an agent, i.e. how the agent plans to react to any possible move by the opponent. We moreover ask our strategies to be total, i.e., that the player cannot refuse to play when it is her turn.

▶ **Definition 5** (Strategies). *A* strategy *on a parametrized game $G = (O_G, P_G, L_G)$ consists of a family $\mathsf{f} = \{\mathsf{f}_n\}_{n\in\mathbb{N}}$, where f_n is a partial function from $\mathsf{Odd}(L_G^n)$ to P_G such that:*

- *for every $s \in \mathsf{Odd}(L_G^n)$, if $\mathsf{f}_n(s) = x$ is defined, then $sx \in L_G^n$;*
- *$sxy \in Dom(\mathsf{f}_n)$ implies that $x = \mathsf{f}_n(s)$;*
- *for every $s \in \overline{\mathsf{f}}_n$, if $sx \in L_G^n$ then $sx \in Dom(\mathsf{f}_n)$;*

where $\overline{\mathsf{f}}$ represents the set of plays characterising f, *defined as the family $\{\overline{\mathsf{f}}_n\}$ where $\overline{\mathsf{f}}_n = \{\varepsilon\} \cup \{s\mathsf{f}_n(s) \mid s \in Dom(\mathsf{f}_n)\} \subseteq L_G^n$.*

Any strategy f is entirely characterized by its set of plays $\overline{\mathsf{f}}$. As such, it does not need to be effective, i.e., it is entirely possible that f, seen as a function of the history and the security parameter, is an uncomputable function.

Up to now, the games we have described are such that their strategies are meant to represent concrete data: think about how a strategy for, e.g., $\mathbf{B}$ or $\mathbf{O}^p$ could look like. It is now time to build games modeling *functions*, this being embodied by the following construction on games:

▶ **Definition 6** (Constructing Functional Games). *The game $G \multimap H$ is given as $O_{G\multimap H} = P_G + O_H$, $P_{G\multimap H} = O_G + P_H$, and $L_{G\multimap H} = \{s \in Alt(O_{G\multimap H}, P_{G\multimap H}) \mid s_G \in L_G, s_H \in L_H\}$.*

Strategies for the game $G \multimap H$ are meant to model any agent that, when interacting with a strategy in G, behaves like a strategy for H. When G and H are ground games, this can indeed be seen as a function between the corresponding sets.

▶ **Example 7.** As an example, we look at the game $\mathbf{O}^p \multimap \mathbf{S}[p]$, which captures functions returning a string of size $p(n)$ after having queried a binary oracle at most $p(n)$ times. First, observe that:

$$\mathbf{O}^p \multimap \mathbf{S}[p] = (\{?_{\mathbf{S}[p]}, 0, 1\}, \{?_{\mathbf{O}^p}\} \cup \{0,1\}^\star, \{L^n_{\mathbf{O}^p\multimap\mathbf{S}[p]}\}_{n\in\mathbb{N}}),$$

where $L_n^{\mathbf{O}^p\multimap\mathbf{S}[p]}$ is generated by the following grammar:

$$\begin{aligned} q \in L_n^{\mathbf{O}^p\multimap\mathbf{S}[p]} &::= ?_{\mathbf{S}[p]}o \;\big|\; ?_{\mathbf{S}[p]}e \;\big|\; ?_{\mathbf{S}[p]}es && s \in \{0,1\}^* \text{ with } |s| \leq p(n) \\ e &::= \epsilon \;\big|\; ?_{\mathbf{O}^p}b_1 \ldots ?_{\mathbf{O}^p}b_m && b_i \in \{0,1\}, m \leq p(n) \\ o &::= ?_{\mathbf{O}^p} \;\big|\; ?_{\mathbf{O}^p}b_1 \ldots ?_{\mathbf{O}^p}b_{m-1}?_{\mathbf{O}^p} && b_i \in \{0,1\}, m \leq p(n) \end{aligned}$$

Of course there are *many* strategies for this game, and we just describe two of them here, both making use of the oracle: the first one – that we will call once_p – queries the oracle for a random boolean, and returns the string $0^{p(n)}$ or the string $1^{p(n)}$ depending on the obtained value. It is represented in Figure 1a. The second strategy – denoted mult_p and represented in Figure 1b – generates a random key of length $p(n)$ by making $p(n)$ calls to the probabilistic oracle.

We now look at how to *compose* strategies: given a strategy on $G \multimap H$, and $H \multimap K$, we want to build a strategy on $G \multimap K$ that combines them. We define composition as in [22, 32], except that we need to take into account the security parameter n.

	$\mathbf{O}^p$	$\multimap$	$\mathbf{S}[p]$
O			$?_{\mathbf{S}[p]}$
P	$?_{\mathbf{O}^p}$		
O	b		
P			$b^{p(n)}$

(a) The strategy once_p.

	$\mathbf{O}^p$	$\multimap$	$\mathbf{S}[p]$
O			$?_{\mathbf{S}[p]}$
P	$?_{\mathbf{O}^p}$		
O	b_1		
$\vdots$			
P	$?_{\mathbf{O}^p}$		
O	$b_{p(n)}$		
P			$b_1 \dots b_{p(n)}$

(b) The strategy mult_p.

Figure 1 Two Distinct Strategies on the Game $\mathbf{O}^p \multimap \mathbf{S}[p]$.

▶ **Definition 8** (Composition of Strategies). *Let G, H, K be parametrized games, and let f, g be two strategies on $G \multimap H$ and $H \multimap K$ respectively. We first define the set of* interaction sequences of f and g *as:*

$$(\mathsf{f} \parallel\!\mid \mathsf{g})_n = \{s \in (M_G + M_H + M_K)^\star \mid s_{G,H} \in \bar{\mathsf{f}}_n \wedge s_{H,K} \in \bar{\mathsf{g}}_n\}.$$

From there, we define the composition of f *and* g *as the unique strategy* $\mathsf{f};\mathsf{g}$ *such that:*

$$\overline{\mathsf{f};\mathsf{g}}_n = \{s_{G,K} \mid s \in (\mathsf{f} \parallel\!\mid \mathsf{g})_n\}.$$

We can check that $\mathsf{f};\mathsf{g}$ is indeed a strategy on $G \multimap K$, and that moreover composition, seen as an operation on strategies, is associative and admits an identity. We can thus define $\mathcal{PG}$ as the category whose objects are parametrized games, and whose set of morphisms $\mathcal{PG}(G, H)$ consists of the parametrized strategies on the game $G \multimap H$.

3.2 Polytime Computable Strategies

Parametrized games have been defined so as to have polynomially bounded *length*. However, there is no guarantee on the effectiveness of its strategies, i.e., that the next player move, can be computed algorithmically from the history, uniformly in the security parameter. This can be however tackled by considering a subcategory of $\mathcal{PG}$ in which strategies are not merely functions, but can be (efficiently) computed:

▶ **Definition 9** (Polytime Computable Strategies). *Let G be a parametrized game, and* f *be a strategy on G. We say that* f *is* polytime computable *when there exists a polynomial time Turing machine which on input $(1^n, s)$ returns* $\mathsf{f}(n)(s)$.

All strategies we have given as examples in the previous section are polytime computable. For example, the two strategies from Example 7 are both computable in linear time.

▶ **Proposition 10** (Stability of Polytime Computable Strategies). *Let G, H, K be polynomially bounded games. If f, g are polytime computable strategies, respectively on $G \multimap H$, and $H \multimap K$, then $\mathsf{f};\mathsf{g}$ is itself a polytime computable strategy.*

For elementary reasons, the identity strategy on any game G is polytime computable. We can thus write $\mathcal{PPG}$ for the the sub-category of $\mathcal{PG}$ whose objects are paramterized games, and whose morphisms are polytime computable strategies.

Let us now consider the algorithm `MAC` from Section 2. Its type can be turned into the parametrized game $\mathbf{S}[\iota] \multimap !_q(\mathbf{S}[r] \multimap \mathbf{B}) \multimap \mathbf{S}[p]$. The bounded exponential $!_q$ serves to model the fact that the argument function can be accessed a number of times which is *polynomially bounded* on n. As a consequence, `MAC` can only query the argument function a number of times which is negligibly smaller than the number of possible queries, itself exponential in n (if $r(n) \geq n$). As we will see in Section 4, this is the key ingredient towards proving security of such a message authentication code to be unattainable.

3.3 Probabilistic Strategies

Both in $\mathcal{PG}$ and in $\mathcal{PPG}$, strategies on any game G are ultimately *functions*, and the way they react to stimuli from the environment is completely deterministic. How could we reconcile all this with our claim that the framework we are introducing adequately models *randomized* higher-order computation? Actually, one could be tempted to define a notion of *probabilistic strategy* in which the underlying function family $\{f_n\}_{n\in\mathbb{N}}$ is such that f_n returns *a probability distribution* $f_n(s)$ of player moves when fed with the history s. This, however, would lead to some technical problems when *composing strategies*: it would be hard to keep the composition of two efficient strategies itself efficient (see [3]).

It turns out that a much more convenient route consists, instead, in defining a *probabilistic* strategy on G simply as a *deterministic* polytime strategy on $\mathbf{O}^p \multimap G$, namely as an ordinary strategy having access to polynomially many random bits. Actually, we have already encountered strategies of this kind, namely once_p and mult_p from Example 7. This will be our way of modeling higher-order probabilistic computations.

But in which sense does any probabilistic strategy behave *probabilistically*, given that, after all, behind it there is a *deterministic* (polytime) Turing machine? The following definition gives an answer to this question, in the particular case of probabilistic strategies on the game $\mathbf{B}$.

▶ **Definition 11.** *Let* f *be a strategy on the game* $\mathbf{O}^p \multimap \mathbf{B}$*. For every* $b \in \mathbb{B}$*, we define the* probability of observing b when executing f *as follows:*

$$\Pr(\mathsf{f} \Downarrow^n b) = \sum_{\substack{(b_1,\ldots,b_k)\in\mathbb{B}^k \\ \textit{with } (?_{\mathbf{B}}\cdot ?_{\mathbf{O}^p}\cdot b_1 \ldots ?_{\mathbf{O}^p}\cdot b_k\cdot b)\in\bar{\mathsf{f}}_n}} \frac{1}{2^k}.$$

Parametrized games and probabilistic strategies can be themselves seen as a category whose morphisms (from the game G to the game H) are pairs in the form (q, f), where f is a strategy in $\mathbf{O}^q \multimap (G \multimap H)$. This category can be proved to be symmetric monoidal closed, although cartesian closure fails: duplication is not available in its full generality, but only in bounded form, which, we conjecture, is enough to get the structure of a bounded exponential situation [6].

Given a probabilistic strategy f on G (i.e. a strategy on $\mathbf{O}^q \multimap G$) and $p \in \mathbf{Pol}$, we indicate as $!_p\mathsf{f}$ the strategy in which $p(n)$ copies of f are played, *but in which* randomness is resolved just once and for all, i.e. $!_p\mathsf{f}$ is the naturally defined strategy on $\mathbf{O}^q \multimap !_pG$ in which the $q(n)$ random bits are all queried for at the beginning of the play, after the first opponent move.

3.4 On the Expressive Power of Probabilistic Strategies

A few words about the expressive power of probabilistic strategies – seen as a model of higher-order randomized computation – are now in order. For trivial reasons, every probabilistic strategy for the game $\mathbf{S}[\iota] \multimap \mathbf{L}[p]$ can be precisely simulated by a probabilistic Turing machine

	$!_q(\mathbf{S}[r]$	$\multimap$	$\mathbf{B})$	$\multimap$	$\mathbf{S}[p]$
O					$?_{\mathbf{S}[p]}$
P			$(1, ?_{\mathbf{B}})$		
O	$(1, ?_{\mathbf{S}[r]})$				
P	$(1, s_1)$				
O			$(1, t_1)$		
		$\vdots$			
P			$(m, ?_{\mathbf{B}})$		
O	$(m, ?_{\mathbf{S}[r]})$				
P	(m, s_m)				
O			(m, t_m)		
P					v

Figure 2 Plays (of Maximal Length) for the game $!_q(\mathbf{S}[r] \multimap \mathbf{B}) \multimap \mathbf{S}[p]$.

working in polynomial time. Conversely, every such machine can be turned into a probabilistic strategy for the aforementioned game, once p is chosen as a sufficiently large polynomial. Similarly, behind any probabilistic strategy for the game $\mathbf{S}[\iota] \multimap !_q(\mathbf{L}[p] \multimap \mathbf{L}[r]) \multimap \mathbf{L}[s]$ there is an probabilistic *oracle* Turing machine working in polynomial time. The converse statement, however, can be proved *only assuming* the oracle with which the machine interacts to produce outputs polynomially related (in size) to the inputs.

More generally, the intrinsic restriction parametrized games impose on the *length* of any interaction indeed poses some limitations as to what strategies can do, and in particular to how they can interact with the environment. This implies that our framework is fundamentally inadequate as a characterization of, say, the basic feasible functionals [9]. We claim, however, that cryptography most often deals with situations in which, even if some of the parties can be computationally *unbounded*, the length of the interaction between them, but also the size of the exchanged messages, are *polynomially* bounded. The interested reader is invited to take a look at, e.g., the cryptographic experiments in [24]. Even in interactive proofs, in which no restrictions is put on complexity of the prover, the amount and size of the exchanged messages is by definition polynomially bounded.

4 The (In)feasibility of Higher-Order Cryptography

In this section, we give both negative and positive results about the possibility of defining a deterministic polytime strategy for the game $\mathbf{S}[\iota] \multimap !_q(\mathbf{S}[r] \multimap \mathbf{B}) \multimap \mathbf{S}[p]$ which could serve to authenticate functions. When r is linear, this is impossible, as proved in Section 4.2 below. When, instead, r is logarithmic (and q is at least linear), a positive result can be given, see Section 4.3.

But how would a strategy for the game $!_q(\mathbf{S}[r] \multimap \mathbf{B}) \multimap \mathbf{S}[p]$ look like? Plays for this game are in Figure 2. A strategy for such a game is required to determine the value of the query $s_{i+1} \in \mathbb{S}[r(n)]$ based on $t_1, \ldots, t_i \in \mathbb{B}$. Moreover, based on $t_1, \ldots, t_m$ (where $m \leq q(n)$), the strategy should be able to produce the value $v \in \mathbb{S}[p(n)]$. Strictly speaking, the strategy should also be able to respond to a situation in which the opponent directly replies to a move $(i, ?_{\mathbf{B}})$ by way of a truth value (i, t_i), without querying the argument. This is however a signal that the agent with which the strategy is interacting represents a *constant* function, and we will not consider it in the following.

The way we will prove deterministic authentication impossible when r is linear consists in showing that since q is polynomially bounded (thus negligibly smaller than the number of possible queries of type $\mathbf{S}[r]$ any function is allowed to make to its argument), there are many argument functions $\mathbb{S}[r(n)] \to \mathbb{B}$ which are indistinguishable, and would thus receive the same tag. In the following, we prove that the (relatively few) coordinates on which the argument function is queried can even be efficiently determined.

4.1 Efficiently Determining Influential Variables

A key step towards proving our negative result comes from the theory of influential variables in decision trees. In this section, we are going to give some preliminary definitions about it, without any aim at being comprehensive (see, e.g., [29]).

From now on, metavariables like N, M, L stand for natural number unrelated to the security parameter, unless otherwise specified. Given a natural number $N \in \mathbb{N}$, $[N]$ denotes the set $\{1, \dots, N\}$. Whenever $j \in [N]$, $e_j \in \mathbb{S}[N]$ is the binary string which is everywhere 0 except on the j-th component, in which it is 1.

▶ **Definition 12** (Variance and Influence). *For every distribution $\mathcal{D}$ over $\mathbb{S}[N]$, and $F : \mathbb{S}[N] \to \mathbb{B}$, we write $\mathbf{Var}_{\mathcal{D}}(F)$ for the value $\mathbb{E}(F(\mathcal{D})^2) - \mathbb{E}(F(\mathcal{D}))^2 = \Pr_{x,y\sim\mathcal{D}}(F(x) \neq F(y))$, called the* variance of F under $\mathcal{D}$. *For every distribution $\mathcal{D}$ over $\mathbb{S}[N]$, $F : \mathbb{S}[N] \to \mathbb{B}$, and $j \in [N]$, we define the* influence of j on F under $\mathcal{D}$, *written $\mathbf{Inf}^j_{\mathcal{D}}(F)$, as $\Pr_{x\sim D}[F(x) \neq F(x \oplus e_j)]$.*

The quantity $\mathbf{Inf}^j_{\mathcal{D}}(F)$ measures how much, on the average, changing the j-th input to F influences its output. If F does not depend too much on the j-th input, then $\mathbf{Inf}^j_{\mathcal{D}}(F)$ is close to 0, while it is close to 1 when switching the j-th input has a strong effect on the output.

▶ **Example 13.** Let $PARITY_N : \mathbb{S}[N] \to \mathbb{B}$ be the parity function on N bits. It holds that

$$\begin{aligned}\mathbf{Inf}^j_{\mathcal{D}}(PARITY_N) &= \Pr_{x\sim\mathcal{D}}[PARITY_N(x) \neq PARITY_N(x \oplus e_j)] \\ &= \sum_x \mathcal{D}(x) \cdot |PARITY_N(x) - PARITY_N(x \oplus e_j)| = \sum_x \mathcal{D}(x) = 1.\end{aligned}$$

This indeed matches the intuition: changing any one coordinate makes the output to change, independently on the distribution from which the input is drawn.

If $F : A \to \mathbb{S}[L]$, and $t \in [L]$, we define $F_t : A \to \mathbb{B}$ to be the function that on input $x \in A$ outputs the t-th bit of $F(x)$. The kind of distributions over $\mathbb{S}[N]$ we will be mainly interested at are the so-called *semi-uniform* ones, namely those in which some of the N bits have a fixed value, while the others take all possible values with equal probability. It is thus natural to deal with them by way of partial functions. For every partial function $g : [N] \to \mathbb{B}$ we define $Dom(g) \subseteq [N]$ to be the set of inputs on which g is defined, and U_g to be the uniform distribution of x over $\mathbb{S}[N]$ *conditioned on* $x_j = g(j)$ for every $j \in Dom(g)$, i.e., the distribution defined as follows:

$$U_g(x) = \begin{cases} \frac{1}{2^{N-|Dom(g)|}} & \text{if } x_j = g(j) \text{ for every } j \in Dom(g); \\ 0 & \text{otherwise.} \end{cases}$$

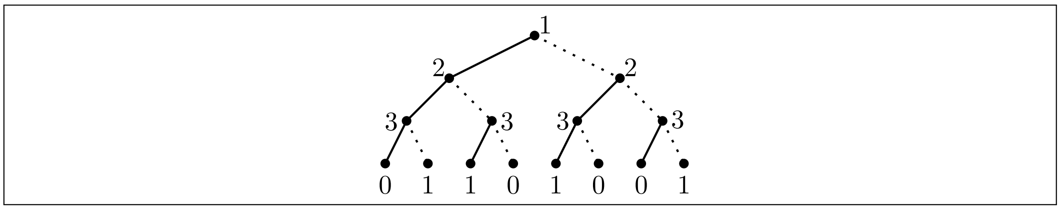


Figure 3 A Decision Tree for $PARITY_3$.

If a distribution $\mathcal{D}$ can be written as U_g, where $g : [N] \to \mathbb{B}$, we say that $\mathcal{D}$ is an $Dom(g)$-*distribution*, or a *semi-uniform* distribution. Given a distribution $\mathcal{D} : \mathbb{S}[N] \to \mathbb{R}_{[0,1]}$, some index $j \in [N]$ and a bit $b \in \mathbb{B}$, the expression $\mathcal{D}[j \leftarrow b]$ stands for the conditioning of $\mathcal{D}$ to the fact that the j-th boolean argument is b. Note that if $\mathcal{D}$ is an S-distribution and $j \in [N] \setminus S$, then $\mathcal{D}[j \leftarrow b]$ is an $S \cup \{j\}$-distribution.

A crucial concept in the following is that of a decision tree, which is a model of computation for boolean functions in which the interactive aspects are put upfront, while the merely computational aspects are somehow abstracted away.

▶ **Definition 14** (Decision Tree). *Given a function F, a* decision tree T for F *is a finite ordered binary tree whose internal nodes are labelled with an index $i \in [N]$, whose leaves are labelled with a bit $b \in \mathbb{B}$, and such that whenever a path ending in a leaf labelled with b is consistent with $x \in \mathbb{S}[N]$, it holds that $F(x) = b$. The* depth *of any decision tree T is defined the same as that of any tree.*

▶ **Example 15.** An example of a decision tree that computes the function $PARITY_3 : \mathbb{S}[3] \to \mathbb{B}$ defined in Example 13 can be found in Figure 3.

The following result, which is an easy corollary of some well-known results in the literature (i.e. Corollary 1.2 from [29]), put the variance and the influence in relation whenever the underlying function can be computed by way of a decision tree of limited depth.

▶ **Lemma 16.** *Suppose that F is computable by a decision tree of depth at most q and $g : [N] \to \mathbb{B}$ is a partial function. Then there exists $j \in [N] \setminus Dom(g)$ such that*

$$\mathbf{Inf}^j_{U_g}(F) \geq \frac{\mathbf{Var}_{U_g}(F)}{q}.$$

Every decision tree T makes on any input a certain number of queries, which of course can be different for different inputs. If $\mathcal{D}$ is a distribution, S is a subset of $[N]$ and T is a decision tree, we define $\Delta_{\mathcal{D},S}(T)$ as the expectation over $x \sim \mathcal{D}$ of the number of queries that T makes on input x outside of S, which is said to be *the average query complexity of T on $\mathcal{D}$ and S*. The following result relates the query complexity before and after the underlying semi-uniform distribution is updated: if we fix the value of a variable, then the average query complexity goes down (on the average) by at least the variable's influence:

▶ **Lemma 17.** *For every decision tree T computing a function F, $S \subseteq [N]$, $j \in [N] \setminus S$, and S-distribution $\mathcal{D}$, it holds that*

$$\tfrac{1}{2}\Delta_{\mathcal{D}[j \leftarrow 0], S \cup \{j\}}(T) + \tfrac{1}{2}\Delta_{\mathcal{D}[j \leftarrow 1], S \cup \{j\}}(T) \leq \Delta_{\mathcal{D},S}(T) - \mathbf{Inf}^j_{\mathcal{D}}(F).$$

By somehow iterating over Lemma 17, we can get the following result, which states that fixing enough coordinates, the variance can be made arbitrarily low, and that those coordinates can be efficiently determined:

▶ **Theorem 18.** *For every $F : \mathbb{S}[N] \to \mathbb{S}[L]$ such that for every $t \in [L]$, F_t is computable by a decision tree of depth at most Q, and every $\varepsilon > 0$, there exist a natural number $m \leq LQ^2/\varepsilon$ and a partial function $g : [N] \to \mathbb{B}$ where $|Dom(g)| \leq m$ such that $\mathbf{Var}_{U_g}(F_t) \leq \varepsilon$ for every $t \in \{1, \ldots, L\}$. Moreover, there is a polytime randomized algorithm A that on input F, $\delta > 0$, and $\varepsilon > 0$, makes at most $\mathcal{O}(LN) \cdot poly(Q/(\delta\varepsilon))$ queries to F and outputs such a partial function g with $|Dom(g)| \leq \mathcal{O}((LQ^2)/(\varepsilon\delta))$ with probability at least $1 - \delta$.*

Proof. We here give the main ingredients of the proof, referring to [3] for a more detailed account. The algorithm A proceeds by iteratively fixing new coordinates between the N many ones the function F depends on, stopping when the variance of all the functions F_t on the obtained semi-uniform distribution falls significantly below ε. The next coordinate to be fixed is chosen by estimating, using statistical methods, the influence of *all* the possible coordinates. Using similar methods, A can also estimate accurately the variance, and stop when enough coordinates are fixed. The role of Lemma 16 is to guarantee that if the variance is not too low, an influential variable can always be found, while the one of Lemma 17 consists in guaranteeing that a bounded number of iterations is enough. ◀

4.2 On the Impossibility of Authenticating Functions

Theorem 18 tells us that for every first-order boolean function which can be computed by a decision tree of low depth, there exist relatively few of its coordinates that, once fixed, determine the function's output with very high probability. If N is exponentially larger than Q, in particular, there is no hope for such a function to be a secure message authentication code. In this section, we aim at proving the aforementioned claim. In order to do it, we build a third-order randomized algorithm, which will be shown to fit into our game-theoretic framework.

More specifically, we are concerned with the cryptographic properties of strategies for the parametrized game $SOF_{q,r,p} =!_q(\mathbf{S}[r] \multimap \mathbf{B}) \multimap \mathbf{S}[p]$ and, in particular, with the case in which r is the identity ι, i.e. we are considering the game $LINSOF_{q,p} = SOF_{q,\iota,p}$. Any such strategy, when deterministic, can be seen as computing a family of functions $\{F_n\}_{n\in\mathbb{N}}$ where $F_n : (\mathbb{S}[n] \to \mathbb{B}) \to \mathbb{S}[p(n)]$. How could we fit all this into the hypotheses of Theorem 18?

The definitions of variance, influence, and decision tree can be easily generalised to functions in the form $F : (\mathbb{S}[N] \to \mathbb{B}) \to \mathbb{S}[M]$. Of course the underlying distribution $\mathcal{D}$ must be a distribution over functions $\mathbb{S}[N] \to \mathbb{B}$. The parameter N can be fixed in such a way that $n < N < 2^n$, where n is the security parameter. For simplicity we will choose N to be a power of 2, which hence divides 2^n.

▶ **Definition 19** (Extensions). *For every $x \in \mathbb{S}[N]$, we define the* extension *of x, denoted by f_x as the function $f_x : \mathbb{S}[n] \to \mathbb{B}$ such that for every $i \in [2^n]$ (identifying $\mathbb{S}[n]$ with the numbers $\{0, \ldots, 2^n - 1\}$ in the natural way), it holds that $f_x(i) = x_{\lfloor i/N \rfloor + 1}$. That is, f_x is the function that outputs x_1 on the first $2^n/N$ inputs, outputs x_2 on the second $2^n/N$ inputs, and so on and so forth. Given a distribution $\mathcal{D}$ over $\mathbb{S}[N]$, a distribution over functions $\mathbb{S}[n] \to \mathbb{B}$ can be formed in the natural way as $f_{\mathcal{D}}$.*

We will also make use of the following slight variation on the classic notion of Hamming distance: define $H(\cdot, \cdot)$ to be the so-called *normalized Hamming distance.* In fact, we overload the symbol H and use it for both *strings* in $\mathbb{S}[N]$ and *functions* in $\mathbb{S}[n] \to X$ for some set X. That is, if $x, y \in \{0, 1\}^N$ then $H(x, y) = \Pr_{j\in[N]}[x_j \neq y_j]$ while if $f, g \in \mathbb{S}[n] \to X$ then $H(f, g) = \Pr_{i\in\mathbb{S}[n]}[f(i) \neq g(i)]$.

Finally, the following ground game will be useful in the following as a way to represent partial functions as ground objects. The game $\mathbf{T}[q]$ is a slight variation on $\mathbf{S}[q]$ in which the returned string is in the ternary alphabet $\{0,1,\bot\}$. Any strategy for $\mathbf{T}[q]$ can thus be seen as representing a (family of) partial functions from $[q(n)]$ to $\mathbb{B}$.

▶ **Theorem 20.** *For every $\varepsilon,\delta>0$, there is a polytime probabilistic strategy* $\mathsf{infvar}_{\varepsilon,\delta}$ *on the game $!_s(\mathit{LINSOF}_{q,p})\multimap\mathbf{T}[t]$ such that for every deterministic strategy* f *on $\mathit{LINSOF}_{q,p}$ computing $\{F_n\}_{n\in\mathbb{N}}$, the composition $(!_s\mathsf{f});\mathsf{infvar}_{\varepsilon,\delta}$, with probability at least $1-\delta$, computes some functions $g_n:[t(n)]\to\mathbb{B}$ such that $\mathbf{Var}_{f_{U_{g_n}}}(F_n)\leq\varepsilon$ and $|\mathit{Dom}(g_n)|\leq\mathcal{O}(p(n)q^2(n)/(\delta\epsilon))$.*

Proof. This is a corollary to Theorem 18, since the functions F_n can be seen as first-order functions, and can thus be queried on functions in the form f_x (see Definition 19), where $x\in\mathbb{S}[N]$, and N is appropriately chosen so as to be significantly smaller than 2^n. Since F_n is computed by f, it can be computed by a decision tree having depth $q(n)$. Please refer to [3] for a more detailed proof. ◀

Remarkably, the strategy $\mathsf{infvar}_{\varepsilon,\delta}$ infers the "influential variables" of f *without* looking at how the latter queries its argument function, something which would anyway be available in the history of the interaction. This is reminiscent of *innocence* [23], a key concept in game semantics. We can now state the main result of this section.

▶ **Theorem 21.** *For every δ there is a polytime probabilistic strategy* coll_δ *on a game $!_s(\mathit{LINSOF}_{q,p})\multimap(\mathbf{S}[\iota]\multimap\mathbf{B})\otimes(\mathbf{S}[\iota]\multimap\mathbf{B})$ such that for every deterministic strategy* f *on $\mathit{LINSOF}_{q,p}$ computing $\{F_n\}_{n\in\mathbb{N}}$, the composition $(!_s\mathsf{f});\mathsf{coll}_\delta$, with probability at least $1-\delta$, computes two function families g,h with $g_n,h_n:\mathbb{S}[n]\to\mathbb{B}$ such that*
1. *$H(g_n,h_n)\geq 0.1$ for every n.*
2. *$F_n(g_n)=F_n(h_n)$ for every n.*
3. *For every function f on which* coll_δ *queries its argument, it holds that $H(f,g_n)\geq 0.1$ and $H(f,h_n)\geq 0.1$.*

Proof. The strategy coll_δ can be easily built from $\mathsf{infvar}_{\varepsilon,\delta}$: the former calls the latter, and then draws two strings independently at random from U_{k_n}, where k_n is the function the latter produces in output, and obtaining x,y. The two required outputs are thus f_x and f_y, and have all the required properties. ◀

This shows that coll_δ finds a *collision* for F_n as a pair of functions that are different from each other (and in fact significantly different in Hamming distance) but for which F_n outputs the same value, and hence F cannot be a collision-resistant hash function. Moreover, because the functions are far from those queried, this means that F_n cannot be a secure message authentication code either, since by querying F_n on g_n, the adversary can predict the value of the tag on h_n.

4.3 A Positive Result on Higher-Order Pseudorandomness

We conclude this paper by giving a positive result. More specifically, we prove that pseudorandomness can indeed be attained at second order, but at a high price, namely by switching to the type $\mathit{LOGSOF}_p=\mathit{SOF}_{\iota,\lfloor\lg\rfloor,p}$. This indeed has the same structure of $\mathit{LINSOF}_{q,p}$, but the argument function takes in input strings of *logarithmic size* rather than linear size. Moreover, the argument function can be accessed a linear number of times, which is enough to query it on *every possible* coordinate.

The fact that a strategy on LOGSOF_p can query its argument on every possible coordinate renders the attacks described in the previous section unfeasible. Actually, LOGSOF_p can be seen as an *interactive* variation of the game $\mathbf{S}[\iota]\multimap\mathbf{S}[p]$, for which pseudorandomness is

	$!_{\iota+1}(\mathbf{L}[\iota]$	$\multimap$	$\mathbf{S}[w])$	$\multimap$	$!_{\iota}(\mathbf{S}[\lfloor\lg\rfloor]$	$\multimap$	$\mathbf{B})$	$\multimap$	$\mathbf{S}[p])$
O									$?_{\mathbf{S}[p]}$
			$\vdots$						
P			$(i, ?_{\mathbf{S}[w]})$						
O	$(i, ?_{\mathbf{L}[\iota]})$								
P	$(i, (z_1 \cdots z_{i-1})$								
O			$(i, j_i))$						
P							$(i, ?_{\mathbf{B}})$		
O					$(i, ?_{\mathbf{S}[\lfloor\lg\rfloor]})$				
P					$(i, \alpha(j_i, \{j_1, \ldots, j_{i-1}\})$				
O							(i, z_i)		
			$\vdots$						
P			$(n, ?_{\mathbf{S}[w]})$						
O	$(n, ?_{\mathbf{L}[\iota]})$								
P	$(n, (z_1 \cdots z_n))$								
O			(n, v)						
P									$\beta(v)$

Figure 4 Plays in $\overline{\mathsf{fo2so}}$.

well known to be attainable starting from standard cryptographic assumptions [19]: simply, instead of taking in input *the whole* string *at once*, it queries it *bit by bit*, in a certain order. A *random strategy* of that type, then, would be one that, using the notation from Figure 2,

- Given $t_1, \ldots, t_i \in \mathbb{B}$, returns a string s_{i+1} uniformly chosen at random from $\mathbb{S}[r(n)] - \{s_1, \ldots, s_i\}$, this for every $i < q(n)$.
- Moreover, based on $t_1, \ldots, t_{r(n)}$, it produces a string v chosen uniformly at random from $\mathbb{S}[p(n)]$.

Please notice that this random strategy can be considered as a *random* functional in $(\mathbb{S}[r(n)] \to \mathbb{B}) \to \mathbb{S}[p(n)]$ only if $r(n)$ is logarithmic, because this way the final result v is allowed to depend on the value of the input function in *all possible coordinates.* The process of generating such a random strategy uniformly can be seen[1] as a probabilistic strategy $\mathsf{randsof}$. We are now ready to formally define pseudorandom functions:

▶ **Definition 22** (Second-Order Pseudorandom Function). *A deterministic polytime strategy* f *on* $\mathbf{S}[\iota] \multimap LOGSOF_p$ *is said to be* pseudorandom *iff for every probabilistic polytime strategy* $\mathcal{A}$ *on* $!_s LOGSOF_p \multimap \mathbf{B}$ *there is a negligible function* $\varepsilon : \mathbb{N} \to \mathbb{R}_{[0,1]}$ *such that*

$$|\Pr(!_s(\mathsf{mult}_\iota; \mathsf{f}); \mathcal{A} \Downarrow^n 1) - \Pr(!_s\mathsf{randsof}; \mathcal{A} \Downarrow^n 1)| \leq \varepsilon(n).$$

The way we build a pseudorandom function consists in constructing a *deterministic* polytime strategy $\mathsf{fo2so}$ for the game $!_{\iota+1}(\mathbf{L}[\iota] \multimap \mathbf{S}[w]) \multimap LOGSOF_p$, where $w \in \mathbf{Pol}$ is such that $w \geq \lfloor\lg\rfloor$ and $w \geq p$. The strategy is represented in Figure 4. The function α interprets its first argument (a string in $\mathbb{S}[w(n)]$), as an element of $\mathbb{S}[\lfloor\lg\rfloor(n)]$ distinct from those it takes as second argument, and distributing the probabilities uniformly. The function β, instead, possibly discards some bits of the input and produces a possibly shorter string.

[1] the strategy at hand would, strictly speaking, need exponentially many random bits, which are not allowed in our model; this could be accomodated without any major problem.

The way the strategy $\mathsf{fo2so}$ is defined makes the composition $\mathsf{f};\mathsf{fo2so}$ statistically very close to the random strategy whenever f is chosen uniformly at random among the strategies for the parametric game $\mathbf{L}[\iota] \multimap \mathbf{S}[w]$. This allows us to prove the following:

▶ **Theorem 23.** *Let $F : \{0,1\}^n \times \{0,1\}^{\leq n} \to \{0,1\}^{w(n)}$ be a pseudorandom function and let f_F be the deterministic polytime strategy for the game $\mathbf{S}[\iota] \multimap !_{\iota+1}(\mathbf{L}[\iota] \multimap \mathbf{S}[w])$ obtained from F. Then, $\mathsf{f}_F;\mathsf{fo2so}$ is second-order pseudorandom.*

5 Related Work

Game semantics and the geometry of interaction are among the best-studied program semantic frameworks (see, e.g. [17, 2, 23]), and can also be seen as computational models, given their operational flavor. This is particularly apparent in the work on abstract machines [10, 14], but also in the so-called geometry of synthesis [15]. In this paper, we are particularly interested in the latter use of game semantics, and take it as the underlying computational model. The definition of our game model has been strongly inspired by works by Hyland [22] and Wolverson [32], the main novelties being parametrization and the bounded exponential construction, which together allow us to account for efficient randomized higher-order computations of the kinds used in cryptography. As a consequence, our definition of an acceptable strategy is more permissive than the ones from so-called AJM games [2] and HO games [23], the former being history-free, the latter essentially relying on so-called justification pointers.

This is certainly not the first paper in which cryptography is generalized to computational models beyond the one of first-order functions. One should of course cite Canetti's universally composable security [7], but also Mitchell et al.'s framework, the latter based on process algebras [28]. None of them deals with security properties of higher-order functions, though. A precursor of the aforementioned work [27] deals with first-order probabilistic polynomial time by way of oracles in an higher-order calculus, but lacks any claim about how probabilistic polynomial time would look like for genuinely higher-order functions.

Various ways to generalize the so-called formal model [12] to higher-order computation have been proposed. As an example, this is what Sumii and Pierce [30] do with their system of logical relations, which is shown to guarantee a form of non-interference. Similarly for Bhargavan et al.'s type system [4] in which cryptographic primitives are seen as libraries for the language F#. All this is is however fundamentally different from what we do here, namely extending the so-called *computational* model to higher-order computation: randomized behaviours and time-bounds are abstracted away.

References

1 Samson Abramsky and Radha Jagadeesan. Games and full completeness for multiplicative linear logic. *J. Symb. Log.*, 59(2):543–574, 1994.

2 Samson Abramsky, Radha Jagadeesan, and Pasquale Malacaria. Full abstraction for PCF. *Inf. Comput.*, 163(2):409–470, 2000.

3 Boaz Barak, Raphaëlle Crubillé, and Ugo Dal Lago. On higher-order cryptography (long version). *CoRR*, abs/2002.07218, 2020. arXiv:2002.07218.

4 Karthikeyan Bhargavan, Cédric Fournet, and Andrew D. Gordon. Modular verification of security protocol code by typing. In *Proc. of POPL 2010*, pages 445–456, 2010.

5 Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM J. Comput.*, 13(4):850–864, 1984.

6 Aloïs Brunel, Marco Gaboardi, Damiano Mazza, and Steve Zdancewic. A core quantitative coeffect calculus. In *Proc. of ESOP 2014*, pages 351–370, 2014.

7 Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proc. of FOCS 2001*, pages 136–145, 2001.

8 Pierre Clairambault and Hugo Paquet. Fully abstract models of the probabilistic lambda-calculus. In *Proc. of CSL 2018*, pages 16:1–16:17, 2018.

9 Stephen A. Cook and Bruce M. Kapron. Characterizations of the basic feasible functionals of finite type. In *Proc. of FOCS 1989*, pages 154–159, 1989.

10 Pierre-Louis Curien and Hugo Herbelin. Abstract machines for dialogue games. *CoRR*, abs/0706.2544, 2007.

11 Vincent Danos and Russell Harmer. Probabilistic game semantics. *ACM Trans. Comput. Log.*, 3(3):359–382, 2002.

12 Danny Dolev and Andrew Chi-Chih Yao. On the security of public key protocols. *IEEE Trans. Inf. Theory*, 29(2):198–207, 1983.

13 Hugo Férée. Game semantics approach to higher-order complexity. *J. Comput. Syst. Sci.*, 87:1–15, 2017.

14 Olle Fredriksson and Dan R. Ghica. Abstract machines for game semantics, revisited. In *Proc. of LICS 2013*, pages 560–569, 2013.

15 Dan R. Ghica. Geometry of synthesis: a structured approach to VLSI design. In *Proc. of POPL 2007*, pages 363–375, 2007.

16 Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–101, 1987.

17 Jean-Yves Girard. Geometry of interaction 1: Interpretation of system F. *Logic Colloquium 88*, 1989.

18 Jean-Yves Girard, Andre Scedrov, and Philip J. Scott. Bounded linear logic: A modular approach to polynomial-time computability. *Theor. Comput. Sci.*, 97(1):1–66, 1992.

19 Oded Goldreich. *Foundations of Cryptography: Volume 1*. Cambridge University Press, 2006.

20 Oded Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, 2009.

21 Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986.

22 Martin Hyland. Game semantics. In Andy Pitts and Peter Dybjer, editors, *Semantics and Logics of Computation*. Cambridge University Press, 1997.

23 Martin Hyland and Luke Ong. On full abstraction for PCF: I, II, and III. *Inf. Comput.*, 163(2):285–408, 2000.

24 Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. Chapman & Hall/CRC, 2007.

25 Akitoshi Kawamura and Stephen A. Cook. Complexity theory for operators in analysis. *TOCT*, 4(2):5:1–5:24, 2012.

26 John Longley and Dag Normann. *Higher-Order Computability*. Theory and Applications of Computability. Springer, 2015.

27 John C. Mitchell, Mark Mitchell, and Andre Scedrov. A linguistic characterization of bounded oracle computation and probabilistic polynomial time. In *Proc. of FOCS 1998*, pages 725–733, 1998.

28 John C. Mitchell, Ajith Ramanathan, Andre Scedrov, and Vanessa Teague. A probabilistic polynomial-time process calculus for the analysis of cryptographic protocols. *Theor. Comput. Sci.*, 353(1-3):118–164, 2006.

29 Ryan O'Donnell, Michael E. Saks, Oded Schramm, and Rocco A. Servedio. Every decision tree has an in.uential variable. In *Proc. of FOCS 2005*, pages 31–39, 2005.

30 Eijiro Sumii and Benjamin C. Pierce. Logical relations for encryption. *Journal of Computer Security*, 11(4):521–554, 2003.

31 Klaus Weihrauch. *Computable Analysis: An Introduction*. Springer Publishing Company, Incorporated, 2013.

32 Nicholas Wolverson. *Game semantics for an object-oriented language*. PhD thesis, University of Edinburgh, UK, 2009.

Cost Automata, Safe Schemes, and Downward Closures

David Barozzini
Institute of Informatics, University of Warsaw, Poland
dbarozzini@mimuw.edu.pl

Lorenzo Clemente
Institute of Informatics, University of Warsaw, Poland
clementelorenzo@gmail.com

Thomas Colcombet
IRIF-CNRS-Université de Paris, France
thomas.colcombet@irif.fr

Paweł Parys
Institute of Informatics, University of Warsaw, Poland
parys@mimuw.edu.pl

Abstract

Higher-order recursion schemes are an expressive formalism used to define languages of possibly infinite ranked trees. They extend regular and context-free grammars, and are equivalent to simply typed λY-calculus and collapsible pushdown automata. In this work we prove, under a syntactical constraint called safety, decidability of the model-checking problem for recursion schemes against properties defined by alternating B-automata, an extension of alternating parity automata for infinite trees with a boundedness acceptance condition. We then exploit this result to show how to compute downward closures of languages of finite trees recognized by safe recursion schemes.

2012 ACM Subject Classification Theory of computation → Logic and verification; Theory of computation → Rewrite systems

Keywords and phrases Cost logics, cost automata, downward closures, higher-order recursion schemes, safe recursion schemes

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.109

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

Related Version A full version of the paper is available at [4], `https://arxiv.org/abs/2004.12187`.

Funding *David Barozzini*: Author supported by the National Science Centre, Poland (grant no. 2016/22/E/ST6/00041).
Lorenzo Clemente: Author supported by the National Science Centre, Poland (grant no. 2016/22/E/ST6/00041).
Thomas Colcombet: Supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No.670624), and the DeLTA ANR project (ANR-16-CE40-0007).
Paweł Parys: Author supported by the National Science Centre, Poland (grant no. 2016/22/E/ST6/00041).

1 Introduction

Higher-order functions are nowadays widely used not only in functional programming languages such as Haskell and the OCAML family, but also in mainstream languages such as Java, JavaScript, Python, and C++. *Recursion schemes* are faithful and algorithmically

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 109; pp. 109:1–109:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

manageable abstractions of the control flow of higher-order programs [36]. A deterministic recursion scheme normalises into a possibly infinite Böhm tree, and in this respect recursion schemes can equivalently be presented as simply-typed lambda-terms using a higher-order fixpoint combinator Y [50]. There are also nontrivial inter-reductions between recursion schemes and the equi-expressive collapsible higher-order pushdown automata [30] and ordered tree-pushdown automata [13]. In another semantics, also used in this paper, nondeterminstic recursion schemes are recognisers of languages of finite trees, and in this view they are also known as higher-order OI grammars [23, 38], generalising indexed grammars [2] (which are recursion schemes of order two) and ordered multi-pushdown automata [8].

The most celebrated algorithmic result in the analysis of recursion schemes is the decidability of the model-checking problem against properties expressed in monadic second-order logic (MSO): given a recursion scheme $\mathcal{G}$ and an MSO sentence φ, one can decide whether the Böhm tree generated by $\mathcal{G}$ satisfies φ [43]. This fundamental result has been reproved several times, that is, using collapsible higher-order pushdown automata [30], intersection types [37], Krivine machines [48], and it has been extended in diverse directions such as global model checking [11], logical reflection [9], effective selection [12], and a transfer theorem via models of lambda-calculus [49]. When the input property is given as an MSO formula, the model-checking problem is non-elementary already for trees of order 0 (regular trees) [51]; when the input property is presented as a parity tree automaton (which is equi-expressive with MSO on trees, but less succinct), the MSO model-checking problem for recursion schemes of order n is complete for n-fold exponential time [43]. Despite these hardness results, the model-checking problem can be solved efficiently on multiple nontrivial examples, thanks to the development of several recursion-scheme model checkers [36, 35, 10, 47, 42].

Unboundedness problems. Recently, an increasing interest has arose for model checking quantitative properties going beyond the expressive power of MSO. The *diagonal problem* is an example of a quantitative property not expressible in MSO. Over words, the problem asks, for a given set of letters Σ and a language of finite words $\mathcal{L}$, whether for every $n \in \mathbb{N}$ there is a word in $\mathcal{L}$ where every letter from Σ occurs at least n times. Over full trios (classes of languages closed under regular transductions), decidability of the diagonal problem over finite words has interesting algorithmic consequences, such as computability of downward closures [54] and decidability of separability by piecewise testable languages [21]. The diagonal problem for languages of words recognised by recursion schemes is decidable [29, 14, 45].

Over full trios of finite words, the diagonal problem is equivalent to the computability of downward closures [22], which is an important problem in its own right. The *downward closure* of a language $\mathcal{L}$ of finite trees is the set $\mathcal{L}{\downarrow}$ of all trees that can be homeomorphically embedded into some tree in $\mathcal{L}$. By Higman's lemma [31], the embedding relation on finite ranked trees is a well quasi-order. Consequently, the downward closure $\mathcal{L}{\downarrow}$ of an arbitrary set of trees $\mathcal{L}$ is always a regular language. The downward closure of a language offers a nontrivial regular abstraction thereof: even though the actual count of letters is lost, their limit properties are preserved, as well as their order of appearance.

We say that the downward closure is *computable* when a finite automaton for $\mathcal{L}{\downarrow}$ can be effectively constructed (which is not true in general). Downward closures are computable for a wide class of languages of finite words such as those recognised by context-free grammars [20, 41, 3], Petri nets [27], stacked counter automata [55], context-free FIFO rewriting systems and 0L-systems [1], second-order pushdown automata [54], higher-order pushdown automata [29], and (possibly unsafe) recursion schemes over words [14]. Over finite trees, it is known that downward closures are computable for the class of regular tree languages [25]. We are not aware of other such computability results for other classes of languages of finite trees.

In another line of research, B-automata, and among them *alternating B-automata*, have been put forward as a quantitative extension to MSO [15, 18, 52, 39]. They extend alternating automata over infinite trees [26, Chapter 9] by nonnegative integer counters that can be incremented or reset to zero. The extra counters do not constrain the availability of transitions during a run (unlike in other superficially similar models, such as counter machines), but are used in order to define the acceptance condition: an infinite tree is *n-accepted* if n is a bound on the values taken by the counters during an accepting run of the automaton over it.

The *universality problem* consists in deciding whether for every tree there is a bound n for which it is n-accepted. The *boundedness problem* asks whether there exists a bound n for which all trees are n-accepted. These two problems are closely related. Their decidability is an important open problem in the field, and proving the decidability of the boundedness problem would solve the long standing nondeterministic Mostowski index problem [17]. However, though open in general, the boundedness problem is known to be decidable over finite words [15] and trees [18] and infinite words [39], as well as over infinite trees for its weak [53] and the more general quasi-weak [40] version.

Another expressive formalism expressing unboundedness properties beyond MSO is MSO+U, which extends MSO by a new quantifier "$\mathsf{U}X.\varphi$" [7] stating that there exist arbitrarily large finite sets X satisfying φ. This logic is incomparable with B-automata. The model-checking problem of recursion schemes against its weak fragment WMSO+U, where monadic second-order quantifiers are restricted to finite sets, is decidable [46].

Contributions. Our first contribution is the decidability of the model-checking problem of properties expressed by alternating B-automata for an expressive class of recursion schemes called *safe recursion schemes*. As generators of infinite trees, safe recursion schemes are equivalent to higher-order pushdown automata without the collapse operation [34] and are strictly less expressive than general (unsafe) recursion schemes [44, Corollary I.2]. Here, the model-checking problem asks whether a concrete infinite tree (the Böhm tree generated by the safe recursion scheme) is accepted by the B-automaton for some bound. This problem happens to be significantly simpler than the universality/boundedness problem above described. The proof goes by reducing the order of the safe recursion scheme similarly as done in Knapik, Niwiński, and Urzyczyn [34] to show decidability of the MSO model-checking problem, at the expense of making the property automaton two-way. We then rely on the fact that two-way alternating B-automata can be converted to equivalent one-way alternating B-automata [6]. Our result is incomparable with the result of Ong [43], since

(1) alternating B-automata are strictly more expressive than MSO, however

(2) we obtain it under the more restrictive safety assumption.

Whether the safety assumption can be dropped while preserving decidability of the model-checking problem against B-automata properties remains open.

Our second contribution is to define the following generalization of the diagonal problem from words to trees: given a language of finite trees $\mathcal{L}$ and a set of letters Σ, decide whether for every $n \in \mathbb{N}$ there is a tree $T \in \mathcal{L}$ such that every letter from Σ occurs at least n times on every branch of T. This generalization is designed in order to reduce the computation of downward closures to the diagonal problem, in the same fashion as for finite words. Our proof strategy is to represent downward-closed sets of trees $\mathcal{L}\downarrow$ by simple tree regular expressions, which are a subclass of regular expressions for finite trees [24, 25]. By further analysing and simplifying the structure of these expressions, the computation of the downward closure can be reduced to finitely many instances of the diagonal problem. Unlike in the case of finite words, we do not know whether for full trios there exists a converse reduction from the diagonal problem to the problem of computing downward closures.

Our third contribution is decidability of the diagonal problem for languages of finite trees recognised by safe recursion schemes (and thus computability of downward closures of those languages). The diagonal problem can directly be expressed in a logic called *weak cost monadic second order logic* (WCMSO) [53], which extends weak MSO with atomic formulae of the form $|X| \leq N$ stating that the cardinality of the monadic variable X is at most N. Since WCMSO can be translated to alternating B-automata [53], the diagonal problem reduces to the model-checking problem of safe recursion schemes against alternating B-automata, which we have shown decidable in the first part. Note that it seems difficult to express the diagonal problem using alternating B-automata directly, and indeed the fact that alternating B-automata can express all WCMSO properties is nontrivial.

Outline. In Section 2, we define recursion schemes and B-automata. In Section 3, we present our first result, namely decidability of model checking of safe recursion schemes against B-automata. In Section 4, we introduce the diagonal problem, and we show how it can be used to compute downward closures. In Section 5, we solve the diagonal problem for schemes. We conclude in Section 6 with some open problems. A full technical report containing full proofs it also available [4].

2 Preliminaries

Recursion schemes. A *ranked alphabet* is a (usually finite) set $\mathbb{A}$ of letters, together with a function *rank*: $\mathbb{A} \to \mathbb{N}$, assigning a *rank* to every letter. When we define trees below, we require that a node labeled by a letter a has exactly $\mathit{rank}(a)$ children. In the sequel, we usually assume some fixed finite ranked alphabet $\mathbb{A}$. The set of *(simple) types* is constructed from a unique ground type o using a binary operation $\to$; namely o is a type, and if α and β are types, so is $\alpha \to \beta$. By convention, $\to$ associates to the right, that is, $\alpha \to \beta \to \gamma$ is understood as $\alpha \to (\beta \to \gamma)$. A type $\mathsf{o} \to \dots \to \mathsf{o}$ with k occurrences of $\to$ is also written as $\mathsf{o}^k \to o$. The *order* of a type α, denoted $\mathit{ord}(\alpha)$ is defined by induction: $\mathit{ord}(\mathsf{o}) = 0$ and $\mathit{ord}(\alpha_1 \to \dots \to \alpha_k \to \mathsf{o}) = \max_i(\mathit{ord}(\alpha_i)) + 1$ for $k \geqslant 1$.

We coinductively define both *lambda-terms* and the two-argument relation "M is a *lambda-term of type* α" as follows (cf. [32, 5]):

- a letter $a \in \mathbb{A}$ is a lambda-term of type $\mathsf{o}^{\mathit{rank}(a)} \to \mathsf{o}$;
- for every type α there is a countable set $\{x, y, \dots\}$ of *variables of type* α which can be used as lambda-terms of type α;
- if M is a lambda-term of type β and x a variable of type α, then $\lambda x.M$ is a lambda-term of type $\alpha \to \beta$; this construction is called a *lambda-binder*;
- if M is a lambda-term of type $\alpha \to \beta$, and N is a lambda-term of type α, then $M\,N$ is a lambda-term of type β, called an *application*.

As usual, we identify lambda-terms up to *alpha-conversion* (i.e., renaming of bound variables). We use here the standard notions of *free variable*, *subterm*, (capture-avoiding) *substitution*, and *beta-reduction* (see for instance [32, 5]). A *closed* lambda-term does not have free variables. For a lambda-term M of type α, the *order* of M, denoted $\mathit{ord}(M)$, is defined as $\mathit{ord}(\alpha)$. It is *first-order* if its order is one. An *applicative term* is a lambda-term not containing lambda-binders (it contains only letters, applications, and variables).

A lambda-term M is *superficially safe* if all its free variables x have order $\mathit{ord}(x) \geqslant \mathit{ord}(M)$. A lambda-term M is *safe* if it is superficially safe, and if for every subterm of the form $K\,L_1\,\dots\,L_k$, where K is not an application and $k \geqslant 1$, all subterms $K, L_1, \dots, L_k$ are superficially safe.

A *(higher-order, deterministic) recursion scheme* over the alphabet $\mathbb{A}$ is a tuple $\mathcal{G} = \langle \mathbb{A}, \mathcal{N}, X_0, \mathcal{R} \rangle$, where $\mathcal{N}$ is a finite set of typed *nonterminals*, $X_0 \in \mathcal{N}$ is the *initial nonterminal*, and $\mathcal{R}$ is a function assigning to every nonterminal $X \in \mathcal{N}$ of type $\alpha_1 \to \cdots \to \alpha_k \to \mathsf{o}$ a finite lambda-term of the form $\lambda x_1 . \cdots . \lambda x_k . K$, of the same type $\alpha_1 \to \cdots \to \alpha_k \to \mathsf{o}$, in which K is an applicative term with free variables in $\mathcal{N} \uplus \{x_1, \ldots, x_k\}$. We refer to $\mathcal{R}(X)$ as the *rule* for X. The *order* of a recursion scheme $\mathit{ord}(\mathcal{G})$ is the maximum order of its nonterminals.

The lambda-term *represented* by a recursion scheme $\mathcal{G}$ as above, denoted $\Lambda(\mathcal{G})$, is the limit of applying recursively the following operation to X_0: take an occurrence of some nonterminal X, and replace it with $\mathcal{R}(X)$ (the nonterminals should be chosen in a fair way, so that every nonterminal is eventually replaced). Thus, $\Lambda(\mathcal{G})$ is a (usually infinite) regular lambda-term obtained by unfolding the nonterminals of $\mathcal{G}$ according to their definition. We remark that when substituting $\mathcal{R}(X)$ for a nonterminal X there is no need for any renaming of variables (capture-avoiding substitution), since $\mathcal{R}(X)$ does not contain free variables other than nonterminals. We only consider recursion schemes for which $\Lambda(\mathcal{G})$ is well-defined (e.g. by requiring that $\mathcal{R}(X)$ is not a single nonterminal). A recursion scheme $\mathcal{G}$ is *safe* if $\Lambda(\mathcal{G})$ is safe.

A *tree* is a closed applicative term of type o. Such lambda-terms are coinductively of the form $a\, M_1 \cdots M_r$, where $a \in \mathbb{A}$ is of rank r, and where $M_1, \ldots, M_r$ are again trees. Thus, such a lambda-term can be identified with a tree understood in the traditional sense: a is the label of its root, and $M_1, \ldots, M_r$ describe subtrees attached in the r children of the root, from left to right. For trees we also use the notation $a(M_1, \ldots, M_r)$ instead of $a\, M_1 \ldots M_r$. A tree is *regular* if it has finitely many subtrees (subterms) up to isomorphism.

The *Böhm tree* of a lambda-term M of type o, denoted $BT(M)$, is defined coinductively as follows: if there is a sequence of beta-reductions from M to a lambda-term of the form $a\, M_1 \ldots M_r$ (where $a \in \mathbb{A}$ is a letter), then $BT(M) = a(BT(M_1), \ldots, BT(M_r))$; otherwise $BT(M) = \bot()$, where $\bot \in \mathbb{A}$ is a distinguished letter of rank 0. It is a classical result that $BT(M)$ exists, and is uniquely defined [32, 5]. Clearly, $BT(M)$ is indeed a tree. The tree *generated* by a recursion scheme $\mathcal{G}$, denoted $BT(\mathcal{G})$, is $BT(\Lambda(\mathcal{G}))$.

A lambda-term N is *normalizing* if $BT(N)$ does not contain the special letter $\bot$; a recursion scheme $\mathcal{G}$ is *normalizing* if $\Lambda(\mathcal{G})$ is normalizing. In other words, in a normalizing recursion scheme/lambda-term beta-reduction always produces a letter. It is possible to transform every recursion scheme $\mathcal{G}$ into a normalizing recursion scheme $\mathcal{G}'$ generating the same tree as $\mathcal{G}$, up to renaming $\bot$ into some non-special letter $\bot'$ (cf. [28, Section 5]). Moreover, the construction preserves safety and the order.

Recursion schemes as recognizers of languages of finite trees. The standard semantics of a recursion scheme $\mathcal{G} = \langle \mathbb{A}, \mathcal{N}, X_0, \mathcal{R} \rangle$ is the single infinite tree $BT(\mathcal{G})$ generated by the scheme. An alternative view is to consider a recursion scheme as a recognizer of a language of finite trees $\mathcal{L}(\mathcal{G})$. This alternative view is relevant when discussing downward closures of languages of finite trees. We employ a special letter $\mathsf{nd} \in \mathbb{A}$ of rank 2 in order to represent $\mathcal{L}(\mathcal{G})$ by resolving the nondeterministic choice of nd in the infinite tree $BT(\mathcal{G})$ in all possible ways. Formally, for two trees T, U, we write $T \to_{\mathsf{nd}} U$ if U is obtained from T by choosing an nd-labeled node u of T and a child v thereof, and replacing the subtree rooted at u with the subtree rooted at v. The relation $\to_{\mathsf{nd}}^*$ is the reflexive and transitive closure of $\to_{\mathsf{nd}}$. We define the language of finite trees *recognized* by $\mathcal{G}$ as $\mathcal{L}(\mathcal{G}) = \mathcal{L}(BT(\mathcal{G}))$, where

$$\mathcal{L}(T) = \{U \mid T \to_{\mathsf{nd}}^* U, \text{ with } U \text{ finite and not containing “}\mathsf{nd}\text{” or “}\bot\text{”}\}\,.$$

For an illustration of this encoding, and simultaneously for an example of a recursion scheme, consider the ranked alphabet $\mathbb{A}$ containing a letter a of rank 2, two letters b_1, b_2 of rank 1, and a letter c of rank 0. We use an initial nonterminal S of order-0 type o, and an additional nonterminal A of order-2 type $(\mathsf{o} \to \mathsf{o}) \to (\mathsf{o} \to \mathsf{o}) \to \mathsf{o} \to \mathsf{o} \to \mathsf{o}$, together with the following two rules:

$$\mathcal{R}(S) = A\, b_1\, b_2\, c\, c,$$
$$\mathcal{R}(A) = \lambda f.\lambda g.\lambda x.\lambda y.\mathsf{nd}\,(a\, x\, y)\,(A\, f\, g\,(f\, x)\,(g\, x)).$$

Then, $BT(\mathcal{G})$ is the infinite non-regular tree $\mathsf{nd}\,(a\, c\, c)\,(\mathsf{nd}\,(a\,(b_1\, c)\,(b_2\, c))\,(\cdots))$, and $\mathcal{L}(\mathcal{G})$ is the non-regular language of all finite trees of the form $a\,(b_1^n\, c)\,(b_2^n\, c)$ with $n \in \mathbb{N}$.

Alternating B-automata. We introduce the model of automata used in this paper, namely *alternating one-way/two-way B-automata* over trees (over a ranked alphabet). We consider counters which can be *incremented* $\mathtt{i}$, *reset* $\mathtt{r}$, or left *unchanged* ε. Let Γ be a finite set of *counters* and let $\mathbb{C} = \{\mathtt{i}, \mathtt{r}, \varepsilon\}$ be the *alphabet of counter actions*. Each counter starts with value zero, and the *value of a sequence* of actions is the supremum of the values achieved during this sequence. For instance $\mathtt{iir}\varepsilon\mathtt{i}\varepsilon$ has value 2, $(\mathtt{ir})^\omega$ has value 1, and $\mathtt{iri}^2\mathtt{ri}^3\mathtt{r}\cdots$ has value ∞. For an infinite sequence of counter actions $w \in \mathbb{C}^\omega$, let $val(w) \in \mathbb{N} \cup \{\infty\}$ be its *value*. In case of several counters, $w = c_1 c_2 \cdots \in (\mathbb{C}^\Gamma)^\omega$, we take the counter with the maximal value: $val(w) = \sup_{c \in \Gamma} val(w(c))$, where $w(c) = c_1(c) c_2(c) \cdots$.

An *(alternating, two-way) B-automaton* over a finite ranked alphabet $\mathbb{A}$ is a tuple $\langle \mathbb{A}, Q, q_0, pr, \Gamma, \delta \rangle$ consisting of a finite set of *states* Q, an *initial state* $q_0 \in Q$, a function $pr \colon Q \to \mathbb{N}$ assigning *priorities* to states, a finite set Γ of counters, and a *transition function*

$$\delta : Q \times \mathbb{A} \to \mathcal{B}^+(\{\uparrow, \circlearrowleft, \downarrow_1, \downarrow_2, \ldots\} \times \mathbb{C}^\Gamma \times Q)$$

mapping a state and a letter a to a (finite) positive Boolean combination of triples of the form (d, c, q); it is assumed that if $d = \downarrow_i$ then $i \leqslant rank(a)$. Such a triple encodes the instruction to send the automaton to state q in direction d while performing action c. The direction $\downarrow_i$ moves to the i-th child, $\uparrow$ moves to the parent, and $\circlearrowleft$ stays in place. We assume that $\delta(q, a)$ is written in disjunctive normal form for all q and a.

The acceptance of an infinite input tree T by an alternating B-automaton $\mathcal{A}$ is defined in terms of a game $(\mathcal{A}, T)$ between two players, called Eve and Adam. Eve is in charge of disjunctive choices and tries to minimize the counter values while satisfying the parity condition. Adam, on the other hand, is in charge of conjunctive choices and tries to either maximize counter values, or to sabotage the parity condition. Since the transition function is given in disjunctive normal form, each turn of the game consists of Eve choosing a disjunct and Adam selecting a single tuple (d, c, q) thereof. In order to guarantee that from every position there is some move, we assume that each disjunction is nonempty and that each disjunct contains a tuple with some direction other than $\uparrow$. A *play* of $\mathcal{A}$ on the tree T is a sequence $q_0, (d_1, c_1, q_1), (d_2, c_2, q_2), \ldots$ compatible with T and δ: q_0 is the initial state, and for all $i \in \mathbb{N}$, $(d_{i+1}, c_{i+1}, q_{i+1})$ appears in $\delta(q_i, T(x_i))$ where x_i is the node of T after following the directions $d_1 d_2 \ldots d_i$ starting from the root. The value $val(\pi)$ of a play π is the value $val(c_1 c_2 \cdots)$ as defined above if the largest number appearing infinitely often among the priorities $pr(q_0), pr(q_1), \ldots$ is even; otherwise, $val(\pi) = \infty$. We say that the play π is *n-winning* (for Eve) if $val(\pi) \leqslant n$.

A *strategy* for one of the players in the game $(\mathcal{A}, T)$ is a function that returns the next choice given the history of the play. Note that choosing a strategy for Eve and a strategy for Adam fixes a play in $(\mathcal{A}, T)$. We say that a play π is *compatible* with a strategy σ if there is some strategy σ' for the other player such that σ and σ' together yield the play π. A strategy for Eve is *n-winning* if every play compatible with it is n-winning. We say that Eve *n-wins* the game if there is some n-winning strategy for Eve. A B-automaton *n-accepts* a tree T if Eve n-wins the game $(\mathcal{A}, T)$; it *accepts* T if it n-accepts T for some $n \in \mathbb{N}$. The language *recognized* by $\mathcal{A}$ is the set of all trees accepted by $\mathcal{A}$.

If no $\delta(q, a)$ uses the direction $\uparrow$, then we call $\mathcal{A}$ *one-way*. The following theorem essentially follows from a result of Blumensath, Colcombet, Kuperberg, Parys, and Vanden Boom [6, Theorem 6] modulo some cosmetic changes (c.f. [4, Appendix A] for more details).

▶ **Theorem 2.1** (c.f. [6, Theorem 6]). *Given an alternating two-way B-automaton, one can compute an alternating one-way B-automaton that recognizes the same language.*

As a special case of a result by Colcombet and Göller [16] we obtain the following fact.

▶ **Fact 2.2.** *One can decide whether a given B-automaton accepts a given regular tree.*

3 Model-checking safe recursion schemes against alternating B-automata

In this section we prove the first main theorem of our paper, the decidability of the *model-checking problem* of safe recursion schemes against properties described by B-automata:

▶ **Theorem 3.1.** *Given an alternating B-automaton $\mathcal{A}$ and a safe recursion scheme $\mathcal{G}$, one can decide whether $\mathcal{A}$ accepts the tree generated by $\mathcal{G}$.*

It is worth noticing that this theorem generalises the result of Knapik et al. [34] on safe recursion schemes from regular (MSO) properties to the more general quantitative realm of properties described by B-automata. On the other hand, our result is incomparable with the celebrated theorem of Ong [43] showing decidability of model checking regular properties of possibly unsafe recursion schemes. Whether model checking of possibly unsafe recursion schemes against properties described by B-automata is decidable remains an open problem.

By Theorem 2.1, every B-automaton can be effectively transformed into an equivalent one-way B-automaton, so it is enough to prove Theorem 3.1 for a one-way B-automaton $\mathcal{A}$. The proof of Theorem 3.1 is based on the following lemma, where we use in an essential way the assumption that the recursion scheme is safe.

▶ **Lemma 3.2.** *For every safe recursion scheme $\mathcal{G}$ of order m and for every alternating one-way B-automaton $\mathcal{A}$, one can effectively construct a safe recursion scheme $\mathcal{G}'$ of order $m - 1$ and an alternating two-way B-automaton $\mathcal{A}'$ such that*

$$\mathcal{A} \text{ accepts } BT(\mathcal{G}) \quad \text{if and only if} \quad \mathcal{A}' \text{ accepts } BT(\mathcal{G}').$$

Theorem 3.1 follows easily: Using Lemma 3.2 we can reduce the order of the considered safe recursion scheme by one. We obtain a two-way B-automaton, which we convert back to a one-way B-automaton using Theorem 2.1. It is then sufficient to repeat this process, until we end up with a recursion scheme of order 0. A recursion scheme of order 0 generates a regular tree and, by Fact 2.2, we can decide whether the resulting B-automaton accepts this tree, answering our original question.

Lambda-trees. We now come to the proof of Lemma 3.2. The construction of $\mathcal{G}'$ from of $\mathcal{G}$ follows an analogous result for MSO [33, 34], which we generalise to B-automata. We represent some lambda-terms as trees. For a *finite* set $\mathcal{X}$ of variables of type o, we define a new ranked alphabet $\mathbb{A}_{\mathcal{X}}$ that contains

1) a letter $\overline{a}$ of rank 0 for every letter $a \in \mathbb{A}$;
2) a letter $\overline{x}$ of rank 0 for every variable $x \in \mathcal{X}$;
3) a letter $\overline{\lambda x}$ of rank 1 for every variable $x \in \mathcal{X}$;
4) a letter @ of rank 2.

We remark that $\mathbb{A}_{\mathcal{X}}$ is a usual finite ranked alphabet. A *lambda-tree* is a tree over the alphabet $\mathbb{A}_{\mathcal{X}}$, where $\mathcal{X}$ is clear from the context. Intuitively, a lambda-tree is a tree representation of a first-order lambda-term.

The semantics $[\![T]\!]_{\mathcal{X},s}$ of a lambda-tree T is defined in such a way that if T "corresponds" to the lambda-term M, then $[\![T]\!]_{\mathcal{X},s} = BT(M)$. Since T uses only variables of type o we can read the resulting Böhm tree directly, without performing any reduction. Essentially, we walk down through T, skipping all lambda-binders and choosing the left branch in all applications. Whenever we reach some variable x, we go up to the corresponding lambda-binder, then up to the corresponding application, and then we again start going down in the argument of this application. Formally, let $\mathcal{X}$ be a finite set of variables of type o, and let $s \in \mathbb{N}$. The intended meaning is that $\mathcal{X}$ contains variables that may potentially appear in the considered lambda-tree T, and that s is a bound for the arity of types in the lambda-term represented by T (types of all its subterms should be of the form $\mathsf{o}^k \to \mathsf{o}$ for $k \leq s$). We take $\mathit{Dirs}_{\mathcal{X},s} = \{\downarrow\} \cup \{\uparrow_x \mid x \in \mathcal{X}\} \cup \{\uparrow_i \mid 1 \leq i \leq s\}$. Intuitively, $\downarrow$ means to go down to the left child, $\uparrow_x$ means that we are looking for the value of (lambda)variable x, and $\uparrow_i$ means that we are looking for the i-th argument of an application. For a node v of T denote its parent by $\mathit{par}(v)$, and its i-th child by $\mathit{ch}_i(v)$. For $d \in \mathit{Dirs}_{\mathcal{X},s}$, and for a node v of T labeled by $a \in \mathbb{A}$, we define the $(\mathcal{X}, s)$*-successor* of (d, v) as

1. $(\downarrow, \mathit{ch}_1(v))$ if $d = \downarrow$ and $a = \overline{\lambda x}$ (for some x) or @,
2. $(\uparrow_x, v)$ if $d = \downarrow$ and $a = \overline{x}$ (for some x),
3. $(\uparrow_x, \mathit{par}(v))$ if $d = \uparrow_x$ and $a \neq \overline{\lambda x}$ (including the case when $a = \overline{\lambda y}$ for $y \neq x$),
4. $(\uparrow_1, \mathit{par}(v))$ if $d = \uparrow_x$ and $a = \overline{\lambda x}$,
5. $(\uparrow_{i+1}, \mathit{par}(v))$ if $d = \uparrow_i$ for $i < s$ and $a = \overline{\lambda y}$ (for some y),
6. $(\uparrow_{i-1}, \mathit{par}(v))$ if $d = \uparrow_i$ for $i > 1$ and $a =$ @,
7. $(\downarrow, \mathit{ch}_2(v))$ if $d = \uparrow_1$ and $a =$ @.

Rule 1 allows us go to down to the first child in the case of lambda-binders and applications. Rule 2 records that we have seen x, and thus we need to find its value by going up. Rule 3 climbs the tree upwards as long as we do not see the corresponding binder $\overline{\lambda x}$. Rule 4 records that we have seen $\overline{\lambda x}$ and initialises its level to 1. We now need to find the corresponding application. Rule 5 increments the level and goes up when we encounter a binder $\overline{\lambda y}$, and Rule 6 decrements it for applications @. Finally, when we see an application at level 1 we apply Rule 7 which searches for the value of x in the right child. An $(\mathcal{X}, s)$*-maximal path* from (d_1, v_1) is a sequence of pairs $(d_1, v_1), (d_2, v_2), \ldots$ in which every (d_{i+1}, v_{i+1}) is the $(\mathcal{X}, s)$-successor of (d_i, v_i), and which is either infinite or ends in a pair that has no $(\mathcal{X}, s)$-successor. For $d \in \mathit{Dirs}_{\mathcal{X},s}$, and for a node v of T, we define the $(\mathcal{X}, s)$*-derived tree* from (T, d, v), denoted by $[\![T, d, v]\!]_{\mathcal{X},s}$, by coinduction:

- if the $(\mathcal{X}, s)$-maximal path from (d, v) is finite and ends in $(\downarrow, w)$ for a node w labeled by $\overline{a}$, then $[\![T, d, v]\!]_{\mathcal{X},s} = a([\![T, \uparrow_1, w]\!]_{\mathcal{X},s}, \ldots, [\![T, \uparrow_{\mathit{rank}(a)}, w]\!]_{\mathcal{X},s})$;
- otherwise, $[\![T, d, v]\!]_{\mathcal{X},s} = \bot$.

The $(\mathcal{X}, s)$*-derived tree* from T is $[\![T]\!]_{\mathcal{X},s} = [\![T, \downarrow, v_0]\!]_{\mathcal{X},s}$, where v_0 is the root of T. We say that T is *normalizing* if $[\![T]\!]_{\mathcal{X},s}$ does not contain $\bot$.

The following lemma performs the order reduction. It crucially relies on the safety assumption. It is a variant of results proved in Knapik et al. [33, 34] (c.f. [4, Appendix C] for more details). Intuitively, it says that a lambda-tree representation T of a safe recursion scheme $\mathcal{G}$ of order m can be computed by a safe recursion scheme of order $m - 1$ in a semantic-preserving way.

▶ **Lemma 3.3** ([33, 34])**.** *For every safe recursion scheme $\mathcal{G}$ of order $m \geq 1$ one can construct a safe recursion scheme $\mathcal{G}'$ of order $m-1$, a finite set of variables $\mathcal{X}$, and a number $s \in \mathbb{N}$ such that*

$$[\![BT(\mathcal{G}')]\!]_{\mathcal{X},s} = BT(\mathcal{G}).$$

▶ Remark 3.4. In Knapik et al. [33, 34] the lambda-tree T is denoted $\daleth_{\mathcal{G}}(X_0)$ (where X_0 is the starting nonterminal of $\mathcal{G}$), and the recursion scheme $\mathcal{G}'$ is denoted $\mathcal{G}^\alpha$. The set $\mathcal{X}$ is just the set of variables appearing in the letters used in $\mathcal{G}^\alpha$; the number s can be also read out of $\mathcal{G}^\alpha$. They prove that the $(\mathcal{X}, s)$-derived tree of $\daleth_{\mathcal{G}}(X_0)$ equals the tree generated by $\mathcal{G}$ [33, Proposition 4], that $\mathcal{G}^\alpha$ is safe [34, Lemma 3.5], and that $\mathcal{G}^\alpha$ generates $\daleth_{\mathcal{G}}(X_0)$.

In order to prove Lemma 3.3, one needs to replace in $\mathcal{G}$ every variable x of type o by $\overline{x}$, every lambda-binder concerning such a variable by $\overline{\lambda x}$, and every application with an argument of type o by a construct creating a @-labeled node. Types of subterms change and the order of the recursion scheme decreases by one. Notice, however, that while computing $BT(\Lambda(\mathcal{G}))$ we may need to rename variables during capture-avoiding substitutions, while in the tree generated by the modified recursion scheme we leave original variable names. In general (i.e., when the transformation is applied to an arbitrary recursion scheme) this causes a problem of overlapping variable names. The assumption that $\mathcal{G}$ is safe is crucial here and there is no need to rename variables when applying the transformation to a safe recursion scheme.

Having Lemma 3.3, it remains to transform a one-way B-automaton $\mathcal{A}$ operating on the tree generated by $\mathcal{G}$ into a two-way B-automaton $\mathcal{A}'$ operating on the lambda-tree generated by $\mathcal{G}'$, as described by the following lemma (as mentioned on page 5, we can assume that $\mathcal{G}$ is normalizing, which implies that $BT(\mathcal{G}')$ is normalizing: the tree $[\![BT(\mathcal{G}')]\!]_{\mathcal{X},s} = BT(\mathcal{G})$ does not contain $\perp$).

▶ **Lemma 3.5.** *Let $\mathcal{A}$ be an alternating one-way B-automaton over a finite alphabet $\mathbb{A}$, let $\mathcal{X}$ be a finite set of variables, and let $s \in \mathbb{N}$. One can construct an alternating two-way B-automaton $\mathcal{A}'$ such that for every normalizing lambda-tree T over $\mathbb{A}_{\mathcal{X}}$,*

$$\mathcal{A} \text{ accepts } [\![T]\!]_{\mathcal{X},s} \quad \text{if and only if} \quad \mathcal{A}' \text{ accepts } T.$$

Proof. The B-automaton $\mathcal{A}'$ simulates $\mathcal{A}$ on the lambda-tree. Whenever $\mathcal{A}$ wants to go down to the i-th child, $\mathcal{A}'$ has to follow the $(\mathcal{X}, s)$-maximal path from $(\uparrow_i, v)$ (where v is the current node). To this end, it has to remember the current pair (d, v), and repeatedly find its $(\mathcal{X}, s)$-successor. Here v is always just the current node visited by the B-automaton; the d component comes from the (finite) set $\mathit{Dirs}_{\mathcal{X},s}$, and thus it can be remembered in the state. It is straightforward to encode the definition of an $(\mathcal{X}, s)$-successor in transitions of an automaton. We do not have to worry about infinite $(\mathcal{X}, s)$-maximal paths, because by assumption the $(\mathcal{X}, s)$-derived tree does not contain $\perp$-labeled nodes. ◀

4 Downward closures of tree languages

In this section we lay down a method for the computation of the downward closure for classes of languages of finite trees closed under linear FTT transductions. This method is analogous to the one of Zetzsche [54] for the case of finite words. In Section 4.1 we define the downward closure of languages of finite ranked trees with respect to the embedding well-quasi order and in Section 4.2 we define the simultaneous unboundedness problem for trees and show how computing the downward closure reduces to it. In Section 4.3 we define the diagonal problem for finite trees and show how the previous problem reduces to it. We will then solve the diagonal problem for languages of finite trees recognized by safe recursion schemes in Section 5.

Let us emphasize that this section can be applied to any class of languages of finite trees closed under linear FTT transductions, not just those recognized by safe recursion schemes.

4.1 Preliminaries

Given two finite trees $S = a(S_1, \dots, S_k)$ and $T = b(T_1, \dots, T_r)$, we say that S *homeomorphically embeds into* T, written $S \sqsubseteq T$, if, either

1) there exists $i \in \{1, \dots, r\}$ such that $S \sqsubseteq T_i$, or

2) $a = b$, $k = r$, and $S_i \sqsubseteq T_i$ for all $i \in \{1, \dots, r\}$.

For a language of finite trees $\mathcal{L}$, its *downward closure*, denoted by $\mathcal{L}{\downarrow}$, is the set of trees S such that $S \sqsubseteq T$ for some tree $T \in \mathcal{L}$.

Pure products. Goubault-Larrecq and Schmitz [25] describe downward-closed sets of trees using so-called *simple tree regular expressions*. Among those expressions they distinguish *products*, which describe *ideals* of trees. Because every downward-closed set of trees is a finite union of ideals, such a set can be described by a finite list of products. Since their definition of a product is rather indirect, we consider the stronger notion of *pure products*.

A *context* is a tree possibly containing one or more occurrences of a special leaf $\square$, called a *hole*. Given a context C and a set of trees $\mathcal{L}$, we write $C[\mathcal{L}]$ for the set of trees obtained from C by replacing every occurrence of the hole $\square$ by some tree from $\mathcal{L}$. Different occurrences of $\square$ are replaced by possibly different trees from $\mathcal{L}$. The definition readily extends to a set of contexts $\mathcal{C}$, by writing $\mathcal{C}[\mathcal{L}]$ for $\bigcup_{C \in \mathcal{C}} C[\mathcal{L}]$. If C does not have any $\square$, then $C[\mathcal{L}]$ is just $\{C\}$. A *pure product* is defined according to the following abstract syntax:

$$P ::= a^?(P, \dots, P) \mid I^*.P, \qquad C ::= a(P_\square, \dots, P_\square),$$
$$I ::= C + \dots + C, \qquad P_\square ::= \square \mid P,$$

where the sum of contexts is nonempty, and where in a context $C = a(P_{\square,1}, \dots, P_{\square,r})$ it is required that at least one $P_{\square,i}$ is a hole $\square$. A pure product P denotes a set of trees $[\![P]\!]$ downward-closed for $\sqsubseteq$, which is defined recursively as follows:

$$\begin{aligned}
[\![a^?(P_1, \dots, P_r)]\!] &= \{a(T_1, \dots, T_r) \mid \forall i \,.\, T_i \in [\![P_i]\!]\} \cup [\![P_1]\!] \cup \dots \cup [\![P_r]\!],\\
[\![I^*.P]\!] &= \bigcup_{n \in \mathbb{N}} \underbrace{[\![I]\!][\dots[[\![I]\!]}_{n}[\![P]\!]]]\dots],\\
[\![C_1 + \dots + C_k]\!] &= [\![C_1]\!] \cup \dots \cup [\![C_k]\!],\\
[\![a(P_{\square,1}, \dots, P_{\square,r})]\!] &= \{a(T_1, \dots, T_r) \mid \forall i \,.\, T_i \in [\![P_{\square,i}]\!]\} \cup [\![P_{\square,1}]\!] \cup \dots \cup [\![P_{\square,r}]\!],\\
[\![\square]\!] &= \{\square\}\,.
\end{aligned}$$

For example, $[\![(a(b(), \square))^*.c^?()]\!]$ is the set of trees of the form either $b()$, or $c()$, or $a(b(), a(b(), \dots a(b(), x) \dots))$ with x either $b()$ or $c()$. Based on the results of Goubault-Larrecq and Schmitz [25] it is not difficult to deduce the following lemma (see [4, Appendix D] for a proof).

▶ **Lemma 4.1.** *Every set of trees $\mathcal{L}$ downward-closed for $\sqsubseteq$ can be represented as $\mathcal{L} = [\![P_1]\!] \cup \dots \cup [\![P_k]\!]$, in which $P_1, \dots, P_k$ are pure products.*

This decomposition result strengthens the results of Goubault-Larrecq and Schmitz [25] by showing that pure products (instead of just products) suffice in order to represent downward-closed sets of trees.

Transductions. A (nondeterministic) *finite tree transducer* (*FTT*) is a tuple $\mathcal{A} = (\mathbb{A}_{in}, \mathbb{A}_{out}, S, p^I, \delta)$, where $\mathbb{A}_{in}, \mathbb{A}_{out}$ are the input and output alphabets (finite, ranked), S is a finite set of *control states*, $p^I \in S$ is an *initial state*, and δ is a finite set of *transition rules* of the form either $(p, a(\mathsf{x}_1, \ldots, \mathsf{x}_r)) \to T$ or $(p, \mathsf{x}) \to T$, where $p \in S$ is a control state, $a \in \mathbb{A}_{in}$ is a letter of rank r, and T is a finite tree over the alphabet $\mathbb{A}_{out} \cup (S \times \{\mathsf{x}_1, \ldots, \mathsf{x}_r\})$ or $\mathbb{A}_{out} \cup (S \times \{\mathsf{x}\})$, respectively. The rank of all the pairs from $S \times \{\mathsf{x}_1, \ldots, \mathsf{x}_r\}$ or $S \times \{\mathsf{x}\}$ is 0. An FTT is *linear* if for each rule of the form $(p, a(\mathsf{x}_1, \ldots, \mathsf{x}_r)) \to T$ and for each $i \in \{1, \ldots, r\}$, in T there is at most one letter from $S \times \{\mathsf{x}_i\}$, and moreover for each rule of the form $(p, \mathsf{x}) \to T$, in T there is at most one letter from $S \times \{\mathsf{x}\}$. An FTT $\mathcal{A}$ defines in a natural way a relation between finite trees, also denoted $\mathcal{A}$ (c.f. Comon et al. [19]). For a language $\mathcal{L}$ we write $\mathcal{A}(\mathcal{L})$ for the set of trees U such that $(T, U) \in \mathcal{A}$ for some $T \in \mathcal{L}$. A function that maps $\mathcal{L}$ to $\mathcal{A}(\mathcal{L})$ for some linear FTT $\mathcal{A}$ is called a *linear FTT transduction.*

▶ **Fact 4.2.** *The downward closure operation $\mathcal{L} \mapsto \mathcal{L}{\downarrow}$ and the regular restriction operation $\mathcal{L} \mapsto \mathcal{L} \cap \mathcal{R}$ (for every regular language $\mathcal{R}$) are effectively linear FTT transductions.*

▶ **Lemma 4.3** (c.f. [4, Appendix E]). *The class of languages of finite trees recognized by safe recursion schemes is effectively closed under linear FTT transductions.*

4.2 The simultaneous unboundedness problem for trees

We say that a pure product P is *diversified*, if no letter appears in P more than once. The *simultaneous unboundedness problem* (*SUP*) for a class $\mathfrak{C}$ of finite trees asks, given a diversified pure product P and a language $\mathcal{L} \in \mathfrak{C}$ such that $\mathcal{L} \subseteq [\![P]\!]$, whether $[\![P]\!] \subseteq \mathcal{L}{\downarrow}$.

▶ Remark 4.4. This is a generalization of SUP over finite words. In the latter problem, one is given a language of finite words $\mathcal{L}$ such that $\mathcal{L} \subseteq a_1^* \ldots a_k^*$, and must check whether $a_1^* \ldots a_k^* \subseteq \mathcal{L}{\downarrow}$. A word in $a_1^* \ldots a_k^*$ can be represented as a linear tree by interpreting $a_1, \ldots, a_k$ as unary letters and by appending a new leaf e at the end. Thus $a_1^* \ldots a_k^*$ can be represented as the language of the diversified pure product $(a_1(\Box))^*.(a_2(\Box))^*. \cdots .(a_k(\Box))^*.e^?()$.

Following Zetzsche [54], we can reduce the computation of the downward closure to SUP.

▶ **Theorem 4.5** (c.f. [4, Appendix F]). *Let $\mathfrak{C}$ be a class of languages of finite trees closed under linear FTT transductions. One can compute a finite tree automaton recognizing the downward closure of a given language from $\mathfrak{C}$ if and only if SUP is decidable for $\mathfrak{C}$.*

▶ Remark 4.6. Pure products for trees correspond to expressions of the form $a_0^? A_1^* a_1^? \ldots A_k^* a_k^?$ for words (where A_i are sets of letters). In SUP for words simpler expressions of the form $b_1^* \ldots b_k^*$ suffice. This is not possible for trees:

1) expressions of the form $a^?(\cdot, \cdot)$ cannot be removed since they are responsible for branching, and
2) reducing the two contexts in $(a(P_1, \Box) + b(P_2, \Box))^*.P_3$ to a single one would require changing trees of the form $a(T_1, b(T_2, T_3))$ into trees of the form $c(T_1, T_2, T_3)$, which is not a linear FTT transduction.

4.3 The diagonal problem for trees

In SUP for words, instead of checking whether $a_1^* \dots a_k^* \subseteq \mathcal{L}{\downarrow}$, one can equivalently check whether, for each $n \in \mathbb{N}$, there is a word $a_1^{x_1} \dots a_k^{x_k} \in \mathcal{L}$ such that $x_1, \dots, x_k \geq n$. The latter problem is known as the *diagonal problem* for words. In this section, we define an analogous diagonal problem for trees, and we show how to reduce SUP to it.

Given a set of letters Σ, we say that a language of finite trees $\mathcal{L}$ is Σ-*diagonal* if, for every $n \in \mathbb{N}$, there is a tree $T \in \mathcal{L}$ such that for every letter $a \in \Sigma$ and every branch B of T there are at least n occurrences a in B. The *diagonal problem* for a class $\mathfrak{C}$ of finite trees asks, given a language $\mathcal{L} \in \mathfrak{C}$ and a set of letters Σ, whether $\mathcal{L}$ is Σ-diagonal.

Versatile trees. Contrary to the case of words, the presence of sums in our expressions creates some complications in reducing from SUP to the diagonal problem. We deal with these sums by introducing the notion of *versatile trees*. Intuitively, in order to obtain a versatile tree of a pure product P, for every sum $I = C_1 + \dots + C_k$ in P we fix some order of the contexts $C_1, \dots, C_k$, and we allow the contexts to be appended in this order. Formally, the set $(\!|P|\!)$ of versatile trees of a pure product P is defined by structural induction on P:

$$\begin{aligned}
(\!|I^*.P|\!) &= \bigcup_{n\in\mathbb{N}} (\!|I|\!)[\underbrace{((\!|I|\!) \cup \{\square\})[\dots[((\!|I|\!) \cup \{\square\})[}_{n}(\!|P|\!)]]\dots]],\\
(\!|a^?(P_1, \dots, P_r)|\!) &= (\!|a(P_1, \dots, P_r)|\!),\\
(\!|C_1 + \dots + C_k|\!) &= (\!|C_1|\!)[\dots[(\!|C_k|\!)]\dots],\\
(\!|a(P_{\square,1}, \dots, P_{\square,r})|\!) &= \{a(T_1, \dots, T_r) \mid \forall i\,.\,T_i \in (\!|P_{\square,i}|\!)\}\,,\\
(\!|\square|\!) &= \{\square\}\,.
\end{aligned}$$

For example, if $I = a(S_1, \square, \square) + b(\square, S_2)$, then $(\!|I|\!) = \{a(S_1, b(\square, S_2), b(\square, S_2))\}$. Notice that all trees in $(\!|P|\!)$ have the same root's label; denote this label by *root*(P).

From SUP to the diagonal problem. Assuming that P is diversified, for a number $n \in \mathbb{N}$ we say that a tree T is *n-large with respect to P* if, for every subexpression of P of the form $I^*.P'$, above every occurrence of *root*(P') in T there are at least n ancestors labeled by *root*$(I^*.P')$. In other words, for $T \in (\!|P|\!)$ this means that in T every context appearing in P was appended at least n times, on all branches where it was possible to append it. Clearly $(\!|P|\!) \subseteq [\![P]\!]$. On the other hand, every tree from $[\![P]\!]$ can be embedded into every large enough versatile tree. We thus obtain the following lemma.

▶ **Lemma 4.7.** *For every diversified pure product P, and for every sequence of trees $T_1, T_2, \dots \in (\!|P|\!)$ such that every T_n is n-large, $\{T_n \mid n \in \mathbb{N}\}{\downarrow} = [\![P]\!]$.*

Using versatile trees we can reduce from SUP to the diagonal problem.

▶ **Lemma 4.8** (c.f. [4, Appendix G]). *Let $\mathfrak{C}$ be a class of languages of finite trees closed under linear FTT transductions. SUP for $\mathfrak{C}$ reduces to the diagonal problem for $\mathfrak{C}$.*

▶ Remark 4.9. Another formulation of the diagonal problem for languages of finite trees [29, 14, 45] requires that, for every $n \in \mathbb{N}$, there is a tree $T \in \mathcal{L}$ containing at least n occurrences of every letter $a \in \Sigma$ (not necessarily on the same branch, unlike in our case). Such a formulation of the diagonal problem seems too weak to compute downward closures for languages of finite trees.

5 Languages of safe recursion schemes

In the previous section, we have developed a general machinery allowing one to compute downward closures for classes of languages of finite trees closed under linear FTT transductions. In this section, we apply this machinery to the particular case of languages recognized by safe recursion schemes. The following is the main theorem of this section.

▶ **Theorem 5.1.** *Finite tree automata recognizing downward closures of languages of finite trees recognised by safe recursion schemes are computable.*

In order to prove the theorem we need to recall a formalism necessary to express the diagonal problem in logic.

Cost logics. *Cost monadic logic* (*CMSO*) was introduced in Colcombet [15] as a quantitative extension of monadic second-order logic. As usual, the logic can be defined over any relational structure, but we restrict our attention to CMSO over trees. In addition to *first-order variables* ranging over nodes of the tree and *monadic second-order variables* (also called *set variables*) ranging over sets of nodes, CMSO uses a single additional variable N, called the *numeric variable*, which ranges over $\mathbb{N}$. The atomic formulas in CMSO are those from *MSO* (the membership relation $x \in X$ and relations $a(x, x_1, \dots, x_r)$ asserting that $a \in \mathbb{A}$ of rank r is the label at node x with children $x_1, \dots, x_r$ from left to right), as well as a new predicate $|X| \leqslant N$, where X is any set variable and N is the numeric variable. Arbitrary CMSO formulas are built inductively by applying Boolean connectives and by quantifying (existentially or universally) over first-order or set variables. We require that any predicates of the form $|X| \leqslant N$ appear positively in the formula (i.e., within the scope of an even number of negations). We regard N as a parameter. As usual, a *sentence* is a formula without first-order or monadic *free variables*; however, the parameter N is allowed to occur in a sentence. If we fix a value $n \in \mathbb{N}$ for N, the semantics of $|X| \leqslant N$ is what one would expect: the predicate holds when X has cardinality at most n. We say that a sentence φ *n-accepts* a tree T if it holds in T when n is used as value of N; it *accepts* T if it n-accepts T for some $n \in \mathbb{N}$. The language *defined* by φ is the set of all trees (over a fixed alphabet $\mathbb{A}$) accepted by φ.

Weak cost monadic logic (*WCMSO* for short) is the variant of CMSO where the second-order quantification is restricted to finite sets. Vanden Boom [53, Theorem 2] proves that WCMSO is effectively equivalent to a subclass of alternating B-automata, called *weak B-automata*. Thanks to Theorem 3.1, we obtain the following corollary.

▶ **Corollary 5.2.** *The model-checking problem of safe recursion schemes against WCMSO properties is decidable.*

▶ Remark 5.3. The same holds for a more expressive logic called *quasi-weak cost monadic logic* (*QWCMSO*) [6], whose expressive power lies between WCMSO and the CMSO. Indeed, Blumensath et al. [6, Theorem 2] prove that QWCMSO is effectively equivalent to a subclass of alternating B-automata called *quasi-weak B-automata*, and thus by Theorem 3.1 even model checking of safe recursion schemes against QWCMSO properties is decidable.

Solving the diagonal problem. By Theorem 4.5 and Lemma 4.8, all we need to do is to show that the diagonal problem is decidable for languages recognized by safe recursion schemes, that is, that given a safe recursion scheme $\mathcal{G}$ and a set of letters Σ, one can check whether for every $n \in \mathbb{N}$ there is a tree $T \in \mathcal{L}(\mathcal{G})$ such that there are at least n occurrences of every letter $a \in \Sigma$ on every branch of T (we say that such a tree T is *n-large with respect* *to* Σ). In order to do this, given a set of letters Σ, we write a WCMSO sentence φ_Σ that n-accepts an (infinite) tree T if and only if no tree in $\mathcal{L}(T)$ is n-large with respect to Σ. Consequently, φ_Σ accepts T if for some n no tree in $\mathcal{L}(T)$ is n-large with respect to Σ, that is, if $\mathcal{L}(T)$ is not Σ-diagonal. Thus, in order to solve the diagonal problem, it is enough to check whether φ_Σ accepts $BT(\mathcal{G})$ (recall that $\mathcal{L}(\mathcal{G})$ is defined as $\mathcal{L}(BT(\mathcal{G}))$), which is decidable by Corollary 5.2. It remains to construct the aforementioned sentence φ_Σ.

First, observe that the process of producing a finite tree recognized by $\mathcal{G}$ from the infinite tree $BT(\mathcal{G})$ generated by $\mathcal{G}$ is expressible by a formula of WCMSO (actually, by a first-order formula). More precisely we can write a WCMSO formula $\mathsf{tree}(X)$ that holds in a tree T if and only if X is instantiated to a set of nodes of a tree $T' \in \mathcal{L}(T)$, together with their nd-labeled ancestors. See [4, Appendix H] for more details. Using $\mathsf{tree}(X)$ we now construct the desired formula φ_Σ, and thus we finish the proof of Theorem 5.1.

▶ **Lemma 5.4.** *Given a set of letters Σ, one can compute a WCMSO sentence φ_Σ that, for every $n \in \mathbb{N}$, n-accepts a tree T if and only if no tree in $\mathcal{L}(T)$ is n-large with respect to Σ.*

Proof. We can reformulate the property as follows: for every tree $T' \in \mathcal{L}(T)$ there is a letter $a \in \Sigma$, and a leaf x that has less than n a-labeled ancestors. This is expressed by the following formula of WCMSO (where $\mathsf{leaf}(x)$ states that the node x is a leaf, $a(x)$ that x has label a, and $z \le x$ that z is an ancestor of x, all being easily expressible):

$$\forall X.\Big(\mathsf{tree}(X) \to \bigvee_{a\in\Sigma} \exists x \exists Z.\big(x \in X \wedge \mathsf{leaf}(x) \wedge \forall z.(z \le x \wedge a(z) \to z \in Z) \wedge |Z| < N\big)\Big). \quad ◀$$

6 Conclusions

A tantalising direction for further work is to drop the safety assumption from Theorem 3.1, that is, to establish whether the model-checking problem against B-automata is decidable for trees generated by (not necessarily safe) recursion schemes. We also leave open whether downward closures are computable for this more expressive class. Another direction for further work is to analyse the complexity of the considered model-checking problem. The related problem described in Remark 4.9 is k-EXP-complete for languages of finite trees recognised by recursion schemes of order k [45], and thus not harder than the nonemptiness problem [43]. Does the same upper bound hold for the more general diagonal problem that we consider in this paper? Zetzsche [56] has shown that the downward closure inclusion problem is co-k-NEXP-hard for languages of finite trees recognised by safe recursion schemes of order k. Is it possible to obtain a matching upper bound?

References

1 Parosh Aziz Abdulla, Luc Boasson, and Ahmed Bouajjani. Effective lossy queue languages. In Fernando Orejas, Paul G. Spirakis, and Jan van Leeuwen, editors, *Automata, Languages and Programming, 28th International Colloquium, ICALP 2001, Crete, Greece, July 8-12, 2001, Proceedings*, volume 2076 of *Lecture Notes in Computer Science*, pages 639–651. Springer, 2001. doi:10.1007/3-540-48224-5_53.

2 Alfred V. Aho. Indexed grammars - an extension of context-free grammars. *J. ACM*, 15(4):647–671, 1968. doi:10.1145/321479.321488.

3 Georg Bachmeier, Michael Luttenberger, and Maximilian Schlund. Finite automata for the sub- and superword closure of CFLs: Descriptional and computational complexity. In Adrian-Horia Dediu, Enrico Formenti, Carlos Martín-Vide, and Bianca Truthe, editors, *Language and Automata Theory and Applications - 9th International Conference, LATA 2015, Nice, France, March 2-6, 2015, Proceedings*, volume 8977 of *Lecture Notes in Computer Science*, pages 473–485. Springer, 2015. doi:10.1007/978-3-319-15579-1_37.

4 David Barozzini, Lorenzo Clemente, Thomas Colcombet, and Paweł Parys. Cost Automata, Safe Schemes, and Downward Closures. *arXiv e-prints*, page arXiv:2004.12187, April 2020. arXiv:2004.12187.

5 Alessandro Berarducci and Mariangiola Dezani-Ciancaglini. Infinite lambda-calculus and types. *Theor. Comput. Sci.*, 212(1-2):29–75, 1999. doi:10.1016/S0304-3975(98)00135-2.

6 Achim Blumensath, Thomas Colcombet, Denis Kuperberg, Paweł Parys, and Michael Vanden Boom. Two-way cost automata and cost logics over infinite trees. In Thomas A. Henzinger and Dale Miller, editors, *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*, pages 16:1–16:9. ACM, 2014. doi:10.1145/2603088.2603104.

7 Mikołaj Bojańczyk. A bounding quantifier. In Jerzy Marcinkowski and Andrzej Tarlecki, editors, *Computer Science Logic, 18th International Workshop, CSL 2004, 13th Annual Conference of the EACSL, Karpacz, Poland, September 20-24, 2004, Proceedings*, volume 3210 of *Lecture Notes in Computer Science*, pages 41–55. Springer, 2004. doi:10.1007/978-3-540-30124-0_7.

8 Luca Breveglieri, Alessandra Cherubini, Claudio Citrini, and Stefano Crespi-Reghizzi. Multi-push-down languages and grammars. *Int. J. Found. Comput. Sci.*, 7(3):253–292, 1996. doi:10.1142/S0129054196000191.

9 Christopher H. Broadbent, Arnaud Carayol, C.-H. Luke Ong, and Olivier Serre. Recursion schemes and logical reflection. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS 2010, 11-14 July 2010, Edinburgh, United Kingdom*, pages 120–129. IEEE Computer Society, 2010. doi:10.1109/LICS.2010.40.

10 Christopher H. Broadbent and Naoki Kobayashi. Saturation-based model checking of higher-order recursion schemes. In Simona Ronchi Della Rocca, editor, *Computer Science Logic 2013 (CSL 2013), CSL 2013, September 2-5, 2013, Torino, Italy*, volume 23 of *LIPIcs*, pages 129–148. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2013. doi:10.4230/LIPIcs.CSL.2013.129.

11 Christopher H. Broadbent and C.-H. Luke Ong. On global model checking trees generated by higher-order recursion schemes. In Luca de Alfaro, editor, *Foundations of Software Science and Computational Structures, 12th International Conference, FOSSACS 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings*, volume 5504 of *Lecture Notes in Computer Science*, pages 107–121. Springer, 2009. doi:10.1007/978-3-642-00596-1_9.

12 Arnaud Carayol and Olivier Serre. Collapsible pushdown automata and labeled recursion schemes: Equivalence, safety and effective selection. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25-28, 2012*, pages 165–174. IEEE Computer Society, 2012. doi:10.1109/LICS.2012.73.

13 Lorenzo Clemente, Paweł Parys, Sylvain Salvati, and Igor Walukiewicz. Ordered tree-pushdown systems. In Prahladh Harsha and G. Ramalingam, editors, *35th IARCS Annual Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2015, December 16-18, 2015, Bangalore, India*, volume 45 of *LIPIcs*, pages 163–177. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2015. doi:10.4230/LIPIcs.FSTTCS.2015.163.

14 Lorenzo Clemente, Paweł Parys, Sylvain Salvati, and Igor Walukiewicz. The diagonal problem for higher-order recursion schemes is decidable. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 96–105. ACM, 2016. doi:10.1145/2933575.2934527.

15 Thomas Colcombet. Regular cost functions, part I: Logic and algebra over words. *Logical Methods in Computer Science*, 9(3), 2013. doi:10.2168/LMCS-9(3:3)2013.

16 Thomas Colcombet and Stefan Göller. Games with bound guess actions. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 257–266. ACM, 2016. doi:10.1145/2933575.2934502.

17 Thomas Colcombet and Christof Löding. The non-deterministic Mostowski hierarchy and distance-parity automata. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part II - Track B: Logic, Semantics, and Theory of Programming & Track C: Security and Cryptography Foundations*, volume 5126 of *Lecture Notes in Computer Science*, pages 398–409. Springer, 2008. doi:10.1007/978-3-540-70583-3_33.

18 Thomas Colcombet and Christof Löding. Regular cost functions over finite trees. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS 2010, 11-14 July 2010, Edinburgh, United Kingdom*, pages 70–79. IEEE Computer Society, 2010. doi:10.1109/LICS.2010.36.

19 Hubert Comon, Max Dauchet, Remi Gilleron, Florent Jacquemard, Denis Lugiez, Christof Löding, Sophie Tison, and Marc Tommasi. Tree automata techniques and applications, 2007. URL: http://tata.gforge.inria.fr/.

20 Bruno Courcelle. On constructing obstruction sets of words. *Bulletin of EATCS*, 1991.

21 Wojciech Czerwiński, Wim Martens, Lorijn van Rooijen, and Marc Zeitoun. A note on decidable separability by piecewise testable languages. In Adrian Kosowski and Igor Walukiewicz, editors, *Fundamentals of Computation Theory - 20th International Symposium, FCT 2015, Gdańsk, Poland, August 17-19, 2015, Proceedings*, volume 9210 of *Lecture Notes in Computer Science*, pages 173–185. Springer, 2015. doi:10.1007/978-3-319-22177-9_14.

22 Wojciech Czerwiński, Wim Martens, Lorijn van Rooijen, Marc Zeitoun, and Georg Zetzsche. A Characterization for Decidable Separability by Piecewise Testable Languages. *Discrete Mathematics & Theoretical Computer Science*, Vol. 19 no. 4, FCT '15, December 2017. doi:10.23638/DMTCS-19-4-1.

23 Werner Damm. The IO- and OI-hierarchies. *Theor. Comput. Sci.*, 20:95–207, 1982. doi:10.1016/0304-3975(82)90009-3.

24 Alain Finkel and Jean Goubault-Larrecq. Forward analysis for WSTS, part I: Completions. In Susanne Albers and Jean-Yves Marion, editors, *26th International Symposium on Theoretical Aspects of Computer Science, STACS 2009, February 26-28, 2009, Freiburg, Germany, Proceedings*, volume 3 of *LIPIcs*, pages 433–444. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, Germany, 2009. doi:10.4230/LIPIcs.STACS.2009.1844.

25 Jean Goubault-Larrecq and Sylvain Schmitz. Deciding piecewise testable separability for regular tree languages. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 97:1–97:15. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.ICALP.2016.97.

26 Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002. doi:10.1007/3-540-36387-4.

27 Peter Habermehl, Roland Meyer, and Harro Wimmel. The downward-closure of Petri net languages. In Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul G. Spirakis, editors, *Automata, Languages and Programming, 37th International Colloquium, ICALP 2010, Bordeaux, France, July 6-10, 2010, Proceedings, Part II*, volume 6199 of *Lecture Notes in Computer Science*, pages 466–477. Springer, 2010. doi:10.1007/978-3-642-14162-1_39.

28 Axel Haddad. IO vs OI in higher-order recursion schemes. In Dale Miller and Zoltán Ésik, editors, *Proceedings 8th Workshop on Fixed Points in Computer Science, FICS 2012, Tallinn, Estonia, 24th March 2012.*, volume 77 of *EPTCS*, pages 23–30, 2012. doi:10.4204/EPTCS.77.4.

29 Matthew Hague, Jonathan Kochems, and C.-H. Luke Ong. Unboundedness and downward closures of higher-order pushdown automata. In Rastislav Bodík and Rupak Majumdar, editors, *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, pages 151–163. ACM, 2016. doi:10.1145/2837614.2837627.

30 Matthew Hague, Andrzej S. Murawski, C.-H. Luke Ong, and Olivier Serre. Collapsible pushdown automata and recursion schemes. In *Proceedings of the Twenty-Third Annual IEEE Symposium on Logic in Computer Science, LICS 2008, 24-27 June 2008, Pittsburgh, PA, USA*, pages 452–461. IEEE Computer Society, 2008. doi:10.1109/LICS.2008.34.

31 Graham Higman. Ordering by divisibility in abstract algebras. *Proc. London Math. Soc.*, s3-2(1):326–336, January 1952. doi:10.1112/plms/s3-2.1.326.

32 Richard Kennaway, Jan Willem Klop, M. Ronan Sleep, and Fer-Jan de Vries. Infinitary lambda calculus. *Theor. Comput. Sci.*, 175(1):93–125, 1997. doi:10.1016/S0304-3975(96)00171-5.

33 Teodor Knapik, Damian Niwiński, and Paweł Urzyczyn. Deciding monadic theories of hyperalgebraic trees. In Samson Abramsky, editor, *Typed Lambda Calculi and Applications, 5th International Conference, TLCA 2001, Kraków, Poland, May 2-5, 2001, Proceedings*, volume 2044 of *Lecture Notes in Computer Science*, pages 253–267. Springer, 2001. doi:10.1007/3-540-45413-6_21.

34 Teodor Knapik, Damian Niwiński, and Paweł Urzyczyn. Higher-order pushdown trees are easy. In Mogens Nielsen and Uffe Engberg, editors, *Foundations of Software Science and Computation Structures, 5th International Conference, FOSSACS 2002. Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2002 Grenoble, France, April 8-12, 2002, Proceedings*, volume 2303 of *Lecture Notes in Computer Science*, pages 205–222. Springer, 2002. doi:10.1007/3-540-45931-6_15.

35 Naoki Kobayashi. A practical linear time algorithm for trivial automata model checking of higher-order recursion schemes. In Martin Hofmann, editor, *Foundations of Software Science and Computational Structures - 14th International Conference, FOSSACS 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings*, volume 6604 of *Lecture Notes in Computer Science*, pages 260–274. Springer, 2011. doi:10.1007/978-3-642-19805-2_18.

36 Naoki Kobayashi. Model checking higher-order programs. *J. ACM*, 60(3):20:1–20:62, 2013. doi:10.1145/2487241.2487246.

37 Naoki Kobayashi and C.-H. Luke Ong. A type system equivalent to the modal mu-calculus model checking of higher-order recursion schemes. In *Proceedings of the 24th Annual IEEE Symposium on Logic in Computer Science, LICS 2009, 11-14 August 2009, Los Angeles, CA, USA*, pages 179–188. IEEE Computer Society, 2009. doi:10.1109/LICS.2009.29.

38 Gregory M. Kobele and Sylvain Salvati. The IO and OI hierarchies revisited. *Inf. Comput.*, 243:205–221, 2015. doi:10.1016/j.ic.2014.12.015.

39 Denis Kuperberg and Michael Vanden Boom. Quasi-weak cost automata: A new variant of weakness. In Supratik Chakraborty and Amit Kumar, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2011, December 12-14, 2011, Mumbai, India*, volume 13 of *LIPIcs*, pages 66–77. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2011.

40 Denis Kuperberg and Michael Vanden Boom. Quasi-weak cost automata: A new variant of weakness. In Supratik Chakraborty and Amit Kumar, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2011, December 12-14, 2011, Mumbai, India*, volume 13 of *LIPIcs*, pages 66–77. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2011. doi:10.4230/LIPIcs.FSTTCS.2011.66.

41 Jan van Leeuwen. Effective constructions in well-partially- ordered free monoids. *Discrete Mathematics*, 21(3):237–252, 1978. doi:10.1016/0012-365X(78)90156-5.

42 Robin P. Neatherway and C.-H. Luke Ong. TravMC2: Higher-order model checking for alternating parity tree automata. In Neha Rungta and Oksana Tkachuk, editors, *2014 International Symposium on Model Checking of Software, SPIN 2014, Proceedings, San Jose, CA, USA, July 21-23, 2014*, pages 129–132. ACM, 2014. doi:10.1145/2632362.2632381.

43 C.-H. Luke Ong. On model-checking trees generated by higher-order recursion schemes. In *21th IEEE Symposium on Logic in Computer Science (LICS 2006), 12-15 August 2006, Seattle, WA, USA, Proceedings*, pages 81–90. IEEE Computer Society, 2006. doi:10.1109/LICS.2006.38.

44 Paweł Parys. On the significance of the collapse operation. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25-28, 2012*, pages 521–530. IEEE Computer Society, 2012. doi:10.1109/LICS.2012.62.

45 Paweł Parys. The complexity of the diagonal problem for recursion schemes. In Satya Lokam and R. Ramanujam, editors, *Proc. of FSTTCS'17*, volume 93 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 45:1–45:14, Dagstuhl, Germany, 2017. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.FSTTCS.2017.45.

46 Paweł Parys. Recursion schemes and the WMSO+U logic. In Rolf Niedermeier and Brigitte Vallée, editors, *35th Symposium on Theoretical Aspects of Computer Science, STACS 2018, February 28 to March 3, 2018, Caen, France*, volume 96 of *LIPIcs*, pages 53:1–53:16. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.STACS.2018.53.

47 Steven J. Ramsay, Robin P. Neatherway, and C.-H. Luke Ong. A type-directed abstraction refinement approach to higher-order model checking. In Suresh Jagannathan and Peter Sewell, editors, *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, pages 61–72. ACM, 2014. doi:10.1145/2535838.2535873.

48 Sylvain Salvati and Igor Walukiewicz. Krivine machines and higher-order schemes. *Inf. Comput.*, 239:340–355, 2014. doi:10.1016/j.ic.2014.07.012.

49 Sylvain Salvati and Igor Walukiewicz. A model for behavioural properties of higher-order programs. In Stephan Kreutzer, editor, *24th EACSL Annual Conference on Computer Science Logic, CSL 2015, September 7-10, 2015, Berlin, Germany*, volume 41 of *LIPIcs*, pages 229–243. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2015. doi:10.4230/LIPIcs.CSL.2015.229.

50 Sylvain Salvati and Igor Walukiewicz. Simply typed fixpoint calculus and collapsible pushdown automata. *Mathematical Structures in Computer Science*, 26(7):1304–1350, 2016. doi:10.1017/S0960129514000590.

51 Larry J. Stockmeyer. *The complexity of decision problems in automata theory and logic.* PhD thesis, MIT, 1974.

52 Michael Vanden Boom. Weak cost monadic logic over infinite trees. In Filip Murlak and Piotr Sankowski, editors, *Mathematical Foundations of Computer Science 2011 - 36th International Symposium, MFCS 2011, Warsaw, Poland, August 22-26, 2011. Proceedings*, volume 6907 of *Lecture Notes in Computer Science*, pages 580–591. Springer, 2011.

53 Michael Vanden Boom. Weak cost monadic logic over infinite trees. In Filip Murlak and Piotr Sankowski, editors, *Mathematical Foundations of Computer Science 2011 - 36th International Symposium, MFCS 2011, Warsaw, Poland, August 22-26, 2011. Proceedings*, volume 6907 of *Lecture Notes in Computer Science*, pages 580–591. Springer, 2011. doi:10.1007/978-3-642-22993-0_52.

54 Georg Zetzsche. An approach to computing downward closures. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Proc. of ICALP'15*, volume 9135 of *LNCS*, pages 440–451. Springer, 2015. doi:10.1007/978-3-662-47666-6_35.

55 Georg Zetzsche. Computing downward closures for stacked counter automata. In Ernst W. Mayr and Nicolas Ollinger, editors, *32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015, March 4-7, 2015, Garching, Germany*, volume 30 of *LIPIcs*, pages 743–756. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2015. doi:10.4230/LIPIcs.STACS.2015.743.

56 Georg Zetzsche. The complexity of downward closure comparisons. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 123:1–123:14. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.ICALP.2016.123.

Sensitive Instances of the Constraint Satisfaction Problem

Libor Barto

Department of Algebra, Faculty of Mathematics and Physics,
Charles University, Praha 8, Czech Republic
barto@karlin.mff.cuni.cz

Marcin Kozik

Theoretical Computer Science, Faculty of Mathematics and Computer Science,
Jagiellonian University, Kraków, Poland
marcin.kozik@uj.edu.pl

Johnson Tan

Department of Mathematics, University of Illinois, Urbana-Champaign, Urbana, IL, USA
jgtan2@illinois.edu

Matt Valeriote

Department of Mathematics and Statistics, McMaster University, Hamilton, Ontario, Canada
matt@math.mcmaster.ca

Abstract

We investigate the impact of modifying the constraining relations of a Constraint Satisfaction Problem (CSP) instance, with a fixed template, on the set of solutions of the instance. More precisely we investigate sensitive instances: an instance of the CSP is called sensitive, if removing any tuple from any constraining relation invalidates some solution of the instance. Equivalently, one could require that every tuple from any one of its constraints extends to a solution of the instance.

Clearly, any non-trivial template has instances which are not sensitive. Therefore we follow the direction proposed (in the context of strict width) by Feder and Vardi in [13] and require that only the instances produced by a local consistency checking algorithm are sensitive. In the language of the algebraic approach to the CSP we show that a finite idempotent algebra $\mathbf{A}$ has a $k+2$ variable near unanimity term operation if and only if any instance that results from running the $(k,k+1)$-consistency algorithm on an instance over $\mathbf{A}^2$ is sensitive.

A version of our result, without idempotency but with the sensitivity condition holding in a variety of algebras, settles a question posed by G. Bergman about systems of projections of algebras that arise from some subalgebra of a finite product of algebras.

Our results hold for infinite (albeit in the case of $\mathbf{A}$ idempotent) algebras as well and exhibit a surprising similarity to the strict width k condition proposed by Feder and Vardi. Both conditions can be characterized by the existence of a near unanimity operation, but the arities of the operations differ by 1.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness; Theory of computation → Complexity theory and logic; Theory of computation → Constraint and logic programming

Keywords and phrases Constraint satisfaction problem, bounded width, local consistency, near unanimity operation, loop lemma

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.110

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

Related Version `https://arxiv.org/abs/2005.00266`

Funding *Libor Barto*: Research partially supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 771005), and by the Czech Science Foundation grant 18-20123S.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 110; pp. 110:1–110:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Marcin Kozik: Research partially supported by the National Science Centre Poland grants 2014/13/B/ST6/01812 and 2014/14/A/ST6/00138.
Matt Valeriote: Research partially supported by the Natural Sciences and Engineering Research Council of Canada.

1 Introduction

One important algorithmic approach to deciding if a given instance of the Constraint Satisfaction Problem (CSP) has a solution is to first consider whether it has a consistent set of local solutions. Clearly, the absence of local solutions will rule out having any (global) solutions, but in general having local solutions does not guarantee the presence of a solution. A major thrust of the recent research on the CSP has focused on coming up with suitable notions of local consistency and then characterizing those CSPs for which local consistency implies outright consistency or some stronger property. A good source for background material is the survey article [7].

Early results of Feder and Vardi [13] and also Jeavons, Cooper, and Cohen [15] establish that when a template (i.e., a relational structure) $\mathbb{A}$ has a special type of polymorphism, called a near unanimity operation, then not only will an instance of the CSP over $\mathbb{A}$ that has a suitably consistent set of local solutions have a solution, but that any partial solution of it can always be extended to a solution. The notion of local consistency that we investigate in this paper is related to that considered by these researchers but that, as we shall see, is weaker.

The following operations are central to our investigation.

▶ **Definition 1.** *An operation $n(x_1, \dots, x_{k+1})$ on a set A of arity $k+1$ is called a* near unanimity operation on A *if it satisfies the equalities*

$$n(b, a, a, \dots, a) = n(a, b, a, \dots, a) = \dots = n(a, a, \dots, a, b) = a$$

for all a, $b \in A$.

Near unanimity operations have played an important role in the development of universal algebra and first appeared in the 1970's in the work of Baker and Pixley [1] and Huhn [14]. More recently they have been used in the study of the CSP [13, 15] and related questions [2, 12]. The main results of this paper can be expressed in terms of the CSP and also in algebraic terms and we start by presenting them from both perspectives. In the concluding section, Section 6, a translation of parts of our results into a relational language is provided, along with some open problems.

1.1 CSP viewpoint

In their seminal paper, Feder and Vardi [13] introduced the notion of bounded width for the class of CSP instances over a finite template $\mathbb{A}$. Their definition of bounded width was presented in terms of the logic programming language DATALOG but there is an equivalent formulation using local consistency algorithms, also given in [13]. Given a CSP instance $\mathcal{I}$ and $k < l$, the (k, l)-consistency algorithm will produce a new instance having all k variable constraints that can be inferred by considering l variables at a time of $\mathcal{I}$. This algorithm rejects $\mathcal{I}$ if it produces an empty constraint. The class of CSP instances over a finite template $\mathbb{A}$ will have width (k, l) if the (k, l)-consistency algorithm rejects all instances from the class that do not have solutions, i.e., the (k, l)-consistency algorithm can be used to decide if a given instance from the class has a solution or not. The class has bounded width if it has width (k, l) for some $k < l$.

A lot of effort, in the framework of the algebraic approach to the CSP, has gone in to analyzing various properties of instances that are the outputs of these types of local consistency algorithms. On one end of the spectrum of the research is a rather wide class of templates of bounded width [5] and on the other a very restrictive class of templates having bounded strict width [13].

To be more precise, we now formally introduce instances of the CSP.

▶ **Definition 2.** *An* instance $\mathcal{I}$ *of the* CSP *is a pair* $(V, \mathcal{C})$ *where* V *is a finite set of variables, and* $\mathcal{C}$ *is a set of constraints of the form* $((x_1, \dots, x_n), R)$ *where all* x_i *are in* V *and* R *is an* n*-ary relation over (possibly infinite) sets* A_i *associated to each variable* x_i*.*

A solution *of* $\mathcal{I}$ *is an evaluation* f *of variables such that, for every* $((x_1, \dots, x_n), R) \in \mathcal{C}$ *we have* $(f(x_1), \dots, f(x_n)) \in R$*;* a partial solution *is a partial function satisfying the same condition.*

The CSP over a relational structure $\mathbb{A}$*, written* $\mathsf{CSP}(\mathbb{A})$*, is the class of* CSP *instances whose constraint relations are from* $\mathbb{A}$*.*

▶ **Example 3.** For $k > 1$, the template associated with the graph k-colouring problem is the relational structure $\mathbb{D}_{k\text{colour}}$ that has universe $\{0, 1, \dots, k-1\}$ and a single relation $\neq_k = \{(x, y) \mid x, y < k \text{ and } x \neq y\}$. The template associated with the HORN-3-SAT problem is the relational structure $\mathbb{D}_{\text{horn}}$ that has universe $\{0, 1\}$ and two ternary relations R_0, R_1, where R_i contains all the triples but $(1, 1, i)$. It is known that $\mathsf{CSP}(\mathbb{D}_{\text{horn}})$ has width $(1, 2)$, that $\mathsf{CSP}(\mathbb{D}_{2\text{colour}})$ has width $(2, 3)$, and that for $k > 2$, $\mathsf{CSP}(\mathbb{D}_{k\text{colour}})$ does not have bounded width (see [7]).

Instances produced by the (k, l)-consistency algorithm have uniformity and consistency properties that we highlight.

▶ **Definition 4.** *The* CSP *instance* $\mathcal{I}$ *is* k-uniform *if all of its constraints are* k*-ary and every set of* k *variables is constrained by a single constraint.*

An instance is a (k, l)-instance *if it is* k*-uniform and for every choice of a set* W *of* l *variables no additional information about the constraints can be derived by restricting the instance to the variables in* W*.*

This last, very important, property can be rephrased in the following way: for every set $W \subseteq V$ of size l, every tuple in every constraint of $\mathcal{I}_{|W}$ participates in a solution to $\mathcal{I}_{|W}$ (where $\mathcal{I}_{|W}$ is obtained from $\mathcal{I}$ by removing all the variables outside of W and all the constraints that contain any such variables).

Consider the notion of strict width k introduced by Feder and Vardi [13, Section 6.1.2]. Let $\mathbb{A}$ be a template and let us assume, to avoid some technical subtleties, that every relation in $\mathbb{A}$ has arity at most k. The class $\mathsf{CSP}(\mathbb{A})$ has *strict width* (k, l) if whenever the (k, l)-consistency algorithm does not reject an instance $\mathcal{I}$ from the class then “it should be possible to obtain a solution by greedily assigning values to the variables one at a time while satisfying the inferred k-constraints.” In other words, if $\mathcal{I}$ is the result of applying the (k, l)-consistency algorithm to an instance of $\mathsf{CSP}(\mathbb{A})$, then any partial solution of $\mathcal{I}$ over at least k variables can be extended to a solution. The template $\mathbb{A}$ is said to have *strict width* k if it has strict width (k, l) for some $l > k$.

A *polymorphism* of a template $\mathbb{A}$ is a function on A that preserves all of the relations of $\mathbb{A}$. Feder and Vardi prove the following.

▶ **Theorem 5** (see Theorem 25, [13])**.** *Let* $k > 1$ *and let* $\mathbb{A}$ *be a finite relational structure with relations of arity at most* k*. The class* $\mathsf{CSP}(\mathbb{A})$ *has strict width* k *if and only if it has strict width* $(k, k+1)$ *if and only if* $\mathbb{A}$ *has a* $(k+1)$*-ary near unanimity operation as a polymorphism.*

Using this Theorem we can conclude that $\mathsf{CSP}(\mathbb{D}_{\mathrm{2colour}})$ from Example 3 has strict width 2 since the ternary majority operation preserves the relation $\neq_2$. In fact this operation preserves all binary relations over the set $\{0,1\}$. On the other hand, $\mathsf{CSP}(\mathbb{D}_{\mathrm{horn}})$ does not have strict width k for any $k \geq 3$.

Following the algebraic approach to the CSP we replace templates $\mathbb{A}$ with algebras $\mathbf{A}$.

▶ **Definition 6.** *An* algebra $\mathbf{A}$ *is a pair* $(A, \mathcal{F})$ *where* A *is a non-empty set, called the* universe *of* $\mathbf{A}$ *and* $\mathcal{F} = (f_i \mid i \in I)$ *is a set of finitary operations on* A *called the* set of basic operations of $\mathbf{A}$. *The function that assigns the arity of the operation* f_i *to* i *is called the* signature of $\mathbf{A}$. *If* $t(x_1, \dots, x_n)$ *is a term in the signature of* $\mathbf{A}$ *then the interpretation of* t *by* $\mathbf{A}$ *as an operation on* A *is called a* term operation of $\mathbf{A}$ *and is denoted by* $t^{\mathbf{A}}$.

The CSP over $\mathbf{A}$, *written* $\mathsf{CSP}(\mathbf{A})$, *is the class of* CSP *instances whose constraint relations are amongst those relations over* A *that are preserved by the operations of* $\mathbf{A}$ *(i.e., they are subuniverses of powers of* $\mathbf{A}$*).*

A number of important questions about the CSP can be reduced to considering templates that have all of the singleton unary relations [7]; the algebraic counterpart to these types of templates are the *idempotent algebras.*

▶ **Definition 7.** *An operation* $f : A^n \to A$ *on a set* A *is* idempotent *if* $f(a, a, \dots, a) = a$ *for all* $a \in A$. *An algebra* $\mathbf{A}$ *is* idempotent *if all of its basic operations are.*

It follows that if $\mathbf{A}$ is idempotent then every term operation of $\mathbf{A}$ is an idempotent operation. As demonstrated in Example 22, several of the results in this paper do not hold in the absence of idempotency.

The characterization of strict width in Theorem 5 has the following consequence in terms of algebras.

▶ **Corollary 8.** *Let* $k > 1$ *and let* $\mathbb{A}$ *be a finite relational structure with relations of arity at most* k. *Let* $\mathbf{A}$ *be the algebra with the same universe as* $\mathbb{A}$ *whose basic operations are exactly the polymorphisms of* $\mathbb{A}$. *The following are equivalent:*

1. $\mathbf{A}$ *has a near unanimity term operation of arity* $k+1$*;*
2. *in every* $(k, k+1)$*-instance over* $\mathbf{A}$, *every partial solution extends to a solution.*

The implication “1 implies 2” in Corollary 8 remains valid for general algebras, not necessarily coming from finite relational structures with restricted arities of relations. However, the converse implication fails even if $\mathbf{A}$ is assumed to be finite and idempotent.

▶ **Example 9.** Consider the rather trivial algebra $\mathbf{A}$ that has universe $\{0,1\}$ and no basic operations. If $\mathcal{I}$ is a $(2,3)$-instance over $\mathbf{A}$ then since, as noted just after Theorem 5, every binary relation over $\{0,1\}$ is invariant under the ternary majority operation on $\{0,1\}$ it follows that every partial solution of $\mathcal{I}$ can be extended to a solution. Of course, $\mathbf{A}$ does not have a near unanimity term operation of any arity.

What this example demonstrates is that in general, for a fixed k, the k-ary constraint relations arising from an algebra do not capture that much of the structure of the algebra. Example 22 provides further evidence for this.

Our first theorem shows that for finite idempotent algebras $\mathbf{A}$, by considering a slightly bigger set of $(k, k+1)$-instances, over $\mathsf{CSP}(\mathbf{A}^2)$, rather than over $\mathsf{CSP}(\mathbf{A})$, we can detect the presence of a $(k+1)$-ary near unanimity term operation. Moreover, it is enough to consider only instances with $k+2$ variables. We note that every $(k, k+1)$-instance over $\mathbf{A}$ can be easily encoded as a $(k, k+1)$-instance over $\mathbf{A}^2$.

▶ **Theorem 10.** *Let* $\mathbf{A}$ *be a finite, idempotent algebra and* $k > 1$*. The following are equivalent:*

1. $\mathbf{A}$ *(or equivalently* $\mathbf{A}^2$*) has a near unanimity term operation of arity* $k + 1$*;*
2. *in every* $(k, k + 1)$*-instance over* $\mathbf{A}^2$*, every partial solution extends to a solution;*
3. *in every* $(k, k + 1)$*-instance over* $\mathbf{A}^2$ ***on*** $k + 2$ ***variables****, every partial solution extends to a solution.*

In Theorem 20 we extend our result to infinite idempotent algebras by working with local near unanimity term operations.

Going back the original definition of strict width: “it should be possible to obtain a solution by greedily assigning values to the variables one at a time while satisfying the inferred k-constraints” we note that the requirement that the assignment should be greedy is rather restrictive. The main theorem of this paper investigates an arguably more natural concept where the assignment need not be greedy.

▶ **Definition 11.** *An instance of the* CSP *is called* sensitive*, if removing any tuple from any constraining relation invalidates some solution of the instance.*

In other words, an instance is sensitive if every tuple in every constraint of the instance extends to a solution. For $(k, k + 1)$-instances, being sensitive is equivalent to the instance being a (k, n)-instance, where n is the number of variables present in the instance. We provide the following characterization.

▶ **Theorem 12.** *Let* $\mathbf{A}$ *be a finite, idempotent algebra and* $k > 1$*. The following are equivalent:*

1. $\mathbf{A}$ *(or equivalently* $\mathbf{A}^2$*) has a near unanimity term operation of arity* $k + 2$*;*
2. *every* $(k, k + 1)$*-instance over* $\mathbf{A}^2$ *is sensitive;*
3. *every* $(k, k + 1)$*-instance over* $\mathbf{A}^2$ ***on*** $\boldsymbol{k + 2}$ ***variables*** *is sensitive.*

Exactly as in Theorem 10 we can consider infinite algebras at the cost of using local near unanimity term operations (see Theorem 21).

In conclusion we investigate a natural property of instances motivated by the definition of strict width and provide a characterization of this new condition in algebraic terms. A surprising conclusion is that the new concept is, in fact, very close to the strict width concept, i.e., for a fixed k one characterization is equivalent to a near unanimity operation of arity $k + 1$ and the second of arity $k + 2$.

1.2 Algebraic viewpoint

Our work has as an antecedent the papers of Baker and Pixley [1] and of Bergman [8] on algebras having near unanimity term operations. In these papers the authors considered subalgebras of products of algebras and systems of projections associated with them. Baker and Pixley showed that in the presence of a near unanimity term operation, such a subalgebra is closely tied with its projections onto small sets of coordinates.

▶ **Definition 13.** *A* variety of algebras *is a class of algebras of the same signature that is closed under taking homomorphic images, subalgebras, and direct products. For* $\mathbf{A}$ *an algebra,* $\mathcal{V}(\mathbf{A})$ *denotes the smallest variety that contains* $\mathbf{A}$ *and is called the* variety generated by $\mathbf{A}$*. A variety* $\mathcal{V}$ *has a near unanimity term of arity* $k + 1$ *if there is some* $(k + 1)$*-ary term in the signature of* $\mathcal{V}$ *whose interpretation in each member of* $\mathcal{V}$ *is a near unanimity operation.*

Here is one version of the Baker-Pixley Theorem:

▶ **Theorem 14** (see Theorem 2.1 from [1]). *Let $\mathbf{A}$ be an algebra and $k > 1$. The following are equivalent:*

1. $\mathbf{A}$ *has a $(k+1)$-ary near unanimity term operation;*
2. *for every $r > k$ and every $\mathbf{A}_i \in \mathcal{V}(\mathbf{A})$, $1 \leq i \leq r$, every subalgebra $\mathbf{R}$ of $\prod_{i=1}^{r} \mathbf{A}_i$ is **uniquely** determined by the projections of R on all products $A_{i_1} \times \cdots \times A_{i_k}$ for $1 \leq i_1 < i_2 < \cdots < i_k \leq r$;*
3. *the same as condition 2, with r set to $k+1$.*

In other words, an algebra has a $(k+1)$-ary near unanimity term operation if and only if every subalgebra of a product of algebras from $\mathcal{V}(\mathbf{A})$ is uniquely determined by its system of k-fold projections into its factor algebras. A natural question, extending the result above, was investigated by Bergman [8]: when does a given "system of k-fold projections" arise from a product algebra?

Note that such a system can be viewed as a k-uniform CSP instance: indeed, following the notation of Theorem 14, we can introduce a variable x_i for each $i \leq r$ and a constraint $((x_{i_1}, \ldots, x_{i_k}); \mathrm{proj}_{i_1,\ldots,i_k} R)$ for each $1 \leq i_1 < i_2 < \cdots < i_k \leq r$. In this way the original relation R consists of solutions of the created instance (but in general will not contain all of them). In this particular instance, different variables can be evaluated in different algebras. Note that the instance is sensitive, if and only if it "arises from a product algebra" in the sense investigated by Bergman.

We will say that $\mathcal{I}$ is a CSP instance *over the variety $\mathcal{V}$ (denoted $\mathcal{I} \in \mathsf{CSP}(\mathcal{V})$)* if all the constraining relations of $\mathcal{I}$ are algebras in $\mathcal{V}$. In the language of the CSP, Bergman proved the following:

▶ **Theorem 15** ([8]). *If $\mathcal{V}$ is a variety that has a $(k+1)$-ary near unanimity term then every $(k, k+1)$-instance over $\mathcal{V}$ is sensitive.*

In commentary that Bergman provided on his proof of this theorem he noted that a stronger conclusion could be drawn from it and he proved the following theorem. We note that this theorem anticipates the results from [13] and [15] dealing with templates having near unanimity operations as polymorphisms.

▶ **Theorem 16** ([8]). *Let $k > 1$ and $\mathcal{V}$ be a variety. The following are equivalent:*

1. $\mathcal{V}$ *has a $(k+1)$-ary near unanimity term;*
2. *any partial solution of a $(k, k+1)$-instance over $\mathcal{V}$ extends to a solution.*

Theorem 15 provides a partial answer to the question that Bergman posed in [8], namely that in the presence of a $(k+1)$-ary near unanimity term, a necessary and sufficient condition for a k-fold system of algebras to arise from a product algebra is that the associated CSP instance is a $(k, k+1)$-instance.

In [8] Bergman asked whether the converse to Theorem 15 holds, namely, that if all $(k, k+1)$-instances over a variety are sensitive, must the variety have a $(k+1)$-ary near unanimity term? He provided examples that suggested that the answer is no, and we confirm this by proving that the condition is actually equivalent to the variety having a near unanimity term of arity $k+2$. The main result of this paper, viewed from the algebraic perspective (but stated in terms of the CSP), is the following:

▶ **Theorem 17.** *Let $k > 1$. A variety $\mathcal{V}$ has a $(k+2)$-ary near unanimity term if and only if each $(k, k+1)$-instance of the* CSP *over $\mathcal{V}$ is sensitive.*

The "if" direction of this theorem is proved in Section 3, while a sketch of a proof of the "only if" direction can be found in Section 5 (the complete reasoning is included in the full version of this paper). We note that a novel and significant feature of this result is that it does not assume any finiteness or idempotency of the algebras involved.

1.3 Structure of the paper

The paper is structured as follows. In the next section we introduce local near unanimity operations and state Theorem 10 and Theorem 12 in their full power. In Section 3 we collect the proofs that establish the existence of (local) near unanimity operations. Section 4 contains a proof of a new loop lemma, which can be of independent interest, and is necessary in the proof in Section 5. In Section 5 we provide a sketch of the proof showing that, in the presence of a near unanimity operation of arity $k+2$, the $(k, k+1)$-instances are sensitive. A complete proof of this fact, which is our main contribution, can be found in the full version of this paper. Finally, Section 6 contains conclusions.

2 Details of the CSP viewpoint

In order to state our results in their full strength, we need to define local near unanimity operations. This special concept of local near unanimity operations is required, when considering infinite algebras.

▶ **Definition 18.** *Let $k > 1$. An algebra $\mathbf{A}$ has* local near unanimity term operations of arity $k+1$ *if for every finite subset S of A there is some $(k+1)$-ary term operation n_S of $\mathbf{A}$ such that*

$$n_S(b, a, \ldots, a, a) = n_S(a, b, a, \ldots, a) = \cdots = n_S(a, a, \ldots, b, a) = n_S(a, a, \ldots, a, b) = a.$$

for all a, $b \in S$.

It should be clear that, for finite algebras, having local near unanimity term operations of arity $k+1$ and having a near unanimity term operation of arity $k+1$ are equivalent, but for arbitrary algebras they are not. The following provides a characterization of when an idempotent algebra has local near unanimity term operations of some given arity; it will be used in the proofs of Theorems 20 and 21. It is similar to Theorem 14 and is proved in the full version of this paper.

▶ **Theorem 19.** *Let $\mathbf{A}$ be an idempotent algebra and $k > 1$. The following are equivalent:*

1. *$\mathbf{A}$ has local near unanimity term operations of arity $k+1$;*
2. *for every $r > k$, every subalgebra of $\mathbf{A}^r$ is uniquely determined by its projections onto all k-element subsets of coordinates;*
3. *every subalgebra of $\mathbf{A}^{k+1}$ is uniquely determined by its projections onto all k-element subsets of coordinates.*

We are ready to state Theorem 10 in its full strength:

▶ **Theorem 20.** *Let $\mathbf{A}$ be an idempotent algebra and $k > 1$. The following are equivalent:*

1. *$\mathbf{A}$ (or equivalently $\mathbf{A}^2$) has local near unanimity term operations of arity $k+1$;*
2. *in every $(k, k+1)$-instance over $\mathbf{A}^2$, every partial solution extends to a solution;*
3. *in every $(k, k+1)$-instance over $\mathbf{A}^2$ **on** $k+2$ **variables**, every partial solution extends to a solution.*

Proof. Obviously condition 2 implies condition 3. A proof of condition 3 implying condition 1 can be found in Section 3. The implication from 1 to 2 is covered by Theorem 16. ◀

Analogously, the main result of the paper, for idempotent algebras, and the full version of Theorem 12 states:

▶ **Theorem 21.** *Let* $\mathbf{A}$ *be an idempotent algebra and* $k > 1$. *The following are equivalent:*

1. $\mathbf{A}$ *(or equivalently* $\mathbf{A}^2$*) has local near unanimity term operations of arity* $k+2$*;*
2. *every* $(k, k+1)$*-instance over* $\mathbf{A}^2$ *is sensitive;*
3. *every* $(k, k+1)$*-instance over* $\mathbf{A}^2$ ***on*** $\boldsymbol{k+2}$ ***variables*** *is sensitive.*

Proof. Obviously condition 2 implies condition 3. For a proof that condition 3 implies condition 1 see Section 3. A sketch of the proof of the remaining implication can be found in Section 5 (see the full version of this paper for a complete proof). ◀

The following examples show that in Theorems 19, 20, and 21 the assumption of idempotency is necessary.

▶ **Example 22.** For $n > 2$, let $\mathbf{S}_n$ be the algebra with domain $[n] = \{1, 2, \dots, n\}$ and with basic operations consisting of all unary operations on $[n]$ and all non-surjective operations on $[n]$ of arbitrary arity. The collection of such operations forms a finitely generated clone, called the Słupecki clone. Relevant details of these algebras can be found in [16, Example 4.6] and [20]. It can be shown that for $m < n$, the subuniverses of $\mathbf{S}_n^m$ consist of all m-ary relations R_θ over $[n]$ determined by a partition θ of $[m]$ by

$$R_\theta = \{(a_1, \dots, a_m) \mid a_i = a_j \text{ whenever } (i, j) \in \theta\}.$$

These rather simple relations are preserved by any operation on $[n]$, in particular by any majority operation or more generally, by any near unanimity operation.

It follows from Theorem 16 that if $k > 1$ and $\mathcal{I}$ is a $(k, k+1)$-instance of $\mathsf{CSP}(\mathbf{S}_{2k+1}^2)$ then any partial solution of $\mathcal{I}$ extends to a solution. This also implies that $\mathcal{I}$ is sensitive. Furthermore any subalgebra of $\mathbf{S}_{k+2}^{k+1}$ is determined by it projections onto all k-element sets of coordinates. As noted in [16, Example 4.6], for $n > 2$, $\mathbf{S}_n$ does not have a near unanimity term operation of any arity, since the algebra $\mathbf{S}_n^n$ has a quotient that is a 2-element essentially unary algebra.

3 Constructing near unanimity operations

In this section we collect the proofs providing, under various assumptions, near unanimity or local near unanimity operations. That is: the proofs of “3 implies 1” in Theorems 20 and Theorem 21 as well as a proof of the “if” direction from Theorem 17.

In the following proposition we construct instances over $\mathbf{A}^2$ (for some algebra $\mathbf{A}$). By a minor abuse of notation, we allow in such instances two kinds of variables: variables x evaluated in A and variables y evaluated in A^2. The former kind should be formally considered as variables evaluated in A^2 where each constraint enforces that x is sent to $\{(b, b) \mid b \in A\}$.

Moreover, dealing with k-uniform instances, we understand the condition “every set of k variables is constrained by a single constraint” flexibly: in some cases we allow for more constraints with the same set of variables, as long as the relations are proper permutations so that every constraint imposes the same restriction.

▶ **Proposition 23.** *Let* $k > 1$ *and let* $\mathbf{A}$ *be an algebra such that, for every* $(k, k+1)$*-instance* $\mathcal{I}$ *over* $\mathbf{A}^2$ *on* $k+2$ *variables every partial solution of* $\mathcal{I}$ *extends to a solution. Then each subalgebra of* $\mathbf{A}^{k+1}$ *is determined by its* k*-ary projections.*

Proof. Let $\mathbf{R} \leq \mathbf{A}^{k+1}$ and we will show that it is determined by the system of projections $\mathrm{proj}_I(R)$ as I ranges over all k elements subsets of coordinates. Using $\mathbf{R}$ we define the following instance $\mathcal{I}$ of $\mathsf{CSP}(\mathbf{A}^2)$. The variables of $\mathcal{I}$ will be the set $\{x_1, x_2, \dots, x_{k+1}, y_{12}\}$ and the domain of each x_i is A, while the domain of y_{12} is A^2.

For $U \subseteq \{x_1, \ldots, x_{k+1}\}$ of size k, let C_U be the constraint with scope U and constraint relation $R_U = \mathrm{proj}_U(R)$. For U a $(k-1)$-element subset of $\{x_1, \ldots, x_{k+1}\}$, let $C_{U\cup\{y_{12}\}}$ be the constraint with scope $U \cup \{y_{12}\}$ and constraint relation $R_{U\cup\{y_{12}\}}$ that consists of all tuples $(b_v \mid v \in U \cup \{y_{12}\})$ such that there is some $(a_1, \ldots, a_{k+1}) \in R$ with $b_v = a_i$ if $v = x_i$ and with $b_{y_{12}} = (a_1, a_2)$.

The instance $\mathcal{I}$ is k-uniform and we will show that it is sensitive. Indeed every tuple in every constraining relation originates in some tuple $\mathbf{b} \in R$. Setting $x_i \mapsto b_i$ and $y_{12} \mapsto (b_1, b_2)$ defines a solution that extends such a tuple.

In particular $\mathcal{I}$ is a $(k, k+1)$-instance over $\mathbf{A}^2$ with $k+2$ variables and so any partial solution of it can be extended to a solution. Let $\mathbf{b} \in A^{k+1}$ such that $\mathrm{proj}_I(\mathbf{b}) \in \mathrm{proj}_I(R)$ for all k element subsets I of $[k+1]$. Then $\mathbf{b}$ is a partial solution of $\mathcal{I}$ over the variables $\{x_1, \ldots, x_{k+1}\}$ and thus there is some extension of it to the variable y_{12} that produces a solution of $\mathcal{I}$. But there is only one consistent way to extend $\mathbf{b}$ to y_{12} namely by setting y_{12} to the value (b_1, b_2). By considering the constraint with scope $\{x_3, \ldots, x_{k+1}, y_{12}\}$ it follows that $\mathbf{b} \in R$, as required. ◀

Now we are ready to prove the first implication tackled in this section: 3 implies 1 in Theorem 20.

Proof of "3 implies 1" in Theorem 20. By Theorem 19 it suffices to show that each subalgebra of $\mathbf{A}^{k+1}$ is determined by its k-ary projections. Fortunately, Proposition 23 provides just that. ◀

We move on to proofs of "3 implies 1" in Theorem 21 and the "if" direction of Theorem 17. Similarly, as in the theorem just proved, we start with a proposition.

▶ **Proposition 24.** *Let $k > 1$ and let $\mathbf{A}$ be an algebra such that every $(k, k+1)$-instance $\mathcal{I}$ over $\mathbf{A}^2$ on $k+2$ variables is sensitive. Then each subalgebra of $\mathbf{A}^{k+2}$ is determined by its $(k+1)$-ary projections.*

Proof. We will show that if $\mathbf{R}$ is a subalgebra of $\mathbf{A}^{k+2}$ then $R = R^*$ where

$$R^* = \{a \in A^{k+2} \mid \mathrm{proj}_I(a) \in \mathrm{proj}_I(R) \text{ whenever } |I| = k+1\}.$$

In other words, we will show that the subalgebra $\mathbf{R}$ is determined by its projections into all $(k+1)$-element sets of coordinates.

We will use R and R^* from the previous paragraph to construct a $(k, k+2)$-instance $\mathcal{I} = (V, \mathcal{C})$ with $V = \{x_5, \ldots, x_{k+2}, y_{12}, y_{34}, y_{13}, y_{24}\}$ where each x_i is evaluated in A while all the y's are evaluated in A^2.

The set of constraints is more complicated. There is a *special constraint* on a *special variable set* $((y_{12}, y_{34}, x_5, \ldots, x_{k+2}), C)$ where

$$C = \{((a_1, a_2), (a_3, a_4), a_5, \ldots, a_{k+2}) \mid (a_1, \ldots, a_{k+2}) \in R^*\}.$$

The remaining constraints are defined using the relation R. For each set of variables $S = \{v_1, \ldots, v_k\} \subseteq V$ (which is different than the set for the special constraint) we define a constraint $((v_1, \ldots, v_k), D_S)$ with $(b_1, \ldots, b_k) \in D_S$ if and only if there exists a tuple $(a_1, \ldots, a_{k+2}) \in R$ such that:

- if v_i is x_j then $b_i = a_j$, and
- if v_i is y_{lm} then $b_i = (a_l, a_m)$.

Note that the instance $\mathcal{I}$ is k-uniform.

▷ Claim 25. $\mathcal{I}$ is a $(k, k+1)$-instance.

Let $S \subseteq V$ be a set of size k. If S is not the special variable set, then every tuple in the relation constraining S originates in some $(b_1, \ldots, b_{k+2}) \in R$ and, as in Proposition 23, sending $x_i \mapsto b_i$ and $y_{lm} \mapsto (b_l, b_m)$ defines a solution that extends such a tuple. We immediately conclude, that the potential failure of the $(k, k+1)$ condition must involve the special constraint.

Thus $S = \{y_{12}, y_{34}, x_5, \ldots, x_{k+2}\}$ and if $\mathbf{b}$ is a tuple from the special constraint C then there is some $(a_1, \ldots, a_{k+2}) \in R^*$ with

$$\mathbf{b} = ((a_1, a_2), (a_3, a_4), a_5, \ldots, a_{k+2}).$$

The extra variable that we want to extend the tuple $\mathbf{b}$ to is either y_{13} or y_{24}. Both cases are similar and we will only work through the details when it is y_{13}. In this case, assigning the value (a_1, a_3) to the variable y_{13} will produce an extension $\mathbf{b}'$ of $\mathbf{b}$ to a tuple over $S \cup \{y_{13}\}$ that is consistent with all constraints of $\mathcal{I}$ whose scopes are subsets of $\{y_{12}, y_{34}, x_5, \ldots, x_{k+2}, y_{13}\}$.

To see this, consider a k element subset S' of $\{y_{12}, y_{34}, x_5, \ldots, x_{k+2}, y_{13}\}$ that excludes some variable x_j. Then, by the definition of R^* there exists some tuple of the form $(a_1, a_2, \ldots, a_{j-1}, a'_j, a_{j+1}, \ldots, a_{k+2}) \in R$. This tuple from R can be used to witness that the restriction of $\mathbf{b}'$ to S' satisfies the constraint $D_{S'}$ since the scope of this constraint does not include the variable x_j.

Suppose that S' is a k element subset of $\{y_{12}, y_{34}, x_5, \ldots, x_{k+2}, y_{13}\}$ that excludes y_{12}. By the definition of R^* there is some tuple of the form $(a_1, a'_2, a_3, \ldots, a_{k+2}) \in R$. Using this tuple it follows that the restriction of $\mathbf{b}'$ to S' satisfies the constraint $D_{S'}$. This is because neither of the variables y_{12} and y_{24} are in S' and so the value $a'_2 \in A_2$ does not matter. A similar argument works when S' is assumed to exclude y_{34} and the claim is proved.

Since $\mathcal{I}$ is a $(k, k+1)$-instance over $\mathbf{A}^2$ and it has $k+2$ variables then by assumption, $\mathcal{I}$ is sensitive. We can use this to show that $R^* \subseteq R$ to complete the proof of this proposition. Let $(a_1, \ldots, a_{k+2}) \in R^*$ and consider the associated tuple $\mathbf{b} = ((a_1, a_2), (a_3, a_4), a_5, \ldots, a_{k+2}) \in C$. Since $\mathcal{I}$ is sensitive then this k-tuple can be extended to a solution $\mathbf{b}'$ of $\mathcal{I}$. Using any constraints of $\mathcal{I}$ whose scopes include combinations of y_{12} or y_{34} with y_{13} or y_{24} it follows that the value of $\mathbf{b}'$ on the variables y_{13} and y_{24} are (a_1, a_3) and (a_2, a_4) respectively. Then considering the restriction of $\mathbf{b}'$ to $S = \{x_5, \ldots, x_{k+2}, y_{13}, y_{24}\}$ it follows that $(a_1, \ldots, a_{k+2}) \in R$ since this restriction lies in the constraint relation D_S. ◀

We are in a position to provide the two final proofs in this section.

Proof of "3 implies 1" in Theorem 21. By Theorem 19 it suffices to show that each subalgebra of $\mathbf{A}^{k+2}$ is determined by its $(k+1)$-ary projections. Fortunately Propositions 24 provides just that. ◀

Proof of the "if" direction in Theorem 17. For this direction we apply Proposition 24 to a special member of $\mathcal{V}$, namely the $\mathcal{V}$-free algebra freely generated by $\mathbf{x}$ and $\mathbf{y}$, which we will denote by $\mathbf{F}$. Up to isomorphism, this algebra is unique and its defining property is that $\mathbf{F} \in \mathcal{V}$ and for any algebra $\mathbf{A} \in \mathcal{V}$, any map $f : \{\mathbf{x}, \mathbf{y}\} \to A$ extends uniquely to a homomorphism from $\mathbf{F}$ to $\mathbf{A}$. Consequently, for any two terms $s(x, y)$ and $t(x, y)$ in the signature of $\mathcal{V}$ if $s^{\mathbf{F}}(\mathbf{x}, \mathbf{y}) = t^{\mathbf{F}}(\mathbf{x}, \mathbf{y})$ then the equation $s(x, y) \approx t(x, y)$ holds in $\mathcal{V}$.

Let $\mathbf{R}$ be the subalgebra of $\mathbf{F}^{k+2}$ generated by the tuples $(\mathbf{y}, \mathbf{x}, \mathbf{x}, \ldots, \mathbf{x})$, $(\mathbf{x}, \mathbf{y}, \mathbf{x}, \ldots, \mathbf{x})$, $\ldots$, $(\mathbf{x}, \ldots, \mathbf{x}, \mathbf{y})$. By Proposition 24, the algebra $\mathbf{R}$ is determined by its $(k+1)$-ary projections and so the constant tuple $(\mathbf{x}, \ldots, \mathbf{x})$ belongs to R. The term generating this tuple from the given generators of $\mathbf{R}$ defines the required $(k+2)$-ary near unanimity operation. ◀

4 New loop lemmata

A *loop lemma* is a theorem stating that a binary relation satisfying certain structural and algebraic requirements necessarily contains a *loop* – a pair (a, a). In this section we provide two new loop lemmata, Theorem 31 and Theorem 32, which generalize an "infinite loop lemma" of Olšák [18] and may be of independent interest. Theorem 32 is a crucial tool for the proof presented in Section 5.

The algebraic assumptions in the new loop lemmata concern absorption, a concept that has proven to be useful in the algebraic theory of CSPs and in universal algebra [6]. We adjust the standard definition to our specific purposes. We begin with a very elementary definition.

▶ **Definition 26.** *Let R and S be sets. We call a tuple $(a_1, \dots, a_n)$ a* one-S-in-R tuple *if for exactly one i we have $a_i \in S$ and all the other a_i's are in R.*

Next we proceed to define a relaxation of the standard absorbing notion. We follow a standard notation, silently extending operations of an algebra to powers (by computing them coordinate-wise).

▶ **Definition 27.** *Let $\mathbf{A}$ be an algebra, $\mathbf{R} \leq \mathbf{A}^k$ and $S \subseteq A^k$. We say that R locally n-absorbs S if, for every finite set $\mathcal{C}$ of one-S-in-R tuples of length n, there is a term operation t of $\mathbf{A}$ such that $t(\mathbf{a^1}, \dots, \mathbf{a^n}) \in R$ whenever $(\mathbf{a^1}, \dots, \mathbf{a^n}) \in \mathcal{C}$. We will say that R* locally absorbs *S, if R locally n-absorbs S for some n.*

Absorption, even in this form, is stable under various constructions. The following lemma lists some of them and we leave it without a proof (the reasoning is identical to the one in e.g. Proposition 2 in [6]).

▶ **Lemma 28.** *Let $\mathbf{A}$ be an algebra and $\mathbf{R} \leq \mathbf{A}^2$ such that R locally n-absorbs S. Then R^{-1} locally n-absorbs S^{-1}; and $R \circ R$ locally n-absorbs $S \circ S$, and $R \circ R \circ R$ locally n-absorbs $S \circ S \circ S$ etc.*

Let us prove a first basic property of local absorption.

▶ **Lemma 29.** *Let $\mathbf{A}$ be an idempotent algebra and $\mathbf{R} \leq \mathbf{A}^2$ such that R locally n-absorbs S. Let $(a_1, \dots, a_n)$ and $(b_1, \dots, b_n)$ be directed walks in R, and let $(a_i, b_i) \in S$ for each i (see Figure 1). Then there exists a directed walk from a_1 to b_n of length n in R.*

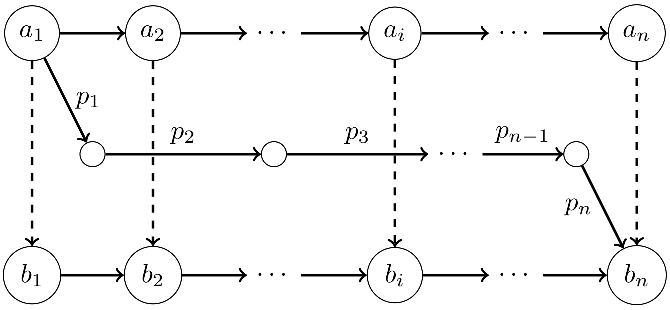


Figure 1 Solid arrows represent tuples from R and dashed arrows represent tuples from S.

Proof. We will show that there is a term operation t of the algebra $\mathbf{A}$ such that the following $(n+1)$-tuple of elements of A is a walk of length n in R from a_1 to b_n.

$$\begin{aligned}(a_1 =&t(a_1, a_1, a_1, \dots, a_1),\\ &t(b_1, a_2, a_2, \dots, a_2),\\ &t(b_2, b_2, a_3, \dots, a_3),\\ &\vdots\\ &t(b_{n-1}, b_{n-1}, \dots, b_{n-1}, a_n),\\ b_n =&t(b_n, b_n, b_n, \dots, b_n)).\end{aligned}$$

In order to choose a proper t we apply the definition of local absorption to the set of $(n+1)$ one-S-in-R tuples corresponding to the steps in the path. ◀

The loop lemma of Olšák concerns symmetric relations absorbing the equality relation $\{(a, a) \mid a \in A\}$, which is denoted $=_A$. The original result, stated in a slightly different language, does not cover the case of local absorption. However, a typographical modification of a proof mentioned in [18] shows that the theorem holds. For completeness sake, we present this proof in the full version of this paper.

▶ **Theorem 30** ([18])**.** *Let $\mathbf{A}$ be an **idempotent** algebra and $\mathbf{R} \leq \mathbf{A}^2$ be nonempty and symmetric. If R locally absorbs $=_A$, then R contains a loop.*

In order to apply this theorem in the case of sensitive instances, we need to generalize it. In the following two theorems we will gradually relax the requirement that R is symmetric. In the first step, we substitute it with a condition requiring a closed, directed walk in the graph (i.e., a sequence of possibly repeating vertices, with consecutive vertices connected by forward edges and the first and last vertex identical). Recall that R^{-1} is the inverse relation to R and let us denote by $R^{\circ l}$ the l-fold relational composition of R with itself.

▶ **Theorem 31.** *Let $\mathbf{A}$ be an **idempotent** algebra and $\mathbf{R} \leq \mathbf{A}^2$ contain a directed closed walk. If R locally absorbs $=_A$, then R contains a loop.*

Proof. Let n denote the arity of the absorbing operations. The proof is by induction on $l \geq 0$, where l is a number such that there exists a directed closed walk from a_1 to a_1 of length 2^l.

We start by verifying that such an l exists. Take a directed walk $(a_1, \dots, a_{k-1}, a_k = a_1)$ in R. We may assume that its length k is at least n, since we can, if necessary, traverse the walk multiple times. An application of Lemma 29 to the relations $R, =_A$ and tuples $(a_1, \dots, a_n)$, $(a_1, \dots, a_n)$ gives us a directed walk from a_1 to a_n of length n. Appending this walk with the walk $(a_n, a_{n+1}, \dots, a_k = a_1)$ yields a directed walk from a_1 to a_1 of length $k+1$. In this way, we can get a directed walk from a_1 to a_1 of any length greater than k.

Now we return to the inductive proof and start with the base of induction for $l = 0$ or $l = 1$. If $l = 0$, then we have found a loop. If $l = 1$ we have a closed walk of length 2, that is, a pair (a, b) which belongs to both R and R^{-1}. We set $R' = R \cap R^{-1}$ and observe that R' is nonempty and symmetric, and it is not hard to verify that R' locally absorbs $=_A$. Olšák's loop lemma, in the form of Theorem 30, gives us a loop in R.

Finally, we make the induction step from $l-1$ to l. Take a closed walk $(a_1, a_2, \dots)$ of length 2^l and consider $R' = R^{\circ 2}$. Observe that R' contains a directed closed walk of length 2^{l-1} (namely $(a_1, a_3, \dots)$), and that R' locally absorbs $=_A$ (by Lemma 28), so, by the inductive hypothesis, R' has a loop. In other words, R has a directed closed walk of length 2 and we are done by the case $l = 1$. ◀

Note that we cannot further relax the assumption on the graph by requiring that, for example, it has an infinite directed walk. Indeed the natural order of the rationals (taken for R) locally 2-absorbs the equality relation by the binary arithmetic mean operation $(a+b)/2$ (i.e., all the absorbing evaluations are realized by a single operation). The same relation locally 4-absorbs equality with the near unanimity operation $n(x,y,z,w)$ which, when applied to $a \leq b \leq c \leq d$, in any order, returns $(b+c)/2$.

Nevertheless, we can strengthen the algebraic assumption and still provide a loop; the following theorem is one of the key components in the proof sketch provided in Section 5 (albeit applied there with $l = 1$).

▶ **Theorem 32.** *Let* $\mathbf{A}$ *be an* ***idempotent*** *algebra and* $\mathbf{R} \leq \mathbf{A}^2$ *contain a directed walk of length* $n-1$*. If* R *locally* n*-absorbs* $=_A$ *and* $R^{\circ l}$ *locally* n*-absorbs* R^{-1} *for some* $l \in \mathbb{N}$ *then* R *contains a loop.*

Proof. By applying Lemma 29 similarly as in the proof of Theorem 31, we can get, from a directed walk of length $n-1$, a directed walk $(a_1, a_2, \dots)$ of an arbitrary length. Moreover, by the same reasoning, for each i and j with $j \geq i+n-1$, there is a directed walk from a_i to a_j of any length greater than or equal to $j-i$.

Consider the relations $R' = R^{\circ ln^2}$ and $S = (R^{-1})^{\circ n^2}$, and tuples

$$\mathbf{c} = (c_1, \dots, c_n) := (a_{n^2}, a_{(n+1)n}, \dots a_{(2n-1)n}), \text{ and}$$
$$\mathbf{d} = (d_1, \dots, d_n) := (a_n, a_{2n} \dots, a_{n^2})$$

By the previous paragraph and the definitions, both $\mathbf{c}$ and $\mathbf{d}$ are directed walks in R', and $(c_i, d_i) \in S$ for each i. Moreover, since $R^{\circ l}$ locally n-absorbs R^{-1}, Lemma 28 implies that R' locally absorbs S. We can thus apply Lemma 29 to the relations R', S and the tuples $\mathbf{c}, \mathbf{d}$ and obtain a directed walk from $c_1 = a_{n^2}$ to $d_{n-1} = a_{n^2}$ in R'. This closed walk in turn gives a closed directed walk in R and we are in a position to finish the proof by applying Theorem 31. ◀

5 Consistent instances are sensitive (sketch of a proof)

In this section we present the main ideas that are used to prove the "only if" direction in Theorem 17 and "1 implies 2" in Theorem 21. These ideas are shown in a very simplified situation, in particular, only the case that $k = 2$ and $\mathbf{A}$ is finite is considered. In the end of this section we briefly discuss the necessary adjustments in the general situation. A complete proof is given in the full version of this paper.

Consider a finite idempotent algebra $\mathbf{A}$ with a 4-ary near unanimity term operation and a $(2,3)$-instance $\mathcal{I} = (V, \mathcal{C})$ over $\mathbf{A}$. Each pair $\{x, y\}$ of variables is constrained by a unique constraint $((x,y), R_{xy})$ or $((y,x), R_{yx})$. For convenience we also define $R_{yx} = R_{yx}^{-1}$ (or $R_{xy} = R_{yx}^{-1}$ in the latter case) and R_{xx} to be the equality relation on A. Our aim is to show that every pair in every constraint relation extends to a solution. The overall structure of the proof is by induction on the number of variables of $\mathcal{I}$.

We fix a pair of variables $\{x_1, x_2\}$ and a pair $(a_1, a_2) \in R_{x_1 x_2}$ that we want to extend. The strategy is to consider the instance $\mathcal{J}$ obtained by removing x_1 and x_2 from the set of variables and shrinking the constraint relations R_{uv} to R'_{uv} so that only the pairs consistent with the fixed choice remain, that is,

$$R'_{uv} = \{(b,c) \in R_{uv} \mid (a_1, b) \in R_{x_1 u}, (a_2, b) \in R_{x_2 u}, (a_1, c) \in R_{x_1 v}, (a_2, c) \in R_{x_2 v}\}.$$

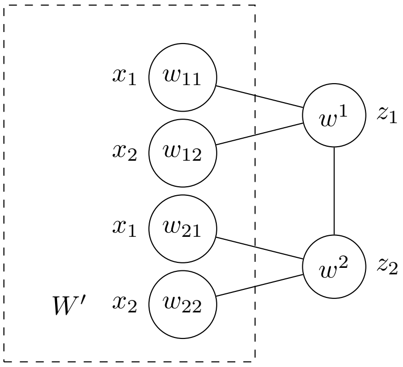


Figure 2 Pattern $\mathbb{P}$ in Lemma 35.

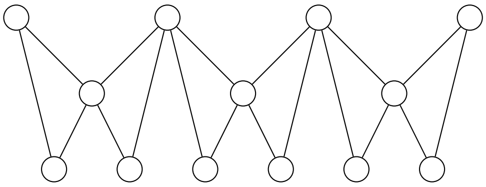


Figure 3 Path of three bow ties.

We will show that $\mathcal{J}$ contains a nonempty $(2,3)$-subinstance, that is, an instance whose constraint relations are nonempty subsets of the original ones. The induction hypothesis then gives us a solution to $\mathcal{J}$ which, in turn, yields a solution to $\mathcal{I}$ that extends the fixed choice.

Having a nonempty $(2,3)$-subinstance can be characterized by the solvability of certain relaxed instances. The following concepts will be useful for working with relaxations of $\mathcal{I}$ and $\mathcal{J}$.

▶ **Definition 33.** *A* pattern *is a triple $\mathbb{P} = (W; \mathcal{F}, l)$, where $(W; \mathcal{F})$ is an undirected graph, and l is a mapping $l : W \to V$. The variable $l(i)$ is referred to as the* label *of i.*

A realization *(*strong realization*, respectively) of $\mathbb{P}$ is a mapping $\alpha : W \to A$, which* satisfies *every edge $\{w_1, w_2\} \in \mathcal{F}$, that is, $(\alpha(w_1), \alpha(w_2)) \in R_{l(w_1),l(w_2)}$ $((\alpha(w_1), \alpha(w_2)) \in R'_{l(w_1),l(w_2)}$, respectively). (Strong realization only makes sense if $l(W) \subseteq V \setminus \{x_1, x_2\}$.)*

*A pattern is (*strongly*)* realizable *if it has a (strong) realization.*

The most important patterns for our purposes are *2-trees*, these are patterns obtained from the empty pattern by gradually adding triangles (patterns whose underlying graph is the complete graph on 3 vertices) and merging them along a vertex or an edge to the already constructed pattern. Their significance stems from the following well known fact.

▶ **Lemma 34.** *An instance (over a finite domain) contains a nonempty (2,3)-subinstance if and only if every 2-tree is realizable in it.*

The "only if" direction of the lemma applied to the instance $\mathcal{I}$ implies that every 2-tree is realizable. The "if" direction applied to the instance $\mathcal{J}$ tells us that our aim boils down to proving that every 2-tree is strongly realizable. This is achieved by an induction on a suitable measure of complexity of the tree using several constructions. We will not go into full technical details here, we rather present several lemmata whose proofs contain essentially all the ideas that are necessary for the complete proof.

▶ **Lemma 35.** *Every edge (i.e., a pattern whose underlying graph is a single edge) is strongly realizable.*

Proof sketch. Let $\mathbb{Q}$ be the pattern formed by an undirected edge with vertices w^1 and w^2 labeled z_1 and z_2, respectively. Let $\mathbb{P}$ be the pattern obtained from $\mathbb{Q}$ by adding a set of four fresh vertices $W' = \{w_{11}, w_{12}, w_{21}, w_{22}\}$ labeled x_1, x_2, x_1, x_2, respectively, and adding the edges $\{w^i, w_{i1}\}$ and $\{w^i, w_{i2}\}$ for $i = 1, 2$, see Figure 2. Observe that the restriction of a realization β of $\mathbb{P}$, such that $\beta(w_{ij}) = a_j$ for each $i, j \in \{1, 2\}$, to the set $\{w^1, w^2\}$ is a strong realization of $\mathbb{Q}$.

We consider the set T of restrictions of realizations of $\mathbb{P}$ to the set W'. Since constraint relations are subuniverses of $\mathbf{A}^2$, it follows that T is a subuniverse of $\mathbf{A}^4$.

$$T = \{(\beta(w_{11}), \beta(w_{12}), \beta(w_{21}), \beta(w_{22})) \mid \beta \text{ realizes } \mathbb{P}\} \leq \mathbf{A}^4$$

We need to prove that the tuple $\mathbf{a} = (a_1, a_2, a_1, a_2)$ is in T. By the Baker-Pixley theorem, Theorem 14, it is enough to show that for any 3-element set of coordinates, the relation T contains a tuple that agrees with $\mathbf{a}$ on this set. This is now our aim.

For simplicity, consider the set of the first three coordinates. We will build a realization β of $\mathbb{P}$ in three steps. After each step, β will satisfy all the edges where it is defined. First, since $(a_1, a_2) \in R_{x_1x_2}$ and $\mathcal{I}$ is a (2,3)-instance, we can find $b_1 \in A$ such that $(a_1, b_1) \in R_{x_1z_1}$ and $(a_2, b_1) \in R_{x_2z_1}$, and we set $\beta(w_{11}) = a_1$, $\beta(w_{12}) = a_2$, and $\beta(w^1) = b_1$. Second, we find $b_2 \in A$ such that $(a_1, b_2) \in R_{x_1z_2}$ and $(b_1, b_2) \in R_{z_1z_2}$ (here we use $(a_1, b_1) \in R_{x_1z_1}$ and that $\mathcal{I}$ is a (2,3)-instance), and set $\beta(w_{21}) = a_1$, $\beta(w^2) = b_2$. Third, using $(a_1, b_2) \in R_{x_1z_2}$ we find a_2' such that $(b_2, a_2') \in R_{z_2x_2}$ and set $\beta(w_{22}) = a_2'$. By construction, β is a realization of $\mathbb{P}$ and $(\beta(w_{11}), \beta(w_{12}), \beta(w_{21})) = (a_1, a_2, a_1)$, so our aim has been achieved. ◀

Using Lemma 35, one can go a step further and prove that every pattern built on a graph which is a triangle is strongly realizable. We are not going to prove this fact here.

▶ **Lemma 36.** *Every bow tie (a pattern whose underlying graph is formed by two triangles with a single common vertex) is strongly realizable.*

Proof sketch. Let $\mathbb{W}_1'$ and $\mathbb{W}_2'$ be two triangles (viewed as undirected graphs) with a single common vertex w. Let $\mathbb{Q}'$ be any pattern over $W_1' \cup W_2'$ with labelling l' sending $W_1' \cup W_2'$ to $V \setminus \{x_1, x_2\}$. Similarly as in the proof of Lemma 35 we form a pattern $\mathbb{Q}$ by adding to $\mathbb{Q}'$ ten additional vertices (five of them labeled x_1, the other five x_2) and edges so that the restriction of a realization α of $\mathbb{Q}$ to the set $W_1' \cup W_2'$ is a strong realization of $\mathbb{Q}'$ whenever the additional vertices have proper values (that is, value a_i for vertices labeled x_i).

We will gradually construct a realization α of $\mathbb{Q}$, which sends all the vertices labeled by x_1 to a_1, and all the vertices labeled by x_2 and adjacent to a vertex in W_1' to a_2. First use the discussion after Lemma 35 to find a strong realization of $\mathbb{Q}'$ restricted to W_1'. This defines α on W_1' and its adjacent vertices labeled by x_1 and x_2.

Next, we want to use Lemma 35 for assigning values to the two remaining vertices of W_2'. However, in order to accomplish that, we need to shift the perspective: the role of x_1 is played by x_1, but the role of x_2 is played by $l'(w)$; and the role of (a_1, a_2) is played by $(a_1, \alpha(w))$. In this new context, we use Lemma 35 to find a strong realization of the edge-pattern formed by the two remaining vertices of W_2' (with a proper restriction of l'). This defines α on all the vertices of $\mathbb{Q}$, except for the two vertices adjacent to $W_2' \setminus \{w\}$ and labeled by x_2. Finally, similarly as in the third step in the proof of Lemma 35, we define α on the remaining two vertices (labeled x_2) to get a sought after realization of $\mathbb{Q}$.

Now α assigns proper values (a_1 or a_2) to all additional vertices, except those two coming from the non-central vertices of W_2' and labeled by x_2. We apply the 4-ary near unanimity term operation to the realization α and its 3 variants obtained by exchanging the roles of W_1' and W_2' and x_1 and x_2. The result of this application is a realization of $\mathbb{Q}$ which defines a strong realization of $\mathbb{Q}'$. ◀

In the same way it is possible to prove strong realizability of further patterns, such as those in the following corollary.

▶ **Corollary 37.** *Every "path of 3 bow ties" (i.e., a pattern whose underlying graph is as in Figure 3) is strongly realizable.*

The application of the loop lemma is illustrated by the final lemma in this section.

▶ **Lemma 38.** *Every diamond (i.e., a pattern whose underlying graph is formed by two triangles with a single common edge) is strongly realizable.*

Proof sketch. The idea is to merge two vertices in a bow tie using the loop lemma. Let $\mathbb{Q}'$ be a pattern over a graph which is a bow tie on two triangles W_1' and W_2' (just like in the proof of Lemma 36). Let $w_1 \in W_1' \setminus W_2'$ and $w_2 \in W_2' \setminus W_1'$ be such that $l(w_1) = l(w_2)$.

Let $\mathbb{Q}$ be obtained from $\mathbb{Q}'$ exactly as in the proof of Lemma 36 and notice that a proper realization α of $\mathbb{Q}$ with $\alpha(w_1) = \alpha(w_2)$ gives us a strong realization of a diamond. Let $\mathbb{Q}^3$ be the pattern obtained by taking the disjoint union of 3 copies of $\mathbb{Q}$ and identifying the vertex w_2 in the i-th copy with the vertex w_1 in the $(i+1)$-first copy, for each $i \in \{1, 2\}$ (Figure 3 shows $\mathbb{Q}^3$ without the additional vertices).

Denote by T the set of all the realizations β of $\mathbb{Q}$ and denote by $S \subseteq T$ the set of those $\beta \in T$ that are proper. By a straightforward argument, both T and S are subuniverses of $\prod_{w \in Q} \mathbf{A}$. Using the near unanimity term operation of arity 4, S clearly 4-absorbs T.

The plan is to apply Theorem 32 to the binary relation $\mathrm{proj}_{w_1,w_2} S \subseteq A \times A$. As noted above, a loop in this relation gives us the desired strong realization of a diamond, so it only remains to verify the assumptions of Theorem 32. By Corollary 37, the pattern $\mathbb{Q}^3$ has a proper realization. The images of copies of vertices w_1 and w_2 in such a realization yield a directed walk in $\mathrm{proj}_{w_1,w_2}(S)$ of length 3. Next, since S 4-absorbs T, then $\mathrm{proj}_{w_1,w_2}(S)$ 4-absorbs $\mathrm{proj}_{w_1,w_2}(T)$, so it is enough to verify that the latter relation contains $=_A$ and $\mathrm{proj}_{w_1,w_2}(S)^{-1}$. We only look at the latter property. Consider any $(b_1, b_2) \in \mathrm{proj}_{w_1,w_2}(S)^{-1}$. By the definition of S, the pattern $\mathbb{Q}$ has a realization α such that $\alpha(w_1) = b_2$ and $\alpha(w_2) = b_1$. We flip the values $\alpha(w_1)$ and $\alpha(w_2)$, restrict α to $\{w_1, w_2\}$ together with the middle vertex of the bow tie, and then extend this assignment to a realization of $\mathbb{Q}$, giving us $(b_1, b_2) \in \mathrm{proj}_{w_1,w_2}(T)$. ◀

There are two major adjustments needed for the general case. First, the "if" direction of Lemma 34 (and its analogue for a general k) is no longer true over infinite domains. This is resolved by working directly with the realizability of k-trees and proving a more general claim by induction: instead of "a $(k, k+1)$-instance is sensitive" we prove, roughly, that any evaluation, which extends to a sufficiently deep k-tree, extends to a solution. Second, for higher values of k than 2 we do not prove strong realizability in one step as in, e.g., Lemma 35, but rather go through a sequence of intermediate steps between realizability and strong realizability.

6 Conclusion

We have characterized varieties that have sensitive $(k, k+1)$-instances of the CSP as those that possess a near unanimity term of arity $k + 2$. From the computational perspective, the following corollary is perhaps the most interesting consequence of our results.

▶ **Corollary 39.** *Let $\mathbb{A}$ be a finite* CSP *template whose relations all have arity at most k and which has a near unanimity polymorphism of arity $k+2$. Then every instance of the* CSP *over $\mathbb{A}$, after enforcing $(k, k+1)$-consistency, is sensitive.*

Therefore not only is the $(k, k+1)$-consistency algorithm sufficient to detect global inconsistency, we also additionally get the sensitivity property. Let us compare this result to some previous results as follows. Consider a template $\mathbb{A}$ that, for simplicity, has only unary and binary relations and that has a near unanimity polymorphism of arity $k + 2 \geq 4$. Then any instance of the CSP over $\mathbb{A}$ satisfies the following.

1. After enforcing $(2,3)$-consistency, if no contradiction is detected, then the instance has a solution [4] (this is the bounded width property).
2. After enforcing $(k,k+1)$-consistency, every partial solution on k variables extends to a solution (this is the sensitivity property).
3. After enforcing $(k+1,k+2)$-consistency, every partial solution extends to a solution [13] (this is the bounded strict width property).

For $k+2>4$ there is a gap between the first and the second item. Are there natural conditions that can be placed there?

The properties of a template $\mathbb{A}$ from the first and the third item (holding for every instance) can be characterized by the existence of certain polymorphisms: a near unanimity polymorphism of arity $k+2$ for the third item [13] and weak near unanimity polymorphisms of all arities greater than 2 for the first item [5, 11, 17]. This paper does not give such a direct characterization for the second item (essentially, since Theorem 21 involves a square). Is there any? Moreover, there are characterizations for natural extensions of the first and the third to relational structures with higher arity relations [13, 3]. This remains open for the second item as well.

In parallel with the flurry of activity around the CSP over finite templates, there has been much work done on the CSP over infinite ω-categorical templates [9, 19]. These templates cover a much larger class of computational problems but, on the other hand, share some pleasant properties with the finite ones. In particular, the $(k,k+1)$-consistency of an instance can still be enforced in polynomial time. Corollary 39 can be extended to this setting as follows.

▶ **Corollary 40.** *Let $\mathbb{A}$ be an ω-categorical* CSP *template whose relations all have arity at most k and which has local idempotent near unanimity polymorphisms of arity $k+2$. Then every instance of the* CSP *over $\mathbb{A}$, after enforcing the $(k,k+1)$-consistency, is sensitive.*

Bounded strict width k of an ω-categorical template was characterized in [10] by the existence of a *quasi-near unanimity* polymorphism n of arity $k+1$, i.e.,

$$n(y,x,\dots,x)\approx n(x,y,\dots,x)\approx\cdots\approx n(x,x,\dots,y)\approx n(x,x,\dots,x),$$

which is, additionally, *oligopotent*, i.e., the unary operation $x\mapsto n(x,x,\dots,x)$ is equal to an automorphism on every finite set. This result extends the characterization of Feder and Vardi since an oligopotent quasi-near unanimity polymorphism generates a near unanimity polymorphism as soon as the domain is finite. On an infinite domain, however, oligopotent quasi-near unanimity polymorphisms generate local near unanimity polymorphisms which, unfortunately, do not need to be idempotent on the whole domain. Our results thus fall short of proving the following natural generalization of Corollary 39 to the infinite.

▶ **Conjecture 41.** *Let $\mathbb{A}$ be an ω-categorical* CSP *template whose relations all have arity at most k and which has an oligopotent quasi-near unanimity polymorphism of arity $k+2$. Then every instance of the* CSP *over $\mathbb{A}$, after enforcing $(k,k+1)$-consistency, is sensitive.*

To confirm the conjecture, a new approach, that does not use a loop lemma, will be needed since there are examples of ω-categorical structures having oligopotent quasi-near unanimity polymorphisms for which the counterpart to Theorem 30 does not hold. Indeed, one such an example is the infinite clique.

References

1 Kirby A. Baker and Alden F. Pixley. Polynomial interpolation and the Chinese remainder theorem for algebraic systems. *Math. Z.*, 143(2):165–174, 1975. doi:10.1007/BF01187059.

2 Libor Barto. Finitely related algebras in congruence distributive varieties have near unanimity terms. *Canad. J. Math.*, 65(1):3–21, 2013. doi:10.4153/CJM-2011-087-3.

3 Libor Barto. The collapse of the bounded width hierarchy. *J. Logic Comput.*, 26(3):923–943, 2016. doi:10.1093/logcom/exu070.

4 Libor Barto and Marcin Kozik. Congruence distributivity implies bounded width. *SIAM J. Comput.*, 39(4):1531–1542, December 2009. doi:10.1137/080743238.

5 Libor Barto and Marcin Kozik. Constraint satisfaction problems solvable by local consistency methods. *J. ACM*, 61(1):3:1–3:19, January 2014. doi:10.1145/2556646.

6 Libor Barto and Marcin Kozik. Absorption in Universal Algebra and CSP. In Andrei Krokhin and Stanislav Zivny, editors, *The Constraint Satisfaction Problem: Complexity and Approximability*, volume 7 of *Dagstuhl Follow-Ups*, pages 45–77. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2017. doi:10.4230/DFU.Vol7.15301.45.

7 Libor Barto, Andrei Krokhin, and Ross Willard. Polymorphisms, and How to Use Them. In Andrei Krokhin and Stanislav Zivny, editors, *The Constraint Satisfaction Problem: Complexity and Approximability*, volume 7 of *Dagstuhl Follow-Ups*, pages 1–44. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2017. doi:10.4230/DFU.Vol7.15301.1.

8 George M. Bergman. On the existence of subalgebras of direct products with prescribed d-fold projections. *Algebra Universalis*, 7(3):341–356, 1977. doi:10.1007/BF02485443.

9 Manuel Bodirsky. Complexity classification in infinite-domain constraint satisfaction. Mémoire d'habilitation à diriger des recherches, Université Diderot – Paris 7, 2012. arXiv:1201.0856.

10 Manuel Bodirsky and Víctor Dalmau. Datalog and constraint satisfaction with infinite templates. *Journal of Computer and System Sciences*, 79(1):79–100, 2013. doi:10.1016/j.jcss.2012.05.012.

11 Andrei A. Bulatov. Bounded relational width. Preprint, 2009.

12 H. Chen, M. Valeriote, and Y. Yoshida. Testing assignments to constraint satisfaction problems. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 525–534, October 2016. doi:10.1109/FOCS.2016.63.

13 Tomás Feder and Moshe Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: a study through Datalog and group theory. *SIAM J. Comput.*, 28(1):57–104 (electronic), 1999. doi:10.1137/S0097539794266766.

14 András Huhn. Schwach distributive Verbände. *Acta Fac. Rerum Natur. Univ. Comenian. Math.*, pages 51–56, 1971.

15 Peter Jeavons, David Cohen, and Martin C. Cooper. Constraints, consistency and closure. *Artificial Intelligence*, 101(1-2):251–265, 1998. doi:10.1016/S0004-3702(98)00022-8.

16 Emil W. Kiss and Péter Pröhle. Problems and results in tame congruence theory. A survey of the '88 Budapest Workshop. *Algebra Universalis*, 29(2):151–171, 1992. doi:10.1007/BF01190604.

17 Marcin Kozik, Andrei Krokhin, Matt Valeriote, and Ross Willard. Characterizations of several Maltsev conditions. *Algebra Universalis*, 73(3-4):205–224, 2015. doi:10.1007/s00012-015-0327-2.

18 Miroslav Olšák. The weakest nontrivial idempotent equations. *Bulletin of the London Mathematical Society*, 49(6):1028–1047, 2017. doi:10.1112/blms.12097.

19 Michael Pinsker. Algebraic and model theoretic methods in constraint satisfaction, 2015. arXiv:1507.00931.

20 Ágnes Szendrei. Rosenberg-type completeness criteria for subclones of Slupecki's clone. *Proceedings of The International Symposium on Multiple-Valued Logic*, pages 349–354, 2012. doi:10.1109/ISMVL.2012.54.

The Complexity of Bounded Context Switching with Dynamic Thread Creation

Pascal Baumann
Max Planck Institute for Software Systems, Kaiserslautern, Germany
pbaumann@mpi-sws.org

Rupak Majumdar
Max Planck Institute for Software Systems, Kaiserslautern, Germany
rupak@mpi-sws.org

Ramanathan S. Thinniyam
Max Planck Institute for Software Systems, Kaiserslautern, Germany
thinniyam@mpi-sws.org

Georg Zetzsche
Max Planck Institute for Software Systems, Kaiserslautern, Germany
georg@mpi-sws.org

Abstract

Dynamic networks of concurrent pushdown systems (DCPS) are a theoretical model for multi-threaded recursive programs with shared global state and dynamical creation of threads. The (global) state reachability problem for DCPS is undecidable in general, but Atig et al. (2009) showed that it becomes decidable, and is in 2EXPSPACE, when each thread is restricted to a fixed number of context switches. The best known lower bound for the problem is EXPSPACE-hard and this lower bound follows already when each thread is a finite-state machine and runs atomically to completion (i.e., does not switch contexts). In this paper, we close the gap by showing that state reachability is 2EXPSPACE-hard already with only one context switch. Interestingly, state reachability analysis is in EXPSPACE both for pushdown threads without context switches as well as for finite-state threads with arbitrary context switches. Thus, recursive threads together with a single context switch provide an exponential advantage.

Our proof techniques are of independent interest for 2EXPSPACE-hardness results. We introduce *transducer-defined* Petri nets, a succinct representation for Petri nets, and show coverability is 2EXPSPACE-hard for this model. To show 2EXPSPACE-hardness, we present a modified version of Lipton's simulation of counter machines by Petri nets, where the net programs can make explicit recursive procedure calls up to a bounded depth.

2012 ACM Subject Classification Theory of computation → Distributed computing models; Theory of computation → Problems, reductions and completeness

Keywords and phrases Dynamic thread creation, Bounded context switching, Asynchronous Programs, Safety verification, State reachability, Petri nets, Complexity, Succinctness, Counter Programs

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.111

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

Funding This research was funded in part by the Deutsche Forschungsgemeinschaft project 389792660-TRR 248 (see `https://perspicuous-computing.science`) and by the European Research Council under the Grant Agreement 610150 (ERC Synergy Grant ImPACT).

Acknowledgements We would like to thank Ranko Lazić, who made us aware of "counter systems with chained counters" from [3] after submission of this work.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 111; pp. 111:1–111:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

There is a complexity gap between EXPSPACE and 2EXPSPACE that shows up in several problems in the safety verification of multithreaded programs.

Atig, Bouajjani, and Qadeer [1] study safety verification for *dynamic networks of concurrent pushdown systems* (DCPS), a theoretical model for multithreaded recursive programs with a finite shared global state, where threads can be recursive and can dynamically spawn additional threads. Unrestricted reachability is undecidable in this model. To ensure decidability, like many other works [16, 11, 14, 10], they assume a bound K that restricts each thread to have at most K context switches. For safety verification in this model, formulated as global state reachability, they show a lower bound of EXPSPACE and an upper bound of 2EXPSPACE, "closing the gap" is left open.

Kaiser, Kroening, and Wahl [8] study safety verification of multithreaded *non-recursive* programs with local and global Boolean variables. In this model, an arbitrary number of non-recursive threads execute over shared global state, but each thread can maintain local state in Boolean variables. Although their paper does not provide an explicit complexity bound, a lower bound of EXPSPACE and an upper bound of 2EXPSPACE can be derived from a reduction from Petri net coverability and their algorithm respectively.

Interestingly, when we restrict the models to disallow either context switches (i.e., each thread runs atomically to completion) or local state in the form of the pushdown stack or local variables (but allow arbitrary context switches), safety verification is in EXPSPACE [1, 7].

Thus, the complexity gap asks whether or not the combination of *local state* (maintained in local variables or in the stack) and bounded *context switching* provides additional power to computation. In this paper, we show that indeed it does. In fact, the combination of local state and just *one* context switch is sufficient to achieve 2EXPSPACE lower bounds for these problems. This closes the complexity gap.

We believe the constructions and models that we use along the way are of independent interest. We introduce *transducer-defined* Petri nets (TDPNs), a succinct representation for Petri nets. The places in a TDPN are encoded using words over a fixed alphabet, and the transitions are described by length-preserving transducers. We show that coverability for TDPNs is 2EXPSPACE-complete[1] and give a polynomial-time reduction from coverability for TDPNs to safety verification for DCPS with one context switch.

The idea of the latter reduction is to map a (compressed) place to the stack of a thread and a marking to the set of currently spawned threads. A key obstacle in the simulation is to "transfer" potentially exponential amount of information from before a transition to after it through a polynomial-sized global store. We present a "guess and verify" procedure, using non-determinism and the use of additional threads to verify a stack content letter-by-letter.

In order to show 2EXPSPACE-hardness for TDPNs, we introduce the model of *recursive net programs* (RNPs), which add the power of making possibly recursive procedure calls to the model of net programs (i.e., programs with access to Petri net counters). The addition of recursion enables us to replace the "copy and paste code" idea in Lipton's construction to show EXPSPACE-hardness of Petri net coverability [13] with a more succinct and cleaner program description where the copies are instead represented by different values of the local variables of the procedures. The net effect is to push the requirement for copies into the call stack of the RNP while maintaining a syntax which gives us a RNP which is polynomial in the

[1] After submitting this work, the authors were made aware of "(level 1) counter systems with chained counters" from [3], for which 2EXPSPACE-hardness of state reachability is shown in [3, Theorem 14]. The 2EXPSPACE-hardness of coverability in TDPN could also be deduced from that result.

size of a given counter program. When the stack size is bounded by an exponential function of the size of the program, we get a 2EXPSPACE-lower bound. We show that recursive net programs with exponentially large stacks can be simulated by TDPNs.

Finally, we note that the 2EXPSPACE lower bound holds for DCPS where each stack is bounded by a linear function of the size. Such stacks can be encoded by polynomially many local Boolean variables, giving us a 2EXPSPACE lower bound for the model of Kaiser et al.

In summary, we introduce a number of natural 2EXPSPACE-complete problems and, through a series of reductions, close an exponential gap in the complexity of safety verification for multithreaded recursive programs.

2 Dynamic Networks of Concurrent Pushdown Systems (DCPS)

In this section, we define the model of DCPS and then state our main result. Intuitively, a DCPS consists of a finite state control and several pushdown threads with local configurations, one of them being the active thread. A local configuration contains the number of context switches the thread has already performed, as well as the contents of its local stack. An action of a thread may specify a new thread with initially one symbol on the stack to be spawned as an inactive thread. The active thread can be switched out for one of the inactive threads at any time. When a thread is switched out, its context switch number increases by one. One can view this model as a collection of dynamically created recursive threads (with a call stack each), that communicate using some finite shared memory (the state control).

A *multiset* $\mathbf{m}\colon S \to \mathbb{N}$ over a set S maps each element of S to a natural number. Let $\mathbb{M}[S]$ be the set of all multisets over S. We treat sets as a special case of multisets where each element is mapped onto 0 or 1. We sometimes write $\mathbf{m} = [\![a_1, a_1, a_3]\!]$ for the multiset $\mathbf{m} \in \mathbb{M}[S]$ such that $\mathbf{m}(a_1) = 2$, $\mathbf{m}(a_3) = 1$, and $\mathbf{m}(a) = 0$ for each $a \in S \backslash \{a_1, a_3\}$. The empty multiset is denoted $\emptyset$. The *size* of a multiset $\mathbf{m}$, denoted $|\mathbf{m}|$, is given by $\sum_{a \in S} \mathbf{m}(a)$. Note that this definition applies to sets as well.

Given two multisets $\mathbf{m}, \mathbf{m}' \in \mathbb{M}[S]$ we define $\mathbf{m} \oplus \mathbf{m}' \in \mathbb{M}[S]$ to be a multiset such that for all $a \in S$, we have $(\mathbf{m} \oplus \mathbf{m}')(a) = \mathbf{m}(a) + \mathbf{m}'(a)$. We also define the natural order $\preceq$ on $\mathbb{M}[S]$ as follows: $\mathbf{m} \preceq \mathbf{m}'$ iff there exists $\mathbf{m}^{\Delta} \in \mathbb{M}[S]$ such that $\mathbf{m} \oplus \mathbf{m}^{\Delta} = \mathbf{m}'$. We also define $\mathbf{m} \ominus \mathbf{m}'$ for $\mathbf{m}' \preceq \mathbf{m}$ analogously: for all $a \in S$, we have $(\mathbf{m} \ominus \mathbf{m}')(a) = \mathbf{m}(a) - \mathbf{m}'(a)$.

A *Dynamic Network of Concurrent Pushdown Systems (*DCPS*)* $\mathcal{A} = (G, \Gamma, \Delta, g_0, \gamma_0)$ consists of a finite set of *(global) states* G, a finite alphabet of *stack symbols* Γ, an *initial state* $g_0 \in G$, an *initial stack symbol* $\gamma_0 \in \Gamma$, and a finite set of *transition rules* Δ. Elements of Δ have one of the two forms (1) $g|\gamma \hookrightarrow g'|w'$, or (2) $g|\gamma \hookrightarrow g'|w' \triangleright \gamma'$, where $g, g' \in G$, $\gamma, \gamma' \in \Gamma$, $w' \in \Gamma^*$, and $|w'| \leq 2$. Rules of the first kind allow the DCPS to take a single step in one of the pushdown threads while the second additionally spawn a new thread with top of stack symbol γ'. The *size* of $\mathcal{A}$ is defined as $|\mathcal{A}| = |G| + |\Gamma| + |\Delta|$.

The set of *configurations* of $\mathcal{A}$ is $G \times (\Gamma^* \times \mathbb{N}) \times \mathbb{M}[\Gamma^* \times \mathbb{N}]$. Given a configuration $\langle g, (w, i), \mathbf{m}\rangle$, we call g the *(global) state*, (w, i) the *local configuration* of the *active thread*, and $\mathbf{m}$ the multiset of the *local configurations* of the *inactive threads*. The initial configuration of $\mathcal{A}$ is $\langle g_0, (\gamma_0, 0), \emptyset\rangle$. For a configuration c of $\mathcal{A}$, we will sometimes write $c.g$ for the state of c and $c.\mathbf{m}$ for the multiset of threads of c (both active and inactive). The *size* of a configuration $c = \langle g, (w, i), \mathbf{m}\rangle$ is defined as $|c| = |w| + \sum_{(w',j) \in \mathbf{m}} |w'|$.

For $i \in \mathbb{N}$ we define the relation $\Rightarrow_i = \rightarrow_i \cup \mapsto_i$ on configurations of $\mathcal{A}$, where $\rightarrow_i$ and $\mapsto_i$ are defined as follows:

- $\langle g, (\gamma.w, i), \mathbf{m}\rangle \rightarrow_i \langle g', (w'.w, i), \mathbf{m}'\rangle$ for all $w \in \Gamma^*$ iff (1) there is a rule $g|\gamma \hookrightarrow g'|w' \in \Delta$ and $\mathbf{m}' = \mathbf{m}$ or (2) there is a rule $g|\gamma \hookrightarrow g'|w' \triangleright \gamma' \in \Delta$ and $\mathbf{m}' = \mathbf{m} \oplus [\![(\gamma', 0)]\!]$.

- $\langle g, (w, i), \mathbf{m} \oplus [\![(w', j)]\!]\rangle \mapsto_i \langle g, (w', j), \mathbf{m} \oplus [\![(w, i+1)]\!]\rangle$ for all $j \in \mathbb{N}$, $g \in G$, $\mathbf{m} \in \mathbb{M}[\Gamma^* \times \mathbb{N}]$, and $w, w' \in \Gamma^*$.

For $b \in \mathbb{N}$ we define the relation $\Rightarrow_{\leq b} := \bigcup_{i=0}^{b} \Rightarrow_i$. We use $\Rightarrow_i^*$ and $\Rightarrow_{\leq b}^*$ to denote the reflexive, transitive closure of $\Rightarrow_i$ and $\Rightarrow_{\leq b}$, respectively.

Given $K \in \mathbb{N}$, a state g of $\mathcal{A}$ is *K-bounded reachable* iff $\langle g_0, (\gamma_0, 0), \emptyset\rangle \Rightarrow_{\leq K}^* \langle g, (w, i), \mathbf{m}\rangle$ for some $(w, i) \in \Gamma^* \times \{0, \ldots, K\}$ and $\mathbf{m} \in \mathbb{M}[\Gamma^* \times \{0, \ldots, K+1\}]$.

Intuitively, a local configuration (w, i) describes a pushdown thread with stack content w that has already performed i context switches. The relation $\to_i$ corresponds to applying the two kinds of transition rules at i context switches. Both of them define pushdown transitions, which the active thread can perform. Type (2) also spawns a new inactive pushdown thread with 0 context switches, whose initial stack content consists of a single specified symbol. For each $i \in \mathbb{N}$, the relation $\mapsto_i$ corresponds to switching out the active thread and raising its number of context switches from i to $i+1$, while also switching in a previously inactive thread. For a fixed K, the *K-bounded state reachability problem* (SRP[K]) for a DCPS is :

Input A DCPS $\mathcal{A}$ and a global state g

Question Is g K-bounded reachable in $\mathcal{A}$?

This corresponds to asking whether the global state g is reachable if each thread can perform at most K context switches.

▶ **Theorem 1** (Main Result). *For each $K \geq 1$, the problem* SRP[K] *is* 2EXPSPACE*-complete.*

The fact that SRP[K] is in 2EXPSPACE for any fixed K follows from the results of Atig et al. [1]. They use a slightly different variant of DCPS. However, it is possible to show a reduction from SRP[K] for our variant to SRP[$K+2$] for theirs.

Our main result is to show 2EXPSPACE-hardness for SRP[1]. One may also adapt the results of Atig et al. to the problem where K is part of the input (encoded in unary), to derive an EXPSPACE lower bound and a 2EXPSPACE upper bound. Our result immediately implies 2EXPSPACE-hardness for this problem as well.

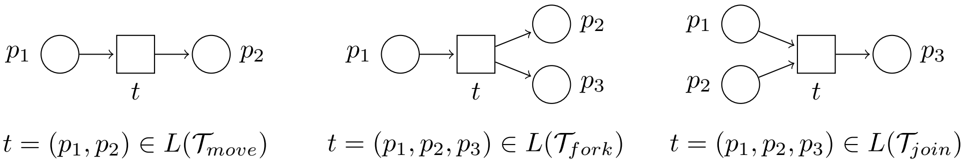
In the remaining sections we prove the lower bound in Theorem 1. In Section 3, we introduce *transducer-defined Petri nets* (TDPN), a succinct representation for Petri nets for which we prove the coverability problem is 2EXPSPACE-complete. Then, we show a reduction from the coverability problem for TDPNs to the SRP[1] problem. In Section 4, we prove hardness for coverability of TDPNs, completing the proof.

3 Transducer Defined Petri Nets (TDPN)

In this section, we prove the lower bound in Theorem 1 by reducing coverability for a succinct representation of Petri nets, namely TDPN, to SRP[1] for DCPS. We first recall some definitions about Petri nets, transducers and problems related to them.

▶ **Definition 2.** *A **Petri net** is a tuple $N = (P, T, F, p_0, p_f)$ where P is a finite set of* places, *T is a finite set of* transitions *with $T \cap P = \varnothing$, $F \subseteq (P \times T) \cup (T \times P)$ is its* flow relation, *and $p_0 \in P$ (resp. $p_f \in P$) its* initial place *(resp.* final place*). A* marking *of N is a multiset $\mathbf{m} \in \mathbb{M}[P]$. For a marking $\mathbf{m}$ and a place p we say that there are $\mathbf{m}(p)$* tokens *on p. Corresponding to the initial (resp. final) place we have the initial marking $\mathbf{m}_0 = [\![p_0]\!]$ (resp. final marking $\mathbf{m}_f = [\![p_f]\!]$). The* size *of N is defined as $|N| = |P| + |T|$.*

A transition $t \in T$ is enabled at a marking $\mathbf{m}$ if $\{p \mid (p, t) \in F\} \preceq \mathbf{m}$. If t is enabled in $\mathbf{m}$, t can be fired, which leads to a marking $\mathbf{m}'$ with $\mathbf{m}' = \mathbf{m} \oplus \{p \mid (t, p) \in F\} \ominus \{p \mid (p, t) \in F\}$. In this case we write $\mathbf{m} \xrightarrow{t} \mathbf{m}'$. A marking $\mathbf{m}$ is coverable *in N if there is a sequence $\mathbf{m}_0 \xrightarrow{t_1} \mathbf{m}_1 \xrightarrow{t_2} \ldots \xrightarrow{t_l} \mathbf{m}_l$ such that $\mathbf{m} \preceq \mathbf{m}_l$. We call such a sequence a* run *of N.*



$t = (p_1, p_2) \in L(\mathcal{T}_{move})$ $t = (p_1, p_2, p_3) \in L(\mathcal{T}_{fork})$ $t = (p_1, p_2, p_3) \in L(\mathcal{T}_{join})$

Figure 1 The types of transitions defined by the three transducers.

The *coverability problem* for Petri nets is defined as:
Input A Petri net N.
Question Is $\mathbf{m}_f$ coverable in N?

▶ **Definition 3.** *For $n \in \mathbb{N}$, a* (length preserving) ***n-ary transducer*** $\mathcal{T} = (\Sigma, Q, q_0, Q_f, \Delta)$ *consists of an alphabet Σ, a finite set of* states Q*, an* initial state $q_0 \in Q$*, a set of* final states $Q_f \subseteq Q$*, and a* transition relation $\Delta \subseteq Q \times \Sigma^n \times Q$*. For a* transition $(q, a_1, \dots, a_n, q') \in \Delta$ *we also write $q \xrightarrow{(a_1,\dots,a_n)} q'$. The size of $\mathcal{T}$ is defined as $|\mathcal{T}| = n \cdot |\Delta|$.*

The language *of $\mathcal{T}$ is the n-ary relation $L(\mathcal{T}) \subseteq (\Sigma^*)^n$ containing precisely those n-tuples $(w_1, \dots, w_n)$, for which there is a transition sequence $q_0 \xrightarrow{(a_{1,1},\dots,a_{n,1})} q_1 \xrightarrow{(a_{1,2},\dots,a_{n,2})} \dots \xrightarrow{(a_{1,m},\dots,a_{n,m})} q_m$ with $q_m \in Q_f$ and $w_i = a_{i,1}a_{i,2}\cdots a_{i,m}$ for all $i \in \{1, \dots, n\}$. Such a transition sequence is called an* accepting run *of $\mathcal{T}$.*

We note that in the more general (i.e. non-length-preserving) definition of a transducer, the transition relation Δ is a subset of $Q \times (\Sigma \cup \varepsilon)^n \times Q$. All transducers we consider in this paper are length-preserving.

▶ **Definition 4.** *A* ***transducer-defined Petri net*** $\mathcal{N} = (w_{init}, w_{final}, \mathcal{T}_{move}, \mathcal{T}_{fork}, \mathcal{T}_{join})$ *consists of two words $w_{init}, w_{final} \in \Sigma^l$ for some $l \in \mathbb{N}$, a binary transducer $\mathcal{T}_{move}$ and two ternary transducers $\mathcal{T}_{fork}$ and $\mathcal{T}_{join}$. Additionally, all three transducers share Σ as their alphabet. This defines an* explicit *Petri net $N(\mathcal{N}) = (P, T, F, p_0, p_f)$:*

- $P = \Sigma^l$.
- *T is the disjoint union of T_{move}, T_{join} and T_{fork}*[2] *where*
  - $T_{move} = \{(w, w') \in \Sigma^l \times \Sigma^l \mid (w, w') \in L(\mathcal{T}_{move})\}$,
  - $T_{fork} = \{(w, w', w'') \in \Sigma^l \times \Sigma^l \times \Sigma^l \mid (w, w', w'') \in L(\mathcal{T}_{fork})\}$, *and*
  - $T_{join} = \{(w, w', w'') \in \Sigma^l \times \Sigma^l \times \Sigma^l \mid (w, w', w'') \in L(\mathcal{T}_{join})\}$.
- *$p_0 = w_{init}$ and $p_f = w_{final}$.*
- $\forall t \in T$:
  - *If $t = (p_1, p_2) \in T_{move}$ then $(p_1, t), (t, p_2) \in F$.*
  - *If $t = (p_1, p_2, p_3) \in T_{fork}$ then $(p_1, t), (t, p_2), (t, p_3) \in F$.*
  - *If $t = (p_1, p_2, p_3) \in T_{join}$ then $(p_1, t), (p_2, t), (t, p_3) \in F$.*

An accepting run of one of the transducers, which corresponds to a single transition of N, is called a transducer-move*. The size of $\mathcal{N}$ is defined as $|\mathcal{N}| = l + |\mathcal{T}_{move}| + |\mathcal{T}_{fork}| + |\mathcal{T}_{join}|$.*

A Petri net defined by transducers in this way can only contain three different types of transitions, each type corresponding to one of the three transducers. These transition types are depicted in Figure 1. The *coverability problem* for TDPN is given by:
Input A TDPN $\mathcal{N}$.
Question Is $\mathbf{m}_f = [\![w_{final}]\!]$ coverable in the corresponding explicit Petri net $N(\mathcal{N})$?

[2] Note that a tuple $(w, w', w'') \in T_{join}$ is different from the same tuple in T_{fork}. In the interest of readability, we have chosen not to introduce a 4^{th} coordinate to distinguish the two.

Observe that the exlicit Petri net $N(\mathcal{N})$ has $|\Sigma|^l$ places, which is exponential in the size of $\mathcal{N}$. This means that TDPN are exponentially succinct representations of Petri nets.

It is a common theme in complexity theory to consider succinct versions of decision problems [12, 6, 15]. The resulting complexity is usually one exponent higher than the original version. In fact, certain types of hardness proofs can be lifted generically [15] (but such a simple argument does not seem to apply in our case). The hardness proof in the following is deferred to Section 4.

▶ **Theorem 5.** *The coverability problem for* TDPN *is* 2EXPSPACE*-complete.*

Traditionally, succinct versions of graphs and automata feature a compression using circuits [6, 15] or formulas [12]. One could also compress Petri nets by using circuits to accept binary encodings of elements (p, t) or (t, p) of the flow relation. It is relatively easy to reduce coverability for TDPN to this model by encoding transitions t as the pair or triple of places that they correspond to, yielding 2EXPSPACE-hardness. We consider transducers because they make the reduction to DCPS more natural. 2EXPSPACE-membership for any such representation follows by first unravelling the Petri net and then checking coverability [17].

We now show that coverability for TDPN can be reduced in polynomial time to SRP[1] for DCPS. The goal of the reduction is, given a TDPN $\mathcal{N} = (w_{init}, w_{final}, \mathcal{T}_{move}, \mathcal{T}_{fork}, \mathcal{T}_{join})$, to produce a DCPS $\mathcal{A}(\mathcal{N})$ with a global state *halt* such that w_{final} is coverable in $\mathcal{N}$ iff *halt* is 1-bounded reachable in $\mathcal{A}(\mathcal{N})$. We outline the main ideas and informally explain the solution to some technical issues that arise.

Representation of Markings. The main idea behind the simulation of a TDPN $\mathcal{N}$ by a DCPS $\mathcal{A}(\mathcal{N})$ is that a token on a place w of $\mathcal{N}$ is represented by a thread with stack content w. Extending this idea, a marking is represented by a multiset of threads, one for each token.

Initialization. The initial marking of $\mathcal{N}$ is $[\![w_{init}]\!]$ and $\mathcal{A}(\mathcal{N})$ starts by going into a special state where it always fills its stack with w_{init} and then moving to a global state *main*. We need $O(l)$ states in the global memory for the initialization.

Simulation of one Transducer-move. In the sequel, we explain the simulation of a single transducer-move from $\mathcal{T}_{move}$; the changes required to be made in the case of $\mathcal{T}_{join}$ and $\mathcal{T}_{fork}$ are explained at the end. Remembering the choice of transducer incurs a multiplicative cost of 3 in the global memory. The transducer-move requires us to do two things: Read the stack contents of a particular *input* thread which corresponds to a place w from which a token is removed; after which we need to create an *output* thread which corresponds to a place w' to which a token is added. This results in the following issue regarding input threads:

Issue 1: How can an input thread communicate its stack content w which comes from an exponentially large space of possibilities (since this space is Σ^l) given the requirement for the global state space to be polynomial in $|\mathcal{N}|$?

Solution 1: We pop the contents of the thread while simultaneously spawning *bit-threads*, each of which contains one letter of w along with the index $i \in \{1, \ldots, l\}$ of the letter and the information that w is a place from which a token is being removed; all of which is coded into a single *bit-symbol.*

Note that we have two types of threads: bit-threads and token-threads (i.e., those whose stack contents encode a token's position). Moreover, these two types of threads have disjoint sets of stack symbols: bit-symbols and *token-symbols.* The idea used to solve Issue 1 and read the stack contents, cannot be used in reverse to create an output token-thread since it is not possible to populate a stack with information from bit-threads.

Issue 2: How do we ensure the creation of appropriate output threads?

Solution 2: We implement a "guess-and-verify" procedure whereby we first guess the contents of an output token-thread while simultaneously producing bit-threads corresponding to w'; this is followed by a verification of the transition by comparing bit-threads produced corresponding to w and w', in a bit-by-bit fashion.

In particular, our simulation of a single transducer-move corresponds to a loop on the global state *main* which is broken up into three stages: Read, guess and verify. The implementation of this loop ensures that a configuration c of $\mathcal{A}(\mathcal{N})$ where $c.g = main$ has a multiset $c.\mathbf{m}$ of threads faithfully representing a marking $\tilde{\mathbf{m}}$ of $\mathcal{N}$ in that $c.\mathbf{m}$ contains exactly $\tilde{\mathbf{m}}(w'')$ token-threads with stack content w'' for each place w'' of $\mathcal{N}$ and no other threads.

We note that the discussion so far shows how the run of a $\mathcal{N}$ can be simulated when the schedule switches contexts at appropriate times. We must also ensure that new behaviors cannot arise due to context switches at arbitrary other points. We accomplish this by using global *locks* that ensure unwanted context switches get stuck.

Issue 3: How do we control the effect of arbitrary context switches?

Solution 3: The global state is partitioned in such a way as to only enable operations on bit-symbols while in some states and token-symbols in others. We ensure that for every bit-symbol γ, there is at most one thread with top of stack γ at any given time. Thus with the help of global control, we make sure unwanted context switches to bit-threads get the system stuck. The problem reduces to avoiding unwanted context switches between token-threads.

We use a *locking mechanism.* We add an extra $\top$ symbol at the top of every token-thread when it is first created. A read-stage always begins in a special state used for unlocking a thread (i.e. removing $\top$). While reading a particular thread, the global state disallows any transition on $\top$ or bit-symbols. Since all inactive token-threads have $\top$ as the top of stack symbol, this implies that the system cannot proceed until it switches back to the unlocked token-thread. Similarly, during the guess-stage where we are creating a new token-thread, transitions are disallowed on $\top$ and bit-symbols. The verify-stage only operates on bit-threads and switching to a token-thread is similarly pointless.

We now describe the three stages. Recall that the global state keeps the information that the current step is a transducer-move from $\mathcal{T}_{move}$.

Read-stage. We non-deterministically switch to a token-thread t_0 containing w as stack content, which we need to read. As explained earlier, we produce bit-threads decorated appropriately and at the end of this stage, we have popped all of t_0 and created l bit-threads; t_0 ceases to exist. The number of global states required in the stage is $O(l)$.

Guess-stage. Next, we create a new token-thread with w' as its stack contents by non-deterministic guessing, simultaneously spawning bit-threads for each letter of w'. At the end of this stage l more bit-threads have been added to the task buffer (for a total of $2l$ bit-threads) along with a token-thread containing w'. As in the read-stage, the number of global states used in this stage is $O(l)$.

Verify-stage. We guess a sequence of transitions $\delta_1 \ldots \delta_l$ of $\mathcal{T}_{move}$ on-the-fly; we guess δ_i which must be of the form $q_{i-1} \xrightarrow{(w_i, w_i')} q_i$ where w_i (resp. w_i') the i^{th} letter of w (resp. w'). We verify our guess by comparing each δ_i with the corresponding bit-threads b_i, b_i' with index i produced in the read-stage from w, w' respectively, before moving on to δ_{i+1}. During the verification, the bit-threads are *killed.* We enforce the condition that the target state of δ_i matches the source state of δ_{i+1}.

Claim. Killing a bit-thread t' with a single stack symbol γ' can be simulated by a DCPS. Consider the following sequence of operations starting from global state g with an active thread t which contains only one symbol γ on the stack:

1. Spawn a thread t'' with a special symbol γ_{spawn} and move to a special kill-state *kill* which contains information regarding the state g prior to the kill operation and stack symbols of t and t'.
2. Switch to a thread with symbol γ' and pop its contents while moving to a special state *return* which is forwarded the information contained in *kill*.
3. Switch to the thread with γ_{spawn} as top of stack and replace it with γ and at the same time go to global state g.

This concludes our proof sketch of the claim. Adding a kill operation to a DCPS only incurs a polynomial increase in the size of the DCPS.

In our setting, the net result of the sequence of operations simulating a kill-move is to remove the two bit-threads b_i, b_i' from the multiset of threads without changing the global state or the top of stack symbol γ. The special states *kill* (resp. *return*) ensure that if one switches to a thread whose top of stack is different from γ' in Step 2 (resp. γ_{spawn} in Step 3), no transition can be made. We return to our discussion regarding the sequence of transitions δ_i.

Since this process of checking the transducer-move occurs bit-by-bit, we require $O(l|\mathcal{T}_{move}|)$ many global states in this stage. At the end of the verification process, $\mathcal{A}(\mathcal{N})$ is once again in state *main* and the new multiset is the result of the addition of a w' thread and removal of the w thread from the old multiset of threads. We can now simulate the next transducer-move.

Checking for Coverability. At any point when $\mathcal{A}(\mathcal{N})$ is in the state *main*, it makes a non-deterministic choice between simulating the next transducer-move or checking for coverability. In the latter case, it goes into a special *check* state where the active thread is compared letter by letter with w_{final} in a process similar to initialization. At the end of the checking process, $\mathcal{A}(\mathcal{N})$ reaches the state *halt*. If the check fails at any intermediate point, $\mathcal{A}(\mathcal{N})$ terminates without reaching the *halt* state. We require a further $O(l)$ states for checking coverability.

Fork and Join. We have shown above how a single transducer-move is simulated assuming that it is a transducer-move from $\mathcal{T}_{move}$. In general, the transducer-move could be from $\mathcal{T}_{join}$ or $\mathcal{T}_{fork}$ as well. In these two cases, we have triples of the form (w, w', w'') accepted by the transducer. However, in the former, we read w, w' and guess w'' while in the latter, we read w and guess w', w''. In the case of $\mathcal{T}_{join}$, once we have read w, we non-deterministically switch to a thread containing w' as its contents. Whenever the threads picked during the read-stage and the threads created during the guess-stage do not agree with the guessed transitions of the transducer-move, we encounter a problem during the verify-stage and $\mathcal{A}(\mathcal{N})$ terminates without reaching the *halt* state.

Context Switches. Every thread (other than the initial one for w_{init}) is created during the guess-stage and then switched out once. The next time it is switched in, it is read and ceases to exist. This implies that there exists a run of $\mathcal{A}(\mathcal{N})$ simulating a run of $\mathcal{N}$ where every thread undergoes at most one context switch. Conversely, we show that a run of $\mathcal{A}(\mathcal{N})$ reaching *halt* where every thread is bounded by at most 1 context switch implies the existence of a run in $\mathcal{N}$ which covers the final marking as desired.

This concludes our overview of the construction of $\mathcal{A}(\mathcal{N})$ and completes the reduction of coverability for TDPN to SRP[1] for DCPS. The global memory is polynomial in the size of $\mathcal{N}$. Similarly, the stack alphabet is expanded to include $O(l \cdot |\Sigma|)$ bit symbols, hence the alphabet of $\mathcal{A}(\mathcal{N})$ is polynomial as well. In summary, $\mathcal{A}(\mathcal{N})$ can be produced in time polynomial in the size of the input.

▶ Remark 6. Our lower bound holds already for DCPS where the stack of each thread is bounded by a linear function of the size of the DCPS. Thus, as a corollary, we get 2EXPSPACE-hardness for a related model in which each thread is a *Boolean program*, i.e., where each thread has its stack bounded by a constant but has a polynomial number (in the size of $|G| + |\Gamma| + |\Delta|$) of local Boolean variables. This closes the gap from [8] as well as other similar models studied in the literature [2, 9, 4].

4 Recursive Net Programs (RNP)

We prove Theorem 5 by adapting the Lipton construction [13], as it is explained in [5], to our succinct representation of Petri nets. Our construction requires two steps. First we reduce termination for bounded counter programs to termination for *Petri net* programs which do not allow zero tests. Second, we reduce termination of net programs with to coverability for TDPN.

For the first step, we have to show how we can simulate the operation of a bounded counter program with one without zero tests. In the Lipton construction, this is achieved by constructing a gadget that performs zero tests for counters bounded by some bound B. These gadgets are obtained by transforming a gadget for bound B into a gadget for B^2. Starting with $B = 2$ and applying this transformation n times leads to a gadget for $B = 2^{2^n}$. One then has to argue that the resulting net program still has linear size in the parameter n. For a 2EXPSPACE lower bound, one would need to simulate a program where the bound is triply exponential in n. A naive implementation of the gadget would then lead to a program with triply exponential counter values, but exponential program size in n.

In order to argue later that the resulting program can be encoded in a small TDPN, we will present the Lipton construction in a different way. Instead of growing the program with every gadget transformation, we implement the gadgets recursively using a stack. We call these programs *recursive net programs* (RNP). This way, when we instantiate the model for a triply exponential bound on the counters (to get 2EXPSPACE-hardness instead of EXPSPACE-hardness), the resulting programs still have polynomial size control flow. Note that at run time, such programs can have an exponentially deep stack; however, this very large stack does not form part of the program description. We shall show that RNP have a natural encoding as TDPN.

For the second step, we reduce termination for RNP to coverability for TDPN. To this end, we borrow some techniques from the original construction to translate an RNP into an exponential sized Petri net. We then assign binary addresses to its places and construct transducers for those pairs and triples that correspond to transitions. This results in a TDPN of polynomial size. Finally, we argue that we do not need the whole exponential sized Petri net to reason about the transducers, and that just a polynomial size part suffices. This then gives us a polynomial time procedure.

4.1 From Bounded Counter Programs to RNP

Bounded Counter Programs. A *counter program* is a finite sequence of *labelled commands* separated by semicolons. Let l, l_1, l_2 be *labels* and x be a *variable* (also called a *counter*). The labelled commands have one of the following five forms:

(1) l : **inc** x; // increment
(2) l : **dec** x; // decrement
(3) l : **halt**
(4) l : **goto** l_1; // unconditional jump
(5) l : **if** $x = 0$ **then goto** l_1 **else goto** l_2; // conditional jump

Variables can hold values over the natural numbers, labels have to be pairwise distinct, but can otherwise come from some arbitrary set. For convenience, we require each program to contain exactly one **halt** command at the very end. The *size* $|C|$ of a counter program C is the number of its labelled commands.

During execution, all variables start with initial value 0. The semantics of programs follows from the syntax, except for the case of decrementing a variable whose value is already 0. In this case, the program aborts, which is different from proper termination, i.e., the execution of the **halt** command. It is easy to see that each counter program has only one execution, meaning it is deterministic. This execution is *k-bounded* if none of the variables ever reaches a value greater than k during it.

Let $\exp^{m+1}(x) := \exp(\exp^m(x))$ and $\exp^1(x) = \exp(x) := 2^x$. The N-fold exponentially bounded *halting problem* (also called *termination*) for counter programs ($\mathsf{HP}[N]$) is given by:
Input A unary number $n \in \mathbb{N}$ and a counter program C.
Question Does C have an $\exp^N(n)$-bounded execution that reaches the **halt** command?
We make use of the following well-known result regarding this problem:

▶ **Theorem 7.** *For each $N > 0$, the problem $\mathsf{HP}[N+1]$ is N-EXPSPACE-complete.*

The proof for arbitrary N matches the proof for $N = 1$, which the Lipton construction used.

Recursive Net Programs. The definition of *recursive net programs* (RNP) also involves sequences of labelled commands separated by semicolons. Let l, l_1, l_2 be labels, x be a variable, and `proc` be a procedure name. Then the labelled commands can still have one of the previous forms (1) to (4). However, form (5) changes from a conditional to a nondeterministic jump, and there are two new forms for procedure calls:

(1) l : **inc** x; // increment
(2) l : **dec** x; // decrement
(3) l : **halt**
(4) l : **goto** l_1; // unconditional jump
(5) l : **goto** l_1 **or goto** l_2; // nondeterministic jump
(6) l : **call** `proc`; // procedure call
(7) l : **return**; // end of procedure

In addition to labelled commands, these programs consist of a finite set PROC of procedure names and also a *maximum recursion depth* $k \in \mathbb{N}$. Furthermore, they not only contain one sequence of labelled commands to serve as the main program, but also include two additional sequences of labelled commands for each procedure name `proc` $\in$ PROC. The second sequence for each `proc` is not allowed to contain any **call** commands and serves as a sort of "base case" only to be called at the maximum recursion depth. Each label has to be unique among all sequences and each jump is only allowed to target labels of the sequence it belongs to. Each RNP contains exactly one **halt** command at the end of the main program.

For $\mathtt{proc} \in \mathsf{PROC}$ let $\#c(\mathtt{proc})$ be the number of commands in both of its sequences added together and let $\#c(\mathtt{main})$ be the number of commands in the main program. Then the *size* of an RNP R is defined as $|R| = \lceil \log k \rceil + \#c(\mathtt{main}) + \sum_{\mathtt{proc} \in \mathsf{PROC}} \#c(\mathtt{proc})$.

The semantics here is quite different compared to counter programs: If the command "l : **call** proc" is executed, the label l gets pushed onto the call stack. Then if the stack contains less than k labels, the first command sequence pertaining to proc, which we now call $\mathtt{proc}_{<\max}$, is executed. If the stack already contains k labels, the second command sequence, $\mathtt{proc}_{=\max}$, is executed instead. Since $\mathtt{proc}_{=\max}$ cannot call any procedures by definition, the call stack's height (i.e. the *recursion depth*) is bounded by k. On a return command, the last label gets popped from the stack and we continue the execution at the label occurring right after the popped one.

How increments and decrements are executed depends on the current recursion depth d as well. For each variable x appearing in a command, $k+1$ copies x_0 to x_k are maintained during execution. The commands **inc** x resp. **dec** x are then interpreted as increments resp. decrements on x_d (and not x or any other copy). As before, all these copies start with value 0 and decrements fail at value 0, which is different from proper termination.

Instead of a conditional jump, we now have a nondeterministic one, that allows the program execution to continue at either label. Regarding termination we thus only require there to be at least one execution that reaches the **halt** command. This gives us the following halting problem for RNP:

Input An RNP R

Question Is there an execution of R that reaches the **halt** command?

We now adapt the Lipton construction to recursive net programs. We start with a $\exp^2(n)$-bounded counter program C with a set of counters X and construct an RNP $R(C)$ with maximum recursion depth $n+1$ that terminates iff C terminates. The number of commands in $R(C)$ will be linear in $|C|$.

Auxiliary Variables. The construction of $R(C)$ involves simulating the zero test. To this end, we introduce for each counter $x \in X$ a complementary counter $\bar{x}$ and ensure that the invariant $x_0 + \bar{x}_0 = \exp^2(n)$ always holds. We can then simulate a zero test on x by checking that $\bar{x}$ can be decremented $\exp^2(n)$ times. This requires us to implement a decrement by $\exp^2(n)$ in linearly many commands and also a similar increment to reach a value of $\exp^2(n)$ for $\bar{x}$ from its initial value 0 at the start of the program. Furthermore, we need helper variables s, $\bar{s}$, y, $\bar{y}$, z, and $\bar{z}$. We also sometimes need to increment or decrement the $(d+1)$th copy of one of these six variables at recursion level d. As an example, for incrementing s_{d+1} in this way, we define the procedure s_inc:

$$\mathtt{s\_inc}_{<\max}: \ \textbf{inc } s; \textbf{return} \qquad \mathtt{s\_inc}_{=\max}: \ \textbf{inc } s; \textbf{return}$$

The analogous procedures for $\bar{s}$, y, $\bar{y}$, z, and $\bar{z}$ are defined similarly.

Program Structure. The program $R(C)$ consists of two parts: The initial part $R_{init}(C)$, which initializes all the complementary counters as mentioned above, followed by $R_{sim}(C)$, the part that simulates C. We construct $R_{sim}(C)$ from C by replacing some of its commands. Increments of the form **inc** x are replaced by **dec** $\bar{x}$;**inc** x, decrements **dec** x are replaced by **dec** x;**inc** $\bar{x}$. Unconditional jumps and the **halt** command stay the same. Each conditional jump (form (5) for counter programs) is replaced by

$$\begin{aligned} l &: \mathrm{Test}(x, l_{\text{continue}}, l_2); \\ l_{\text{continue}} &: \mathrm{Test}(\bar{x}, l_1, l_2) \end{aligned}$$

```
Test$(x, l_{\text{zero}}, l_{\text{nonzero}})$ :

                  goto $l_{\text{nztest}}$ or goto $l_{\text{loop}}$;
  $l_{\text{nztest}}$ : dec $x$; inc $x$; goto $l_{\text{nonzero}}$;
    $l_{\text{loop}}$ : dec $\bar{x}$; inc $x$; call $\bar{\texttt{s}}$_dec; call s_inc;
                  goto $l_{\text{exit}}$ or goto $l_{\text{loop}}$;
    $l_{\text{exit}}$ : call dec; goto $l_{\text{zero}}$

Test$_{+1}(v, l_{\text{zero}}, l_{\text{nonzero}})$ :

                  goto $l_{\text{nztest}}$ or goto $l_{\text{loop}}$;
  $l_{\text{nztest}}$ : call v_dec; call v_inc; goto $l_{\text{nonzero}}$;
    $l_{\text{loop}}$ : call $\bar{\texttt{v}}$_dec; call v_inc;
                  call $\bar{\texttt{s}}$_dec; call s_inc;
                  goto $l_{\text{exit}}$ or goto $l_{\text{loop}}$;
    $l_{\text{exit}}$ : call dec; goto $l_{\text{zero}}$
```

```
dec$_{<\text{max}}$ :

   $l_{\text{outer}}$ : call y_dec; call $\bar{\texttt{y}}$_inc;
   $l_{\text{inner}}$ : call z_dec; call $\bar{\texttt{z}}$_inc;
             dec $s$; inc $\bar{s}$;
             Test$_{+1}(z, l_{\text{next}}, l_{\text{inner}})$;
    $l_{\text{next}}$ : Test$_{+1}(y, l_{\text{exit}}, l_{\text{outer}})$;
    $l_{\text{exit}}$ : return

dec$_{=\text{max}}$ :

             dec $s$; inc $\bar{s}$; dec $s$; inc $\bar{s}$;
             return
```

Figure 2 Definitions of the macros Test and Test$_{+1}$ as well as the procedure `dec`. Regarding the second macro we require $v \in \{y, z\}$.

where Test$(x, l_{zero}, l_{nonzero})$ is what we call a *macro*. We use it as syntactic sugar to be replaced by its specification for the actual construction of $R(C)$. This is in contrast to procedures, which refer to specific parts of the program that can be called to increase the recursion depth.

Test Macros and Decrement Procedure. The macro Test is specified in the left part of Figure 2. It involves a call to the procedure `dec`, which is defined in the right part of the same figure. Below Test we have also specified the variant Test$_{+1}$, which is used in `dec`. The main difference is that Test$_{+1}$ can only be invoked on variables y or z and acts on their $(d+1)$th copy at recursion depth d.

Semantically, `dec` at recursion depth d decrements s_d by $\exp^2(n+1-d)$ (and increments $\bar{s}_d$ by the same amount). Both variants of Test simulate a conditional jump and have the side effect of switching the values x_d and $\bar{x}_d$ if the tested variable x_d was 0. Because of this, every conditional jump of C gets replaced by two instances of the Test macro, where the second one reverses the potential side effect.

The decrements of procedure `dec` are performed via two nested loops that each run $\exp^2(n-d)$-times. Each of these loops uses a helper variable y_{d+1} or z_{d+1} that has to be tested for zero at the end, using the Test$_{+1}$ macro. This involves transferring the helper variable's value to s_{d+1} and then calling `dec` at the next recursion depth. Essentially, any decrement by $\exp^2(j)$ for some j is implemented using $\exp^2(j-1)$ many decrements by $\exp^2(j-1)$ via the nested loops. This iterative squaring of the value by which we decrement continues down to the base case of $\exp^2(0) = 2$.

Semantics. Our construction is semantically very similar to the Lipton construction, barring two main differences: Firstly, instead of having $n+1$ different procedure definitions of `dec` (one per level d), we only need two because of recursion. The case for the Test macros is similar, as is the case of the helper variables s, y, z and their complements. Secondly, our variable copies start with index 0 counting upwards, whereas in the Lipton construction the variables start with index n and count downwards. This means that for some index d we have the invariant $s_d + \bar{s}_d = \exp^2(n+1-d)$ in our construction, where it is $s_d + \bar{s}_d = \exp^2(d)$ for Lipton. While the invariant of the Lipton construction is simpler, ours allows us to define the recursion depth starting at 0 and going upwards, which seemed more natural for recursion.

Let us give a more precise analysis regarding the effect of the Test macros and `dec` procedure. During the execution of `dec` at recursion depth d, we begin with $s_d = \exp^2(n+1-d)$, $y_{d+1} = z_{d+1} = \exp^2(n-d)$, and $\bar{s}_d = \bar{y}_{d+1} = \bar{z}_{d+1} = 0$. The invariants $s_d + \bar{s}_d = \exp^2(n+1-d)$, $y_{d+1} + \bar{y}_{d+1} = \exp^2(n-d)$, and $z_{d+1} + \bar{z}_{d+1} = \exp^2(n-d)$ are upheld throughout. At the end we have $s_d = \bar{y}_{d+1} = \bar{z}_{d+1} = 0$, $y_{d+1} = z_{d+1} = \exp^2(n-d)$, and $\bar{s}_d = \exp^2(n+1-d)$, meaning the decrements were performed correctly and all helper variables retain their initial values. The situation is quite similar for Test and Test$_{+1}$, if the variable to be tested was initially 0. In the non-zero case, the tested variable is just decremented and incremented once, whereas no other variables are touched. All executions that differ from the described behavior are guaranteed to get stuck.

Correctness of these semantics can be proven by induction on the recursion depth. It requires the assumptions $x_0 + \bar{x}_0 = \exp^2(n)$, $v_d + \bar{v}_d = \exp^2(n+1-d)$, and $\bar{v}_d = 0$ for all $x \in X$, $v \in \{\bar{s}, y, z\}$, and $d > 0$.

Initialization. We now have to construct $R_{init}(C)$ in such a way, that it performs all the necessary increments for these assumptions to hold at the start of $R_{sim}(C)$. Since this is again achieved using iterative squaring, we omit the precise construction. It involves calling a procedure `inc` to perform the increments on copies of variables at lower recursion depths, while x_0 is incremented in the main program for each $x \in X$.

Size Analysis. To give a brief size analysis of $R(C)$, PROC contains 14 procedure names, whose corresponding definitions have constant size. For each command in C, $R_{sim}(C)$ contains constantly many commands, and $R_{init}(C)$ has linearly many commands in the size of the variable set X. Since wlog. each variable of C is involved in at least one of its commands, the amount of commands in $R(C)$ is linear in $|C|$. Here, for doubly exponential counter values, we would not even need n to be given in unary since only $\lceil \log(n+1) \rceil$ factors into the size of $R(C)$.

Handling Triply Exponential Counter Values. The exact same construction with a maximum recursion depth of $2^n + 1$ can be used to simulate a counter program with counters bounded by $\exp^3(n)$: Starting with 2 and squaring n-times yields $\exp^2(n)$, therefore squaring 2^n times instead yields $\exp^3(n)$. The correctness follows from the same inductive proofs as before. For this changed maximum recursion depth, configurations contain exponentially in n many counter values and also maintain a call stack of size up to 2^n. However, since the maximum recursion depth can be encoded in binary, its size is still polynomial in the unary encoding of n. Thus, the halting problem for recursive net programs is 2EXPSPACE-hard.

4.2 From RNP to TDPN

Figure 3 and Figure 4 show how the commands of recursive net programs can be simulated by Petri net transitions. This is again done in similar fashion to Esparza's description [5] of the Lipton construction [13]. As we can see, this involves only the three types of transitions defined by our transducers.

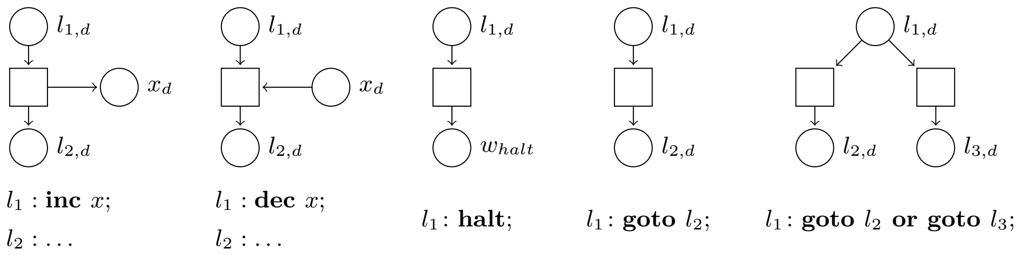


Figure 3 Petri net transitions for five of the seven command types found in recursive net programs. Here, $d \in \{0, \ldots, k\}$, where k is the maximum recursion depth.

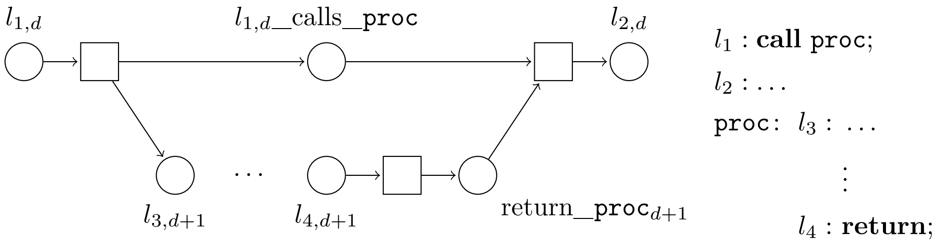


Figure 4 Petri net transitions for procedure calls found in recursive net programs. Here, $d \in \{0, \ldots, k-1\}$, where k is the maximum recursion depth.

Let us give more detail regarding the Petri net construction: Given an RNP R with maximum recursion depth k we construct a transducer-defined Petri net $\mathcal{N} = (w_{init}, w_{final}, \mathcal{T}_{move}, \mathcal{T}_{fork}, \mathcal{T}_{join})$, which defines the Petri net $N(\mathcal{N}) = (P, T, F, p_0, p_f)$, such that $[\![p_f]\!]$ is coverable in $N(\mathcal{N})$ iff there is a terminating execution of R. We begin by arguing about the shape of $N(\mathcal{N})$ and then construct our transducers afterwards.

The idea is for $N(\mathcal{N})$ to start with one place per variable and one place per label, as well as one auxiliary place for each **call** command and each proc $\in$ PROC, which can be seen in Figure 4. Additionally, there is also a single auxiliary place w_{halt} for the **halt** command. Let the number of all these places be h. Then each such place gets copied $k+1$ times, so that a copy exists for each possible recursion depth. Transitions get added at each recursion depth d according to Figure 3 and Figure 4, whereas some transitions in the latter also connect to places of recursion depth $d+1$.

Regarding the transducers, we use the alphabet $\{0, 1\}$. Every place address $w = u.v$ has a prefix u of length $\lceil \log h \rceil$ and a postfix v of length $\lceil \log k \rceil$. We assign each of the h places that $N(\mathcal{N})$ started with a number from 0 to $h-1$. The binary representation of this number (with leading zeros) is used for the u-part of its address. For the v-part, we use the binary representation of the recursion depth d (also with leading zeros), that a particular copy of this place corresponds to. The address of the place corresponding to the first label in the main program at recursion depth 0 is used for w_{init}, whereas the one corresponding to w_{halt} at recursion depth 0 is used for w_{final}.

To accept a particular pair or triple of addresses as a transition, each of the three transducers distinguishes between all possibilities regarding the u-parts. Any pair or triple of $\lceil \log h \rceil$-length words that matches a particular transition of the right type (move, fork, join) has a unique path in the transducer, while all non-matching pairs or triples do not. Then for the v-parts, the transducer needs to either check for equality, if all places correspond to the same recursion depth, or for one binary represented number to be one higher. Since it is clear from the u-parts, whether the recursion depths should all match or not, we can just connect the unique paths to the correct part of the transducer at the end.

The transducer parts for the first $\lceil \log h \rceil$ bits require to distinguish between up to $2^{3 \log h} = 8h$ possibilities, meaning they require polynomially in h many states. The parts for the last $\lceil \log k \rceil$ bits can easily be constructed using polynomially many states in $\log k$. Since h is linear in the number of commands in R and $\lceil \log k \rceil$ is the size of the binary encoding of the maximum recursion depth, $\mathcal{N}$ is of polynomial size compared to R. Because we can construct $\mathcal{N}$ by first constructing $N(\mathcal{N})$ without the copies for each recursion depth, this is feasible in polynomial time. Thus, the coverability problem for transducer-defined Petri nets is 2EXPSPACE-hard.

5 Discussion

The chain of reductions in Sections 3 and 4 complete the 2EXPSPACE lower bound for 1-bounded reachability for DCPS. In fact, an inspection of the reductions show a technical strengthening: the 2EXPSPACE lower bound already holds for SRP[1] of DCPS which satisfy two additional properties, *boundedness* and *local termination*.

▶ **Definition 8.** *A* DCPS $\mathcal{A}$ *is said to be* bounded *if there is a global bound* $B \in \mathbb{N}$ *on the size of every configuration of every run of* $\mathcal{A}$*. It is* locally terminating *if every infinite run of* $\mathcal{A}$ *contains infinitely many context switches.*

Consider the chain of reductions from the halting problem for bounded counter programs to RNP to TDPN to SRP[1]. The configurations of the counter programs, by definition, are bounded by a triply-exponential bound on the parameter n. This bound translates to bounds on the RNP and TDPN instances. In particular, the number of places in the TDPN produced in the reduction is exponentially bounded in n and the number of tokens on these places is triple-exponentially bounded in n. The DCPS constructed from the TDPN uses the stack of a thread to store an address of a place; thus, the height of a stack is bounded by a polynomial in n. In addition, since the number of tokens in the TDPN correspond to the number of in-progress threads in the DCPS, this implies a triple exponential (in n) bound on the number of threads in any execution of the DCPS. Thus, the size of every configuration in every run of the DCPS is bounded.

Second, the rules of the constructed DCPS do not allow any one thread to run indefinitely. In other words, any non-terminating run of the DCPS must involve infinitely many threads and the run contains infinitely many context switches.

▶ **Theorem 9.** *The* SRP[1] *problem for bounded, locally terminating* DCPS *is* 2EXPSPACE-*hard.*

References

1 Mohamed Faouzi Atig, Ahmed Bouajjani, and Shaz Qadeer. Context-bounded analysis for concurrent programs with dynamic creation of threads. In *Tools and Algorithms for the Construction and Analysis of Systems, 15th International Conference, TACAS 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings*, pages 107–123, 2009.

2 Byron Cook, Daniel Kroening, and Natasha Sharygina. Verification of Boolean programs with unbounded thread creation. *Theoretical Computer Science*, 388(1-3):227–242, 2007. doi:10.1016/j.tcs.2007.07.050.

3 Stéphane Demri, Diego Figueira, and M. Praveen. Reasoning about data repetitions with counter systems. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013*, pages 33–42, 2013.

4 Emanuele D'Osualdo, Jonathan Kochems, and C.-H. Luke Ong. Automatic verification of Erlang-style concurrency. In *Static Analysis - 20th International Symposium, SAS 2013, Seattle, WA, USA, June 20-22, 2013, Proceedings*, volume 7935 of *Lecture Notes in Computer Science*, pages 454–476. Springer, 2013.

5 Javier Esparza. Decidability and complexity of Petri net problems – an introduction. In G. Rozenberg and W. Reisig, editors, *Lectures on Petri Nets I: Basic Models. Advances in Petri Nets*, number 1491 in Lecture Notes in Computer Science, pages 374–428, 1998.

6 Hana Galperin and Avi Wigderson. Succinct representations of graphs. *Information and Control*, 56(3):183–198, 1983.

7 Pierre Ganty and Rupak Majumdar. Algorithmic verification of asynchronous programs. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 34(1):6, 2012.

8 Alexander Kaiser, Daniel Kroening, and Thomas Wahl. Dynamic cutoff detection in parameterized concurrent programs. In *22nd International Conference on Computer Aided Verification, CAV 2010, Edinburgh, UK, July 15-19, 2010, Proceedings*, pages 645–659. Springer, 2010.

9 Jonathan Kochems. *Verification of asynchronous concurrency and the shaped stack constraint.* PhD thesis, University of Oxford, UK, 2014. URL: `http://ora.ox.ac.uk/objects/uuid:cd487639-0e7f-4248-9405-e05e8a8383d5`.

10 Salvatore La Torre, Parthasarathy Madhusudan, and Gennaro Parlato. The language theory of bounded context-switching. In *LATIN 2010: Theoretical Informatics, 9th Latin American Symposium, Oaxaca, Mexico, April 19-23, 2010, Proceedings*, volume 6034 of *Lecture Notes in Computer Science*, pages 96–107. Springer, 2010.

11 Akash Lal and Thomas W. Reps. Reducing concurrent analysis under a context bound to sequential analysis. *Formal Methods in System Design*, 35(1):73–97, 2009. `doi:10.1007/s10703-009-0078-9`.

12 Ernst Leiss. Succinct representation of regular languages by Boolean automata. *Theoretical Computer Science*, 13(3):323–330, 1981. `doi:10.1016/S0304-3975(81)80005-9`.

13 Richard Lipton. The reachability problem is exponential-space hard. *Yale University, Department of Computer Science, Report*, 62, 1976.

14 Madanlal Musuvathi and Shaz Qadeer. Iterative context bounding for systematic testing of multithreaded programs. In *Proceedings of the ACM SIGPLAN 2007 Conference on Programming Language Design and Implementation, PLDI 2007, San Diego, CA, USA, June 10-13, 2007*, pages 446–455. ACM, 2007.

15 Christos H. Papadimitriou and Mihalis Yannakakis. A note on succinct representations of graphs. *Information and Control*, 71(3):181–185, 1986.

16 Shaz Qadeer and Jakob Rehof. Context-bounded model checking of concurrent software. In *Tools and Algorithms for the Construction and Analysis of Systems, 11th International Conference, TACAS 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005, Proceedings*, volume 3440 of *Lecture Notes in Computer Science*, pages 93–107. Springer, 2005.

17 Charles Rackoff. The covering and boundedness problems for vector addition systems. *Theoretical Computer Science*, 6(2):223–231, 1978.

Two Variable Logic with Ultimately Periodic Counting

Michael Benedikt
University of Oxford, UK
michael.benedikt@cs.ox.ac.uk

Egor V. Kostylev
University of Oxford, UK
egor.kostylev@cs.ox.ac.uk

Tony Tan
National Taiwan University, Taipei, Taiwan
tonytan@csie.ntu.edu.tw

Abstract

We consider the extension of FO^2 with quantifiers that state that the number of elements where a formula holds should belong to a given ultimately periodic set. We show that both satisfiability and finite satisfiability of the logic are decidable. We also show that the spectrum of any sentence is definable in Presburger arithmetic. In the process we present several refinements to the "biregular graph method". In this method, decidability issues concerning two-variable logics are reduced to questions about Presburger definability of integer vectors associated with partitioned graphs, where nodes in a partition satisfy certain constraints on their in- and out-degrees.

2012 ACM Subject Classification Theory of computation → Logic

Keywords and phrases Presburger Arithmetic, Two-variable logic

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.112

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

Related Version A full version is available at https://arxiv.org/abs/2006.01193.

Funding *Michael Benedikt*: Engineering and Physical Sciences Research Council grant EP/M005852/1
Tony Tan: Taiwan Ministry of Science and Technology under grant no. 107-2221-E-002-026-MY2 and NTU under grant no. 108L891908

Acknowledgements We are grateful to Bartosz Bednarczyk for many useful comments and suggestions on preliminary drafts of this work.

1 Introduction

In the search for expressive logics with decidable satisfiability problem, two-variable logic, denoted here as FO^2, is one yardstick. This logic is expressive enough to subsume basic modal logic and many description logics, while satisfiability and finite satisfiability coincide, and both are decidable [23, 15, 9]. However, FO^2 lacks the ability to count. Two-variable logic with counting, C^2, is a decidable extension of FO^2 that adds *counting quantifiers*. In C^2 one can express, for example, $\exists^5 x\ P(x)$ and $\forall x \exists^{\geqslant 5} y\ E(x, y)$ which, respectively, mean that there are exactly 5 elements in unary relation P, and that every element in a graph has at least 5 adjacent edges. Satisfiability and finite satisfiability do not coincide for C^2, but both are decidable [10, 16]. In [16] the problems were shown to be NEXPTIME-complete under a unary encoding of numbers, and this was extended to binary encoding in [18]. However, the numerical capabilities of C^2 are quite limited. For example, one can not express that the number of outgoing edges of each element in the graph is even.

A natural extension is to combine FO^2 with *Presburger arithmetic* where one is allowed to define collections of tuples of integers from addition and equality using boolean operators and quantifiers. The collections of k-tuples that one can define in this way are the *semi-linear sets*, and the collections of integers (when $k = 1$) definable are the *ultimately periodic sets.* Prior work has considered the addition of *Presburger quantification* to fragments of two-variable logic. For every definable set $\phi(x, y)$ and every ultimately periodic set S, one has a formula $\exists^S y\ \phi(x, y)$ that holds at x when the number of y such that $\phi(x, y)$ is in S. We let $\mathsf{FO}^2_{\mathrm{Pres}}$ denote the logic that adds this construct to FO^2.

On the one hand, the corresponding quantification over general k-tuples (allowing semi-linear rather than ultimately periodic sets) easily leads to undecidability [11, 3]. On the other hand, adding this quantification to modal logic has been shown to preserve decidability [1, 7]. Related *one-variable fragments* in which we have only a unary relational vocabulary and the main quantification is $\exists^S x\ \phi(x)$ are known to be decidable (see, e.g. [2]), and their decidability is the basis for a number of software tools focusing on integration of relational languages with Presburger arithmetic [14]. The decidability of full $\mathsf{FO}^2_{\mathrm{Pres}}$ is, to the best of our knowledge, still open [4]. There are a number of other extensions of C^2 that have been shown decidable; for example it has been shown that one can allow a distinguished equivalence relation [22] or a forest-structured relation [6, 5]. $\mathsf{FO}^2_{\mathrm{Pres}}$ is easily seen to be orthogonal to these other extensions.

In this paper we show that both satisfiability and finite satisfiability of $\mathsf{FO}^2_{\mathrm{Pres}}$ are decidable. Our result makes use of the *biregular graph method* introduced for analyzing C^2 in [13]. The method focuses on the problem of existence of graphs equipped with a partition of vertices based on constraints on the out- and in-degree. Such a partitioned graph can be characterized by the cardinalities of each partition component, and the key step in showing these decidability results is to prove that the set of tuples of integers representing valid sizes of partition components is definable by a formula in Presburger arithmetic. From this "graph constraint Presburger definability" result one can reduce satisfiability in the logic to satisfiability of a Presburger formula, and from there infer decidability using known results on Presburger arithmetic.

The approach is closely-related to the machinery developed by Pratt-Hartmann (the "star types" of [21]) for analyzing the decidability and complexity of C^2, its fragments [19], and its extensions [22, 5]. An advantage of the biregular graph approach is that it is transparent how to extract more information about the shape of witness structures. In particular we can infer that the *spectrum* of any formula is Presburger definable, where the spectrum of a formula ϕ is the set of cardinalities of finite models of ϕ. It is also interesting to note that a more restricted version of our biregular graph method is used to prove the decidability of FO^2 extended with two equivalence relations [12].

Characterising the spectrum for general first order formulas is quite a difficult problem, with ties to major open questions in complexity theory [8]. This work can be seen as a demonstration of the power of the biregular graph method to get new decidability results. We make heavy use of both techniques and results in [13], adapting them to the richer logic. We also require additional inductive arguments to handle the interaction of ordinary counting quantifiers and modulo counting quantification.

2 Preliminaries

Let $\mathbb{N} = \{0, 1, 2, \ldots\}$ and let $\mathbb{N}_\infty = \mathbb{N} \cup \{\infty\}$.

Linear and ultimately periodic sets. A set of the form $\{a + ip \mid i \in \mathbb{N}\}$, for some $a, p \in \mathbb{N}$ is a *linear set.* We will denote such a set by a^{+p}, where a and p are called the *offset* and *period* of the set, respectively. Note that, by definition, $a^{+0} = \{a\}$, which is a linear set. For convenience, we define $\emptyset$ and $\{\infty\}$ (which may be written as ∞^{+p}) to also be linear sets.

An *ultimately periodic set* (*u.p.s.*) S is a finite union of linear sets. Usually we write a u.p.s. $\{c_1\} \cup \cdots \cup \{c_m\} \cup a_1^{+p_1} \cup \cdots \cup a_n^{+p_n}$ as just $\{c_1, \ldots, c_m, a_1^{+p_1}, \ldots, a_n^{+p_n}\}$, and abusing notation, we write $a^{+p} \in S$ for a u.p.s. S if $a + ip \in S$ for every $i \in \mathbb{N}$.

Two-variable logic with ultimately periodic counting quantifiers. An *atomic formula* is either an atom $R(\vec{u})$, where R is a predicate, and $\vec{u}$ is a tuple of variables of appropriate size, or an equality $u = u'$, with u and u' variables, or one of the formulas $\top$ and $\bot$ denoting the True and False values. The logic $\mathsf{FO}^2_{\mathrm{Pres}}$ is a class of first-order formulas using only variables x and y, built up from atomic formulas and equalities using the usual boolean connectives and also *ultimately periodic counting quantification*, which is of the form $\exists^S x\ \phi$ where S is a u.p.s. One special case is where S is a singleton $\{a\}$ with $a \in \mathbb{N}_\infty$, which we write $\exists^a x\ \phi$; in case of $a \in \mathbb{N}$, these are *counting quantifiers.* The semantics of $\mathsf{FO}^2_{\mathrm{Pres}}$ is defined as usual except that, for every $a \in \mathbb{N}$, $\exists^a x\ \phi$ holds when there are *exactly* a number of x's such that ϕ holds, $\exists^\infty\ \phi$ holds when there are infinitely many x's such that ϕ holds, and $\exists^S x\ \phi$ holds when there is some $a \in S$ such that $\exists^a x\ \phi$ holds.

Note that when S is $\{\infty\} \cup 0^{+1} = \mathbb{N}_\infty$, $\exists^S x\ \phi$ is equivalent to $\top$. When S is 0^{+1}, $\exists^S x\ \phi$ semantically means that there are finitely many x such that ϕ holds. We define $\exists^\emptyset x\ \phi$ to be $\bot$ for any formula ϕ. We also note that $\exists^0 x\ \phi$ is equivalent to $\forall x\ \neg\phi$, and $\neg\exists^S x\ \phi$ is equivalent to $\exists^{\mathbb{N}_\infty - S} x\ \phi$.

For example, we can state in $\mathsf{FO}^2_{\mathrm{Pres}}$ that every node in a graph has even degree (i.e., the graph is Eulerian). Clearly $\mathsf{FO}^2_{\mathrm{Pres}}$ extends C^2, the fragment of the logic where only counting quantifiers are used, and FO^2, the fragment where only the classical quantifier $\exists x$ is allowed.

Presburger arithmetic. An *existential Presburger formula* is a formula of the form $\exists x_1 \ldots x_k\ \phi$, where ϕ is a quantifier-free formula over the signature including constants $0, 1$, a binary function symbol $+$, and a binary relation $\leqslant$. Such a formula is a *sentence* if it has no free variables. The notion of a sentence holding in a structure interpreting the function, relations, and constants is defined in the usual way. The structure $\mathcal{N} = (\mathbb{N}, +, \leqslant, 0, 1)$, is defined by interpreting $+, \leqslant, 0, 1$ in the standard way, while the structure $\mathcal{N}_\infty = (\mathbb{N}_\infty, +, \leqslant, 0, 1)$ is the same except that $a + \infty = \infty$ and $a \leqslant \infty$ for each $a \in \mathbb{N}_\infty$.

It is known that the satisfiability of existential Presburger sentences over $\mathcal{N}$ is decidable and belongs to NP [17]. Further, the satisfiability problem for $\mathcal{N}_\infty$ can easily be reduced to that for $\mathcal{N}$. Indeed, we can first guess which variables are mapped to ∞ and then which atoms should be true, then check whether each guessed atomic truth value is consistent with other guesses and determine additional variables which must be infinite based on this choice, and finally restrict to atoms that do not involve variables guessed to be infinite, and check that the conjunction is satisfiable by standard integers.

▶ **Theorem 1.** *The satisfiability problem for existential Presburger sentences over both $\mathcal{N}$ and $\mathcal{N}_\infty$ are both in NP.*

3 Main result

In this section we prove the decidability of $\mathsf{FO}^2_{\mathrm{Pres}}$ satisfiability. Our decision procedure is based on the key notion of regular graphs. Note that whenever we talk about graphs or digraphs (i.e., directed graphs), by default we allow both finite and infinite sets of vertices and edges.

3.1 Regular graphs

In the following we fix an integer $p \geqslant 0$. Let $\mathbb{N}_{\infty,+p}$ denote the set whose elements are either a or a^{+p}, where $a \in \mathbb{N}_\infty$. For integers $t, m \geqslant 1$, let $\mathbb{N}_{\infty,+p}^{t\times m}$ denote the set of matrices with t rows and m columns where each entry is an element from $\mathbb{N}_{\infty,+p}$.

A *t-color bipartite (undirected) graph* is $G = (U, V, E_1, \ldots, E_t)$, where U and V are sets of vertices and $E_1, \ldots, E_t$ are pairwise disjoint sets of edges between U and V. Edges in E_i are called *E_i-edges.* We will write an edge in a bipartite graph as $(u, v) \in U \times V$. For a vertex $u \in U \cup V$, the *E_i-degree* of u is the number of E_i-edges adjacent to u. The degree of u is the sum of the E_i-degrees for $i = 1 \ldots t$. We say that G is *complete*, if $U \times V = \bigcup_{i=1}^{t} E_i$.

For two matrices $A \in \mathbb{N}_{\infty,+p}^{t\times m}$ and $B \in \mathbb{N}_{\infty,+p}^{t\times n}$, the graph G is a *$A|B$-biregular* graph, if there exist partitions $U_1, \ldots, U_m$ of U and $V_1, \ldots, V_n$ of V such that for every $1 \leqslant i \leqslant t$, for every $1 \leqslant k \leqslant m$, for every $1 \leqslant l \leqslant n$, the E_i-degree of every vertex in U_k is $A_{i,k}$ and the E_i degree of every vertex in V_l is $B_{i,l}$.[1] For each such partition, we say that G *has size* $\bar{M}|\bar{N}$, where $\bar{M} = (|U_1|, \ldots, |U_m|)$ and $\bar{N} = (|V_1|, \ldots, |V_n|)$. The partition $U_1, \ldots, U_m$ and $V_1, \ldots, V_n$ is called a *witness partition.* We should remark that some U_i and V_i are allowed to be empty.

The above definition can be easily adapted for the case of directed graphs that are not necessarily bipartite. A t-color *directed graph* (or *digraph*) is $G = (V, E_1, \ldots, E_t)$, where $E_1, \ldots, E_t$ are pairwise disjoint set of directed edges on a set of vertices V. As before, edges in E_i are called E_i-edges. The E_i-indegree and -outdegree of a vertex u, is defined as the number of incoming and outgoing E_i-edges incident to u.

In a t-color digraph G we will assume that (i) there are no self-loops – that is, (v, v) is not an E_i-edge, for every vertex $v \in V$ and every E_i, and (ii) if (u, v) is an E_i-edge, then its inverse (v, u) is not an E_j-edge for any E_j. This will suffice for the digraphs that arise in our decision procedure. We say that a digraph G is *complete*, if for every $u, v \in V$ and $u \neq v$, either (u, v) or (v, u) is an E_i-edge, for some E_i.

We say that G is a *$A|B$-regular* digraph, where $A, B \in \mathbb{N}_{\infty,+p}^{t\times m}$, if there exists a partition $V_1, \ldots, V_m$ of V such that for every $1 \leqslant i \leqslant t$, for every $1 \leqslant k \leqslant m$, the E_i-indegree and -outdegree of every vertex in V_k is $A_{i,k}$ and $B_{i,k}$, respectively. We say that G has size $(|V_1|, \ldots, |V_m|)$, and call $V_1, \ldots, V_m$ a witness partition.

Lemma 2 below will be the main technical tool for our decidability result. Let $\bar{x}$ and $\bar{y}$ be vectors of variables of length m and n, respectively.

▶ **Lemma 2.** *For every $A \in \mathbb{N}_{\infty,+p}^{t\times m}$ and $B \in \mathbb{N}_{\infty,+p}^{t\times n}$, there exists (effectively computable) existential Presburger formula $\textsf{c-bireg}_{A|B}(\bar{x}, \bar{y})$ such that for every $(\bar{M}, \bar{N}) \in \mathbb{N}_\infty^m \times \mathbb{N}_\infty^n$, the following holds: there is complete $A|B$-biregular graph with size $\bar{M}|\bar{N}$ if and only if $\textsf{c-bireg}_{A|B}(\bar{M}, \bar{N})$ holds in $\mathcal{N}_\infty$.*

Lemma 3 below is the analog for digraphs.

▶ **Lemma 3.** *For every $A \in \mathbb{N}_{\infty,+p}^{t\times m}$ and $B \in \mathbb{N}_{\infty,+p}^{t\times m}$, there exists (effectively computable) existential Presburger formula $\textsf{c-reg}_{A|B}(\bar{x})$ such that for every $\bar{M} \in \mathbb{N}_\infty^m$, the following holds. There is complete $A|B$-regular digraph with size $\bar{M}$ if and only if $\textsf{c-reg}_{A|B}(\bar{M})$ holds in $\mathcal{N}_\infty$.*

[1] By abuse of notation, when we say that an integer z equals a^{+p}, we mean that $z \in a^{+p}$. Thus, when writing $A_{i,k} = a^{+p}$, we mean that the degree of the vertex is an element in a^{+p}.

Lemmas 2 and 3 can be easily readjusted when we are interested only in finite sizes, i.e., $\bar{M} \in \mathbb{N}^m$ and $\bar{N} \in \mathbb{N}^n$, by requiring the formulas to hold in $\mathcal{N}$, instead of $\mathcal{N}_\infty$. Alternatively, we can also state inside the formulas that none of the variables in $\bar{x}$ and $\bar{y}$ are equal to ∞.

The proofs of these two lemmas are discussed in Section 4.

3.2 Decision procedure

Theorem 4 below is the main result in this paper.

▶ **Theorem 4.** *For every FO^2_{Pres} sentence ϕ, there is an (effectively computable) existential Presburger formula $PRES_\phi$ such that (i) ϕ has a model iff $PRES_\phi$ holds in $\mathcal{N}_\infty$ and (ii) ϕ has a finite model iff $PRES_\phi$ holds in $\mathcal{N}$.*

From the decision procedure for existential Presburger formulas (Theorem 1) mentioned in Section 2, we immediately will obtain the following corollary.

▶ **Corollary 5.** *Both satisfiability and finite satisfiability for FO^2_{Pres} are decidable.*

We will sketch how Theorem 4 is proven, making use of Lemmas 2 and 3. We start by observing that satisfiability (and spectrum analysis) for an $\mathsf{FO}^2_{\mathrm{Pres}}$ sentence can be converted effectively into the same questions for a sentence in a variant of Scott normal form:

$$\phi := \forall x \forall y \; \alpha(x,y) \;\; \wedge \;\; \bigwedge_{i=1}^{k} \forall x \exists^{S_i} y \; \beta_i(x,y) \wedge x \neq y, \tag{1}$$

where $\alpha(x,y)$ is a quantifier free formula, each $\beta_i(x,y)$ is an atomic formula and each S_i is an u.p.s. The proof, which is fairly standard, will appear in the full version of this paper. By taking the least common multiple, we may assume that all the (non-zero) periods in all S_i are the same.

We recall some standard terminology. A 1-*type* is a maximally consistent set of atomic and negated atomic unary formulas using only variable x. A 1-type can be identified with the quantifier-free formula that is the conjunction of its constituent formulas. Thus, we say that an element a in a structure $\mathcal{A}$ has 1-type π, if π holds on the element a. We denote by A_π the set of elements in $\mathcal{A}$ with 1-type π. Clearly the domain A of a structure $\mathcal{A}$ is partitioned into the sets A_π. Similarly, a 2-*type* is a maximally consistent set of atomic and negated atomic binary formulas using only variables x, y, containing the predicate $x \neq y$. The notion of a pair of elements (a,b) in a structure $\mathcal{A}$ having 2-type E is defined as with 1-types. We denote by $\Pi = \{\pi_1, \pi_2, \ldots, \pi_n\}$ and $\mathcal{E} = \{E_1, \ldots, E_t, \overleftarrow{E_1}, \ldots, \overleftarrow{E_t}\}$ the sets of all 1-types and 2-types, respectively, where $\overleftarrow{E_i}(x,y) = E_i(y,x)$ for each $1 \leqslant i \leqslant t$ – that is, each $\overleftarrow{E_i}$ is the reversal of E_i.

Let $g : \mathcal{E} \times \Pi \to \mathbb{N}_{\infty,+p}$ be a function. We will use such a function g to describe the "behavior" of the elements in the following sense. Let $\mathcal{A}$ be a structure. We say that an element $a \in A$ *behaves according to* g, if for every $E \in \mathcal{E}$ and for every $\pi \in \Pi$, the number of elements $b \in A_\pi$ such that the 2-type of (a,b) is E belongs to $g(E,\pi)$. We denote by $A_{\pi,g}$ the set of all elements in A_π that behave according to g. The restriction of g on 1-type π is the function $g_\pi : \mathcal{E} \to \mathbb{N}_{\infty,+p}$, where $g_\pi(E) = g(E,\pi)$. We call the function g_π the *behavior* (function) towards 1-type π.

We are, of course, only interested in functions g that are consistent with the sentence ϕ in (1), and we formalize this as follows:

- A 1-type $\pi \in \Pi$ and a function $g : \mathcal{E} \times \Pi \to \mathbb{N}_{\infty,+p}$ are *incompatible* (w.r.t. $\forall x \forall y \; \alpha(x,y)$), if there is $E \in \mathcal{E}$ and $\pi' \in \Pi$ such that $\pi(x) \wedge E(x,y) \wedge \pi'(y) \models \neg\alpha(x,y)$ and $g(E,\pi') \neq 0$.

- A function $g : \mathcal{E} \times \Pi \to \mathbb{N}_{\infty,+p}$ is a *good* function (w.r.t. $\bigwedge_{i=1}^{k} \forall x \exists^{S_i} y\ \beta_i(x,y) \wedge x \neq y$), if for every $\pi \in \Pi$ and for every i the following holds:[2]

$$\sum_{E \models \beta_i(x,y)} \sum_{\pi \in \Pi} g(E,\pi) = a \qquad\qquad \text{for some } a \in S_i.$$

If $\mathcal{A} \models \phi$ then $A_{(\pi,g)} = \emptyset$, whenever π and g are incompatible, and in addition every element in A behaves only according to some good function.

The main idea is to construct the sentence PRES_ϕ that "counts" the cardinality $|A_{(\pi,g)}|$ in every structure $\mathcal{A} \models \phi$, for every π and g. Toward this end, let $\mathcal{G} = \{g_1, g_2, \ldots, g_m\}$ enumerate all good functions. Note that $\mathcal{G}$ can be computed effectively from the sentence ϕ, since it suffices to consider functions $g : \mathcal{E} \times \Pi \to \mathbb{N}_{\infty,+p}$ with codomain $\{0, \ldots, a, 0^{+p}, \ldots, a^{+p}, \infty\}$, where a is the maximal offset of the (non-∞) elements in $\bigcup_{i=1}^{k} S_i$.

The sentence PRES_ϕ will be of the form

$$\mathsf{PRES}_\phi := \exists \bar{X}\ \mathsf{consistent}_1(\bar{X})\ \wedge\ \mathsf{consistent}_2(\bar{X}) \tag{2}$$

where $\bar{X}$ is a vector of variables $(X_{(\pi_1,g_1)}, X_{(\pi_1,g_2)}, \ldots, X_{(\pi_n,g_m)})$. Intuitively, each $X_{(\pi_i,g_j)}$ represents $|A_{\pi_i,g_j}|$. By the formulas $\mathsf{consistent}_1(\bar{X})$ and $\mathsf{consistent}_2(\bar{X})$, we capture the consistency of the integers $\bar{X}$ with the formulas $\forall x \forall y\ \alpha(x,y)$ and $\bigwedge_{i=1}^{k} \forall x \exists^{S_i} y\ \beta_i(x,y) \wedge x \neq y$, respectively.

We start by defining the formula $\mathsf{consistent}_1(\bar{X})$. Letting H be the set of all pairs (π, g) where π and g are incompatible, the formula $\mathsf{consistent}_1(\bar{X})$ can be defined as

$$\mathsf{consistent}_1(\bar{X}) := \bigwedge_{(\pi,g) \in H} X_{(\pi,g)} = 0 \tag{3}$$

Towards defining the formula $\mathsf{consistent}_2(\bar{X})$, we introduce some notations. For $\pi \in \Pi$, define the matrices $M_\pi, \overleftarrow{M}_\pi \in \mathbb{N}_{\infty,+p}^{t \times m}$ as follows:

$$M_\pi := \begin{pmatrix} g_1(E_1,\pi) & \cdots & g_m(E_1,\pi) \\ \vdots & \ddots & \vdots \\ g_1(E_t,\pi) & \cdots & g_m(E_t,\pi) \end{pmatrix} \quad \text{and} \quad \overleftarrow{M}_\pi := \begin{pmatrix} g_1(\overleftarrow{E_1},\pi) & \cdots & g_m(\overleftarrow{E_1},\pi) \\ \vdots & \ddots & \vdots \\ g_1(\overleftarrow{E_t},\pi) & \cdots & g_m(\overleftarrow{E_t},\pi) \end{pmatrix}$$

The idea is that M_π captures all possible behavior towards 1-type π, where each column j represents the behavior of g_j towards π. Note that for a structure $\mathcal{A}$ and 1-type π, the restriction of $\mathcal{A}$ on the set A_π can be viewed as a t-color digraph $G = (V, E_1, \ldots, E_t)$. It is sufficient to consider only the 2-types $E_1, \ldots, E_t$, because each E_i determines its reversal $\overleftarrow{E_i}$. Moreover, an element a has an incoming E_i-edge if and only if it has an outgoing $\overleftarrow{E_i}$-edge. Thus, if $\mathcal{A} \models \phi$, the graph G is a complete $M_\pi|\overleftarrow{M}_\pi$-regular digraph.

Now, we explain how to capture the behavior between elements with distinct 1-types. Define matrices $L_\pi, \overleftarrow{L}_\pi \in \mathbb{N}_{\infty,+p}^{2t \times m}$ as follows:

$$L_\pi := \begin{pmatrix} M_\pi \\ \overleftarrow{M}_\pi \end{pmatrix} \quad \text{and} \quad \overleftarrow{L}_\pi := \begin{pmatrix} \overleftarrow{M}_\pi \\ M_\pi \end{pmatrix}$$

That is, in L_π the first t rows come from M_π with the next t rows from $\overleftarrow{M}_\pi$. On the other hand, in $\overleftarrow{L}_\pi$ the first t rows come from $\overleftarrow{M}_\pi$, followed by the t rows from M_π.

[2] Here the operation $+$ on $\mathbb{N}_{\infty,+p}$ is defined to be commutative operation where $a + \infty = a^{+p} + \infty = \infty$ and $a^{+p} + b = a^{+p} + b^{+p} = (a+b)^{+p}$. On integers from $\mathbb{N}$, it is the standard addition operation.

The idea is that for a structure $\mathcal{A}$, the 2-types that are realized between A_π and $A_{\pi'}$ can be viewed as a $2t$-color bipartite graph $G = (A_\pi, A_{\pi'}, E_1, \ldots, E_t, \overleftarrow{E_1}, \ldots, \overleftarrow{E_t})$, where the direction of the edges are ignored. Moreover, a pair (a, b) has 2-type E if and only if (b, a) has 2-type $\overleftarrow{E}$, Thus, if $\mathcal{A} \models \phi$, the graph G is a complete $L_{\pi'}|\overleftarrow{L}_{\pi}$-biregular graph.

Now we are ready to define the formula $\mathsf{consistent}_2(\bar{X})$. We enumerate all the 1-types $\pi_1, \ldots, \pi_n$ and define $\mathsf{consistent}_2$ as follows:

$$\mathsf{consistent}_2(\bar{X}) := \bigwedge_{1 \leqslant i \leqslant n} \mathsf{c\text{-}reg}_{M_{\pi_i}|\overleftarrow{M}_{\pi_i}}(\bar{X}_{\pi_i}) \ \wedge \ \bigwedge_{1 \leqslant i < j \leqslant n} \mathsf{c\text{-}bireg}_{L_{\pi_j}|\overleftarrow{L}_{\pi_i}}(\bar{X}_{\pi_i}, \bar{X}_{\pi_j}). \qquad (4)$$

The formula $\mathsf{consistent}_1(\bar{X})$ is Presburger definable by inspection, while $\mathsf{consistent}_2(\bar{X})$ is Presburger definable using Lemmas 2 and 3. The correctness comes directly from the following lemma.

▶ **Lemma 6.** *For every structure $\mathcal{A} \models \phi$, $\mathsf{consistent}_1(\bar{N}) \wedge \mathsf{consistent}_2(\bar{N})$ holds, where $\bar{N} = (|A_{\pi_1,g_1}|, \ldots, |A_{\pi_n,g_m}|)$. Conversely, for every $\bar{N}$ such that $\mathsf{consistent}_1(\bar{N}) \wedge \mathsf{consistent}_2(\bar{N})$ holds, there is $\mathcal{A} \models \phi$ such that $\bar{N} = (|A_{\pi_1,g_1}|, \ldots, |A_{\pi_n,g_m}|)$.*

Proof. Let ϕ be in Scott normal form as in (1). As before, $\Pi = \{\pi_1, \pi_2, \ldots, \pi_n\}$ denote the set of all 1-types and $\mathcal{E} = \{E_1, \ldots, E_t, \overleftarrow{E_1}, \ldots, \overleftarrow{E_t}\}$ the set of all 2-types, where $\overleftarrow{E_i}(x, y) = E_i(y, x)$ for each $1 \leqslant i \leqslant t$. Recall that each 2-type E contains the predicate $x \neq y$ and that $\mathcal{G} = \{g_1, \ldots, g_m\}$ is the set of all good functions.

Note that for $\pi, \pi' \in \Pi$ and $E \in \mathcal{E}$, the conjunction $\pi(x) \wedge E(x, y) \wedge \pi'(y)$ corresponds to a boolean assignment of the atomic predicates in $\alpha(x, y)$. Thus, either $\pi(x) \wedge E(x, y) \wedge \pi'(y) \models \alpha(x, y)$ or $\pi(x) \wedge E(x, y) \wedge \pi'(y) \models \neg\alpha(x, y)$. Similarly, $\pi(x) \wedge x = y \models \alpha(x, y)$ or $\pi(x) \wedge x = y \models \neg\alpha(x, y)$.

We first prove the first statement in the lemma. Let $\mathcal{A} \models \phi$. Partition A into $A_{\pi,g}$'s. We will show that $\mathsf{consistent}_1(\bar{X}) \ \wedge \ \mathsf{consistent}_2(\bar{X})$ holds when each $X_{\pi,g}$ is assigned with the value $|A_{\pi,g}|$.

Since $\mathcal{A} \models \forall x \forall y \ \alpha(x, y)$, by definition $A_{\pi,g} = \emptyset$, whenever π and g are incompatible. Thus, $\mathsf{consistent}_1(\bar{X})$ holds.

Next, we will show that $\mathsf{consistent}_2(\bar{X})$ holds. Let $\pi \in \Pi$. By definition of A_π, A_π is a complete $M_\pi|\overleftarrow{M}_\pi$-regular digraph $G = (V, E_1, \ldots, E_t)$, with size $(|A_{\pi,g_1}|, \ldots, |A_{\pi,g_m}|)$. Thus, by Lemma 3, $\mathsf{c\text{-}reg}_{M_\pi|\overleftarrow{M}_\pi}(\bar{X}_\pi)$ holds.

For $\pi_i, \pi_j \in \Pi$, where $i < j$, the structure $\mathcal{A}$ restricted to A_{π_i} and A_{π_j} can be viewed as a complete $L_{\pi_j}|\overleftarrow{L}_{\pi_i}$-biregular graph $G = (U, V, E_1, \ldots, E_t, \overleftarrow{E_1}, \ldots, \overleftarrow{E_t})$, where $U = A_{\pi_i}$ and $V = A_{\pi_j}$, and for each $1 \leqslant i \leqslant t$, we have the interpretation denoted (by a slight abuse of notation) as E_i consist of all pairs $(a, b) \in A_{\pi_i} \times A_{\pi_j}$ whose 2-type is E_i, and similarly for $\overleftarrow{E_i}$. By Lemma 2, $\mathsf{c\text{-}bireg}_{L_{\pi_j}|\overleftarrow{L}_{\pi_i}}(\bar{X}_{\pi_i}, \bar{X}_{\pi_j})$ holds.

Now we prove the second statement. Suppose PRES_ϕ holds. By definition, there exists an assignment to the variables in $\bar{X}$ such that $\mathsf{consistent}_1(\bar{X}) \wedge \mathsf{consistent}_2(\bar{X})$ holds. Abusing notation as we often do in this work, we denote the value assigned to each $X_{\pi,g}$ by the variable $X_{\pi,g}$ itself.

For each (π, g), we have a set $V_{\pi,g}$ with cardinality $X_{\pi,g}$. We denote by $V_\pi = \bigcup_g V_{\pi,g}$. We construct a structure $\mathcal{A}$ that satisfies ϕ as follows.

- The domain is $A = \bigcup_{\pi,g} V_{\pi,g}$.
- For each $\pi \in \Pi$, for each $a \in V_\pi$, the unary atomic formulas on a are defined such that the 1-type of a becomes π.

- For each $\pi \in \Pi$, the binary predicates on $(u, v) \in V_\pi \times V_\pi$ are defined as follows. Since $\mathsf{c\text{-}reg}_{M_\pi|\overleftarrow{M}_\pi}(\bar{X}_\pi)$ holds, there is a complete $M_\pi|\overleftarrow{M}_\pi$-regular digraph $G = (V_\pi, E_1, \ldots, E_t)$ with size $\bar{X}_\pi$. The edges $E_1, \ldots, E_t$ define precisely the 2-types among elements in V_π.
- For each π_i, π_j, where $i < j$, the binary predicates on $(u, v) \in V_{\pi_i} \times V_{\pi_j}$ are defined as follows. Since $\mathsf{c\text{-}bireg}_{L_{\pi_j}|\overleftarrow{L}_{\pi_i}}(\bar{X}_{\pi_i}, \bar{X}_{\pi_j})$ holds, there is a $L_{\pi'}|\overleftarrow{L}_\pi$-biregular graph $G = (V_{\pi_i}, V_{\pi_j}, E_1, \ldots, E_t, \overleftarrow{E}_1, \ldots, \overleftarrow{E}_t)$ with size $\bar{X}_\pi|\bar{X}_{\pi'}$. The edges $E_1, \ldots, E_t, \overleftarrow{E}_1, \ldots, \overleftarrow{E}_t$ define precisely the 2-types on $(u, v) \in V_{\pi_i} \times V_{\pi_j}$.

We first show that $\mathcal{A} \models \forall x \forall y\ \alpha(x, y)$. Indeed, suppose there exist $u, v \in A$ such that $\pi(u) \wedge E_(u, v) \wedge \pi'(v) \not\models \alpha(u, v)$. By definition, there is g such that $u \in V_{\pi,g}$ and $g(E, \pi') \neq 0$. Thus, $V_{\pi,g} \neq \emptyset$. This also means that π is incompatible with g, which implies that $X_{\pi,g} = 0$ by $\mathsf{consistent}_1(\bar{X})$, thus, contradicts the assumption that $V_{\pi,g} \neq \emptyset$.

Next, we show that $\mathcal{A} \models \bigwedge_{i=1}^{k} \forall x \exists^{S_i} y\ \beta_i(x, y) \wedge x \neq y$. Note that $\mathcal{G} = \{g_1, \ldots, g_m\}$ consists of only good functions. Thus, for every $g \in \mathcal{G}$, for every β_i, the sum $\sum_\pi \sum_{\beta_i(x,y) \in E}\ g(E, \pi)$ is an element in S_i. ◀

4 Proof ideas for Lemmas 2 and 3

We now discuss the proof of the main biregular graph lemmas. Due to space constraints, we deal only with the 1-color case, which gives the flavor of the arguments. The general case, which is much more involved, is deferred to the full version of this paper.

This section is organized as follows. In Subsection 4.1 we will focus on a relaxation of Lemma 2 where the requirement being complete is dropped. This will then be used to prove the complete case in Subsection 4.2. Finally, in Subsection 4.3 we present a brief explanation on how to modify the proof for the biregular graphs to the one for regular digraphs.

4.1 The case of incomplete 1-color biregular graphs

This subsection is devoted to the proof of the following lemma.

▶ **Lemma 7.** *For every $A \in \mathbb{N}^{1\times m}_{\infty,+p}$ and $B \in \mathbb{N}^{1\times n}_{\infty,+p}$, there exists (effectively computable) existential Presburger formula $\mathsf{bireg}_{A|B}(\bar{x}, \bar{y})$ such that for every $(\bar{M}, \bar{N}) \in \mathbb{N}^m_\infty \times \mathbb{N}^n_\infty$ the following holds: there is an $A|B$-biregular graph with size $\bar{M}|\bar{N}$ if and only if $\mathsf{bireg}_{A|B}(\bar{M}, \bar{N})$ holds in $\mathcal{N}_\infty$.*

The desired formula $\mathsf{c\text{-}bireg}_{A|B}(\bar{x}, \bar{y})$ for complete biregular graphs will be defined using the formula $\mathsf{bireg}_{A|B}(\bar{x}, \bar{y})$.

We will use the following notations. The term vectors always refers to row vectors, and we usually use $\bar{a}, \bar{b}, \ldots$ (possibly indexed) to denote them. We write $(\bar{a}, \bar{b})$ to denote the vector $\bar{a}$ concatenated with $\bar{b}$. Obviously 1-row matrices can be viewed as row vectors. For $\bar{a} = (a_1, \ldots, a_k) \in \mathbb{N}^k_\infty$, we write $\bar{a}^{+p}$ to denote the vector $(a_1^{+p}, \ldots, a_k^{+p})$.

Matrix entries of the form a^{+p} are called *periodic* entries. Otherwise, they are called *fixed* entries. By grouping the entries according to whether they are fixed/periodic, we write a 1-row matrix M as $(\bar{a}, \bar{b}^{+p})$, where $\bar{a}$ and $\bar{b}^{+p}$ correspond to the fixed and periodic entries in M. Matrices that contain only fixed (or, periodic) entries are written as $\bar{a}$ (or, $\bar{a}^{+p}$).

To specify $A|B$-biregular graphs, we write $(\bar{a}, \bar{b}^{+p})|(\bar{c}, \bar{d}^{+p})$-biregular graphs, where $A = (\bar{a}, \bar{b}^{+p})$ and $B = (\bar{c}, \bar{d}^{+p})$. Similarly, when, say, A contains only fixed entries, it is written as $\bar{a}|(\bar{c}, \bar{d}^{+p})$-biregular. The size of $(\bar{a}, \bar{b}^{+p})|(\bar{c}, \bar{d}^{+p})$-biregular graph is written as $(\bar{M}_0, \bar{M}_1)|(\bar{N}_0, \bar{N}_1)$, where the lengths of $\bar{M}_0, \bar{M}_1, \bar{N}_0, \bar{N}_1$ are the same as $\bar{a}, \bar{b}, \bar{c}, \bar{d}$, respectively. The other cases, when some of $\bar{a}, \bar{b}^{+p}, \bar{c}, \bar{d}^{+p}$ are omitted, are treated in similar manner.

As before, we will write $\bar{x}, \bar{y}$ (possibly indexed) to denote a vector of variables. We write $\bar{1}$ to denote the vector with all components being 1. We use $\cdot$ to denote the standard dot product between two vectors. To avoid being repetitive, when dot products are performed, it is implicit that the vector lengths are the same. In particular, $\bar{x} \cdot \bar{1}$ is the sum of all the components in $\bar{x}$.

We now outline the proof of Lemma 7, focusing only on the case where there is no ∞ degree in the matrices. The case where such a degree exists is similar but simpler. Without loss of generality, we can also assume that none of the fixed entries are zero. For vectors $\bar{M}_0, \bar{M}_1, \bar{N}_0, \bar{N}_1$ with the same length as $\bar{a}, \bar{b}, \bar{c}, \bar{d}$, respectively, we say that $(\bar{M}_0, \bar{M}_1)|(\bar{N}_0, \bar{N}_1)$ *is big enough for* $(\bar{a}, \bar{b}^{+p})|(\bar{c}, \bar{d}^{+p})$, if the following holds:

(a) $\bar{M}_0 \cdot \bar{1} + \bar{M}_1 \cdot \bar{1} + \bar{N}_0 \cdot \bar{1} + \bar{N}_1 \cdot \bar{1} \geqslant 2\delta_{\max}^2 + 3$,

(b) $\bar{M}_1 \cdot \bar{1} \geqslant \delta_{\max}^2 + 1$,

(c) $\bar{N}_1 \cdot \bar{1} \geqslant \delta_{\max}^2 + 1$.

Here $\delta_{\max}$ is $\max(p, \bar{a}, \bar{b}, \bar{c}, \bar{d})$ – that is, the maximal element among p and the components in $\bar{a}, \bar{b}, \bar{c}, \bar{d}$. When $\bar{b}^{+p}$ or $\bar{d}^{+p}$ are missing, the same notion can be defined by dropping condition (b) or (c), respectively. For example, we say that $\bar{M}|\bar{N}$ is big enough for $\bar{a}|\bar{b}$, if $\bar{M} \cdot \bar{1} + \bar{N} \cdot \bar{1} \geqslant 2\delta_{\max}^2 + 3$, where $\delta_{\max} = \max(\bar{a}, \bar{b})$. Similarly, $(\bar{M}_0, \bar{M}_1)|\bar{N}$ is big enough for $(\bar{a}, \bar{b}^{+p})|\bar{c}$, if $\bar{M}_0 \cdot \bar{1} + \bar{M}_1 \cdot \bar{1} + \bar{N} \cdot \bar{1} \geqslant 2\delta_{\max}^2 + 3$, and $\bar{M}_1 \cdot \bar{1} \geqslant \delta_{\max}^2 + 1$, where $\delta_{\max} = \max(p, \bar{a}, \bar{b}, \bar{c})$.

The proof idea is as follows. We first construct a formula that deals with big enough sizes. Then, we construct a formula for each of the cases when one of the conditions (a), (b) or (c) is violated. The interesting case will be when condition (b) is violated. This means that the number of vertices with degrees from $\bar{b}^{+p}$ is fixed, and they can be "encoded" inside the Presburger formula.

We start with the big enough case. When there are only fixed entries, we will use the following lemma.

▶ **Lemma 8.** *For $\bar{M}|\bar{N}$ big enough for $\bar{a}|\bar{b}$, there is a $\bar{a}|\bar{b}$-biregular graph with size $\bar{M}|\bar{N}$ if and only if $\bar{M} \cdot \bar{a} = \bar{N} \cdot \bar{b}$.*

Proof. Note that if we have a biregular graph with the desired outdegrees on the left, then the total number of edges must be $\bar{M} \cdot \bar{a}$, and similarly the total number of edges considering the requirement for vertices on the right, we see that the total number of edges must be $\bar{N} \cdot \bar{b}$. Thus this condition is always a necessary one, regardless of whether $\bar{M}|\bar{N}$ is big enough.

When both $\bar{M}$ and $\bar{N}$ do not contain ∞, [13, Lemma 7.2] shows that when $\bar{M}|\bar{N}$ is big enough for $\bar{a}|\bar{b}$, the converse holds: $\bar{M} \cdot \bar{a} = \bar{N} \cdot \bar{b}$ implies that there is a $\bar{a}|\bar{b}$-biregular graph with size $\bar{M}|\bar{N}$. We briefly mention the proof idea there, which we will also see later (e.g., in the proof of Lemma 9). There is a preliminary construction that handles the requirement on vertices on one side in isolation, leaving the vertices on the right with outdegree 1. A follow-up construction merges vertices on the right in order to ensure the necessary number of incoming edges on the right. In doing so we exploit the "big enough" property in order to avoid merging two nodes on the right with a common adjacent edge on the left.

We will now prove that the condition is also sufficient when either $\bar{M}$ or $\bar{N}$ contains ∞. So assume $\bar{M} \cdot \bar{a} = \bar{N} \cdot \bar{b}$, and thus both $\bar{M}, \bar{N}$ contain ∞.

We construct an $\bar{a}|\bar{b}$-biregular graph $G = (U, V, E)$ with size $\bar{M}|\bar{N}$ as follows. Let $\bar{a} = (a_1, \dots, a_m)$ and $\bar{b} = (b_1, \dots, b_n)$. Let $\bar{M} = (M_1, \dots, M_m)$ and $\bar{N} = (N_1, \dots, N_n)$. We pick pairwise disjoint sets $U_1, \dots, U_m$, where each $|U_i| = M_i$ and $V_1, \dots, V_n$, where $|V_i| = N_i$. We set $U = \bigcup_i U_i$ and $V = \bigcup_i V_i$.

The edges are constructed as follows. For each $i \leqslant i \leqslant m$, when $|U_i|$ is finite, we make each vertex $u \in U_i$ have degree a_i, as follows. For each $1 \leqslant j \leqslant t$, we pick a_i "new" vertices from some infinite set V_l – that is, vertices that are not adjacent to any edge, and connect them to u. Likewise, for each vertex $v \in V_i$ when $|V_i|$ is finite. After performing this, every vertex in finite U_i and V_i has degree a_i and b_i, respectively, and every vertex in infinite sets U_i and V_i has degree at most 1.

Finally, we iterate the following process. For every infinite U_i, if $u \in U_i$ has degree other than a_i, we change the degree to a_i by picking "new" vertices from some infinite set V_l, and connect them to u by an appropriate number of edges. Likewise, we can make each vertex v in infinite V_i to have degree b_i. Note that in any iteration, for every infinite set U_i, the degree of a vertex $u \in U_i$ is either a_i, 1, or 0. Likewise, in any iteration, for every infinite set V_i, the degree of a vertex $v \in V_i$ is either b_i, 1, or 0. Since there is an infinite supply of vertices, there are always new vertices that can be picked in any iteration. ◀

Now we move to the case where the entries are still big enough, but some of the entries are periodic on one side. Then we consider the following formula $\Psi_{(\bar{a},\bar{b}^{+p})|\bar{c}}(\bar{x}_0, \bar{x}_1, \bar{y})$:

$$\exists z \quad (z \neq \infty) \ \wedge \ \big(\bar{a} \cdot \bar{x}_0 + \bar{b} \cdot \bar{x}_1 + pz \ = \ \bar{c} \cdot \bar{y}\big). \tag{5}$$

Note that if $G = (U, V, E)$ is a $(\bar{a}, \bar{b}^{+p})|\bar{c}$-biregular graph with size $(\bar{M}_0, \bar{M}_1)|\bar{N}$, then the number of edges $|E|$ should equal the sum of the degrees of the vertices in U, which is $\bar{a} \cdot \bar{M}_0 + \bar{b} \cdot \bar{M}_1 + zp$, for some integer $z \geqslant 0$. Since this quantity must equal the sum of the degrees of the vertices in V, which is $\bar{c} \cdot \bar{N}$, we again conclude that this formula is a necessary condition – regardless of whether the entries are big enough. We again show the converse.

▶ **Lemma 9.** *For $(\bar{M}_0, \bar{M}_1)|\bar{N}$ big enough for $(\bar{a}, \bar{b}^{+p})|\bar{c}$ the following holds. There is a $(\bar{a}, \bar{b}^{+p})|\bar{c}$-biregular graph with size $(\bar{M}_0, \bar{M}_1)|\bar{N}$ if and only if $\Psi_{(\bar{a},\bar{b}^{+p})|\bar{c}}(\bar{M}_0, \bar{M}_1, \bar{N})$ holds.*

Proof. Assume that $\Psi_{(\bar{a},\bar{b}^{+p})|\bar{c}}(\bar{M}_0, \bar{M}_1, \bar{N})$ holds. As before, abusing notation, we denote the value assigned to variable z by z itself. Suppose $\bar{a} \cdot \bar{M}_0 + \bar{b} \cdot \bar{M}_1 + pz = \bar{N} \cdot \bar{c}$. Since $(\bar{M}_0, \bar{M}_1)|\bar{N}$ is big enough for $(\bar{a}, \bar{b}^{+p})|\bar{c}$, it follows immediately that $(\bar{M}_0, \bar{M}_1, z)|\bar{N}$ is big enough for $(\bar{a}, \bar{b}, p)|\bar{c}$. Applying Lemma 8, there is a $(\bar{a}, \bar{b}, p)|\bar{c}$-biregular graph with size $(\bar{M}_0, \bar{M}_1, z)|\bar{N}$. That is, we have a graph that satisfies our requirements, but there is an additional partition class Z on the left of size z where the number of adjacent vertices is p, rather than being $\bar{b}^{+p}$ as we require. Let $G = (U, V, E)$ be such a graph, and let $U = U_0 \cup U_1 \cup Z$, where U_0, U_1, and Z are the sets of vertices whose degrees are from $\bar{a}$, $\bar{b}$, and from p. Note that $|U_0| = \bar{M}_0 \cdot \bar{1}$, $|U_1| = \bar{M}_1 \cdot \bar{1}$ and $|Z| = z$.

We will construct a $(\bar{a}, \bar{b}^{+p})|\bar{c}$-biregular graph with size $(\bar{M}_0, \bar{M}_1)|\bar{N}$. The idea is to merge the vertices in Z with vertices in U_1. Let $z_0 \in Z$. The number of vertices in U_1 reachable from z_0 in distance 2 is at most $\delta^2_{\max}$. Since $(\bar{M}_0, \bar{M}_1)|\bar{N}$ is big enough for $(\bar{a}, \bar{b}^{+p})|\bar{c}$, we have $|U_1| = \bar{M}_1 \cdot \bar{1} \geqslant \delta^2_{\max} + 1$. Thus, there is a vertex $u \in U_1$ not reachable in distance 2. We merge z_0 and u into one vertex. Since the degree of z_0 is p, such merging increases the degree of u by p, which does not break our requirement. We perform such merging for every vertex in Z. ◀

Finally, we turn to the big enough case where there are periodic entries on both sides. There we will deal with the following formula $\Psi_{(\bar{a},\bar{b}^{+p})|(\bar{c},\bar{d}^{+p})}(\bar{x}_0, \bar{x}_1, \bar{y}_0, \bar{y}_1)$:

$$\exists z_1 \exists z_2 \ (z_1 \neq \infty) \ \wedge \ (z_2 \neq \infty) \ \wedge \big(\bar{a} \cdot \bar{x}_0 + \bar{b} \cdot \bar{x}_1 + pz_1 = \bar{c} \cdot \bar{y}_0 + \bar{d} \cdot \bar{y}_1 + pz_2\big). \tag{6}$$

▶ **Lemma 10.** *For $(\bar{M}_0, \bar{M}_1)|(\bar{N}_0, \bar{N}_1)$ big enough for $(\bar{a}, \bar{b}^{+p})|(\bar{c}, \bar{d}^{+p})$ the following holds: there exists a $(\bar{a}, \bar{b}^{+p})|(\bar{c}, \bar{d}^{+p})$-biregular graph with size $(\bar{M}_0, \bar{M}_1)|(\bar{N}_0, \bar{N}_1)$ if and only if $\Psi_{(\bar{a},\bar{b}^{+p})|(\bar{c},\bar{d}^{+p})}(\bar{M}_0, \bar{M}_1, \bar{N}_0, \bar{N}_1)$ holds.*

Proof. As before, the "only if" part is straightforward, so we focus on the "if" part. Suppose $\Psi_{(\bar{a},\bar{b}^{+p})|(\bar{c},\bar{d}^{+p})}(\bar{M}_0,\bar{M}_1,\bar{N}_0,\bar{N}_1)$ holds. Thus, $\bar{a}\cdot\bar{M}_0+\bar{b}\cdot\bar{M}_1+pz_1=\bar{c}\cdot\bar{N}_0+\bar{d}\cdot\bar{N}_1+pz_2$. If $z_1\geqslant z_2$, then the equation can be rewritten as $\bar{a}\cdot\bar{M}_0+\bar{b}\cdot\bar{M}_1+p(z_1-z_2)=\bar{c}\cdot\bar{N}_0+\bar{d}\cdot\bar{N}_1$. By Lemma 9, there is a $(\bar{a},\bar{b}^{+p})|(\bar{c},\bar{d})$-biregular graph with size $(\bar{M}_0,\bar{M}_1)|(\bar{N}_0,\bar{N}_1)$, which of course, is also $(\bar{a},\bar{b}^{+p})|(\bar{c},\bar{d}^{+p})$-biregular. The case when $z_2\geqslant z_1$ is symmetric. ◀

The previous lemmas give formulas that capture the existence of 1-color biregular graphs for big enough sizes. We now turn to sizes that are not big enough – that is, when one of the conditions (a), (b) or (c) is violated. When condition (a) is violated, we have restricted the total size of the graph, and thus we can write a formula that simply enumerate all possible valid sizes. We will consider the case when condition (b) is violated, with the case where condition (c) is violated being symmetric.

If (b) is violated we can fix the value of $\bar{M}_1\cdot\bar{1}$ as some r, and it suffices to find a formula that works for this r. The idea is that a fixed number of vertices in a graph can be "encoded" as formulas. For $\bar{a}=(a_1,\dots,a_k)$, $\bar{b}=(b_1,\dots,b_l)$, $\bar{c}=(c_1,\dots,c_m)$ and $\bar{d}=(d_1,\dots,d_n)$, and for integer $r\geqslant 0$, define the formula $\Phi^r_{(\bar{a},\bar{b}^{+p})|(\bar{c},\bar{d}^{+p})}(\bar{x}_0,\bar{x}_1,\bar{y}_0,\bar{y}_1)$ as follows:

1. when $r=0$, let
$$\Phi^r_{(\bar{a},\bar{b}^{+p})|(\bar{c},\bar{d}^{+p})}(\bar{x}_0,\bar{x}_1,\bar{y}_0,\bar{y}_1) := \ \bar{x}_1\cdot\bar{1}=0 \ \wedge \ \Psi_{\bar{a}|(\bar{c},\bar{d}^{+p})}(\bar{x}_0,\bar{y}_0,\bar{y}_1),$$
where $\Psi_{\bar{a}|(\bar{c},\bar{d}^{+p})}(\bar{x}_0,\bar{y}_0,\bar{y}_1)$ is as defined in equation (5);
2. when $r\geqslant 1$, let $\bar{x}_1=(x_{1,1},\dots,x_{1,l})$ and
$$\Phi^r_{(\bar{a},\bar{b}^{+p})|(\bar{c},\bar{d}^{+p})}(\bar{x}_0,\bar{x}_1,\bar{y}_0,\bar{y}_1) \ :=$$
$$\exists s\exists\bar{z}_0\exists\bar{z}_1\exists\bar{z}_2\exists\bar{z}_3\bigvee_{i=1}^{l}\left(\begin{array}{l} \quad (x_{1,i}\neq 0) \ \wedge \ (b_i+ps \ = \ \bar{z}_1\cdot\bar{1}+\bar{z}_3\cdot\bar{1}) \ \wedge \ (s\neq\infty) \\ \wedge\,(\bar{z}_0+\bar{z}_1=\bar{y}_0) \ \wedge \ (\bar{z}_2+\bar{z}_3=\bar{y}_1) \\ \wedge\,\Phi^{r-1}_{(\bar{a},\bar{b}^{+p})|(\bar{c},\bar{c}-\bar{1},\bar{d}^{+p},(\bar{d}-\bar{1})^{+p})}(\bar{x}_0,\bar{x}_1-\mathbf{e}_i,\bar{z}_0,\bar{z}_1,\bar{z}_2,\bar{z}_3)\end{array}\right),$$
where $\mathbf{e}_i$ denotes the unit vector (with length k) where the i-th component is 1, and the lengths of $\bar{z}_0$ and $\bar{z}_1$ are the same as $\bar{y}_0$, and the lengths of $\bar{z}_2$ and $\bar{z}_3$ are the same as $\bar{y}_1$.

The motivation for these formulas will be explained in the proof of the following lemma.

▶ **Lemma 11.** *For every $\bar{a},\bar{b},\bar{c},\bar{d}$, every integer $r\geqslant 0$ and every $\bar{M}_0,\bar{M}_1,\bar{N}_0,\bar{N}_1$ such that*
1. $\bar{M}_0\cdot\bar{1}+\bar{N}_0\cdot\bar{1}+\bar{N}_1\cdot\bar{1}\geqslant 2\delta^2_{\max}+3$,
2. $\bar{N}_1\cdot\bar{1}\geqslant\delta^2_{\max}+1$,
3. $\bar{M}_1\cdot\bar{1}=r$,

where $\delta_{\max}=\max(p,\bar{a},\bar{c},\bar{d})$, the following holds: there is a $(\bar{a},\bar{b}^{+p})|(\bar{c},\bar{d}^{+p})$-biregular graph with size $(\bar{M}_0,\bar{M}_1)|(\bar{N}_0,\bar{N}_1)$ if and only if $\Phi^r_{(\bar{a},\bar{b}^{+p})|(\bar{c},\bar{d}^{+p})}(\bar{M}_0,\bar{M}_1,\bar{N}_0,\bar{N}_1)$ holds.

Proof. The proof is by induction on r. The base case $r=0$ follows from Lemma 9, so we focus on the induction step.

We begin with the "only if" direction, which provides the intuition for these formulas. Suppose $G=(U,V,E)$ is a $(\bar{a},\bar{b}^{+p})|(\bar{c},\bar{d}^{+p})$-biregular with size $(\bar{M}_0,\bar{M}_1)|(\bar{N}_0,\bar{N}_1)$. We let $U=U_{0,1}\cup\cdots\cup U_{0,k}\cup U_{1,1}\cup\cdots\cup U_{1,l}$, where $\bar{M}_0=(|U_{0,1}|,\dots,|U_{0,k}|)$ and $\bar{M}_1=(|U_{1,1}|,\dots,|U_{1,l}|)$. Likewise, we let $V=V_{0,1}\cup\cdots\cup V_{0,m}\cup V_{1,1}\cup\cdots\cup V_{1,n}$, where $\bar{N}_0=(|V_{0,1}|,\dots,|V_{0,m}|)$ and $\bar{N}_1=(|V_{1,1}|,\dots,|V_{1,n}|)$.

Since we are not in the base case, we can assume $\bar{M}_1\cdot\bar{1}=\sum_{i=1}^{l}|U_{1,i}|=r\neq 0$. Thus we can fix some i with $1\leqslant i\leqslant l$ such that $U_{1,i}\neq\emptyset$, and fix also some $u\in U_{1,i}$. Based on this u, we define, for each $1\leqslant j\leqslant m$, $Z_{0,j}$ to be the set of vertices in $V_{0,j}$ adjacent to u. For each $1\leqslant j\leqslant n$ we let $Z_{1,j}$ be the set of vertices in $V_{1,j}$ adjacent to u. Figure 1 illustrates the situation.

If we omit the vertex u and all its adjacent edges, we have the following:

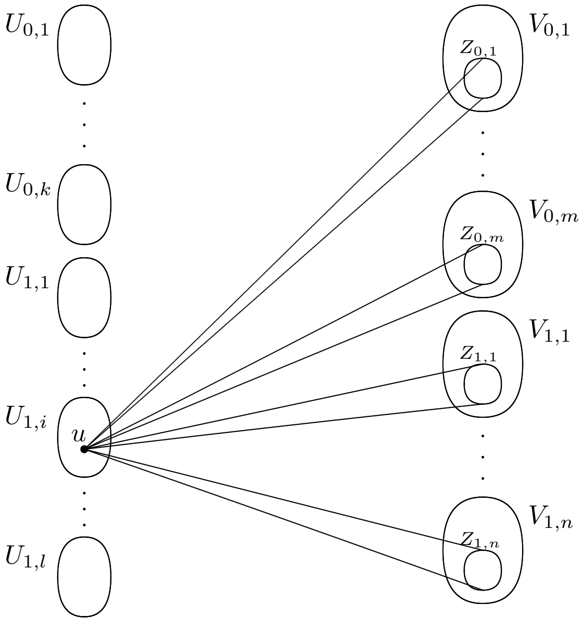


Figure 1 Inductive construction for the "not big enough" case.

1. for every $1 \leqslant j \leqslant m$, every vertex in $Z_{0,j}$ has degree $c_j - 1$,
2. for every $1 \leqslant j \leqslant n$, every vertex in $Z_{1,j}$ has degree $(d_j - 1)^{+p}$.

Thus, we have a $(\bar{a}, \bar{b}^{+p})|(\bar{c}, \bar{c} - \bar{1}, \bar{d}^{+p}, (\bar{d} - \bar{1})^{+p})$-biregular graph with size $(\bar{M}_0, \bar{M}_1 - \mathbf{e}_i)|(\bar{K}_{0,0}, \bar{K}_{0,1}, \bar{K}_{1,0}, \bar{K}_{1,1})$, where

$$\bar{K}_0 = (|V_{0,1}| - |Z_{0,1}|, \ldots, |V_{0,m}| - |Z_{0,m}|), \qquad \bar{K}_1 = (|Z_{0,1}|, \ldots, |Z_{0,m}|),$$
$$\bar{K}_2 = (|V_{1,1}| - |Z_{1,1}|, \ldots, |V_{1,n}| - |Z_{1,n}|), \qquad \bar{K}_3 = (|Z_{1,1}|, \ldots, |Z_{1,n}|).$$

We can check that the sizes allow us to apply the induction hypothesis to this graph, keeping in mind that the sizes on the left have now decreased by one. We conclude that $\Phi^{r-1}_{(\bar{a},\bar{b}^{+p})|(\bar{c},\bar{c}-\bar{1},\bar{d}^{+p},(\bar{d}-\bar{1})^{+p})}(\bar{M}_0, \bar{M}_1 - \mathbf{e}_i)|(\bar{K}_{0,0}, \bar{K}_{0,1}, \bar{K}_{1,0}, \bar{K}_{1,1})$ holds. Moreover, since $u \in U_{1,i}$, and hence the degree of u is b_i^{+p}, we have $\bar{K}_1 \cdot \bar{1} + \bar{K}_3 \cdot \bar{1} = b_i + ps$, for some integer $s \geqslant 0$. Note also that $\bar{K}_0 + \bar{K}_1 = \bar{N}_0$ and $\bar{K}_2 + \bar{K}_3 = \bar{N}_1$. Thus, $\Phi^{r}_{(\bar{a},\bar{b}^{+p})|(\bar{c},\bar{d}^{+p})}(\bar{M}_0, \bar{M}_1)|(\bar{N}_0, \bar{N}_1)$ holds where the variables $\bar{z}_0, \bar{z}_1, \bar{z}_2, \bar{z}_3$ are assigned with $\bar{K}_0, \bar{K}_1, \bar{K}_2, \bar{K}_3$, respectively.

For the "if" direction, suppose $\Phi^{r}_{(\bar{a},\bar{b}^{+p})|(\bar{c},\bar{d}^{+p})}(\bar{M}_0, \bar{M}_1, \bar{N}_0, \bar{N}_1)$ holds. Then we can fix some $s, \bar{z}_0, \bar{z}_1, \bar{z}_2, \bar{z}_3$, and i such that (a) $x_{1,i} \neq 0$, (b) $b_i + ps = \bar{z}_1 \cdot \bar{1} + \bar{z}_3 \cdot \bar{1}$, (c) $\bar{z}_0 + \bar{z}_1 = \bar{N}_0$, (d) $\bar{z}_2 + \bar{z}_3 = \bar{N}_1$, and (e) $\Phi^{r-1}_{(\bar{a},\bar{b}^{+p})|(\bar{c},\bar{c}-\bar{1},\bar{d}^{+p},(\bar{d}-\bar{1})^{+p})}(\bar{M}_0, \bar{M}_1 - \mathbf{e}_i, \bar{z}_0, \bar{z}_1, \bar{z}_2, \bar{z}_3)$ holds.

We prove from this that a biregular graph of the appropriate size exists. Note that the hypothesis requires that $\bar{M}_0 \cdot \bar{1} + \bar{N}_0 \cdot \bar{1} + \bar{N}_1 \cdot \bar{1} \geqslant 2\delta^2_{\max} + 3$, where $\delta_{\max}$ is as defined in the statement of the lemma. Since $\max(p, \bar{a}, \bar{b}, \bar{c}, \bar{c} - \bar{1}, \bar{d}, \bar{d} - \bar{1}) = \delta_{\max}$, the equalities in (c) and (d) imply that $\bar{M}_0 \cdot \bar{1} + \bar{z}_0 \cdot \bar{1} + \bar{z}_1 \cdot \bar{1} + \bar{z}_2 \cdot \bar{1} + \bar{z}_3 \cdot \bar{1}$ is bigger than $2\delta^2_{\max} + 3$.

Note that $(\bar{M}_1 - \mathbf{e}_i) \cdot \bar{1} = r - 1$. Thus we can apply the induction hypothesis and obtain a $(\bar{a}, \bar{b}^{+p})|(\bar{c}, \bar{c} - \bar{1}, \bar{d}^{+p}, (\bar{d} - \bar{1})^{+p})$-biregular graph $G = (U, V, E)$ with size $(\bar{M}_0, \bar{M}_1 - \mathbf{e}_i)|(\bar{z}_0, \bar{z}_1, \bar{z}_2, \bar{z}_3)$. Let $V = V_0 \cup V_1 \cup V_2 \cup V_3$ be the partition of V, where

$$V_0 = V_{0,1} \cup \cdots \cup V_{0,m}, \qquad V_1 = V_{1,1} \cup \cdots \cup V_{1,m},$$
$$V_2 = V_{2,1} \cup \cdots \cup V_{2,n}, \qquad V_3 = V_{3,1} \cup \cdots \cup V_{3,n},$$

and such that

1. for every $1 \leqslant i \leqslant m$, the degree of vertices in $V_{0,j}$ and $V_{1,j}$ are c_j and $c_j - 1$, respectively;
2. for every $1 \leqslant i \leqslant n$, the degree of vertices in $V_{2,j}$ and $V_{3,j}$ are d_j^{+p} and $(d_j - 1)^{+p}$, respectively.

Note also that $\bar{z}_0 = (|V_{0,1}|, \ldots, |V_{0,m}|)$, $\bar{z}_1 = (|V_{1,1}|, \ldots, |V_{1,m}|)$, $\bar{z}_2 = (|V_{2,1}|, \ldots, |V_{2,m}|)$, and $\bar{z}_3 = (|V_{3,1}|, \ldots, |V_{3,m}|)$.

Let u be a fresh vertex. We can construct a $(\bar{a}, \bar{b}^{+p})|(\bar{c}, \bar{d}^{+p})$-biregular graph $G' = (U \cup \{u\}, V, E')$, by connecting the vertex u with every vertex in $V_1 \cup V_3$. Note that the formula states that $\bar{z}_1 \cdot \bar{1} + \bar{z}_3 \cdot \bar{1} = b_i + ps$, which equals to $|V_1| + |V_3|$, thus, the degree of u is $b_i + ps$, which satisfies our requirement for a vertex to be in U_i. Since prior to the connection, the degrees of $V_{1,j}$ and $V_{3,j}$ are $c_j - 1$ and $(d_j - 1)^{+p}$, after connecting u with each vertex in $V_1 \cup V_3$, their degrees become c_j and d_j^{+p}. That is, the right side vertices now have the desired degrees, i.e., G' is $(\bar{a}, \bar{b}^{+p})|(\bar{c}, \bar{d}^{+p})$-biregular. Moreover, $\bar{z}_0 + \bar{z}_1 = \bar{N}_0$ and $\bar{z}_2 + \bar{z}_3 = \bar{N}_1$. Thus, the resulting graph G' has size $(\bar{M}_0, \bar{M}_1)|(\bar{N}_0, \bar{N}_1)$. ◀

The formula $\mathsf{bireg}_{(\bar{a},\bar{b}^{+p})|(\bar{c},\bar{d}^{+p})}(\bar{x}_0, \bar{x}_1, \bar{y}_0, \bar{y}_1)$ characterizing the sizes of $(\bar{a}, \bar{b}^{+p})|(\bar{c}, \bar{d}^{+p})$-biregular graphs can be defined by combining all the cases described above.

4.2 Proof of Lemma 2 for 1-color graphs (the complete case)

We now turn to bootstrapping the biregular case to add the completeness requirement imposed in Lemma 2. Let $\bar{a} = (a_1, \ldots, a_k)$, $\bar{b} = (b_1, \ldots, b_l)$, $\bar{c} = (c_1, \ldots, c_m)$ and $\bar{d} = (d_1, \ldots, d_n)$. Let $\bar{x}_0 = (x_{0,1}, \ldots, x_{0,k})$, $\bar{x}_1 = (x_{1,1}, \ldots, x_{1,l})$, $\bar{y}_0 = (y_{0,1}, \ldots, y_{0,m})$, and $\bar{y}_1 = (y_{1,1}, \ldots, y_{1,n})$.

The formula $\mathsf{c\text{-}bireg}_{(\bar{a},\bar{b}^{+p})|(\bar{c},\bar{d}^{+p})}(\bar{x}_0, \bar{x}_1, \bar{y}_0, \bar{y}_1)$ for the sizes of complete $(\bar{a}, \bar{b}^{+p})|(\bar{c}, \bar{d}^{+p})$-biregular graphs is the conjunction of $\mathsf{bireg}_{(\bar{a},\bar{b}^{+p})|(\bar{c},\bar{d}^{+p})}(\bar{x}_0, \bar{x}_1, \bar{y}_0, \bar{y}_1)$ such that

1. for every $1 \leqslant i \leqslant k$, if $x_{0,i} \neq 0$, then $\bar{y}_0 \cdot \bar{1} + \bar{y}_1 \cdot \bar{1} = a_i$;
2. for every $1 \leqslant i \leqslant l$, if $x_{1,i} \neq 0$, then $\bar{y}_0 \cdot \bar{1} + \bar{y}_1 \cdot \bar{1} = b_i + pz_i$, for some z_i;
3. for every $1 \leqslant i \leqslant m$, if $y_{0,i} \neq 0$, then $\bar{x}_0 \cdot \bar{1} + \bar{x}_1 \cdot \bar{1} = c_i$;
4. for every $1 \leqslant i \leqslant n$, if $y_{1,i} \neq 0$, then $\bar{x}_0 \cdot \bar{1} + \bar{x}_1 \cdot \bar{1} = d_i + pz_i$, for some z_i.

To understand these additional conditions, consider a complete biregular graph meeting the cardinality specification. The completeness criterion for 1-color graphs implies that each element on the left is connected to every element on the right. Thus if the size of a partition required to have fixed outdegree a_i is non-empty, we must have that a_i is exactly the cardinality of the number of elements on the right. This is what is captured in the first item. If we have non-empty size for a partition whose outdegree is constrained to be b_i plus a multiple of p, then the total number of elements on the right must be b_i plus a multiple of p. This is what the second item specifies. Considering elements on the left motivates the third and fourth item. Thus we see that these conditions are necessary.

Suppose $\mathsf{c\text{-}bireg}_{(\bar{a},\bar{b}^{+p})|(\bar{c},\bar{d}^{+p})}(\bar{M}_0, \bar{M}_1, \bar{M}_0, \bar{M}_1)$ holds. Then, there is a $(\bar{a}, \bar{b}^{+p})|(\bar{c}, \bar{d}^{+p})$-biregular graph $G = (U, V, E)$ with size $(\bar{M}_0, \bar{M}_1)|(\bar{N}_0, \bar{N}_1)$, which are not necessarily complete. Note that $\bar{N}_0 \cdot \bar{1} + \bar{N}_1 \cdot \bar{1}$ is precisely the number of vertices in V. The first item states that the existence of a vertex u with degree a_i implies u is adjacent to every vertex in V. Now, suppose there is a vertex $u \in U$ with degree b_i^{+p}. If u is not adjacent to every vertex in V, then we can add additional edges so that u is adjacent to every vertex in V. The second item states that $|V| = b_i^{+p}$. Thus, adding such edges is legal, since the degree of u stays b_i^{+p}. We can make vertices in V adjacent to every vertex in U using the same argument.

4.3 The proof for regular digraphs

Recall that in the prior argument we consider only digraphs without any self-loop. Thus, a digraph can be viewed as a bipartite graph by splitting every vertex u into two vertices, where one is adjacent to all the incoming edges, and the other to all the outgoing edges. Thus, $A|B$-regular digraphs with size $\bar{M}$ can be characterized as $A|B$-biregular graphs with size $\bar{M}|\bar{M}$. For more details, see [13, Section 8].

5 Extensions and applications

A *type/behavior profile* for a model M is the vector of cardinalities of the sets $A_{\pi,g}$ computed in M, where π ranges of 1-types and g over behavior functions (for a fixed ϕ). Recall that in the proof Theorem 4 we actually showed, in Lemma 6, that we can obtain existential Presburger formulas which define exactly the vectors of integers that arise as the type/behavior profiles of models of ϕ. The domain of the model can be broken up as a disjoint union of sets $A_{\pi,g}$, and thus its cardinality is a sum of numbers in this vector. We can thus add one additional integer variable x_{total} in PRES_ϕ, which will be free, with an additional equation stating that x_{total} is the sum of all $X_{\pi,g}$'s. This allows us to conclude definability of the spectrum.

▶ **Theorem 12.** *From an FO^2_{Pres} sentence ϕ, we can effectively construct a Presburger formula $\psi(n)$ such that $\mathcal{N} \models \psi(n)$ exactly when n is the size of a finite structure that satisfies ϕ, and similarly a formulas $\psi_\infty(n)$ such that $\mathcal{N}_\infty \models \psi_\infty(n)$ exactly when n is the size of a finite or countably infinite model of ϕ.*

We say that ϕ has *NP data complexity of (finite) satisfiability* if there is a non-deterministic algorithm that takes as input a set of ground atoms A and determines whether $\phi \wedge \bigwedge A$ is satisfiable, running in time polynomial in the size of A. Pratt-Hartmann [20] showed that C^2 formulas have NP data complexity of both satisfiability and finite satisfiability. Following the general approach to data complexity from [20], while plugging in our Presburger characterization of $\mathsf{FO}^2_{\mathrm{Pres}}$, we can show that the same data complexity bound holds for $\mathsf{FO}^2_{\mathrm{Pres}}$.

▶ **Theorem 13.** *FO^2_{Pres} formulas have NP data complexity of satisfiability and finite satisfiability.*

Proof. We give only the proof for finite satisfiability. We will follow closely the approach used for C^2 in Section 4 of [20], and the terminology we use below comes from that work.

Given a set of facts D, our algorithm guesses a set of facts (including equalities) on elements of D, giving us a finite set of facts D^+ extending D, but with the same domain as D. We check that our guess is consistent with the universal part α and such that equality satisfies the usual transitivity and congruence rules.

Now consider 1-types and 2-types with an additional predicate Observable. Based on this extended language, we consider good functions as before, and define the formulas $\mathsf{consistent}_1$ and $\mathsf{consistent}_2$ based on them. 1-types with that contain the predicate Observable will be referred to as observable 1-types. The restriction of a behavior function to observable 1-types will be called an *observable behavior*. Given a structure M, an observable one-type π, and an observable behavior function g_0, we let M_{π,g_0} be the elements of M having 1-type π and observable behavior g_0, and we analogously let D_{π,g_0} be the elements of D whose 1-type and behavior in D^+ match π and g_0.

We declare that all elements in A are in the predicate Observable. Add to the formulas $\mathsf{consistent}_1$ and $\mathsf{consistent}_2$ additional conjuncts stating that for each observable 1-type π and for each observable behavior function g_0, the total sum of the number of elements with 1-type π and a behavior function g extending g_0 (i.e., the cardinality of M_{π,g_0}) is the same as $|D_{\pi,g_0}|$. with the cardinality being counted modulo equalities of D^+.

At this point our algorithm returns true exactly when the sentence obtained by existentially quantifying this extended set of conjuncts is satisfiable in the integers. The solving procedure is certainly in NP. In fact, since the number of variables is fixed, with only the constants varying, it is in PTIME [17].

We argue for correctness, focusing on the proof that when the algorithm returns true we have the desired model. Assuming the constraints above are satisfied, we get a graph, and from the graph we get a model M. M will clearly satisfy ϕ, but its domain does not contain the domain of D. Letting O be the elements of M satisfying Observable, we know, from the additional constraints imposed, that the cardinality of O matches the cardinality of the domain of D modulo the equalities in D^+, and for each observable 1-type π_o and observable behavior g_0, $|M_{\pi,g_0}| = |D_{\pi,g_0}$.

Fix an isomorphism λ taking each M_{π,g_0} to (equality classes of) D_{π,g_0}. Create M' by redefining M on O by connecting pairs (o_1, o_2) via E exactly when $\lambda(o_1), \lambda(o_2)$ ise connected via E in D^+. We can thus identify O with D^+ modulo equalities in M'.

Clearly M' now satisfies D. To see that M' satisfies ϕ, we simply note that since all of the observable behaviors are unchanged in moving from an element e in M to the corresponding element $\lambda(e)$ in M', and every such e modified has an observable type, it follows that the behavior of every element in M is unchanged in moving from M to M'. Since the 1-types are also unchanged, M' satisfies ϕ. ◀

Note that the data complexity result here is best possible, since even for FO^2 the data complexity can be NP-hard [20].

6 Conclusion

We have shown that we can extend the powerful language two-variable logic with counting to include ultimately periodic counting quantifiers without sacrificing decidability, and without losing the effective definability of the spectrum of formulas within Presburger arithmetic. We believe that by refining our proof we can obtain a 2NEXPTIME bound on complexity. However the only lower bound we know of is NEXPTIME, inherited from FO^2. We leave the analysis of the exact complexity for future work.

References

1 Franz Baader. A new description logic with set constraints and cardinality constraints on role successors. In *FROCOS*, 2017.

2 Bartosz Bednarczyk. One-variable logic meets Presburger arithmetic. *Theor. Comput. Sci.*, 802:141–146, 2020.

3 Bartosz Bednarczyk, Franz Baader, and Sebastian Rudolph. Satisfiability and query answering in description logics with global and local cardinality constraints. In *ECAI*, 2020.

4 Bartosz Bednarczyk and Witold Charatonik. Modulo counting on words and trees. In *FSTTCS*, 2017.

5 Witold Charatonik, Yegor Guskov, Ian Pratt-Hartmann, and Piotr Witkowski. Two-variable first-order logic with counting in forests. In *LPAR*, 2018.

6 Witold Charatonik and Piotr Witkowski. Two-variable logic with counting and trees. *ACM Trans. Comput. Log.*, 17(4):31:1–31:27, 2016.

7 Stéphane Demri and Denis Lugiez. Complexity of modal logics with Presburger constraints. *J. Applied Logic*, 8(3):233–252, 2010.

8 Arnaud Durand, Neil D. Jones, Johann A. Makowsky, and Malika More. Fifty years of the spectrum problem: survey and new results. *The Bulletin of Symbolic Logic*, 18(4):505–553, 2012.

9 Erich Grädel, Phokion G. Kolaitis, and Moshe Y. Vardi. On the decision problem for two-variable first-order logic. *Bull. Symbolic Logic*, 3(1):53–69, March 1997.

10 Erich Grädel, Martin Otto, and Eric Rosen. Two-variable logic with counting is decidable. In *LICS*, 1997.

11 Erich Grädel, Martin Otto, and Eric Rosen. Undecidability results on two-variable logics. *Arch. Math. Logic*, 38:313–354, 1999.

12 Emanuel Kieronski, Jakub Michaliszyn, Ian Pratt-Hartmann, and Lidia Tendera. Two-variable first-order logic with equivalence closure. *SIAM J. Comput.*, 43(3):1012–1063, 2014.

13 Eryk Kopczyński and Tony Tan. Regular graphs and the spectra of two-variable logic with counting. *SIAM J. Comput.*, 44(3):786–818, 2015.

14 Viktor Kuncak and Martin C. Rinard. Towards efficient satisfiability checking for boolean algebra with Presburger arithmetic. In *CADE*, 2007.

15 Michael Mortimer. On language with two variables. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 21:135–140, 1975.

16 Leszek Pacholski, Wieslaw Szwast, and Lidia Tendera. Complexity results for first-order two-variable logic with counting. *SIAM J. Comput.*, 29(4):1083–1117, 2000.

17 Christos H. Papadimitriou. On the complexity of integer programming. *J. ACM*, 28(4):765–768, October 1981.

18 Ian Pratt-Hartmann. Complexity of the two-variable fragment with counting quantifiers. *Journal of Logic Language and Information*, 14:369–395, 2005.

19 Ian Pratt-Hartmann. Complexity of the guarded two-variable fragment with counting quantifiers. *J. Log. Comput.*, 17(1):133–155, 2007.

20 Ian Pratt-Hartmann. Data-complexity of the two-variable fragment with counting quantifiers. *Inf. Comput.*, 207(8):867–888, 2009.

21 Ian Pratt-Hartmann. The two-variable fragment with counting revisited. In *WoLLIC*, 2010.

22 Ian Pratt-Hartmann. The two-variable fragment with counting and equivalence. *Math. Log. Q.*, 61(6):474–515, 2015.

23 Dana Scott. A decision method for validity of sentences in two variables. *The Journal of Symbolic Logic*, page 377, 1962.

Single-Use Automata and Transducers for Infinite Alphabets

Mikołaj Bojańczyk
Institute of Informatics, University of Warsaw, Poland
bojan@mimuw.edu.pl

Rafał Stefański
Institute of Informatics, University of Warsaw, Poland
rafal.stefanski@mimuw.edu.pl

Abstract

Our starting point are register automata for data words, in the style of Kaminski and Francez. We study the effects of the single-use restriction, which says that a register is emptied immediately after being used. We show that under the single-use restriction, the theory of automata for data words becomes much more robust. The main results are: (a) five different machine models are equivalent as language acceptors, including one-way and two-way single-use register automata; (b) one can recover some of the algebraic theory of languages over finite alphabets, including a version of the Krohn-Rhodes Theorem; (c) there is also a robust theory of transducers, with four equivalent models, including two-way single use transducers and a variant of streaming string transducers for data words. These results are in contrast with automata for data words without the single-use restriction, where essentially all models are pairwise non-equivalent.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory

Keywords and phrases Automata, semigroups, data words, orbit-finite sets

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.113

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

Related Version A full version of the paper is available at `https://arxiv.org/abs/1907.10504`.

Funding Supported by the European Research Council under the European Unions Horizon 2020 research and innovation programme (ERC consolidator grant LIPA, agreement no. 683080).

1 Introduction

One of the appealing features of regular languages for finite alphabets is the robustness of the notion: it can be characterised by many equivalent models of automata (one-way, two-way, deterministic, nondeterministic, alternating, etc.), regular expressions, finite semigroups, or monadic second-order logic. A similar robustness appears for transducers, see [11] for a survey; particularly for the class of regular string-to-string functions, which can be characterised using deterministic two-way transducers, streaming string transducers, or MSO transductions.

This robustness vanishes for infinite alphabets. We consider infinite alphabets that are constructed using an infinite set $\mathbb{A}$ of atoms, also called data values. Atoms can only be compared for equality. The literature for infinite alphabets is full of depressing diagrams like [15, Figure 1] or [6, p. 24], which describe countless models that satisfy only trivial relationships such as deterministic ⊆ nondeterministic, one-way ⊆ two-way, etc.

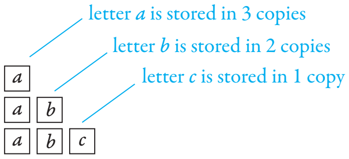
This lack of robustness has caused several authors to ask if there is a notion of "regular language" for infinite alphabets; see [3, p. 703] or [4, p. 2]. This question was probably rhetorical, with the assumed answer being "no". In this paper, we postulate a "yes" answer. The main theme is register automata, as introduced by Kaminski and Francez [13], but with

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 113; pp. 113:1–113:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

the single-use restriction, which says that immediately after a register is used, its value is destroyed. As we show in this paper, many automata constructions, which fail for unrestricted register automata, start to work again in the presence of the single-use restriction.

Before describing the results in the paper, we illustrate the single-use restriction.

▶ **Example 1.** Consider the language "there are at most three distinct letters in the input word, not counting repetitions", over alphabet $\mathbb{A}$. There is a natural register automaton which recognises this language: use three registers to store the distinct atoms that have been seen so far, and if a fourth atom comes up, then reject. This automaton, however, violates the single-use restriction, because each new input letter is compared to all the registers.


letter *a* is stored in 3 copies
letter *b* is stored in 2 copies
letter *c* is stored in 1 copy
a
a *b*
a *b* *c*

Here is a solution that respects the single-use restriction. The idea is that once the automaton has seen three distinct letters a, b, c, it stores them in six registers as explained in the picture on the right. Assume that a new input letter d is read. The behaviour of the automaton (when it already has three atoms in its registers) is explained in the flowchart in Figure 1.

A similar flowchart is used for the corner cases when the automaton has seen less than three letters so far.

Our first main result, Theorem 6 (in Section 3), says that the following models recognise the same languages over infinite alphabets:

1. deterministic one-way single-use automata;
2. deterministic two-way single-use automata;
3. orbit-finite monoids [5];
4. rigidly guarded MSO$^{\sim}$ [9];
5. string-to-boolean regular list functions with atoms.

The equivalence of the models in items 3 and 4 was shown in [9]; the remaining models and their equivalences are new (item 5 is an extension of the regular list functions from [7]).

Just like their classical versions, one-way and two-way single-use automata are equivalent as language acceptors, but they are no longer equivalent as transducers. For example, a two-way single-use transducer can reverse the input string, which is impossible for a one-way single-use transducer. In Sections 4 and 5 we develop the theory of single-use transducers:

In Section 4, we investigate single-use one-way transducers. For finite alphabets, one of the most important results about one-way transducers is the Krohn-Rhodes Theorem [14], which says that every Mealy machine (which is a length preserving one-way transducer) can be decomposed into certain "prime" Mealy machines. We show that the same can be done for infinite alphabets, using a single-use extension of Mealy machines. The underlying prime machines are the all the machines from the original Krohn-Rhodes theorem, plus one additional register machine which moves atoms to later positions.

In Section 5, we investigate single-use two-way transducers, and show that the corresponding class of string-to-string functions enjoys similar robustness properties as the languages discussed in Theorem 6, with four models being equivalent:

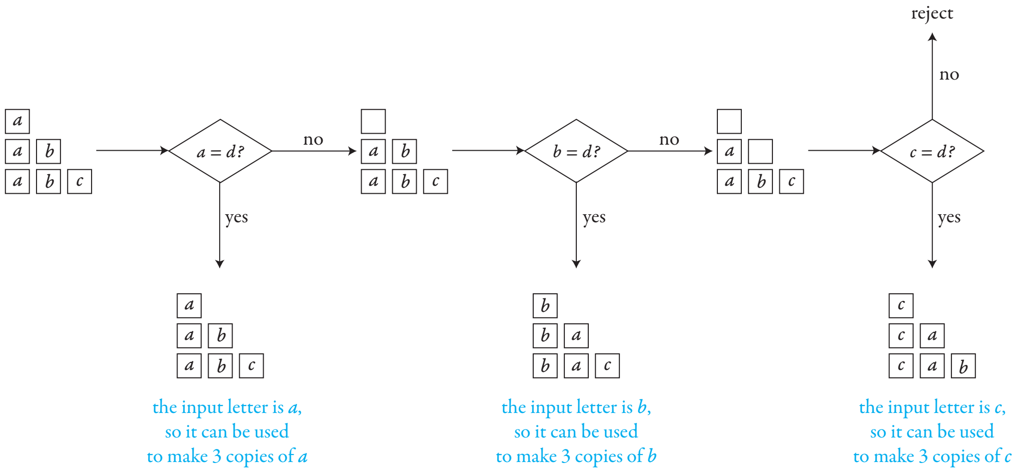


Figure 1 Updating the six registers.

1. single-use two-way transducers;
2. an atom extension of streaming string transducers [2];
3. string-to-string regular list functions with atoms;
4. compositions of certain "prime two-way machines" (Krohn & Rhodes style).

We also show other good properties of the string-to-string functions in the above items, including closure under composition (which follows from item 4) and decidable equivalence.

Summing up, the single-use restriction allows us to identify languages and string-to-string functions with infinite alphabets, which share the robustness and good mathematical theory usually associated with regularity for finite alphabets.

Due to space constraints, and a large number of results, virtually all of the proofs are in an appendix. We use the available space to explain and justify the many new models that are introduced.

2 Automata and transducers with atoms

For the rest of the paper, fix an infinite set $\mathbb{A}$, whose elements are called *atoms*. Atoms will be used to construct infinite alphabets. Intuitively speaking, atoms can only be compared for equality. It would be interesting enough to consider alphabets of the form $\mathbb{A} \times \Sigma$, for some finite Σ, as is typically done in the literature on data words [4, p. 1]. However, in the proofs, we use more complicated sets, such as the set $\mathbb{A}^2$ of pairs of atoms, the set $\mathbb{A} + \{\vdash, \dashv\}$ obtained by adding two endmarkers to the atoms, or the co-product (i.e. disjoint union) $\mathbb{A}^2 + \mathbb{A}^3$. This motivates the following definition.

▶ **Definition 2.** *A* polynomial orbit-finite set[1] *is any set that can be obtained from* $\mathbb{A}$ *and singleton sets by means of finite products and co-products (i.e. disjoint unions).*

We only care about properties of such sets that are stable under atom automorphisms, as described below. Define an *atom automorphism* to be any bijection $\mathbb{A} \to \mathbb{A}$. (This notion of automorphism formalises the intuition that atoms can only be compared for equality). Atom automorphisms form a group. There is a natural action of this group on polynomial

[1] The name "orbit-finite" is used because the above definition is a special case of orbit-finite sets discussed later in the paper, and the name "polynomial" is used to underline that the sets are closed under products and co-products.

orbit-finite sets: for elements of $\mathbb{A}$ we apply the atom automorphism, for singleton sets the action is trivial, and for other polynomial orbit-finite sets the action is lifted inductively along $+$ and $\times$ in the natural way. Let Σ and Γ be sets equipped with an action of the group of atom automorphisms – in particular, these could be polynomial orbit-finite sets. A function $f : \Sigma \to \Gamma$ is called *equivariant* if $f(\pi(x)) = \pi(f(x))$ holds for every $x \in \Sigma$ and every atom automorphism π. The general idea is that equivariant functions can only talk about equality of atoms. In the case of polynomial orbit-finite sets, equivariant functions can also be finitely represented using quantifier-free formulas [6, Lemma 1.3].

The model. We now describe the single-use machine models discussed in this paper. There are four variants: machines can be one-way or two-way, and they can recognise languages or compute string-to-string functions. We begin with the most general form – two-way string-to-string functions – and define the other models as special cases.

The machine reads the input string, extended with left and right endmarkers $\vdash, \dashv$. It uses registers to store atoms that appear in the input string. A register can store either an atom, or the undefined value $\perp$. The single-use restriction, which is highlighted in bold below, says that a register is set to $\perp$ immediately after being used.

▶ **Definition 3.** *The syntax of a* two-way single-use transducer[2] *consists of*

- *input and output alphabets Σ and Γ, both polynomial orbit-finite sets;*
- *a finite set of states Q, with a distinguished initial state $q_0 \in Q$;*
- *a finite set R of register names;*
- *a* transition function *which maps each state $q \in Q$ to an element of:*

$$\underbrace{\textit{questions}}_{\textit{question that is asked}} \times \underbrace{(Q \times \textit{actions})}_{\substack{\textit{what to do if the}\\ \textit{question has a yes answer}}} \times \underbrace{(Q \times \textit{actions})}_{\substack{\textit{what to do if the}\\ \textit{question has a no answer}}}$$

where the allowed questions and actions are taken from the following toolkit:

1. Questions.
   - **a.** *Apply an equivariant function $f : \Sigma + \{\vdash, \dashv\} \to \{yes, no\}$ to the letter under the head, and return the answer.*
   - **b.** *Are the atoms stored in registers r_1, r_2 equal and defined? If any of these registers is undefined, then the run immediately stops and rejects[3].* ***This question has the side effect of setting the values of r_1 and r_2 to $\perp$.***
2. Actions.
   - **a.** *Apply an equivariant function $f : \Sigma + \{\vdash, \dashv\} \to \mathbb{A} + \perp$ to the letter under the head, and store the result in register $r \in R$.*
   - **b.** *Apply an equivariant function $f : \mathbb{A}^k \to \Gamma$ to the contents of distinct registers $r_1, \ldots, r_k \in R$, and append the result to the output string. If any of the registers is undefined, stop and reject.* ***This action has the side effect of setting the values of $r_1, r_2, \ldots, r_k$ to $\perp$.***
   - **c.** *Move the head to the previous/next input position.*
   - **d.** *Accept/reject and finish the run.*

[2] Unless otherwise noted, all transducers and automata considered in this paper are deterministic. The theory of nondeterministic single-use models seems to be less appealing.

[3] By remembering in the state which registers are defined, one can modify an automaton so that this never happens.

The semantics of the transducer is a partial function from strings over the input alphabet to strings over the output alphabet. Consider a string of the form $\vdash w \dashv$ where $w \in \Sigma^*$. A *configuration* over such a string consists of (a) a position in the string; (b) a state; (c) a register valuation, which is a function of type $R \to \mathbb{A} + \bot$; (d) an output string, which is a string over the output alphabet. A *run* of the transducer is defined to be a sequence of configurations, where consecutive configurations are related by applying the transition function in the natural way. The *output* of a run is defined to be the contents of the output string in the last configuration. An *accepting configuration* is one which executes the accept action from item 2d – accepting configurations have no successors. The *initial configuration* is a configuration where the head is over the left endmarker $\vdash$, the state is the initial state, the register valuation maps all registers to the undefined value, and the output string is empty. An *accepting run* is a run that begins in the initial configuration and ends in an accepting one. By determinism, there is at most one accepting run. The semantics of the transducer is defined to be the partial function $\Sigma^* \to \Gamma^*$, which inputs $w \in \Sigma^*$ and returns the output of the accepting run over $\vdash w \dashv$. If there is no accepting run, $f(w)$ has no value.

Special cases. A *one-way single-use transducer* is the special case of Definition 3 which does not use the "previous" action from item 2c. A *two-way single-use automaton* is the special case which does not use the output actions from item 2b. The *language* recognised by such an automaton is defined to be the set of words which admit an accepting run. A *one-way single-use automaton* is the special case of a two-way single-use automaton, which does not use the "previous" action from item 2c.

3 Languages recognised by single-use automata

In this section we discuss languages recognised by single-use automata. The main result is that one-way and two-way single-use automata recognise the same languages, and furthermore these are the same languages that are recognised by orbit-finite monoids [5], the logic rigidly guarded MSO$^\sim$ [9], and a new model called regular list functions with atoms, that will be defined in Section 5.

Orbit-finite monoids. We begin by defining orbit-finite sets and orbit-finite monoids, which play an important technical role in this paper. For more on orbit-finite sets, see the lecture notes [6]. For a tuple $\bar{a} \in \mathbb{A}^*$, an $\bar{a}$-automorphism is defined to be any atom automorphism that maps $\bar{a}$ to itself. Consider set X equipped with an action of the group of atom automorphisms. We say that $x \in X$ is *supported* by a tuple of atoms $\bar{a} \in \mathbb{A}^*$ if $\pi(x) = x$ holds for every $\bar{a}$-automorphism π. We say that a subset of X is $\bar{a}$-supported if it is an $\bar{a}$-supported element of the powerset of X; similarly we define supports of relations and functions. We say that x is finitely supported if it is supported by some tuple $\bar{a} \in \mathbb{A}^*$. Define the *$\bar{a}$-orbit of* x to be its orbit under the action of the group of $\bar{a}$-automorphisms.

▶ **Definition 4** (Orbit-finite sets). *Let X be a set equipped with an action of atom automorphisms. A subset $Y \subseteq X$ is called* orbit-finite *if (a) every element of Y is finitely supported; and (b) there exists some $\bar{a} \in \mathbb{A}^*$ such that Y is a union of finitely many $\bar{a}$-orbits.*

An equivariant orbit-finite set is the special case where the tuple $\bar{a}$ in item (b) is empty. The polynomial orbit-finite sets from Section 2 are a special case of equivariant orbit-finite sets[4]. The following notion was introduced in [5, Section 3].

▶ **Definition 5** (Orbit-finite monoid). *An* orbit-finite monoid *is a monoid where the underlying set is orbit-finite, and the monoid operation is finitely supported. If Σ is an orbit-finite set, then we say that a language $L \subseteq \Sigma^*$ is* recognised *by an orbit-finite monoid M if there is a finitely supported monoid morphism $h : \Sigma^* \to M$ and a finitely supported accepting set $F \subseteq M$ such that L contains exactly the words whose image under h belongs to F.*

In this paper, we are mainly interested in the case where both the morphism and the accepting set are equivariant. In this case, it follows that the alphabet Σ, the image of the morphism, and the recognised language all also have to be equivariant.

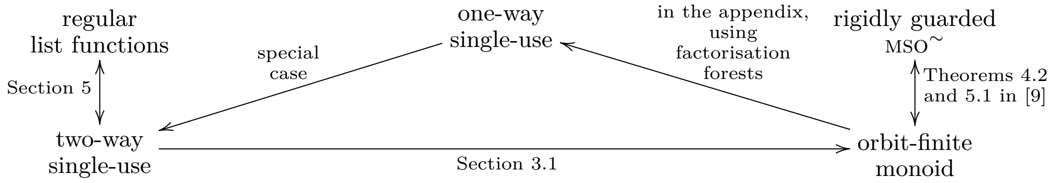
The structural theory of orbit-finite monoids was first developed in [5], where it was shown how the classical results about Green's relations for finite monoids extend to the orbit-finite setting. This theory was further investigated in [9], including a lemma stating that every orbit-finite group is necessarily finite. In the full version of this paper we build on these results, to prove an orbit-finite version of the Factorisation Forest Theorem of Simon [17, Theorem 6.1], which is used in proofs of Theorems 6 and 12.

Main theorem about languages. We are now ready to state Theorem 6, which is our main result about languages.

▶ **Theorem 6.** *Let Σ be a polynomial orbit-finite set. The following conditions are equivalent for every language $L \subseteq \Sigma^*$:*

1. *L is recognised by a single-use one-way automaton;*
2. *L is recognised by a single-use two-way automaton;*
3. *L is recognised by an orbit-finite monoid, with an equivariant morphism and an equivariant accepting set;*
4. *L can be defined in the rigidly guarded $\text{MSO}^\sim$ logic;*
5. *L's characteristic function $\Sigma^* \to \{yes, no\}$ is an orbit-finite regular list function.*

The equivalence of items 4 and 3 has been proved in [9, Theorems 4.2 and 5.1], and since we do not use rigidly guarded $\text{MSO}^\sim$ outside of the this theorem, we do not give a definition here (see [9, Section 3]). The orbit-finite regular list functions from item 5 will be defined in Section 5. The proof outline for Theorem 6 is given in the following diagram



All equivalences in the theorem are effective, i.e. there are algorithms implementing the conversions between any of the models.

The single-use restriction is crucial in the theorem. Automata without the single-use restriction – call them multiple-use – only satisfy the trivial inclusions:

[4] The converse does not hold – there exist sets that are equivariant orbit finite but not polynomial orbit finite e. g. the set of unordered pairs of atoms: $\{\{a, b\} \mid a, b \in \mathbb{A},\ a \neq b\}$.

$$\text{single-use} \overset{\substack{\text{first letter appears again}\\ \text{[6, Exercise 91]}}}{\subsetneq} \text{one-way multiple-use} \overset{\substack{\text{some letter appears twice}\\ \text{[13, Example 11]}}}{\subsetneq} \text{two-way multiple-use.}$$

Two-way multiple-use automata have an undecidable emptiness problem [15, Theorem 5.3]. For one-way (even multiple-use) automata, emptiness is decidable and even tractable in a suitable parametrised understanding [6, Corollary 9.12]. We leave open the following question: given a one-way multiple-use automaton, can one decide if there is an equivalent automaton that is single-use (by Theorem 6, it does not matter whether one-way or two-way)?

3.1 From two-way automata to orbit-finite monoids

In this section, we show the implication 2 ⇒ 3 of Theorem 6. (This is the only proof presented in the conference version of the paper – we chose it, because it illustrates the importance of the single-use restriction). The implication states that the language of every single-use two-way automaton can also be recognised by an equivariant homomorphism into an orbit-finite monoid. In the proof, we use the Shepherdson construction for two-way automata [16] and show that, thanks to the single-use restriction, it produces monoids which are orbit-finite.

Consider a two-way single-use automaton, with k registers and let Q be the set of its states. For a string over the input alphabet (extended with endmarkers), define its *Shepherdson profile* to be the function of the type

$$\overbrace{Q \times (\mathbb{A}+\bot)^k}^{\substack{\text{state and register}\\ \text{valuation at the}\\ \text{start of the run}}} \times \overbrace{\{\leftarrow,\rightarrow\}}^{\substack{\text{does the run}\\ \text{enter from the}\\ \text{left or right}}} \quad \to \quad \{\text{accept}, \text{loop}\} + (\overbrace{Q \times (\mathbb{A}+\bot)^k}^{\substack{\text{state and register}\\ \text{valuation at the}\\ \text{end of the run}}} \times \overbrace{\{\leftarrow,\rightarrow\}}^{\substack{\text{does the run}\\ \text{exit from the}\\ \text{left or right}}} \;)$$

that describes runs of the automaton in the natural way (see [16, Proof of Theorem 2]). The run is taken until the automaton either exits the string from either side, accepts, or enters an infinite loop. By the same reasoning as in Shepherdson's proof, one can equip the set of Shepherdson profiles with a monoid structure so that the function which maps a word to its Shepherdson profile becomes a monoid homomorphism. We use the name Shepherdson monoid for the resulting monoid (it only contains the "achievable" profiles – the image of Σ^*). It is easy to see that whether a word is accepted depends only on an equivariant property of its Shepherdson profile, and therefore the language recognised by the automaton is also recognised by the Shepherdson monoid.

It remains to show that the Shepherdson monoid is orbit-finite, which is the main part of the proof. Unlike the arguments so far, this part of the proof relies on the single-use restriction. To illustrate this, we give an example of a one-way automaton that is not single-use and whose Shepherdson monoid is not orbit-finite.

▶ **Example 7.** Consider the language over $\mathbb{A}$ of words whose first letter appears again. This language is not recognised by any orbit-finite monoid [6, Exercise 91], but it is recognised by a multiple-use one-way automaton, which stores the first letter in a register, and then compares this register with all remaining letters of the input word. For this automaton, the Shepherdson profile needs to remember all of the distinct letters that appear in the word. In particular, if two words have different numbers of distinct letters, then their Shepherdson profiles cannot be in the same orbit. Since input strings can contain arbitrarily many distinct letters, the Shepherdson monoid of this automaton is not orbit-finite.

▶ **Lemma 8.** *For every single-use two-way automaton there is some $N \in \mathbb{N}$ such that every Shepherdson profile is supported by at most N atoms.*

Before proving the lemma, we use it to show that the Shepherdson monoid is orbit-finite. In the full version of the paper, we show that if an equivariant set consists of functions from one orbit-finite set to another orbit-finite set (as is the case for the underling set in the Shepherdson monoid) and all functions in the set have supports of bounded size (as is the case thanks to Lemma 8), then the set is orbit-finite. This leaves us with proving Lemma 8.

Proof. Define a *transition* in a run to be a pair of consecutive configurations. Each transition has a corresponding question and action. A transition in a run is called *important* if its question or action involves a register that has not appeared in any action or question of the run. The number of important transitions is bounded by k – the number of registers. The crucial observation, which relies on the single-use restriction, is that if the input word, head position, and state are fixed (but not the register valuation), then the sequence of actions in the corresponding run depends only on the answers to the questions in the important transitions. This is described in more detail below.

Fix a choice of the following parameters: (a) a string over the input alphabet that might contain endmarkers; (b) an entry point of the automaton – either the left or the right end of the word; (c) a state of the automaton. We do not fix the register valuation. For a register valuation η, define $\rho(\eta)$ to be the run which begins in the configuration described by the parameters (abc) together with η, and which is maximal, i.e. it ends when the automaton either accepts, rejects, or tries to leave the fixed string. For $i \in \{0, 1, \dots, k\}$ define $\alpha_i(\eta)$ to be the sequence of actions that are performed in the maximal prefix of the run $\rho(\eta)$ which uses at most i important transitions. The crucial observation that was stated at the beginning of this proof is that once the parameters (abc) are fixed, then the sequence of actions $\alpha_i(\eta)$ depends only on the answers to the questions asked in the first i important transitions. In particular, the function α_i has at most 2^i possible values. Furthermore, by a simple induction on i, one can show the following claim.

▷ Claim 9. The function α_i is supported by at most 2^{i+1} atoms.

Since there are at most k important transitions in a run, the above claim implies that, for every fixed choice of parameters (abc), at most 2^{k+1} atoms are needed to support the function which maps η to the sequence of actions in the run $\rho(\eta)$. In the arguments for the Shepherdson profile for a fixed word w, parameter (b) can have two values (first or last position) and parameter (c) can have at most $|Q|$ values. Therefore, at most $2|Q|2^{k+1}$ atoms are needed to support the function which takes an argument as in the Shepherdson profile, and returns the sequence of actions in the corresponding run. The lemma follows. ◀

4 A Krohn-Rhodes decomposition of one-way transducers with atoms

In this section, we present a decomposition result for single-use one-way transducers, which is a version of the celebrated Krohn-Rhodes Theorem [14, p. 454]. We think that this result gives further evidence for the good structure of single-use models. In the next section, we give a similar decomposition result for two-way single-use transducers which will be used to prove the equivalence of several other characterisations of the two-way model.

We begin by describing the classical Krohn-Rhodes Theorem. A *Mealy machine* is a deterministic one-way length-preserving transducer, which is obtained from a deterministic finite automaton by labelling transitions with output letters and ignoring accepting states.

The Krohn-Rhodes Theorem says that every function computed by a Mealy machine is a composition of functions computed by certain prime Mealy machines (which are called *reversible* and *reset* in [1, Chapter 6]). In this section, we prove a version of this theorem for orbit-finite alphabets; this version relies crucially on the single-use restriction. To distinguish the original model of Mealy machines from the single-use model described below, we will use the name *classical Mealy machine* for the Mealy machines in the original Krohn-Rhodes Theorem, i.e. the alphabets and state spaces are finite.

Define a *single-use Mealy machine* to have the same syntax as in Definition 3, with the following differences: there are no "next/previous" actions from item 2c, but the output action from item 2b has the side effect of moving the head to the next position. A consequence is that a Mealy machine is length-preserving, i.e. it outputs exactly one letter for each input position. Furthermore, there are no endmarkers and no "accept" or "reject" actions from item 2d; the automaton begins in the first input position and accepts immediately once its head leaves the input word from the right.

▶ **Example 10.** Define *atom propagation* to be the following length-preserving function. The input alphabet is $\mathbb{A} + \{\epsilon, \downarrow\}$ and the output alphabet is $\mathbb{A} + \perp$. If a position i in the input string has label $\downarrow$ and there is some (necessarily unique) position $j < i$ with an atom label such that all positions strictly between j and i have label ϵ, then the output label of position i is the atom in input position j. For all other input positions, the output label is $\perp$. Here is an example of atom propagation:

input	1	2	ϵ	ϵ	$\downarrow$	$\downarrow$	3	ϵ	ϵ	$\downarrow$	ϵ	$\downarrow$
output	$\perp$	$\perp$	$\perp$	$\perp$	2	$\perp$	$\perp$	$\perp$	$\perp$	3	$\perp$	$\perp$

Atom propagation is computed by a single-use Mealy machine, which stores the most recently seen atom in a register, and outputs the register at the nearest appearance of $\downarrow$.

The following example illustrates some of the technical difficulties with single-use Mealy machines: It is often useful to consider a Mealy machine that computes the run of another Mealy machine i. e. decorate every input position with the state and the register valuation that the Mealy machine will have after reading the input up to (but not including) that position. As shown in the following example the single-use restriction makes this construction impossible.

▶ **Example 11.** Consider the single-use Mealy machine that implements the atom propagation function from Example 10. This machine has only one register. Every time it sees an atom value, it stores the value in the register and every time it sees $\downarrow$, and the register is non-empty, the machine outputs the register's content. We claim that the run of this machine cannot be computed by a Mealy machine. If it could, we would be able to use it to construct a Mealy machine that given a word over $\mathbb{A}$, equips every position with the atom from the first position. This would easily lead to a construction of a single-use automaton for the language "the first letter appears again" (from Example 7) which, as we already know, is impossible.

The Krohn-Rhodes Theorem, both in its original version and in our orbit-finite version below, says that every Mealy machine can be decomposed using two types of composition:

$$\frac{\Sigma^* \xrightarrow{f} \Gamma^* \quad \Gamma^* \xrightarrow{g} \Delta^*}{\Sigma^* \xrightarrow{g \circ f} \Delta^*} \text{ sequential} \qquad \frac{\Sigma_1^* \xrightarrow{f_1} \Gamma_1^* \quad \Sigma_2^* \xrightarrow{f_2} \Gamma_2^*}{(\Sigma_1 \times \Sigma_2)^* \xrightarrow{f_1 | f_2} (\Gamma_1 \times \Gamma_2)^*} \text{ parallel}$$

The sequential composition is simply function composition. The parallel composition – which only makes sense for length preserving functions – applies the function f_i to the i-th projection of the input string.

▶ **Theorem 12.** *Every total function computed by a single-use Mealy machine can be obtained, using sequential and parallel composition, from the following* prime functions*:*

1. Length-preserving homomorphisms. *Any function of type $\Sigma^* \to \Gamma^*$, where Σ and Γ are polynomial orbit-finite, obtained by lifting to strings an equivariant function of type $\Sigma \to \Gamma$.*
2. Classic Mealy machines. *Any function computed by a classical Mealy machine.*
3. Atom propagation. *The atom propagation function from Example 10.*

By the original Krohn-Rhodes theorem, classical Mealy machines can be further decomposed.

Define a *composition of primes* to be any function that can be obtained from the prime functions by using a parallel and sequential composition. In this terminology, Theorem 12 says that every function computed by a single-use Mealy machine is a composition of primes. In the full version of the paper we show that the converse is also true: every prime function is computed by a single-use Mealy machine, and single-use Mealy machines are closed under both kinds of composition.

Decomposition of single-use one-way transducers. A Mealy machine is the special case of a single-use one-way transducer which is length preserving, and does not see an endmarker. A corollary of Theorem 12 is that, in order to generate all total functions computed by single-use one-way transducers, it is enough to add two items to the list of prime functions from Theorem 12: (a) a function $w \mapsto w\dashv$ which appends an endmarker[5], and (b) equivariant homomorphisms over polynomial orbit-finite alphabets that are not necessarily length-preserving.

5 Two-way single-use transducers

In this section, we turn to two-way single-use transducers. For them, we show three other equivalent models: (a) compositions of certain two-way prime functions; (b) an atom variant of the streaming string transducer (SST) model of Alur and Černý from [2]; and (c) an atom variant of the regular string functions from [7]. We believe that the atom variants of items (b) and (c), as described in this section, are the natural atom extensions of the original models; and the fact that these extensions are all equivalent to single-use two-way transducers is a further validation of the single-use restriction.

We illustrate the transducer models using the functions from the following example.

▶ **Example 13.** Consider some polynomial orbit-finite alphabet Σ. The input and output alphabets are the same, namely Σ extended with a separator symbol $|$. Define *map reverse* (respectively, *map duplicate*) to be the function which reverses (respectively, duplicates) every string between consecutive separators, as in the following examples:

$$\underbrace{12||345|678|9 \;\mapsto\; 21||543|876|9}_{\text{map reverse}} \qquad \underbrace{12||345|678|9 \;\mapsto\; 1212||345345|678678|99}_{\text{map duplicate}}$$

Both functions can be computed by single-use two-way transducers. These functions will be included in the prime functions for two-way single-use transducers, as discussed in item (a) at the beginning of this section.

[5] This function accounts for the fact that a one-way transducer (contrary to a Mealy machine) may perform some computation and produce some output at the end of the input word.

Streaming string transducers with atoms. A streaming string transducer with atoms has two types of registers: atom registers $r, s, \ldots$ which are the same as in Definition 3, and string registers $A, B, C, \ldots$ which are used to store strings over the output alphabet. Both kinds of registers are subject to the single-use restriction, which is highlighted in bold in the following definition.

▶ **Definition 14** (Streaming string transducer with atoms). *Define the syntax of a* streaming string transducer (SST) with atoms *in the same way as a one-way single-use transducer (variant of Definition 3), except that the model is additionally equipped with a finite set of* string registers*, with a designated* output string register*. The actions are the same as for one-way single-use transducers except that the output action is replaced by two kinds of actions:*

1. *Apply an equivariant function $f : \mathbb{A}^k \to \Gamma$ to the contents of distinct registers $r_1, \ldots, r_k \in R$, and put the result into string register A (overwriting its previous contents). If any of these registers is undefined, then the run immediately stops and rejects.* ***This action has the side effect of setting the values of $r_1, r_2, \ldots, r_{k_j}$ to $\perp$.***
2. *Concatenate string registers A and B, and put the result into string register C.* ***This action has the side effect of setting A and B to the empty string.***

The output of a streaming string transducer is defined to be the contents of the designated output register when the "accept" action is performed. In the atomless case, when no atom registers are allowed and the input and output alphabets are finite, the above definition is equivalent to the original definition of streaming string transducers from [2].

▶ **Example 15.** Consider the map reverse function from Example 13, with alphabet $\mathbb{A}$. To compute it, we use two string registers A and B, with the output register being B. When reading an atom $a \in \mathbb{A}$, the transducer executes an action $A := aA$. (This action needs to be broken into simpler actions as in Definition 14 and requires auxiliary registers). When reading a separator symbol, the automaton executes action $B := B|A$, which erases the content of register A. Similar idea works for map duplicate – it uses two copies of register A.

Regular list functions with atoms. Our last model is based on the regular list functions from [7]. Originally, the regular list functions were introduced to characterise two-way transducers (over finite alphabets), in terms of simple prime functions and combinators [7, Theorem 6.1]. The following definition extends the original definition[6] in only two ways: we add an extra datatype $\mathbb{A}$ and an equality test `eq` : $\mathbb{A}^2 \to$ {yes,no}.

▶ **Definition 16** (Regular list functions with atoms). *Define the* datatypes *to be sets which can be obtained from $\mathbb{A}$ and singleton sets, by applying constructors for products $\tau \times \sigma$, co-products $\tau + \sigma$ and lists τ^*. The class of* regular list functions with atoms *is the least class which:*

1. *contains all equivariant constant functions;*
2. *contains all functions from Figure 2, and an equality test* `eq` $: \mathbb{A}^2 \to$ *{yes,no};*
3. *is closed under applying the following combinators:*
   a. *`comp` function composition $(f, g) \mapsto f \circ g$;*
   b. *`pair` function pairing $(f_0, f_1) \mapsto (x \mapsto (f_0(x), f_1(x)))$;*
   c. *`cases` function co-pairing $(f_0, f_1) \mapsto ((i, a) \mapsto f_i(a))$;*
   d. *`map` lifting functions to lists $f \mapsto ([a_1, \ldots, a_n] \mapsto [f(a_1), \ldots, f(a_n)])$.*

[6] In [7], the group product operation has output type G^*, while this paper uses $(G \times \sigma)^*$. This difference is due to an error in [7].

`project`$_i$	:	$(\sigma_0 \times \sigma_1) \to \sigma_i$ projection $(a_0, a_1) \mapsto a_i$
`coproject`$_i$	:	$\sigma_i \to (\sigma_0 + \sigma_1)$ coprojection $a_i \mapsto (i, a_i)$
`distr`	:	$(\sigma_1 + \sigma_2) \times \tau \to (\sigma_1 \times \tau) + (\sigma_2 \times \tau)$ distribution $((i, a), b) \mapsto (i, (a, b))$
`reverse`	:	$\sigma^* \to \sigma^*$ list reverse $[a_1, \ldots, a_n] \mapsto [a_n, \ldots, a_1]$
`concat`	:	$(\sigma^*)^* \to \sigma^*$ list concatenation, defined by $[] \mapsto []$ and $[a] \cdot l \mapsto a \cdot concat(l)$
`append`	:	$(\sigma \times \sigma^*) \to \sigma^*$ append, defined by $(a, l) \mapsto [a] \cdot l$
`coappend`	:	$\sigma \to (\sigma \times \sigma^*) + \bot$ the opposite of append, defined by $[] \mapsto (1, \bot)$ and $[a] \cdot l \mapsto (0, (a, l))$
`block`	:	$(\sigma + \tau)^* \to (\sigma^* + \tau^*)^*$ group the list into maximal connected blocks from σ^* or τ^*
`group`	:	$(G \times \sigma)^* \to (G \times \sigma)^*$ $[(g_1, a_1), \ldots, (g_n, a_n)] \mapsto [(1, a_1), (g_1, a_2), (g_1 g_2, a_3), \ldots, (g_1 \cdots g_{n-1}, a_n)]$

Figure 2 For every datatypes $\tau, \tau_0, \tau_1, \sigma$, every finite group G, and every $i \in \{0, 1\}$ the above functions are regular list functions with atoms.

Every polynomial orbit-finite set is a datatype (actually, polynomial orbit-finite sets are exactly the datatypes that do not use lists), and therefore it makes sense to talk about regular list functions with atoms that describe string-to-string functions with input and output alphabets that are polynomial orbit-finite sets. Also, one can consider string-to-boolean functions – they describe languages, and are the model mentioned in item 5 of Theorem 6.

▶ **Example 17.** We show that map reverse from Example 13 is a regular list function with atoms. Consider an input string, say

$$[1, 2, |, |, 3, 4, 5, |, 6, 7, 8, |, 9] \in (\mathbb{A} + |)^*.$$

Apply the prime `block` function, yielding

$$[[1, 2], [|, |], [3, 4, 5], [|], [6, 7, 8], [|], [9]] \in (\mathbb{A}^* + |^*)^*.$$

Using the `cases` and `map` combinators, apply `reverse` to all list items, yielding

$$[[2, 1], [|, |], [5, 4, 3], [|], [8, 7, 6], [|], [9]] \in (\mathbb{A}^* + |^*)^*.$$

To get the final output, apply `concat`. A similar idea works for map duplicate, except we need to derive the string duplication function:

$$w \overset{\texttt{pair}(\ldots)}{\mapsto} (w, [w]) \overset{\texttt{append}}{\mapsto} [w, w] \overset{\texttt{concat}}{\mapsto} ww$$

Equivalence of the models. The main result of this section is that all models described above are equivalent, and furthermore admit a decomposition into prime functions in the spirit of the Krohn-Rhodes theorem. Since the functions discussed in this section are no longer length-preserving, the Krohn-Rhodes decomposition uses only sequential composition.

▶ **Theorem 18.** *The following conditions are equivalent for every total function $f : \Sigma^* \to \Gamma^*$, where Σ and Γ are polynomial orbit-finite sets:*

1. *f is computed by a two-way single-use transducer;*
2. *f is computed by a streaming string transducer with atoms;*
3. *f is a regular list function with atoms;*
4. *f is a sequential composition of functions of the following kinds:*
   - **a.** *single-use Mealy machines;*
   - **b.** *equivariant homomorphisms that are not necessarily length-preserving;*
   - **c.** *map reverse and map duplicate functions from Example 13.*

In the future, we plan to extend the above theorem with one more item, namely a variant of MSO transductions based on rigidly guarded MSO$^\sim$. The models in items 3 and 4 are closed under sequential composition, and therefore the same is true for the models in items 1 and 2; we do not know any direct proof of composition closure for items 1 and 2, which contrasts the classical case without atoms [8, Theorem 2]. The Krohn-Rhodes decomposition from item 4, in the case without atoms, was present implicitly in [7]; in this paper we make the decomposition explicit, extend it to atoms, and leverage it to get a relatively simple proof of Theorem 18. Even for the reader interested in transducers but not atoms, our Krohn-Rhodes-based proof of Theorem 18 might be of some independent interest.

Here are some immediate corollaries of Theorem 18:

1. Every function in item 4 is computed by a two-way single-use transducer which is reversible in the sense of [10, p. 2]; hence two-way single-use transducers can be translated into reversible ones.
2. Since the equivalence in Theorem 18 also works for functions with yes/no outputs, it follows that items 2 and 5 in Theorem 6 are equivalent.
3. If f is a transducer from the class described in Theorem 18, then the language class described in Theorem 6 is preserved under inverse images of f.

All conversions between the models in Theorem 18 are effective. Our last result concerns the equivalence problem for these models, which is checking if two transducers compute the same function. Using a reduction to the equivalence problem for *copyful* streaming string transducers without atoms [12, p. 81], we prove the following result:

▶ **Theorem 19.** *Equivalence is decidable for streaming string transducers with atoms (and therefore also for every other of the equivalent models from Theorem 18).*

References

1 Ginzburg Abraham. *Algebraic Theory of Automata.* Elsevier, 1968.

2 Rajeev Alur and Pavol Černý. Expressiveness of streaming string transducers. In *Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2010, Chennai, India*, volume 8 of *LIPIcs*, pages 1–12. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010.

3 Henrik Björklund and Thomas Schwentick. On notions of regularity for data languages. *Theoretical Computer Science*, 411(4):702–715, January 2010.

4 Mikołaj Bojańczyk. Automata for Data Words and Data Trees. In Christopher Lynch, editor, *Rewriting Techniques and Applications, RTA, Edinburgh, Scottland, UK*, volume 6 of *LIPIcs*, pages 1–4. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010.

5 Mikołaj Bojańczyk. Nominal Monoids. *Theory Comput. Syst.*, 53(2):194–222, 2013.

6 Mikołaj Bojańczyk. Slightly infinite sets, 2019. URL: `https://www.mimuw.edu.pl/~bojan/paper/atom-book` [cited version of September 11, 2019].

7 Mikołaj Bojańczyk, Laure Daviaud, and Shankara Narayanan Krishna. Regular and First-Order List Functions. In *Logic in Computer Science, LICS, Oxford, UK*, pages 125–134. ACM, 2018.

8 Michal Chytil and Vojtech Jákl. Serial Composition of 2-Way Finite-State Transducers and Simple Programs on Strings. In *International Colloquium on Automata, Languages and Programming, ICALP, Turku, Finland*, volume 52 of *Lecture Notes in Computer Science*, pages 135–147. Springer, 1977.

9 Thomas Colcombet, Clemens Ley, and Gabriele Puppis. Logics with rigidly guarded data tests. *Logical Methods in Computer Science*, 11(3), 2015.

10 Luc Dartois, Paulin Fournier, Ismaël Jecker, and Nathan Lhote. On reversible transducers. In *International Colloquium on Automata, Languages and Programming, ICALP, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 113:1–113:12. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2017.

11 Emmanuel Filiot and Pierre-Alain Reynier. Transducers, logic and algebra for functions of finite words. *SIGLOG News*, 3(3):4–19, 2016.

12 Emmanuel Filiot and Pierre-Alain Reynier. Copyful Streaming String Transducers. In Matthew Hague and Igor Potapov, editors, *Reachability Problems, RP , London, UK*, volume 10506 of *Lecture Notes in Computer Science*, pages 75–86. Springer, 2017.

13 Michael Kaminski and Nissim Francez. Finite-Memory Automata. *Theor. Comput. Sci.*, 134(2):329–363, 1994.

14 Kenneth Krohn and John Rhodes. Algebraic theory of machines. i. prime decomposition theorem for finite semigroups and machines. *Transactions of the American Mathematical Society*, 116:450–450, 1965.

15 Frank Neven, Thomas Schwentick, and Victor Vianu. Finite state machines for strings over infinite alphabets. *ACM Trans. Comput. Log.*, 5(3):403–435, 2004.

16 J. C. Shepherdson. The reduction of two-way automata to one-way automata. *IBM Journal of Research and Development*, 3(2):198–200, April 1959.

17 Imre Simon. Factorization forests of finite height. *Theoretical Computer Science*, 72(1):65–94, 1990.

Weakly-Unambiguous Parikh Automata and Their Link to Holonomic Series

Alin Bostan
INRIA Saclay Île-de-France, Palaiseau, France
Alin.Bostan@inria.fr

Arnaud Carayol
LIGM, Univ. Gustave Eiffel, CNRS, Marne-la-Vallée, France
arnaud.carayol@u-pem.fr

Florent Koechlin
LIGM, Univ. Gustave Eiffel, CNRS, Marne-la-Vallée, France
florent.koechlin@u-pem.fr

Cyril Nicaud
LIGM, Univ. Gustave Eiffel, CNRS, Marne-la-Vallée, France
cyril.nicaud@u-pem.fr

Abstract

We investigate the connection between properties of formal languages and properties of their generating series, with a focus on the class of *holonomic* power series. We first prove a strong version of a conjecture by Castiglione and Massazza: weakly-unambiguous Parikh automata are equivalent to unambiguous two-way reversal bounded counter machines, and their multivariate generating series are holonomic. We then show that the converse is not true: we construct a language whose generating series is algebraic (thus holonomic), but which is inherently weakly-ambiguous as a Parikh automata language. Finally, we prove an effective decidability result for the inclusion problem for weakly-unambiguous Parikh automata, and provide an upper-bound on its complexity.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory; Mathematics of computing → Generating functions

Keywords and phrases generating series, holonomicity, ambiguity, reversal bounded counter machine, Parikh automata

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.114

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

1 Introduction

This article investigates the link between *holonomic* (or *D-finite*) power series and formal languages. We consider the classical setting in which this connection is established via the generating series $L(x) = \sum_{n\geq 0} \ell_n x^n$ counting the number ℓ_n of words of length n in a given language $\mathcal{L}$.

On the languages side, the Chomsky–Schützenberger hierarchy [10] regroups languages in classes of increasing complexity: regular, context-free, context-sensitive and recursively enumerable. For power series, a similar hierarchy exists, consisting of the rational, algebraic and holonomic series. The first two levels of each hierarchy share a strong connection, as the generating series of a regular (resp. unambiguous context-free) language is a rational (resp. algebraic) power series.

This connection has borne fruits both in formal language theory and in combinatorics. In combinatorics, finite automata and unambiguous grammars are routinely used to establish rationality and algebraicity of particular power series. In formal languages, this connection was (implicitly) used to give polynomial-time algorithms for the inclusion and universality

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 114; pp. 114:1–114:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

tests for unambiguous finite automata [35]. In [15], Flajolet uses the connection between unambiguous context-free grammars and algebraic series to prove the inherent ambiguity of certain context-free languages, solving several conjectures with this tool. Using analytic criteria on the series (for instance, the existence of infinitely many singularities), he establishes that the series of these context-free languages is not algebraic. Hence these languages cannot be described by unambiguous context-free grammars and are therefore inherently ambiguous.

In this article we propose to extend the connection to holonomic series. Holonomic series enjoy non-trivial closure properties whose algorithmic counterparts are actively studied in computer algebra. Our aim is to show that these advances can be leveraged to obtain non-trivial results in the formal languages and verification worlds. These results are particularly noteworthy as the notion of unambiguity in automata theory is not fully understood [11]. The work of extending the connection was already initiated by Massazza in [27], where he introduces two families of languages, named RCM[1] and linearly constrained languages (LCL), whose generating series are holonomic. These classes are, however, not captured by well-known models of automata, and this limits their appeal. Recently, Castiglione and Massazza addressed this issue and conjectured that RCM contains the languages accepted by deterministic one-way reversal bounded machines (RBCM for short) [8]; Massazza proved the result for RBCM for two subclasses of one-way deterministic RBCM [28, 29]. This conjecture hints that the class RCM is related to models of automata such as RBCM, which are used in program verification.

Our first contribution is to prove a stronger version of this conjecture. We show that RCM and LCL respectively correspond to the languages accepted by weakly-unambiguous[2] version of Parikh automata (PA, for short) [24] and pushdown Parikh automata. Intuitively, Parikh automata are[3] finite non-deterministic word automata enriched with the ability to test semilinear constraints between the number of occurrences of each transition in the run. In terms of RBCM, these classes correspond to unambiguous two-way RBCM and unambiguous one-way RBCM enriched with a stack. Parikh automata are also commonly used in program verification. In view of the literature, these results might seem expected but they still require a careful adaptation of the standard techniques in the absence of a stack and become even more involved when a stack is added.

After having established the relevance of the classes of languages under study, we provide two consequences of the holonomicity of their associated generating series.

The first consequence follows Flajolet's approach mentioned previously and gives criteria to establish the inherent weak-ambiguity for languages accepted by PA or pushdown PA, by proving that their generating series are not holonomic. These criteria are sufficient but not necessary; this is not surprising as the inherent weak-ambiguity is undecidable for languages accepted by PA. Yet, the resulting method captures non-trivial examples with quite short and elegant proofs. In contrast, we give an example of inherently weakly-ambiguous PA language having a holonomic series (and therefore not amenable to the analytic method) for which we prove inherent weak-ambiguity *by hand*. The proof is quite involved but shows the inherent ambiguity of this language for a much larger class of automata (i.e., PA whose semi-linear sets are replaced by arbitrary recursive sets).

[1] The name RCM comes from the fact that these languages are defined using a Regular language, a semilinear Constraint and a Morphism.

[2] We use the term weakly-unambiguous here to avoid a possible confusion with the class of unambiguous PA defined in [7] which is strictly contained in our class. Our notion of non-ambiguity is the standard one: every word has at most one accepting computation (this is detailed in Remark 11).

[3] In this article, we use an equivalent definition where the transitions of the automaton are additionally labeled by vectors of natural numbers. A run is accepting if the sum of all the vectors encountered in the run satisfies a semilinear constraint.

The second consequence is of an algorithmic nature. We focus on the inclusion problem for weakly-unambiguous PA, whose decidability can be deduced from Castiglione and Massazza's work [8]. Here our contribution is an effective decidability result: we derive a concrete bound B, depending on the size of the representation of the two PAs, such that the inclusion holds if and only if the languages are included when considering words up to the length B. This bound B is obtained by a careful analysis of the proofs establishing the closure properties of holonomic series (in several variables), notably under Hadamard product and specialization. We do this by controlling various parameters (order, size of the polynomial coefficients, …) of the resulting partial differential equations.

2 Primer on holonomic power series in several variables

In this section, we introduce power series in several variables and the classes of rational, algebraic and holonomic power series. We recall the connection with regular and context-free languages via the notions of generating series in one or several variables.

Let $\mathbb{Q}[x_1, \dots, x_k]$ be the ring of polynomials in the variables $x_1, \dots, x_k$ with coefficients in $\mathbb{Q}$ and let $\mathbb{Q}(x_1, \dots, x_k)$ be the associated field of rational functions.

The *generating series of a sequence* $(f_n)_{n\in\mathbb{N}}$ is the (formal) power series in the variable x defined by $F(x) = \sum_{n\in\mathbb{N}} f_n x^n$. More generally, the generating series of a sequence $a(n_1, \dots, n_k)$ is a multivariate (formal) power series in the variables $x_1, \dots, x_k$ defined by $A(x_1, \dots, x_k) = \sum_{(n_1,\dots,n_k)\in\mathbb{N}^k} a(n_1, \dots, n_k) x_1^{n_1} \dots x_k^{n_k}$. In this article, we only consider power series whose coefficients belong to the field $\mathbb{Q}$. The set of such k-variate power series is denoted $\mathbb{Q}[[x_1, \dots, x_k]]$. Power series are naturally equipped with a sum and a product which generalize those of polynomials, for which $\mathbb{Q}[[x_1, \dots, x_k]]$ is a ring. We use the bracket notation for the coefficient extraction: $[x_1^{n_1} \dots x_k^{n_k}]A(x_1, \dots, x_k) = a(n_1, \dots, n_k)$. The *support* of $A \in \mathbb{Q}[[x_1, \dots, x_k]]$ is the set of $(n_1, \dots, n_k)$ such that $[x_1^{n_1} \dots x_k^{n_k}]A \neq 0$. The inverse $1/A$ of a series $A \in \mathbb{Q}[[x_1, \dots, x_k]]$ is well-defined when its constant term $[x_1^0 \dots x_k^0]A$ is not zero. For instance, the inverse of $A(x_1, x_2) = 1 - x_1 x_2^2$ is $\frac{1}{1-x_1x_2^2} = \sum_{n\geq 0} x_1^n x_2^{2n}$.

The *generating series of a language* $\mathcal{L}$ over the alphabet $\Sigma = \{a_1, \dots, a_k\}$ is the univariate power series $L(x) = \sum_{w\in\mathcal{L}} x^{|w|} = \sum_{n\in\mathbb{N}} \ell_n x^n$, where ℓ_n counts the number of words of length n in $\mathcal{L}$. Similarly the *multivariate generating series* of $\mathcal{L}$ defined by $L(x_{a_1}, \dots, x_{a_k}) = \sum_{(n_1,\dots,n_k)\in\mathbb{N}^k} \ell(n_1, \dots, n_k) x_{a_1}^{n_1} \dots x_{a_k}^{n_k}$ where $\ell(n_1, \dots, n_k)$ denotes the number of words w in $\mathcal{L}$ such that $|w|_{a_1} = n_1$, $|w|_{a_2} = n_2$, …, and $|w|_{a_k} = n_k$, and $|w|_a$ denotes the number of occurrences of $a \in \Sigma$ in w. This way, we create one dimension per letter, so that each letter $a \in \Sigma$ has a corresponding variable x_a.

Observe that the univariate generating series of a language is exactly $L(x, \dots, x)$, obtained by setting each variable to x in its multivariate generating series.

▶ **Example 1.** The generating series of the language $\mathcal{P}$ of well-nested parentheses defined by the grammar $S \to aSbS + \varepsilon$ is $P(x_a, x_b) = \frac{1-\sqrt{1-4x_ax_b}}{2x_ax_b}$ and its counting series is[4] $P(x) = \frac{1-\sqrt{1-4x^2}}{2x^2}$. Indeed the production of the grammar translates to the equation $P(x_a, x_b) = x_a x_b P(x_a, x_b)^2 + 1$. This equation admits only one power series solution, namely $\frac{1-\sqrt{1-4x_ax_b}}{2x_ax_b}$.

[4] As for the inverse, the square root of a power series with nonzero constant term can be defined using the usual Taylor formula $\sqrt{1-x} = \sum_{n\geq 0} \frac{1}{(1-2\,n)\,4^n} \binom{2\,n}{n} x^n$.

A power series $A(x_1,\ldots,x_k) = \sum_{n_1,\ldots,n_k} a(n_1,\ldots,n_k)x_1^{n_1}\ldots x_k^{n_k}$ is *rational* if it satisfies an equation of the form: $P(x_1,\ldots,x_k)A(x_1,\ldots,x_k) = Q(x_1,\ldots,x_k)$, with $P,Q \in \mathbb{Q}[x_1,\ldots,x_k]$ and $P \neq 0$. The generating series (both univariate and multivariate) of regular languages (i.e., languages accepted by a finite state automaton) are rational power series [4]. It is well-known that the generating series can be effectively computed from a deterministic automaton accepting the language (see for instance [17, §I.4.2] for a detailed proof). For example, the multivariate generating series of the regular language $(abc)^*$ is $\frac{1}{1-x_ax_bx_c} = \sum_{n\geq 0} x_a^n x_b^n x_c^n$. Its univariate generating series is $\frac{1}{1-x^3}$.

The connection between rational languages and rational power series is not tight. For instance, the generating series of the non-regular context-free language $\{a^nb^n : n \geq 0\}$ is $\frac{1}{1-x_ax_b}$, which is rational. In fact, it has the same generating series as the regular language $(ab)^*$. Also there exist rational power series with coefficients in $\mathbb{N}$ which are not the generating series of any rational language. This is the case for $\frac{x+5x^2}{1+x-5x^2-125x^3}$ as shown in [4].

A power series $A(x_1,\ldots,x_k)$ is *algebraic* if there exists a non-zero polynomial $P \in \mathbb{Q}[x_1,\ldots,x_k,Y]$ such that $P(x_1,\ldots,x_k,A(x_1,\ldots,x_k)) = 0$. All rational series are algebraic.

▶ **Example 2.** The series $\frac{1}{1-x_1x_2} = \sum_{n\geq 0} x_1^n x_2^n$ is rational, as it satisfies the equation $(1-x_1x_2)A(x_1,x_2) = 1$. The series $A(x_1,x_2) = \sqrt{1-x_1x_2}$ is algebraic but not rational, since $A(x_1,x_2)^2 + (x_1x_2 - 1) = 0$ and there is no similar algebraic equation of degree 1.

The reader is referred to [34, 17] for a detailed account on rational and algebraic series.

In the same manner that rational series satisfy linear equations and algebraic series satisfy polynomial equations, holonomic series satisfy linear differential equations with polynomial coefficients. To give a precise definition, we need to introduce the formal partial differentiation of power series. The differential operator ∂_{x_i} with respect to the variable x_i is defined by

$$\partial_{x_i}A(x_1,\ldots,x_k) = \sum_{n_1,\ldots,n_k} n_i\, a(n_1,\ldots,n_k)x_1^{n_1}\ldots x_{i-1}^{n_{i-1}}x_i^{n_i-1}x_{i+1}^{n_{i+1}}\ldots x_k^{n_k}.$$

The composed operator $\partial_{x_i}^j$ is inductively defined for $j \geq 1$ by $\partial_{x_i}^1 = \partial_{x_i}$ and $\partial_{x_i}^{j+1} = \partial_{x_i} \circ \partial_{x_i}^j$.

▶ **Definition 3** (see [33, 26])**.** *A power series $A(x_1,\ldots,x_k)$ is* holonomic *or* D-finite[5] *if the $\mathbb{Q}(x_1,\ldots,x_k)$-vector space spanned by the family $\{\partial_{x_1}^{i_1}\ldots\partial_{x_k}^{i_k}A(x_1,\ldots,x_k) : (i_1,\ldots,i_k) \in \mathbb{N}^k\}$ has a finite dimension. Equivalently, for every variable $z \in \{x_1,\ldots,x_k\}$, $A(x_1,\ldots,x_k)$ satisfies a linear differential equation of the form $P_r(x_1,\ldots,x_k)\partial_z^r A(x_1,\ldots,x_k) + \ldots + P_0(x_1,\ldots,x_k)A(x_1,\ldots,x_k) = 0$, where the P_i's are polynomials of $\mathbb{Q}[x_1,\ldots,x_k]$ with $P_r \neq 0$.*

In the sequel, except for Section 5, we rely on the closure properties of the holonomic series and will not need to go back to Definition 3.

▶ **Example 4.** A simple example of holonomic series is $A(x) = e^{x^2} = \sum_{n\geq 0} \frac{x^{2n}}{n!}$. It is holonomic (in one variable) since it satisfies $\partial_x A(x) - 2xA(x) = 0$.

For a more involved example, consider the language $\mathcal{L}_3 = \{w \in \{a,b,c\}^* : |w|_a = |w|_b = |w|_c\}$, containing the words having the same number of occurrences of a's, b's and c's. This language is classically not context-free. Moreover there are $\binom{3n}{n,n,n}$ words of length $3n$ in $\mathcal{L}_3$

[5] A priori, these notions differ: a function $A(x_1,\ldots,x_k)$ is called *D-finite* if all its partial derivatives $\partial_{x_1}^{n_1}\cdots\partial_{x_k}^{n_k}\cdot A$ generate a finite dimensional space over $\mathbb{Q}(x_1,\ldots,x_k)$, and *holonomic* if the functions $x_1^{\alpha_1}\cdots x_k^{\alpha_k}\partial_{x_1}^{\beta_1}\cdots\partial_{x_k}^{\beta_k}\cdot A$ subject to the constraint $\alpha_1+\cdots+\alpha_k+\beta_1+\cdots+\beta_k \leq N$ span a vector space whose dimension over $\mathbb{Q}$ grows like $O(N^k)$. The equivalence of these notions is proved by deep results of Bernšteĭn [2] and Kashiwara [21, 36].

and the power series $\sum_{n\geq 0}\binom{3n}{n,n,n}x^n$ is transcendental [15, §7]. Its multivariate generating series $L_3(x_a,x_b,x_c)$ is equal to $\sum_{n\geq 0}\frac{(3\,n)!\,(x_a x_b x_c)^n}{(n!)^3}$ and satisfies the partial differential equation:

$$(27x_a^2 x_b x_c - x_a)\partial^2_{x_a} f(x_a,x_b,x_c) + (54x_a x_b x_c - 1)\partial_{x_a} f(x_a,x_b,x_c) + 6x_b x_c f(x_a,x_b,x_c) = 0,$$

and the symmetric ones for the other variables x_b and x_c.

Holonomic series are an extension of the hierarchy we presented, as stated in the following proposition (see [12] for a proof, and [3] for bounds, algorithms and historical remarks).

▶ **Proposition 5.** *Multivariate algebraic power series are holonomic.*

In the univariate case[6], a power series $A(x)=\sum_n a_n x^n$ is holonomic if and only if its coefficients satisfy a linear recurrence of the form $p_r(n)a_{n+r}+\ldots+p_0(n)a_n=0$, where every p_i is a polynomial with rational coefficients [33, Th. 1.2].

We now focus on these closure properties.

▶ **Proposition 6** ([33])**.** *Multivariate holonomic series are closed under sum and product.*

Holonomic series are also closed under substitution by algebraic series as long as the resulting series is well-defined[7].

▶ **Proposition 7** ([26, Prop. 2.3])**.** *Let $A(x_1,\ldots,x_k)$ be a power series and let $G_i(y_1,\ldots,y_\ell)$ be algebraic power series such that $B(y_1,\ldots,y_\ell)=A(G_1(y_1,\ldots,y_\ell),\ldots,G_k(y_1,\ldots,y_\ell))$ is well-defined as a power series. If A is holonomic, then B is also holonomic.*

A sufficient condition for the substitution to be valid is that $G_i(0,\ldots,0)=0$ for all i (see [33, Th. 2.7]). For the case $G_1=\cdots=G_k=1$, called the *specialization to 1*, a sufficient condition is that for every index $(i_1,\ldots,i_k)$, $[x_1^{i_1}\ldots x_k^{i_k}]A$ is a polynomial in $y_1,\ldots,y_\ell$.

The Hadamard product is the coefficient-wise multiplication of power series. If the series $A(x_1,\ldots,x_k)$ and $B(x_1,\ldots,x_k)$ are the generating series of the sequences $a(n_1,\ldots,n_k)$ and $b(n_1,\ldots,n_k)$, the *Hadamard product* $A\odot B$ of A and B is the power series defined by

$$A\odot B(x_1,\ldots,x_k)=\sum_{(n_1,\ldots,n_k)\in\mathbb{N}^k} a(n_1,\ldots,n_k)b(n_1,\ldots,n_k)x_1^{n_1}\ldots x_k^{n_k}.$$

Observe that the support of $F\odot G$ is the intersection of the supports of F and G.

▶ **Theorem 8** ([25])**.** *Multivariate holonomic series are closed under Hadamard product.*

▶ **Example 9.** The generating series of the language $\mathcal{L}_3$ of Example 4, which is not context-free, can be expressed using the Hadamard product: since $\frac{1}{1-x_a x_b x_c}$ is the support series of the subset $\{(n,n,n) : n\in\mathbb{N}\}$, and since $\frac{1}{1-(x_a+x_b+x_c)}$ is the multivariate series of all the words on $\{a,b,c\}$, we have $L_3(x_a,x_b,x_c)=\frac{1}{1-(x_a+x_b+x_c)}\odot\frac{1}{1-x_a x_b x_c}$, which is not algebraic.

One of our main technical contributions is to provide bounds on the sizes of the polynomials in the differential equations of the holonomic representation of the Hadamard product of two rational series $\frac{P_1}{Q_1}$ and $\frac{P_2}{Q_2}$: we prove that their maxdegree is at most $(kM)^{\mathcal{O}(k)}$ and that the logarithm of their largest coefficient is at most $(kM)^{\mathcal{O}(k^2)}(1+\log S_\infty)$, where M (resp. S_∞) is the maxdegree plus one (resp. largest coefficient) in P_1, Q_1, P_2 and Q_2.

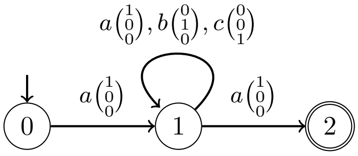
[6] The generalization of this equivalence to the multivariate case is not straightforward (see [26] for more details) and will not be used in this article.

[7] Note that the substitution of a power series into another power series might not yield a power series: for instance substituting x by $1+y$ in $\sum_{n\geq 0}x^n$ does not result in a power series as the constant term would be infinite.

3 Weakly-unambiguous Parikh automata

In this section, we introduce weakly-unambiguous Parikh automata and show that their multivariate generating series are holonomic. We establish that they accept the same languages as unambiguous two-way reversal bounded counter machines [19]. Finally, we prove that the class of accepted languages coincides with Massazza's RCM class [27, 8].

Parikh automata (PA for short) were introduced in [23, 24]. Informally, a PA is a finite automaton whose transitions are labeled by pairs $(a, \boldsymbol{v})$ where a is a letter of the input alphabet and $\boldsymbol{v}$ is a vector in $\mathbb{N}^d$. A run $q_0 \xrightarrow{a_1,\boldsymbol{v_1}} q_1 \xrightarrow{a_2,\boldsymbol{v_2}} q_2 \cdots q_{n-1} \xrightarrow{a_n,\boldsymbol{v_n}} q_n$ computes the word $a_1 \cdots a_n$ and the vector $\boldsymbol{v_1} + \cdots + \boldsymbol{v_n}$ where the sum is done component-wise. The acceptance condition is given by a set of final states and a semilinear set of vectors. A run is accepting if it reaches a final state and if its vector belongs to the semilinear set.


$a\binom{1}{0}{}_{0}, b\binom{0}{1}{}_{0}, c\binom{0}{0}{}_{1}$
$a\binom{1}{0}{}_{0}$
$a\binom{1}{0}{}_{0}$
0
1
2

The PA depicted above, equipped with the semilinear constraint $\{(n_1, n_2, n_1 + n_2) : n_1, n_2 \geq 0\}$, accepts the set of words w over $\{a, b, c\}$ that start and end with a and that are such that $|w|_a + |w|_b = |w|_c$.

3.1 Semilinear sets and their characteristic series

A set $L \subseteq \mathbb{N}^d$ is *linear* if it is of the form $\boldsymbol{c} + P^* := \{\boldsymbol{c} + \lambda_1 \boldsymbol{p_1} + \cdots + \lambda_k \boldsymbol{p_k} : \lambda_1, \dots, \lambda_k \in \mathbb{N}\}$, where $\boldsymbol{c} \in \mathbb{N}^d$ is the *constant* of the set and $P = \{\boldsymbol{p_1}, \dots, \boldsymbol{p_k}\} \subset \mathbb{N}$ its set of *periods.* A set $C \subseteq \mathbb{N}^d$ is *semilinear* if it is a finite union of linear sets. For example, the semilinear set $\{(n_1, n_2, n_1 + n_2) : n_1, n_2 \geq 0\}$ is in fact a linear set $(0, 0, 0) + \{(1, 0, 1), (0, 1, 1)\}^*$.

In [13, 20], it is shown that every semilinear set admits an unambiguous presentation. A presentation $\boldsymbol{c} + P^*$ with $P = \{\boldsymbol{p_1}, \dots, \boldsymbol{p_k}\}$ of a linear set L is unambiguous if for all $\boldsymbol{x} \in L$, the λ_i's such that $\boldsymbol{x} = \boldsymbol{c} + \lambda_1 \boldsymbol{p_1} + \cdots + \lambda_k \boldsymbol{p_k}$ are unique. An *unambiguous presentation of a semilinear set* is given by a disjoint union of unambiguous linear sets. A bound on the size of the equivalent unambiguous presentation is given in [9].

Semilinear sets are ubiquitous in theoretical computer science and admit numerous characterizations. They are the rational subsets of the commutative monoid $(\mathbb{N}^d, +)$, the unambiguous rational subset of $(\mathbb{N}^d, +)$ [13, 20], the Parikh images of context-free languages [30], the sets definable in Presburger arithmetic [31], the sets defined by boolean combinations of linear inequalities, equalities and equalities modulo constants.

For example, the semilinear set $\{(2n, 3n, 5n) : n \geq 0\}$ is equal to $(0, 0, 0) + \{(2, 3, 5)\}^*$. It is also the Parikh image of the regular (hence context-free) language $(aabbbccccc)^*$. In $(\mathbb{N}, +)$, it is defined by the Presburger formula $\phi(x, y, z) = \exists w, x = w + w \wedge y = w + w + w \wedge z = w + w + w + w + w$. Finally it is characterized in $\mathbb{N}^3$ by the equalities $3x = 2y$ and $5x = 2z$.

For a semilinear set $C \subseteq \mathbb{N}^d$, we consider its *characteristic generating series* $C(x_1, \dots, x_d)$ defined by $\sum_{(i_1,\dots,i_d) \in C} x_1^{i_1} \cdots x_d^{i_d}$. It is well-known [13, 20] that this power series is rational[8].

[8] The characteristic series of an unambiguous linear set $\boldsymbol{c} + P^* \subseteq \mathbb{N}$ with $P = \{\boldsymbol{p_1}, \dots, \boldsymbol{p_k}\}$ is $x_1^{\boldsymbol{c}(1)} \dots x_d^{\boldsymbol{c(d)}} \prod_{i=1}^k \left(1 - x_1^{\boldsymbol{p}_i(1)} \cdots x_d^{\boldsymbol{p}_i(d)}\right)^{-1}$ and hence is rational. As an unambiguous semilinear set is the disjoint union of unambiguous linear sets, its characteristic series is the sum of their series and it is therefore rational.

3.2 Weakly-unambiguous PAs and their generating series

We now introduce PA and their weakly-unambiguous variant. We discuss the relationship with the class of unambiguous PA introduced by Cadilhac et al. in [7] and the closure properties of this class.

A *Parikh automaton* of dimension $d \geq 1$ is a tuple $\mathcal{A} = (\Sigma, Q, q_I, F, C, \Delta)$ where Σ is the alphabet, Q is the set of states, $q_I \in Q$ is the initial state, $F \subseteq Q$ is the set of final states, $C \subseteq \mathbb{N}^d$ is the semilinear constraint and $\Delta \subseteq Q \times (\Sigma \times \mathbb{N}^d) \times Q$ is the transition relation.

A run of the automaton is a sequence $q_0 \xrightarrow{a_1,\boldsymbol{v_1}} q_1 \xrightarrow{a_2,\boldsymbol{v_2}} q_2 \cdots q_{n-1} \xrightarrow{a_n,\boldsymbol{v_n}} q_n$ where for all $i \in [1,n]$, $(q_{i-1}, (a_i, \boldsymbol{v_i}), q_i)$ is a transition in Δ. The run is labeled by the pair $(a_1 \cdots a_n, \boldsymbol{v_1} + \cdots + \boldsymbol{v_n}) \in \Sigma^* \times \mathbb{N}^d$. It is accepting if $q_0 = q_I$, the state q_n is final and if the vector $\boldsymbol{v_1} + \cdots + \boldsymbol{v_n}$ belongs to C. The word w is then said to be accepted by $\mathcal{A}$. The language accepted by $\mathcal{A}$ is denoted by $\mathcal{L}(\mathcal{A})$.

To define a notion of size for a PA, we assume that the constraint set is given by an unambiguous presentation $\uplus_{i=1}^{p} \boldsymbol{c_i} + P_i^*$. We denote by $|\mathcal{A}| := |Q| + |\Delta| + p + \sum_i |P_i|$ and by $\|\mathcal{A}\|_\infty$ the maximum coordinate of a vector appearing in Δ, the $\boldsymbol{c_i}$'s and the P_i's.

▶ **Definition 10.** *A Parikh automaton is said to be* weakly-unambiguous *if for every word there is at most one accepting run.*

A language is *inherently weakly-ambiguous* if it cannot be accepted by any weakly-unambiguous PA. The language $\mathcal{S}$ (defined in Section 4.1) is an example[9] of a language accepted by a non-deterministic PA which is inherently weakly-ambiguous.

▶ Remark 11. We consider here the standard notion of unambiguity for finite state machines. However we decided to use the name *weakly-unambiguous* to avoid the confusion with the class of unambiguous PA which appears in the literature. This class was introduced by Cadilhac et al. in [7] for constraint automata, a model equivalent to PA and was latter defined directly on PAs. This notion of unambiguity is more restrictive than ours: they call a Parikh automaton *unambiguous* if the underlying automaton on letters, where the vectors have been erased, is unambiguous. Clearly such automata are weakly-unambiguous. However the converse is not true. Consider the language $\mathcal{L} = \{c^n w \ : \ w = x_1 x_2 \cdots x_m \in \{a,b\}^* \ \wedge \ m \geq n > 0 \wedge \ |x_1 x_2 \cdots x_n|_a < |x_1 x_2 \cdots x_n|_b\}$ over the alphabet $\{a,b,c\}$. Using results from [7], one can show that it is not recognized by any unambiguous Parikh automata. However, it is accepted by the weakly-unambiguous automaton depicted in Fig. 1 below with the semilinear $\{(n_1, n_2, n_3) \ : \ n_1 = n_2 + n_3 \text{ and } n_2 < n_3\}$.

The lack of expressivity of unambiguous PAs is counter-balanced by their closure under boolean operations, which is explained by their link with a class of deterministic PA [7, 14]. It was pointed out to us by a reviewer that the class of weakly-unambiguous PA is briefly considered, under the name OneCA, in Cadilhac's PhD thesis [5, p. 117], where only basic properties are established, in particular the strict inclusion of unambiguous PA in this class.

Using a standard product construction when the vectors are concatenated and using the concatenation of the constraints, it is easy to show that weakly-unambiguous PA are closed under intersection. In [8], the authors claim that the class[10] is closed under union. However their construction has an irrecoverable flaw and we do not know if weakly-unambiguous PA are closed under union or under complementation.

[9] Using the equivalences between weakly-unambiguous PA and RCM established in Proposition 13 and PA and RBCM [23, 6], it also gives an example of a language accepted by a RBCM with a non-holonomic generating series (strengthening Theorem 12 of [8]) and a witness for the strict inclusion of RCM in RBCM announced in Theorem 11 of [8]. Remark that their proof of this theorem only shows that there exists no recursive translation from RBCM to RCM.

[10] Actually, their claim is for the class RCM, which we will show to be equivalent in Section 3.4.

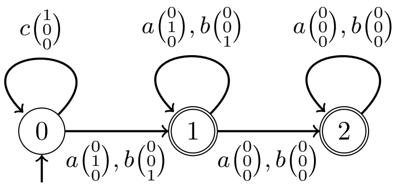

$c\binom{1}{0}{0}$ → $c\left(\begin{smallmatrix}1\\0\\0\end{smallmatrix}\right)$
$a\left(\begin{smallmatrix}0\\1\\0\end{smallmatrix}\right), b\left(\begin{smallmatrix}0\\0\\1\end{smallmatrix}\right)$
$a\left(\begin{smallmatrix}0\\0\\0\end{smallmatrix}\right), b\left(\begin{smallmatrix}0\\0\\0\end{smallmatrix}\right)$
0
1
2
$a\left(\begin{smallmatrix}0\\1\\0\end{smallmatrix}\right), b\left(\begin{smallmatrix}0\\0\\1\end{smallmatrix}\right)$
$a\left(\begin{smallmatrix}0\\0\\0\end{smallmatrix}\right), b\left(\begin{smallmatrix}0\\0\\0\end{smallmatrix}\right)$

Figure 1 A weakly-unambiguous Parikh automaton accepting the language $\mathcal{L} = \{c^n w \ : \ w = x_1 x_2 \cdots x_m \in \{a,b\}^* \ \wedge \ m \geq n > 0 \wedge \ |x_1 x_2 \cdots x_n|_a < |x_1 x_2 \cdots x_n|_b\}$ over the alphabet $\{a,b,c\}$ with the semilinear constraint $\{(n_1,n_2,n_3) \ : \ n_1 = n_2 + n_3$ and $n_2 < n_3\}$.

We now give a very short proof of the fact that weakly-unambiguous languages in PA have holonomic generating series. The idea of the proof can be traced back to [25]. A similar proof was given in [27] for languages in the class RCM but using the closure under algebraic substitutions instead of specialization (see Remark 25).

Our approach puts into light a different multivariate power series associated with a weakly-unambiguous PA $\mathcal{A}$ of dimension d. The multivariate *weighted generating series* $G(x, y_1, \ldots, y_d)$ of $\mathcal{A}$ is such that for all indices $(n, i_1, \ldots, i_d)$, $[x^n y_1^{i_1} \ldots y_d^{i_d}]G$ counts the number of words of length n accepted by $\mathcal{A}$ with a run labeled by the vector $(i_1, \ldots, i_d)$.

▶ **Proposition 12.** *The generating series of the language recognized by a weakly-unambiguous Parikh automaton is holonomic.*

Proof. Let $\mathcal{A}$ be a weakly-unambiguous PA with a constraint set $C \subseteq \mathbb{N}^d$. We first prove that its weighted series $G(x, y_1, \ldots, y_d)$ is holonomic. As holonomic series are closed under Hadamard product (see Theorem 8), it suffices to express G as the Hadamard product of two rational series $\overline{A}$ and $\overline{C}$ in the variables $x, y_1, \ldots, y_d$.

The first series $\overline{A}(x, y_1, \ldots, y_d)$ is such that for all $n, i_1, \ldots, i_d \geq 0$, $[x^n y_1^{i_1} \ldots y_d^{i_d}]\overline{A}$ counts the number of runs of $\mathcal{A}$ starting in q_I, ending in a final state and labeled with a word of length n and the vector $(i_1, \ldots, i_d)$. Note that we do not require that $(i_1, \ldots, i_d)$ belongs to C. As this series simply counts the number of runs in an automaton, its rationality is proved via the standard translation of the automaton into a linear system of equations.

For the second series, we take $\overline{C}(x, y_1, \ldots, y_d) := \frac{1}{1-x}\widetilde{C}(y_1, \ldots, y_d)$ where $\widetilde{C}$ is the support series of C, which is rational (see [13, 20]). A direct computation yields that for all $n, i_1, \ldots, i_d \geq 0$, $[x^n y_1^{i_1} \ldots y_d^{i_d}]\overline{C}$ is equal to 1 if $(i_1, \ldots, i_d)$ belongs to C and 0 otherwise.

The Hadamard product of $\overline{A}$ and $\overline{C}$ counts the number of runs accepting a word of length n with the vector $(i_1, \ldots, i_d)$. As $\mathcal{A}$ is weakly-unambiguous, this quantity is equal to the number of words of length n accepted with this vector. Hence $G = \bar{A} \odot \bar{C}$.

The univariate series $A(x)$ of $\mathcal{A}$ is equal to $G(x, 1, \ldots, 1)$. Indeed, for all $n \geq 0$, $[x^n]G(x, 1, \ldots, 1) = \sum_{\boldsymbol{i}=(i_1,\ldots,i_d)} [x^n y_1^{i_1} \cdots y_d^{i_d}]G(x, y_1, \ldots, y_d)$ is the sum over all vectors $\boldsymbol{i} \in \mathbb{N}^d$ of the number of words of length n accepted with the vector $\boldsymbol{i}$. As $\mathcal{A}$ is weakly-unambiguous, each word is accepted with at most one vector and this sum is therefore equal to the total number of accepted words of length n. Thanks to Proposition 7, $A(x) = G(x, 1, \ldots, 1)$ is holonomic. ◀

3.3 Equivalence with unambiguous reversal bounded counter machines

A *k-counter machine* [19] is informally a Turing machine with one read-only tape that contains the input word, and k counters. Reading a letter a on the input tape, in a state q, the machine can check which of its counters are zero, increment or decrement its counters, change its state, and move its read head one step to the left or right, or stay on its current position. Note that the machine does not have access to the exact value of its counters. A k-counter machine is said (m,n)-*reversal bounded* if its reading head can change direction between left and right at most m times, and if every counter can alternate between incrementing and decrementing at most n times each. Finally, a *reversal bounded counter machine* (RBCM) is a k-counter machine which is (m,n)-reversal bounded for some m and n. A RBCM is *unambiguous* if for every word there is at most one accepting computation.

RBCM are known[11] to recognize the same languages as Parikh automata (see [24, 23, 6]). This equality does not hold anymore for their deterministic versions [6, Prop. 3.14]. However, the proof of the equivalence for the general case can be slightly modified to preserve unambiguity.

▶ **Proposition 13.** *The class of languages accepted by unambiguous* RBCM *and weakly-unambiguous* PA *coincide.*

Proof sketch. Unambiguous RBCMs are shown to be equivalent to one-way unambiguous RBCMs. In turn these are shown to be equivalent to weakly-unambiguous PA with ε-transitions, which in turn are equivalent to weakly-unambiguous PA. This ε-removal step needs to be adapted to preserve weak-unambiguity. ◀

3.4 Equivalence with RCM

If we fix an alphabet $\Gamma = \{a_1, \dots, a_d\}$ with the ordering $a_1 < \dots < a_d$ on the letters, we can associate with every semilinear set C of dimension d, the language $[C] = \{w \in \Gamma^* : (|w|_{a_1}, \dots, |w|_{a_d}) \in C\}$ of words whose numbers of occurrences of each letter satisfy the constraint expressed by C. For instance, if we take the semilinear set $C_0 = \{(n,m,n,m) : n, m \geq 0\}$ and the alphabet $\{a,b,c,d\}$ ordered by $a < b < c < d$, $[C_0]$ consists of all words having as many a's as c's and as many b's as d's.

A language $\mathcal{L}$ over Σ belongs to RCM if there exist a regular language $\mathcal{R}$ over $\Gamma = \{a_1, \dots, a_d\}$, a semilinear set[12] $C \subseteq \mathbb{N}^d$ and a length preserving morphism $\mu\colon \Gamma^* \longrightarrow \Sigma^*$, that is injective over $\mathcal{R} \cap [C]$, so that $\mathcal{L} = \mu(\mathcal{R} \cap [C])$. For example, $\mathcal{L}_{abab} = \{a^n b^m a^n b^m : n, m \in \mathbb{N}\}$ can be shown to be in RCM by taking $\Gamma = \{a,b,c,d\}, \Sigma = \{a,b\}$, $\mu(a) = \mu(c) = a, \mu(b) = \mu(d) = b$, $\mathcal{R} = a^*b^*c^*d^*$ and the semilinear set C_0 defined in the previous paragraph.

▶ **Theorem 14.** $\mathcal{L} \in$ RCM *iff* $\mathcal{L}$ *is recognized by a weakly-unambiguous Parikh automaton.*

Proof sketch. Every language in RCM can be accepted by a weakly-unambiguous PA that guesses the underlying word over Γ: the weak-unambiguity is guaranteed by the injectivity of the morphism. Conversely a language accepted by a weakly-unambiguous PA is in RCM by taking for $\mathcal{R}$ the set of runs of the PA and translating the constraint: the injectivity of the morphism is guaranteed by the weak-unambiguity of the PA. ◀

In [8], the authors conjectured that the class RCM contains the one-way deterministic RBCM. From Theorem 14 and Proposition 13 we get a stronger result:

▶ **Corollary 15.** *The languages in RCM are the languages accepted by unambiguous RBCM.*

[11] The proof in [23] contains a patchable error, that was corrected in [6].

[12] Our definition may seem a little more general than Massazza's, which only uses semilinear defined without modulo constraints, but it can be shown that the classes are equivalent.

3.5 Weakly-unambiguous pushdown Parikh automata

A *pushdown Parikh automaton* ($\mathbb{P}$PA for short) is a PA where the finite automaton is replaced by a pushdown automaton. A *weakly-unambiguous* $\mathbb{P}$PA has at most one accepting run for each word. Most results obtained previously can be adapted for weakly-unambiguous $\mathbb{P}$PA. However, unsurprisingly, the class of languages accepted by weakly-unambiguous $\mathbb{P}$PA is not closed under union and intersection. This can be shown using the inherent weak-ambiguity of the language $\mathcal{D}$ proved in Section 4.1. The closure under complementation is left open.

▶ **Proposition 16.** *The generating series of a weakly-unambiguous* $\mathbb{P}$PA *is holonomic.*

Proof. The proof is almost identical to the proof of Proposition 12. The only difference is that the series $\overline{A}$ is algebraic and not rational. Indeed it counts the number of runs in a pushdown automaton and the language of runs is a deterministic context-free language even if the pushdown automaton is not deterministic. ◀

Remark that using the same techniques, we can prove that the generating series of weakly-unambiguous Parikh tree automata are holonomic. As we proved that all these series are also generating series of $\mathbb{P}$As, we do not elaborate on this model in this extended abstract.

RBCMs can be extended with a pushdown storage to obtain a *RBCM with a stack* [19].

▶ **Theorem 17.** *Weakly-unambiguous* $\mathbb{P}$PA *are equivalent to unambiguous one-way RBCM with a stack.*

Proof sketch. We first establish that unambiguous one-way RBCM with a stack are equivalent to weakly-unambiguous $\mathbb{P}$PA with ε-transitions. Contrarily to the PA case, the removal of ε-transitions is quite involved and uses weighted context-free grammars. ◀

The class LCL of [27] is defined[13] as RCM is, except that the regular language is replaced by an unambiguous context-free[14] language. Similarly to the PA case, one can prove:

▶ **Proposition 18.** *LCL is the set of languages accepted by weakly-unambiguous* $\mathbb{P}$PA.

4 Examples of inherently weakly-ambiguous languages

There is a polynomial-time algorithm to decide whether a given PA is weakly-unambiguous. But inherent weak-unambiguity is undecidable, as a direct application of a general theorem from [18]. This emphasizes that inherent weak-ambiguity is a difficult problem in general.

4.1 Two examples using an analytic criterion

Following an idea from Flajolet [15] for context-free languages, the link between weakly-unambiguous PA and holonomic series yields sufficient criteria to establish inherent weak-ambiguity, of analytic flavor: the contraposition of Proposition 12 indicates that if $\mathcal{L}$ is recognized by a PA but its generating series is not holonomic, then $\mathcal{L}$ is inherently weakly-ambiguous. Hence, any criterion of non-holonomicity can be used to establish the inherent weak-ambiguity. Many such criteria can be obtained when considering the generating series as analytic functions (of complex variables). See [15, 16] for several examples. For the presentation of this method in this extended abstract, we only rely on the following property:

[13] In [27], LCL is defined without the injective morphism but we adapt it following [8].

[14] Using deterministic context-free languages instead of unambiguous ones in the definition of LCL would result in the same class.

▶ **Proposition 19** ([33]). *A holonomic function in one variable has finitely many singularities.*

Our first example is the language $\mathcal{D}$, defined over the alphabet $\{a,b\}$ as follows:

$$\mathcal{D} = \{\underline{n_1}\ \underline{n_2} \dots \underline{n_k}\ :\ k > 0,\ n_1 = 1 \text{ and } \exists j < k, n_{j+1} \neq 2n_j\},\ \text{where } \underline{n} = a^n b.$$

This language is recognized by a weakly-ambiguous Parikh automaton, which guesses the correct j, and then verifies that $n_{j+1} \neq 2n_j$. Let $\overline{\mathcal{D}} = ab(a^*b)^* \setminus \mathcal{D}$, and suppose by contradiction that $\mathcal{D}$ can be recognized by a weakly-unambiguous PA. Then its generating series should be holonomic by Proposition 12. Since the generating series of $\overline{\mathcal{D}}$ is $\overline{D}(x_a, x_b) = \frac{x_a x_b}{1-\frac{x_b}{1-x_a}} - D(x_a, x_b)$, it should be holonomic too. Looking closely at the form of the words of $\overline{\mathcal{D}}$, we get that its generating series is $\sum_{k\geq 1} x_a^{2^k-1} x_b^k$. It is not holonomic as $x\overline{D}(x,1) + x$ has infinitely many singularities, see [15, p. 296–297].

Our second example is Shamir's language $\mathcal{S} = \{a^n b v_1 a^n v_2 : n \geq 1, v_1, v_2 \in \{a,b\}^*\}$. One can easily design a PA recognizing $\mathcal{S}$, where one coordinate stands for the length of the first run of a's and the other one for the second run of a's, the automaton guessing when the second run starts. Flajolet proved that $\mathcal{S}$ is inherently ambiguous as a context-free language, since its generating series $S(z) = \frac{z(1-z)}{1-2z} \sum_{n\geqslant 1} \frac{z^{2n}}{1-2z+z^{n+1}}$ has an infinite number of singularities [15, p. 296–297]. This also yields its inherent weak-ambiguity as a PA language.

4.2 Limit of the method: an example using pumping techniques

As already mentioned, the analytic method presented is not always sufficient to prove inherent ambiguity. In this section, we develop an example where it does not apply. We consider the following language $\mathcal{L}_{even}$, which is accepted both by a deterministic pushdown automaton and a non-deterministic PA (where $\underline{n} = a^n b$ as in Section 4.1):

$$\mathcal{L}_{even} = \left\{\underline{n_1}\ \underline{n_2} \dots \underline{n_{2k}}\ :\ k \in \mathbb{N},\ \forall i \leq 2k, n_i > 0,\ \text{and } \exists j \leq k, n_{2j} = n_{2j-1}\right\}.$$

In other words, $\mathcal{L}_{even}$ is the language of sequences of encoded numbers having two consecutive equal values, the first one being at an odd position. This language is accepted by a non-deterministic PA but is also deterministic context-free. This means that its generating series is algebraic and hence holonomic. This puts it out of the reach of our analytic method.

In this section we establish the following result:

▶ **Theorem 20.** *The language $\mathcal{L}_{even}$ is inherently weakly-ambiguous as a PA language.*

The remainder of this section is devoted to sketch the proof of this proposition. By contradiction, we suppose that $\mathcal{L}_{even}$ is recognized by a weakly-unambiguous PA $\mathcal{A}$.

An *a-piece* ω of $\mathcal{A}$ is a non-empty simple path of a-edges in $\mathcal{A}$, starting and ending at the same state: the states of the path are pairwise distinct, except for its extremities. The *origin* of w is its starting (and ending) state. Let $\Pi(\mathcal{A})$ be the (finite) set of a-pieces in $\mathcal{A}$.

We see a run in $\mathcal{A}$ as a sequence of transitions forming a path in $\mathcal{A}$. An *a-subpath* of a run R in $\mathcal{A}$ is a maximal consecutive subsequence of R whose transitions are all labeled by a's, that is, it cannot be extended further to the left nor to the right in R using a's.

Let R be an accepting run in $\mathcal{A}$. One can show that every a-subpath S of R can be decomposed as $S = w_1 \sigma_1^{s_1} w_2 \sigma_2^{s_2} \cdots w_f \sigma_f^{s_f} w_{f+1}$, where the σ_i's are a-pieces of $\Pi(\mathcal{A})$, the s_i's are positive integers, and the w_i's are paths not using twice the same state. Moreover, this decomposition is unique if we add the condition that if $w_i = \varepsilon$, then $\sigma_i \neq \sigma_{i-1}$ and the only state in common in w_i and σ_i is the origin of σ_i. This is done by repeatedly following the path until a state q is met twice, factorizing this segment of the form $w\sigma$, where σ is an a-piece of origin q. We call this decomposition the *canonical form* of S, and the *signature* of S is the tuple $(w_1, \sigma_1, w_2, \dots, w_f, \sigma_f, w_{f+1})$, i.e., we dropped the s_i's of the canonical form.

From the weak-unambiguity of $\mathcal{A}$ we can prove that there are at most c distinct possible signatures, where c only depends on $\mathcal{A}$, and that f is always at most $|Q_{\mathcal{A}}|$.

Ramsey's Theorem [32] guarantees that there exists an integer r such that any complete undirected graph with at least r vertices, whose edges are colored using c^2 different colors, admits a monochromatic triangle. We fix two positive integers n and k sufficiently large, which will be chosen later on, depending on $\mathcal{A}$ only. For $\ell \in \{1, \ldots, r\}$, let w_ℓ be the word $w_\ell = \underline{n_1}\,\underline{n_2} \ldots \underline{n_{2r}}$, where $n_i = n$ for odd i and $n_{2i} = n + k$ if $i \neq \ell$ and $n_{2\ell} = n$. Each w_ℓ is in $\mathcal{L}_{even}$, with a match at position 2ℓ only. By weak-unambiguity, each w_ℓ has a unique accepting run $\mathcal{R}_\ell$ in $\mathcal{A}$, each such run having $2r$ a-subpaths by construction. For $i \neq j$ in $\{1, \ldots, r\}$, let λ_{ij} be the signature of the $2j$-th a-subpath of $\mathcal{R}_i$, which is a path of length $n + k$ by definition. The complete undirected graph of vertex set $\{1, \ldots, r\}$ where each edge ij, with $i < j$, is colored by the pair $(\lambda_{ij}, \lambda_{ji})$ admits a monochromatic triangle of vertices $\alpha < \beta < \gamma$. In particular, $\lambda_{\alpha\beta} = \lambda_{\alpha\gamma}$ and $\lambda_{\gamma\alpha} = \lambda_{\gamma\beta}$.

We choose $k = \operatorname{lcm}(\{|\sigma| : \sigma \in \Pi(\mathcal{A})\})$, and n sufficiently large so that any a-subpath of an accepting run contains an a-piece σ repeated at least $k + 1$ times. This is possible as the w_i's have bounded length, and there are at most $|Q_{\mathcal{A}}| + 1$ of them. Hence, the 2γ-th a-subpath of w_α contains a a-piece σ that is repeated more than s times, where $s = k/|\sigma|$. As $\lambda_{\alpha\beta} = \lambda_{\alpha\gamma}$, the piece σ is also in the 2β-th a-subpath of w_α. If we alter the accepting path $\mathcal{R}_\alpha$ into $\mathcal{R}'_\alpha$ by looping s more times in σ in the a-subpath at position 2β and s less times in σ at position 2γ, we obtain a run for the word $w = \cdots \underline{n}\,\underset{2\alpha}{\underline{n}} \cdots \underline{n}\,\underset{2\beta}{\underline{n + 2k}} \cdots \underline{n}\,\underset{2\gamma}{\underline{n}} \cdots$. This run is accepting as the PA computes the same vector as for w_α, by commutativity of vectors addition. And the signatures remain unchanged, as there are sufficiently many repetitions of σ at position 2γ in w_α. Similarly, as $\lambda_{\gamma\alpha} = \lambda_{\gamma\beta}$, we can alter the accepting path $\mathcal{R}_\gamma$ into an accepting path $\mathcal{R}'_\gamma$ of same signatures as $\mathcal{R}_\gamma$ for the same word w, by removing $s' = k/|\sigma'|$ iterations of an a-piece σ' at position 2α and adding them at position 2β.

We have built two paths $\mathcal{R}'_\alpha$ and $\mathcal{R}'_\gamma$ that both accept the same word w. Therefore, as $\mathcal{A}$ is weakly-unambiguous, they are equal. As the signatures have not changed, this implies that the signature at position 2α in $\mathcal{R}'_\alpha$ is $\lambda_{\gamma\alpha}$, which is equal to $\lambda_{\gamma\beta}$ (monochromaticity), which is equal to $\lambda_{\alpha\beta}$ ($\mathcal{R}'_\alpha$ and $\mathcal{R}'_\gamma$ have same signatures). This is a contradiction as we could remove one a-piece at position 2α in w_α and add it at position 2β, while computing the same vector with the same starting and ending states: but this word is not in $\mathcal{L}_{even}$.

▶ Remark 21. The proof relies on manipulations of paths in the automaton, and we only use the commutativity of the addition for the vector part. Thus, it still holds if we consider automata where we use a recursively enumerable set instead of a semilinear set for acceptance.

5 Algorithmic consequence of holonomicity

Generating series of languages have already been used to obtain efficient algorithms on unambiguous models of automata. For instance, they were used by Stearns and Hunt as a basic tool to obtain bounds on the length of a word witnessing the non-inclusion between two unambiguous word automata [35]. More precisely, the proof in [35] relies on the recurrence equation satisfied by the coefficients of the generating series (which is guaranteed to exist by holonomicity in one variable). In the rational case, this recurrence relation can be derived from the automaton and does not require advanced results on holonomic series. In this section, our aim is to obtain a similar bound for the inclusion problem for weakly-unambiguous Parikh automata. The inclusion problem for RCM (and hence for weakly-unambiguous PA) is shown to be decidable in [8] but no complexity bound is provided. Note that this problem is known to be undecidable for non-deterministic PA [19]. We follow the same approach as for the rational case [35] and for RCM [8]. In stark contrast with the rational case, it is necessary to closely inspect holonomic closure properties in order to give concrete bounds.

Fix $\mathcal{A}$ and $\mathcal{B}$ two weakly-unambiguous PA. We can construct a weakly-unambiguous PA $\mathcal{C}$ accepting $L(\mathcal{A}) \cap L(\mathcal{B})$. We rely on the key fact that the series $D(x) = A(x) - C(x)$ counts the number of words of length n in $L(\mathcal{A}) \setminus L(\mathcal{B})$. In particular, $L(\mathcal{A}) \subseteq L(\mathcal{B})$ if and only if $D(x) = 0$.

As $D(x)$ is the difference of two holonomic series, it is holonomic. As equality between holonomic series is decidable, [8] concludes that the problem is decidable. But without further analysis, no complexity upper-bound can be derived. The coefficients of $D(x) = \sum_{n \geq 0} d_n x^n$ satisfy a recurrence equation of the form $p_0(n)d_n = \sum_{k=1}^{r} p_k(n)d_{n-k}$ for $n \geq r$ with $p_0(x) \neq 0$. This equation fully determines d_n in terms of its r previous values $d_{n-1}, \dots, d_{n-r}$, *provided that* $p_0(n) \neq 0$. In particular, if the r previous values are all equal to 0, then $d_n = 0$. Consequently, if d_n is equal to 0 for all $n \leq r + R$ where R denotes the largest positive root of p_0 (which is bounded from above by its ∞-norm, as a polynomial on $\mathbb{Z}$) then[15] $D(x) = 0$.

Taking $W := r + R$, we have that if $L(\mathcal{A}) \not\subseteq L(\mathcal{B})$ then there exists a word witnessing this non-inclusion of length at most W. We now aim at computing an upper-bound on W on the size of the inputs $\mathcal{A}$ and $\mathcal{B}$.

For this, we first bound the order of the linear recurrence satisfied by $A(x)$ and $C(x)$, as well as the degrees and norm of the polynomials involved. This is stated in Proposition 22, whose proof follows the one of Proposition 12, while establishing such bounds for the multivariate Hadamard product and the specialization to 1.

▶ **Proposition 22.** *The generating series $A(x)$ of a weakly-unambiguous PA $\mathcal{A}$ of dimension $d \geq 1$ satisfies a non-trivial linear differential equation $q_s(x)\partial_x^s A(x) + \cdots + q_0(x)A(x) = 0$, with $s \leq ((d+1)|\mathcal{A}|\,\|\mathcal{A}\|_\infty)^{O(d)}$ and for all $i \in [0,s]$, $\deg(q_i) \leq ((d+1)|\mathcal{A}|\,\|\mathcal{A}\|_\infty)^{O(d)}$ and $\log \|q_i\|_\infty \leq ((d+1)|\mathcal{A}|\,\|\mathcal{A}\|_\infty)^{O(d^2)}$ using the notations of Section 3.2.*

Finally, we transfer these bounds to the series $D(x)$ of $L(\mathcal{A}) \setminus L(\mathcal{B})$ using the analysis of [22] for the sum of holonomic series in one variable.

▶ **Theorem 23.** *Given two weakly-unambiguous PA $\mathcal{A}$ and $\mathcal{B}$ of respective dimensions $d_\mathcal{A}$ and $d_\mathcal{B}$, if $L(\mathcal{A})$ is not included in $L(\mathcal{B})$ then there exists a word in $L(\mathcal{A}) \setminus L(\mathcal{B})$ of length at most $2^{2^{O(d^2 \log(dM))}}$ where $d = d_\mathcal{A} + d_\mathcal{B}$ and $M = |\mathcal{A}|\,|\mathcal{B}|\,\|\mathcal{A}\|_\infty\,\|\mathcal{B}\|_\infty$.*

Using the bound of Theorem 23, the inclusion problem can be solved in triply exponential time by a naive counting of all words up to the bound. Using dynamic programming to compute the number of accepted words, we can decide inclusion in doubly exponential time.

▶ **Corollary 24.** *Given two weakly-unambiguous PA $\mathcal{A}$ and $\mathcal{B}$ of dimensions $d_\mathcal{A}$ and $d_\mathcal{B}$, we can decide if $L(\mathcal{A})$ is included in $L(\mathcal{B})$ in time $2^{2^{O(d^2 \log(dM))}}$ where $d = d_\mathcal{A} + d_\mathcal{B}$ and $M = |\mathcal{A}|\,|\mathcal{B}|\,\|\mathcal{A}\|_\infty\,\|\mathcal{B}\|_\infty$.*

▶ Remark 25. In [8], the authors propose a different construction to prove the holonomicity of the generating series of languages in RCM. This proof uses the closure of holonomic series under Hadamard product and algebraic substitution $x_1 = x_2 = \cdots = x_n = x$. It is natural to wonder if this approach would lead to better bounds in Proposition 22 (using the

[15] The proof of Theorem 7 in [8] wrongly suggests that we can take the order of the differential equation for D as a bound on the length of a witness for non-inclusion. In general, this is not the case. For instance consider $D(x) = x^{1000}$ which satisfies the first-order differential equation $1000D(x) - x\partial_x D(x) = 0$. It is clear that the coefficients $D_0 = 0$ and $D_1 = 0$ are not enough to decide that D is not zero.

equivalence between weakly-unambiguous PA and RCM). It turns out that the operation $x_1 = x_2 = \cdots = x_n = x$ is more complicated than it seems at a first glance. Indeed, to our knowledge, no proof of the closure under algebraic substitution explains what happens if, during the substitution process, the equations become trivial. This issue can be overcome by doing the substitution step by step: $x_2 = x_1$, then $x_3 = x_1$, etc. However, this naive approach would produce worse bounds.

6 Perspectives

The bounds obtained in Section 5 are derived directly from constructions given in the proofs of the closure properties. In particular, we did not use any information on the special form of our series. The bounds are certainly perfectible using more advanced tools from computer algebra. Also it seems that the complexity of the closure under the algebraic substitution deserves more investigation, as discussed in Remark 25.

A more ambitious perspective is to find larger classes of automata whose generating series are holonomic. This would certainly require new ideas, as for instance any holonomic power series with coefficients in $\{0, 1\}$ is known to be the characteristic series of some semilinear set [1].

References

1 Jason P. Bell and Shaoshi Chen. Power series with coefficients from a finite set. *J. Comb. Theory, Ser. A*, 151:241–253, 2017. doi:10.1016/j.jcta.2017.05.002.

2 Joseph N. Bernstein. Analytic continuation of generalized functions with respect to a parameter. *Funct. Anal. Appl.*, 6(4):273–28, 1972. doi:10.1007/BF01077645.

3 Alin Bostan, Frédéric Chyzak, Grégoire Lecerf, Bruno Salvy, and Éric Schost. Differential equations for algebraic functions. In *International Symposium on Symbolic and Algebraic Computation, ISSAC 2007*, pages 25–32. ACM, 2007. doi:10.1145/1277548.1277553.

4 Mireille Bousquet-Mélou. Rational and algebraic series in combinatorial enumeration. In *International Congress of Mathematicians (ICM 2006)*, volume 3, pages 789–826. Eur. Math. Soc., Zürich, 2006. doi:10.4171/022-3/40.

5 Michaël Cadilhac. *Automata with a semilinear constraint.* PhD thesis, Université de Montréal, 2013.

6 Michaël Cadilhac, Alain Finkel, and Pierre McKenzie. Affine Parikh automata. *RAIRO – Theor. Inf. and Applic.*, 46(4):511–545, 2012. doi:10.1051/ita/2012013.

7 Michaël Cadilhac, Alain Finkel, and Pierre McKenzie. Unambiguous constrained automata. *Int. J. Found. Comput. Sci.*, 24(7):1099–1116, 2013. doi:10.1142/S0129054113400339.

8 Giusi Castiglione and Paolo Massazza. On a class of languages with holonomic generating functions. *Theor. Comput. Sci.*, 658:74–84, 2017. doi:10.1016/j.tcs.2016.07.022.

9 Dmitry Chistikov and Christoph Haase. The taming of the semi-linear set. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016*, volume 55 of *LIPIcs*, pages 128:1–128:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.ICALP.2016.128.

10 Noam Chomsky and Marcel-Paul Schützenberger. *The Algebraic Theory of Context-Free Languages*, volume 35 of *Studies in Logic and the Foundations of Mathematics*. Elsevier, 1963. doi:10.1016/S0049-237X(08)72023-8.

11 Thomas Colcombet. Unambiguity in automata theory. In Jeffrey Shallit and Alexander Okhotin, editors, *Descriptional Complexity of Formal Systems - 17th International Workshop, DCFS 2015, Waterloo, ON, Canada, June 25-27, 2015. Proceedings*, volume 9118 of *Lecture Notes in Computer Science*, pages 3–18. Springer, 2015. doi:10.1007/978-3-319-19225-3_1.

12 Louis Comtet. Calcul pratique des coefficients de Taylor d'une fonction algébrique. *Enseignement Math. (2)*, 10:267–270, 1964.

13 Samuel Eilenberg and Marcel-Paul Schützenberger. Rational sets in commutative monoids. *J. Algebra*, 13(2):173–191, 1969. doi:10.1016/0021-8693(69)90070-2.

14 Emmanuel Filiot, Shibashis Guha, and Nicolas Mazzocchi. Two-Way Parikh Automata. In Arkadev Chattopadhyay and Paul Gastin, editors, *39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019)*, volume 150 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 40:1–40:14, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.FSTTCS.2019.40.

15 Philippe Flajolet. Analytic models and ambiguity of context-free languages. *Theor. Comput. Sci.*, 49(2):283–309, 1987. doi:10.1016/0304-3975(87)90011-9.

16 Philippe Flajolet, Stefan Gerhold, and Bruno Salvy. On the non-holonomic character of logarithms, powers, and the nth prime function. *Electr. J. Comb.*, 11(2), 2005. doi:10.37236/1894.

17 Philippe Flajolet and Robert Sedgewick. *Analytic Combinatorics*. Cambridge University Press, first edition, 2009.

18 Sheila Greibach. A note on undecidable properties of formal languages. *Mathematical Systems Theory*, 2(1):1–6, 1968. doi:10.1007/BF01691341.

19 Oscar H. Ibarra. Reversal-bounded multicounter machines and their decision problems. *J. ACM*, 25(1):116–133, 1978. doi:10.1145/322047.322058.

20 Ryuichi Ito. Every semilinear set is a finite union of disjoint linear sets. *J. Comput. Syst. Sci.*, 3(2):221–231, 1969. doi:10.1016/S0022-0000(69)80014-0.

21 Masaki Kashiwara. On the holonomic systems of linear differential equations. II. *Invent. Math.*, 49(2):121–135, 1978. doi:10.1007/BF01403082.

22 Manuel Kauers. Bounds for D-finite closure properties. In *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation, (ISSAC 2014)*, pages 288–295. ACM, New York, 2014. doi:10.1145/2608628.2608634.

23 Felix Klaedtke and Harald Rueß. Parikh automata and Monadic Second-Order logics with linear cardinality constraints. Technical Report 177, Freiburg University, 2002.

24 Felix Klaedtke and Harald Rueß. Monadic second-order logics with cardinalities. In *Automata, Languages and Programming, 30th International Colloquium, ICALP 2003*, volume 2719 of *Lecture Notes in Computer Science*, pages 681–696. Springer, 2003. doi:10.1007/3-540-45061-0_54.

25 Leonard Lipshitz. The diagonal of a D-finite power series is D-finite. *J. Algebra*, 113(2):373–378, 1988. doi:10.1016/0021-8693(88)90166-4.

26 Leonard Lipshitz. D-finite power series. *J. Algebra*, 122(2):353–373, 1989. doi:10.1016/0021-8693(89)90222-6.

27 Paolo Massazza. Holonomic functions and their relation to linearly constrained languages. *ITA*, 27(2):149–161, 1993. doi:10.1051/ita/1993270201491.

28 Paolo Massazza. On the conjecture $\mathcal{L}_{\text{dfcm}} \subsetneq$ RCM. In *Implementation and Application of Automata - 22nd International Conference, CIAA 2017*, volume 10329 of *Lecture Notes in Computer Science*, pages 175–187. Springer, 2017. doi:10.1007/978-3-319-60134-2_15.

29 Paolo Massazza. On the generating functions of languages accepted by deterministic one-reversal counter machines. In *Proceedings of the 19th Italian Conference on Theoretical Computer Science, (ICTCS 2018)*, volume 2243 of *CEUR Workshop Proceedings*, pages 191–202. CEUR-WS.org, 2018.

30 Rohit J. Parikh. On context-free languages. *J. ACM*, 13(4):570–581, 1966. doi:10.1145/321356.321364.

31 Mojżesz Presburger. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. *C.R. 1er Congrès des Mathématiciens des pays slaves*, pages 92–101, 1929.

32 Frank P. Ramsey. On a Problem of Formal Logic. *Proc. London Math. Soc. (2)*, 30(4):264–286, 1929. doi:10.1112/plms/s2-30.1.264.

33 Richard P. Stanley. Differentiably finite power series. *Eur. J. Comb.*, 1(2):175–188, 1980. doi:10.1016/S0195-6698(80)80051-5.

34 Richard P. Stanley. *Enumerative combinatorics. Vol. 2*, volume 62 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, Cambridge, 1999. doi:10.1017/CBO9780511609589.

35 Richard Edwin Stearns and Harry B. Hunt III. On the equivalence and containment problems for unambiguous regular expressions, regular grammars and finite automata. *SIAM J. Comput.*, 14(3):598–611, 1985. doi:10.1137/0214044.

36 Nobuki Takayama. An approach to the zero recognition problem by Buchberger algorithm. *J. Symbolic Comput.*, 14(2-3):265–282, 1992. doi:10.1016/0747-7171(92)90039-7.

On the Size of Finite Rational Matrix Semigroups

Georgina Bumpus
University of Oxford, UK

Christoph Haase
University College London, UK

Stefan Kiefer
University of Oxford, UK

Paul-Ioan Stoienescu
University of Oxford, UK

Jonathan Tanner
University of Oxford, UK

Abstract

Let n be a positive integer and $\mathcal{M}$ a set of rational $n \times n$-matrices such that $\mathcal{M}$ generates a finite multiplicative semigroup. We show that any matrix in the semigroup is a product of matrices in $\mathcal{M}$ whose length is at most $2^{n(2n+3)}g(n)^{n+1} \in 2^{O(n^2 \log n)}$, where $g(n)$ is the maximum order of finite groups over rational $n \times n$-matrices. This result implies algorithms with an elementary running time for deciding finiteness of weighted automata over the rationals and for deciding reachability in affine integer vector addition systems with states with the finite monoid property.

2012 ACM Subject Classification Mathematics of computing → Discrete mathematics; Theory of computation → Algebraic language theory; Theory of computation → Quantitative automata

Keywords and phrases Matrix semigroups, Burnside problem, weighted automata, vector addition systems

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.115

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

Related Version An extended version of the paper is available at https://arxiv.org/abs/1909.04756.

Funding This work is part of a project that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant agreement No. 852769, ARiAT).
Georgina Bumpus: Work supported by St John's College, Oxford.
Stefan Kiefer: Work supported by a Royal Society University Research Fellowship.
Paul-Ioan Stoienescu: Work supported by St John's College, Oxford.
Jonathan Tanner: Work supported by St John's College, Oxford.

Acknowledgements The authors thank James Worrell for discussing [17] with them.

1 Introduction

The Burnside Problem

An element g of a semigroup G is called *torsion* if $g^i = g^j$ holds for some naturals $i < j$, and G *torsion* if all its elements are torsion. Burnside [6] asked in 1902 a question which became known as the *Burnside problem* for groups: is every finitely generated torsion group finite? Schur [28] showed in 1911 that this holds true for groups of invertible complex matrices, i.e., any finitely generated torsion subgroup of $GL(n, \mathbb{C})$ is finite. This was generalised by

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 115; pp. 115:1–115:13
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Kaplansky [21, p. 105] to matrices over arbitrary fields. The Burnside problem for groups has a negative answer in general: in 1964 Golod and Shafarevich exhibited a finitely generated infinite torsion group [13, 14].

The Maximal Order of Finite Matrix Groups

Schur's result [28] assures that finitely generated torsion matrix groups are finite, but does not bound the group order. Indeed, it is easy to see that any finite cyclic group is isomorphic to a group generated by a matrix in $GL(2, \mathbb{R})$. The same is not true for $GL(n, \mathbb{Q})$: An elementary proof, see e.g. [23], shows that any finite subgroup of $GL(n, \mathbb{Q})$ is conjugate to a finite subgroup of $GL(n, \mathbb{Z})$. Another elementary proof shows that the order of any finite subgroup of $GL(n, \mathbb{Z})$ divides $(2n)!$; see, e.g., [27, Chapter IX]. Thus, denoting the order of the largest finite subgroup of $GL(n, \mathbb{Q})$ by $g(n)$, we have $g(n) \leq (2n)!$. It is shown in a paper by Friedland [12] that $g(n) = 2^n n!$ holds for all sufficiently large n. This bound is attained by the group of signed permutation matrices. Friedland's proof rests on an article by Weisfeiler [34] which in turn is based on the classification of finite simple groups. Feit showed in an unpublished manuscript [9] that $g(n) = 2^n n!$ holds if and only if $n \in \mathbb{N} \setminus \{2, 4, 6, 7, 8, 9, 10\}$.[1] Feit's proof relies on an unpublished manuscript [33], also based on the classification of finite simple groups, which Weisfeiler left behind before his tragic disappearance.

Deciding Finiteness of Matrix Groups

Bounds on group orders give a straightforward, albeit inefficient, way of deciding whether a given set of matrices generates a finite group: starting from the set of generators, enlarge it with products of matrices in the set, until either it is closed under product or the bound on the order has been exceeded. One can do substantially better: it is shown in [2] that, using computations on quadratic forms, one can decide in polynomial time if a given finite set of rational matrices generates a finite group.

Deciding Finiteness of Matrix Semigroups

The Burnside problem has a natural analogue for semigroups. In 1975, McNaughton and Zalcstein [26] positively solved the Burnside problem for matrix semigroups, i.e., they showed, for any field $\mathbb{F}$, that any finitely generated torsion subsemigroup of $\mathbb{F}^{n \times n}$ is finite, using the result for groups by Schur and Kaplansky as a building block. From a computational point of view, McNaughton and Zalcstein's result suggests an approach for deciding finiteness of the semigroup generated by a given set of rational matrices: finiteness is recursively enumerable, by closing the set of generators under product, as described above for groups. On the other hand, infiniteness is recursively enumerable by enumerating elements in the generated semigroup and checking each element whether it is torsion. By the contrapositive of McNaughton and Zalcstein's result, if the generated matrix semigroup is infinite, it has a non-torsion element, witnessing infiniteness. However, deciding whether a given matrix has finite order is nontrivial. Only in 1980 did Kannan and Lipton [19, 20] show that the so-called *orbit problem* is decidable (in polynomial time), implying an algorithm for checking whether a matrix has finite order.

[1] A list of the maximal-order finite subgroups of $GL(n, \mathbb{Q})$ for $n \in \{2, 4, 6, 7, 8, 9, 10\}$ can be found in [3, Table 1].

Avoiding this problem, Mandel and Simon [25] showed in 1977 that there exists a function $f : \mathbb{N}^3 \to \mathbb{N}$ such that if S is a finite subsemigroup of $\mathbb{F}^{n\times n}$, generated by m of its elements, and the subgroups of S have order at most g, then S has size (cardinality) at most $f(n, m, g)$. For rational matrices, one may use the function $g(n)$ from above for g. By making, in a sense, McNaughton and Zalcstein's proof quantitative, Mandel and Simon explicitly construct such a function f, which implies an algorithm, with bounded runtime, for deciding finiteness of a finitely generated rational matrix semigroup. A similar result about the decidability of this problem was obtained independently and concurrently by Jacob [18].

Size Bounds

Unlike the function g for rational matrix groups, Mandel and Simon's function $f(n, m, g)$ depends on m, the number of generators. This is unavoidable: the semigroup generated by the set $\mathcal{M}_m := \left\{\begin{pmatrix} 0 & i \\ 0 & 0 \end{pmatrix} : i \in \{0, \ldots, m-1\}\}\right\}$ is the set $\mathcal{M}_m$ itself, with $|\mathcal{M}_m| = m$ for any $m \in \mathbb{N}$. Further, the growth in n of Mandel and Simon's f is, roughly, a tower of exponentials of height n. They write in [25, Section 3]: "However, it is likely that our upper bound $[f(n, m, g)]$ can be significantly improved."

In [4, Chapter VI], Berstel and Reutenauer also show, for the rational case, the existence of a function in n and m that bounds the semigroup size. They write: "As we shall see, the function [...] grows extremely rapidly." An analysis of their proof shows that the growth of their function is comparable with the growth of Mandel and Simon's function. A related approach is taken in [31]. Further proofs of McNaughton and Zalcstein's result can be found, e.g., in [24, 11, 8, 30], but they do not lead to better size bounds.

Length Bounds

In 1991, Weber and Seidl [32] considered semigroups over *nonnegative* integer matrices. Using combinatorial and automata-theoretic techniques, they showed that if a finite set $\mathcal{M} \subseteq \mathbb{N}^{n\times n}$ generates a finite monoid, then for any matrix M of that monoid there are $M_1, \ldots, M_\ell \in \mathcal{M}$ with $\ell \leq \lceil e^2 n! \rceil - 2$ such that $M = M_1 \cdots M_\ell$; i.e., any matrix in the monoid is a product of matrices in $\mathcal{M}$ whose *length* is at most $\lceil e^2 n! \rceil - 2$. Note that this bound does not depend on the number of generators. Weber and Seidl also give an example that shows that such a length bound cannot be smaller than 2^{n-2}.

Almeida and Steinberg [1] proved in 2009 a length bound for rational matrices and expressing the zero matrix: if a finite set $\mathcal{M} \subseteq \mathbb{Q}^{n\times n}$ (with $n > 1$) generates a *finite* semigroup that includes the zero matrix 0, then there are $M_1, \ldots, M_\ell \in \mathcal{M}$ with $\ell \leq (2n-1)^{n^2} - 1$ such that $0 = M_1 \cdots M_\ell$. A length bound of n^5 for expressing the zero matrix was recently given in the nonnegative integer case [22]. It is open whether there is a polynomial length bound for expressing the zero matrix in the rational case.

Our Contribution

We prove a $2^{O(n^2 \log n)}$ length bound for the rational case:

▶ **Theorem 1.** *Let $\mathcal{M} \subseteq \mathbb{Q}^{n\times n}$ be a finite set of rational matrices such that $\mathcal{M}$ generates a finite semigroup $\overline{\mathcal{M}}$. Then for any $M \in \overline{\mathcal{M}}$ there are $M_1, \ldots, M_\ell \in \mathcal{M}$ with $\ell \leq 2^{n(2n+3)} g(n)^{n+1} \in 2^{O(n^2 \log n)}$ such that $M = M_1 \cdots M_\ell$. (Here $g(n) \leq (2n)!$ denotes the order of the largest finite subgroup of $GL(n, \mathbb{Q})$.)*

The example by Weber and Seidl mentioned above shows that any such length bound must be at least 2^{n-2}. A length bound trivially implies a size bound, and Theorem 1 allows us to obtain the first significant improvement over the fast-growing function of Mandel and Simon.

▶ **Corollary 2.** *Let $\mathcal{M} \subseteq \mathbb{Q}^{n\times n}$ be a finite set of m rational matrices that generate a finite semigroup $\overline{\mathcal{M}}$. Then $|\overline{\mathcal{M}}| \le m^{2^{O(n^2 \log n)}}$.*

The proof of Theorem 1 is largely based on linear-algebra arguments, specifically on the structure of a certain graph of vector spaces obtained from $\mathcal{M}$. This graph was introduced and analysed by Hrushovski et al. [17] for the computation of the *Zariski closure* of the generated matrix semigroup.

After the preliminaries (section 2) and the proof of Theorem 1 (section 3), we discuss applications in automata theory (section 4). In particular we show that our result implies the first elementary-time algorithm for deciding finiteness of weighted automata over the rationals.

2 Preliminaries

We write $\mathbb{N} = \{0, 1, 2, \ldots\}$. For a finite alphabet Σ, we write $\Sigma^* = \{a_1 \cdots a_k : k \ge 0,\ a_i \in \Sigma\}$ and $\Sigma^+ = \{a_1 \cdots a_k : k \ge 1,\ a_i \in \Sigma\}$ for the free monoid and the free semigroup generated by Σ. The elements of Σ^* are called *words*. For a word $w = a_1 \cdots a_k$, its *length* $|w|$ is k. We denote by ε the empty word, i.e., the word of length 0. For $L \subseteq \Sigma^*$, we also write $L^* = \{w_1 \cdots w_k : k \ge 0,\ w_i \in L\} \subseteq \Sigma^*$ and $L^+ = \{w_1 \cdots w_k : k \ge 1,\ w_i \in L\} \subseteq \Sigma^*$.

We denote by I_n the $n\times n$-identity matrix, and by $\vec{0}$ the zero vector. For vectors $v_1, \ldots, v_k$ from a vector space, we denote their span by $\langle v_1, \ldots, v_k\rangle$. In this article, we view elements of $\mathbb{Q}^n$ as *row* vectors.

For some $n \in \mathbb{N} \setminus \{0\}$, let $\mathcal{M} \subseteq \mathbb{Q}^{n\times n}$ be a finite set of rational matrices, generating a finite semigroup $\overline{\mathcal{M}}$. For notational convenience, throughout the paper, we associate to $\mathcal{M}$ an alphabet Σ with $|\mathcal{M}| = |\Sigma|$, and a bijection $M : \Sigma \to \mathcal{M}$ which we extend to the monoid morphism $M : \Sigma^* \to \overline{\mathcal{M}} \cup \{I_n\}$. Thus we may write $M(\Sigma)$ and $M(\Sigma^*)$ for $\mathcal{M}$ and $\overline{\mathcal{M}} \cup \{I_n\}$, respectively.

We often identify a matrix $A \in \mathbb{Q}^{n\times n}$ with its linear transformation $A : \mathbb{Q}^n \to \mathbb{Q}^n$ such that $x \mapsto xA$ for row vectors $x \in \mathbb{Q}^n$. To avoid clutter, we extend linear-algebra notions from matrices to words, i.e., we may write $\operatorname{im} w$, $\ker w$, $\operatorname{rk} w$ for the image $\operatorname{im}(M(w)) = \mathbb{Q}^n M(w)$, the kernel $\ker(M(w)) = \{x \in \mathbb{Q}^n : xM(w) = \vec{0}\}$, and the rank of $M(w)$.

If all matrices in $M(\Sigma)$ are invertible and $M(\Sigma^*)$ is finite, then $M(\Sigma^*)$ is a finite subgroup of $GL(n, \mathbb{Q})$. For $n \in \mathbb{N}$, let us write $g(n)$ for the size of the largest finite subgroup of $GL(n, \mathbb{Q})$. As discussed in the introduction, a non-trivial but elementary proof shows $g(n) \le (2n)!$, and it is known that $g(n) = 2^n n!$ holds for sufficiently large n.

Exterior Algebra

This brief introduction is borrowed and slightly extended from [17, Section 3]. Let V be an n-dimensional vector space over a field $\mathbb{F}$. (We will only consider $V = \mathbb{Q}^n$.) For any $r \in \mathbb{N}$, let $\mathcal{A}_r$ denote the set of maps $B : V^r \to \mathbb{F}$ so that B is linear in each argument and further $B(v_1, \ldots, v_r) = 0$ holds whenever $v_i = v_{i+1}$ holds for some $i \in \{1, \ldots, r-1\}$. These conditions imply that swapping two adjacent arguments changes the sign, i.e.,

$$B(v_1, \ldots, v_{i-2}, v_{i-1}, v_{i+1}, v_i, v_{i+2}, v_{i+3}, \ldots, v_r) \ = \ -B(v_1, \ldots, v_r)\,.$$

These properties of $\mathcal{A}_r$ imply that, given an arbitrary basis $\{e_1, \ldots, e_n\}$ of V, any $B \in \mathcal{A}_r$ is uniquely determined by all $B(e_{i_1}, \ldots, e_{i_r})$ where $1 \leq i_1 < i_2 < \ldots < i_r \leq n$. For any $v_1, \ldots, v_r \in V$, define the *wedge product*

$$v_1 \wedge \cdots \wedge v_r : \mathcal{A}_r \to \mathbb{F} \quad \text{by} \quad (v_1 \wedge \cdots \wedge v_r)(B) = B(v_1, \ldots, v_r)\,.$$

It follows from the properties of $\mathcal{A}_r$ above that the wedge product is linear in each argument: if $v_i = \lambda u + \lambda' u'$ then

$$\Big(\bigwedge_{1\leq i\leq k} v_i\Big)(B) = \lambda\Big(\bigwedge_{1\leq j<i} v_j \wedge u \wedge \bigwedge_{i<j\leq k} v_j\Big)(B) + \lambda'\Big(\bigwedge_{1\leq j<i} v_j \wedge u' \wedge \bigwedge_{i<j\leq k} v_j\Big)(B)$$

Moreover, $(v_1 \wedge \cdots \wedge v_r)(B) = 0$ if $v_i = v_j$ holds for some i, j with $i \neq j$.

For $r \in \mathbb{N}$ define $\Lambda^r V$ as the vector space generated by the length-r wedge products $v_1 \wedge \cdots \wedge v_r$ with $v_1, \ldots, v_r \in V$. For any basis $\{e_1, \ldots, e_n\}$ of V, the set $\{e_{i_1} \wedge \cdots \wedge e_{i_r} : 1 \leq i_1 < \ldots < i_r \leq n\}$ is a basis of $\Lambda^r V$; hence $\dim \Lambda^r V = \binom{n}{r}$. Note that $\Lambda^1 V = V$ and $\binom{n}{r} = 0$ for $r > n$. One can view the wedge product as an associative operation $\wedge : \Lambda^r V \times \Lambda^\ell V \to \Lambda^{r+\ell} V$. Define the *exterior algebra of* V as the direct sum $\Lambda V = \Lambda^0 V \oplus \Lambda^1 V \oplus \cdots$. Then also $\wedge : \Lambda V \times \Lambda V \to \Lambda V$.

It follows that for $u_1, \ldots, u_r \in V$, we have $u_1 \wedge \cdots \wedge u_r \neq \vec{0}$ if and only if $\{u_1, \ldots, u_r\}$ is linearly independent. Furthermore, for $u_1, \ldots, u_r, v_1, \ldots, v_r \in V$ and $u = u_1 \wedge \cdots \wedge u_r \neq \vec{0}$ and $v = v_1 \wedge \cdots \wedge v_r \neq \vec{0}$, we have that u, v are scalar multiples if and only if $\langle u_1, \ldots, u_r \rangle = \langle v_1, \ldots, v_r \rangle$.

The Grassmannian $\mathrm{Gr}(n)$ is the set of subspaces of $\mathbb{Q}^n$. By the above-stated properties of the wedge product there is an injective function

$$\iota : \mathrm{Gr}(n) \to \Lambda\, \mathbb{Q}^n$$

such that, for all $W \in \mathrm{Gr}(n)$, we have $\iota(W) = v_1 \wedge \cdots \wedge v_r$ where $\{v_1, \ldots, v_r\}$ is an arbitrarily chosen basis of W. Note that the particular choice of a basis for W only changes the value of $\iota(W)$ up to a constant. Given subspaces $W_1, W_2 \in \mathrm{Gr}(n)$, we moreover have $W_1 \cap W_2 = \{\vec{0}\}$ if and only if $\iota(W_1) \wedge \iota(W_2) \neq \vec{0}$.

3 Proof of Theorem 1

It is convenient to state and prove our main result in terms of monoids rather than semigroups:

▶ **Theorem 3.** *Let $M : \Sigma^* \to \mathbb{Q}^{n\times n}$ be a monoid morphism whose image $M(\Sigma^*)$ is finite. Then for any $w \in \Sigma^*$ there is $u \in \Sigma^*$ with $M(w) = M(u)$ and*

$$|u| \;\leq\; 2^{n(2n+3)} g(n)^{n+1} \in 2^{O(n^2 \log n)}\,.$$

With this theorem at hand, Theorem 1 follows immediately:

Proof of Theorem 1. Let $M \in \overline{\mathcal{M}}$ be an element of the semigroup generated by $\mathcal{M}$. If $M \neq I_n$, by Theorem 3, M can be written as a short product. Otherwise, $M = I_n \in G$, where $G = \overline{\mathcal{M}} \cap GL(n, \mathbb{Q})$ is a finite group of order at most $g(n)$. For any product $M_1 \cdots M_\ell$ with $\ell > g(n)$, there are $1 \leq i < j \leq \ell$ such that $M_1 \cdots M_i = M_1 \cdots M_j$, and so $M_1 \cdots M_\ell = M_1 \cdots M_i M_{j+1} \cdots M_\ell$. Hence, there are $\ell \in \{1, \ldots, g(n)\}$ and $M_1, \ldots, M_\ell \in \mathcal{M}$ such that $M = I_n = M_1 \cdots M_\ell$. ◀

▶ **Remark 4.** *The same argument as in the proof above shows that in a finite* monoid $(H, \cdot)$, *generated by* $G \subseteq H$, *for any* $h \in H$ *there are* $\ell \in \{0, \ldots, |H|-1\}$ *and* $g_1, \ldots, g_\ell \in G$ *with* $h = g_1 \cdots g_\ell$.

In the remainder of this section, we prove Theorem 3. We assume that $M : \Sigma^* \to \mathbb{Q}^{n\times n}$ is a monoid morphism with finite image $M(\Sigma^*)$.

3.1 The Maximum-Rank Case

In this subsection we prove:

▶ **Proposition 5.** *Suppose that there is* $r \le n$ *with* $\operatorname{rk} a = r$ *for all* $a \in \Sigma$. *Let* $w \in \Sigma^*$ *with* $\operatorname{rk} w = r$. *Then there is* $u \in \Sigma^*$ *with* $M(w) = M(u)$ *and*

$$|u| \ \le\ 2^{2n+3} g(n) - 1 \ \in\ 2^{O(n \log n)} .$$

In this subsection we assume that $\operatorname{rk} a = r$ holds for all $a \in \Sigma$. For the proof of Proposition 5, we define a directed labelled graph G whose vertices are the vector spaces $\operatorname{im} w$ for $w \in \Sigma^*$ such that $\operatorname{rk} w = r$, and whose edges are triples (V_1, a, V_2) such that $a \in \Sigma$ and $V_1 M(a) = V_2$. Let (V_1, a, V_2) be an edge; then $V_2 \subseteq \operatorname{im} a$, but $\dim V_2 = r = \operatorname{rk} a = \dim \operatorname{im} a$, hence $V_2 = \operatorname{im} a$, i.e., the edge label determines the edge target. We will implicitly use the fact that any path in G is determined by its start vertex and the sequence of its edge labels. Note that if V_1 is a vertex and $a \in \Sigma$, the edge $(V_1, a, \operatorname{im} a)$ is present in G if and only if $\operatorname{rk} V_1 M(a) = r$ if and only if $V_1 \cap \ker a = \{\vec{0}\}$.

The following two lemmas, which are variants of lemmas in [17, Section 6], are statements about the structure of G in terms of its strongly connected components (SCCs).

▶ **Lemma 6.** *Let* $w = w_1 \cdots w_k$ *for* $w_1, \ldots, w_k \in \Sigma^+$ *with* $\operatorname{rk} w = r$ *such that the* k *vertices* $\operatorname{im} w_1, \ldots, \operatorname{im} w_k$ *are all in different SCCs of* G. *Then* $k \le 2\binom{n}{r}$.

Proof. Let $i \in \{2, \ldots, k-1\}$. Since $\operatorname{rk} w_i = r = \operatorname{rk}(w_i w_{i+1})$, we have $\operatorname{im} w_i \cap \ker w_{i+1} = \{\vec{0}\}$, thus $\iota(\operatorname{im} w_i) \wedge \iota(\ker w_{i+1}) \ne \vec{0}$. On the other hand, for any $j < i$, since $\operatorname{im} w_i, \operatorname{im} w_j$ are in different SCCs and $\operatorname{im} w_i$ is reachable from $\operatorname{im} w_j$, the vertex $\operatorname{im} w_j$ is not reachable from $\operatorname{im} w_i$; therefore we have $\operatorname{im} w_i \cap \ker w_j \ne \{\vec{0}\}$, thus $\iota(\operatorname{im} w_i) \wedge \iota(\ker w_j) = \vec{0}$. It follows that $\iota(\ker w_{i+1}) \notin \langle \iota(\ker w_j) : j < i\rangle$. Indeed, if $\iota(\ker w_{i+1}) = \sum_{j<i} \lambda_j \iota(\ker w_j)$ for some $\lambda_1, \ldots, \lambda_{i-1}$ then, by linearity of the wedge product, $\iota(\operatorname{im} w_i) \wedge \iota(\ker w_{i+1}) = \sum_{j<i} \lambda_j (\iota(\operatorname{im} w_i) \wedge \iota(\ker w_j)) = \vec{0}$, a contradiction.

We show by induction on i that $\dim \langle \iota(\ker w_j) : j \in \{1, \ldots, i\}\rangle \ge i/2$ for all $i \in \{1, \ldots, k\}$. This is clear for $i = 1, 2$. For the induction step, we have $\dim \langle \iota(\ker w_j) : j \in \{1, \ldots, i+1\}\rangle \ge \dim \langle \iota(\ker w_{i+1}), \iota(\ker w_j) : j \in \{1, \ldots, i-1\}\rangle \ge 1 + (i-1)/2 = (i+1)/2$. Hence $k/2 \le \dim \langle \iota(\ker w_j) : j \in \{1, \ldots, k\}\rangle \le \dim \Lambda^{n-r}\mathbb{Q}^n = \binom{n}{r}$. ◀

▶ **Lemma 7.** *Let* $a_1 \cdots a_k \in \Sigma^*$ *be (the edge labels of) a shortest path in* G *from a vertex* $\operatorname{im} a_0$ *to* $\operatorname{im} a_k$. *Then* $k \le \binom{n}{r}$.

Proof. Let $i \in \{0, \ldots, k-2\}$. We have $\operatorname{im} a_i \cap \ker a_{i+1} = \{\vec{0}\}$, thus $\iota(\operatorname{im} a_i) \wedge \iota(\ker a_{i+1}) \ne \vec{0}$. On the other hand, for any $j > i+1$, since $a_{i+1} \cdots a_j$ is a shortest path from $\operatorname{im} a_i$ to $\operatorname{im} a_j$, there is no edge from $\operatorname{im} a_i$ to $\operatorname{im} a_j$; therefore we have $\operatorname{im} a_i \cap \ker a_j \ne \{\vec{0}\}$, thus $\iota(\operatorname{im} a_i) \wedge \iota(\ker a_j) = \vec{0}$. It follows that $\iota(\ker a_{i+1}) \notin \langle \iota(\ker a_j) : j > i+1\rangle$.

By induction it follows that $\dim \langle \iota(\ker a_j) : j \in \{i+1, \ldots, k\}\rangle \ge k - i$ holds for all $i \in \{0, \ldots, k-1\}$. Hence $k \le \dim \langle \iota(\ker a_j) : j \in \{1, \ldots, k\}\rangle \le \dim \Lambda^{n-r}\mathbb{Q}^n = \binom{n}{r}$. ◀

The next lemmas discuss *cycles* $w \in \Sigma^+$ in G, i.e., (the edge labels of) paths in G such that $\operatorname{im} w \cap \ker w = \{\vec{0}\}$. A cycle w is said to be *around* $\operatorname{im} w_0$ if $\operatorname{im} w = \operatorname{im} w_0$. The following lemma says, loosely speaking, that cycles around a single vertex "generate a group".

▶ **Lemma 8.** *Let $w_0 \in \Sigma^+$ with $\operatorname{rk} w_0 = r$, and let $P \in \mathbb{Q}^{r \times n}$ be a matrix with $\operatorname{im} P = \operatorname{im} w_0$. Then for every cycle $w \in \Sigma^+$ around $\operatorname{im} w_0$ there exists a unique invertible matrix $M'(w) \in GL(r, \mathbb{Q})$ such that $PM(w) = M'(w)P$. Moreover, for any nonempty set $C \subseteq \Sigma^+$ of cycles around $\operatorname{im} w_0$, $M'(C^+)$ is a finite subgroup of $GL(r, \mathbb{Q})$.*

Proof. Let $w \in \Sigma^+$ be a cycle around $\operatorname{im} w_0$. Since $\operatorname{im} P \cap \ker(M(w)) = \{\vec{0}\}$, it follows that $\operatorname{im}(PM(w)) = \operatorname{im} w = \operatorname{im} P$. So the rows of $PM(w)$ are linear combinations of rows of P, and vice versa, hence there is a unique $M'(w) \in GL(r, \mathbb{Q})$ with $PM(w) = M'(w)P$.

Let $C \subseteq \Sigma^+$ be a nonempty set of cycles around $\operatorname{im} w_0$. For any $w_1, w_2 \in C$ we have $M'(w_1 w_2)P = PM(w_1 w_2) = PM(w_1)M(w_2) = M'(w_1)PM(w_2) = M'(w_1)M'(w_2)P$, and since the rows of P are linearly independent, it follows that $M'(w_1 w_2) = M'(w_1)M'(w_2)$. Thus, $M'(C^+)$ is a semigroup.

Towards a contradiction, suppose $M'(C^+)$ were infinite. Since the rows of P are linearly independent, it follows that $M'(C^+)P$ is infinite, thus $PM(C^+)$ is infinite. Since $\operatorname{im} w_0 = \operatorname{im} P$, there is a matrix $B \in \mathbb{Q}^{n \times r}$ with $M(w_0) = BP$. Since the columns of B are linearly independent, the set $BPM(C^+)$ is infinite. But this set equals $M(w_0 C^+)$, contradicting the finiteness of $M(\Sigma^*)$. Thus the semigroup $M'(C^+)$ is finite. As $M'(C^+) \subseteq GL(r, \mathbb{Q})$, it follows that $M'(C^+)$ is a finite group. ◀

The following lemma allows us, loosely speaking, to limit the number of cycles in a word.

▶ **Lemma 9.** *Let $w_0, w_1, \ldots, w_k \in \Sigma^+$ such that $w_1, \ldots, w_k$ are cycles around $\operatorname{im} w_0$. Then there exist $\ell \leq g(n) - 1$ and $\{u_1, \ldots, u_\ell\} \subseteq \{w_1, \ldots, w_k\}$ such that $M(w_0 w_1 \cdots w_k) = M(w_0 u_1 \cdots u_\ell)$.*

Proof. We can assume $k \geq 1$. Let $C = \{w_1, \ldots, w_k\}$. Let P and $M'(w)$ for $w \in C$ as in Lemma 8. By Lemma 8, the set $M'(C^+)$ is a finite subgroup of $GL(r, \mathbb{Q})$, so we have $|M'(C^+)| \leq g(r) \leq g(n)$. By Remark 4, there are $\ell \leq g(n) - 1$ and $u_1, \ldots, u_\ell \in C$ such that $M'(w_1) \cdots M'(w_k) = M'(u_1) \cdots M'(u_\ell)$. Since $\operatorname{im} w_0 = \operatorname{im} P$, there is a matrix $B \in \mathbb{Q}^{n \times r}$ with $M(w_0) = BP$. Hence we have $M(w_0 w_1 \cdots w_k) = BPM(w_1) \cdots M(w_k) = BM'(w_1) \cdots M'(w_k)P = BM'(u_1) \cdots M'(u_\ell)P = BPM(u_1) \cdots M(u_\ell) = M(w_0 u_1 \cdots u_\ell)$. ◀

The following lemma allows us to add cycles to a word.

▶ **Lemma 10.** *Let $w \in \Sigma^+$ be a cycle in G. Then there exists $\rho(w) \in \mathbb{N} \setminus \{0\}$ such that $M(w_0) = M(w_0 w^{\rho(w)})$ holds for all $w_0 \in \Sigma^+$ with $\operatorname{im} w_0 = \operatorname{im} w$.*

Proof. Let $P \in \mathbb{Q}^{r \times n}$ be a matrix with $\operatorname{im} P = \operatorname{im} w$. By Lemma 8, there exists $M'(w) \in GL(r, \mathbb{Q})$ such that $PM(w) = M'(w)P$ and $\{M'(w)^i : i \in \mathbb{N}\}$ is a finite group. Define $\rho(w)$ to be the order of this group, i.e., $M'(w)^{\rho(w)} = I_r$. Let $w_0 \in \Sigma^+$ with $\operatorname{im} w_0 = \operatorname{im} w$. Since $\operatorname{im} w_0 = \operatorname{im} P$, there is a matrix $B \in \mathbb{Q}^{n \times r}$ with $M(w_0) = BP$. Hence $M(w_0) = BP = BI_r P = BM'(w)^{\rho(w)}P = BPM(w)^{\rho(w)} = M(w_0)M(w)^{\rho(w)} = M(w_0 w^{\rho(w)})$. ◀

The following lemma allows us to limit the length of paths within an SCC.

▶ **Lemma 11.** *Let $a \in \Sigma$, and let $w \in \Sigma^*$ be a path in G from $\operatorname{im} a$ such that $\operatorname{im} a$ and $\operatorname{im} w$ are in the same SCC. Then there exists $u \in \Sigma^*$ with $M(aw) = M(au)$ and*

$$|u| \;\leq\; 2^{n+2} g(n) - 2 \;\in\; 2^{O(n \log n)}\,.$$

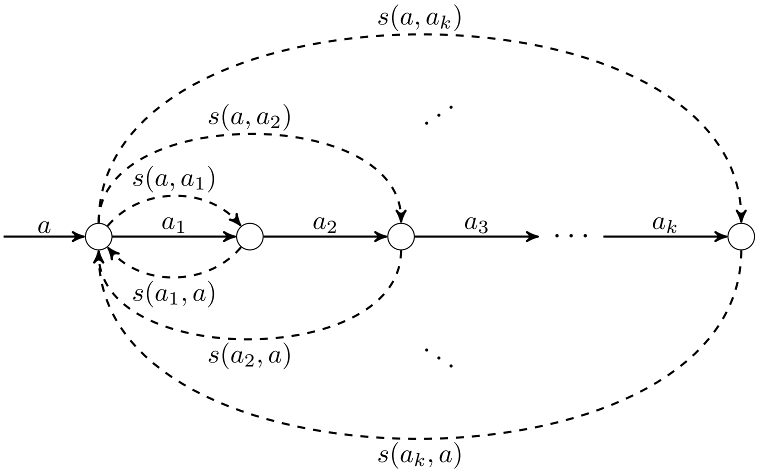


Figure 1 Illustration of the paths w and w' in Lemma 11. Edges are depicted as solid arrows, paths as dashed arrows.

Proof. For any $b_1, b_2 \in \Sigma$ such that $\operatorname{im} b_1, \operatorname{im} b_2$ are in the SCC of $\operatorname{im} a$, let $s(b_1, b_2) \in \Sigma^*$ be a shortest path from $\operatorname{im} b_1$ to $\operatorname{im} b_2$. By Lemma 7, we have $|s(b_1, b_2)| \leq \binom{n}{r}$.

Suppose $w = a_1 \cdots a_k$ for $a_i \in \Sigma$. For $i \in \{1, \ldots, k\}$ define the cycle $w_i := s(a_i, a)s(a, a_i)$ around $\operatorname{im} a_i$. By Lemma 10, we have $M(aw) = M(aw')$ for

$$w' := a_1 w_1^{\rho(w_1)} a_2 w_2^{\rho(w_2)} \cdots a_k w_k^{\rho(w_k)} .$$

For $i \in \{1, \ldots, k\}$ also define the cycle $v_i := s(a, a_i)s(a_i, a)$ around $\operatorname{im} a$. Then we have:

$$w' = a_1 s(a_1, a) v_1^{\rho(w_1)-1} s(a, a_1) a_2 s(a_2, a) v_2^{\rho(w_2)-1} s(a, a_2) \cdots a_k s(a_k, a) v_k^{\rho(w_k)-1} s(a, a_k)$$

Figure 1 illustrates the paths w and w'. Define a set of cycles $C \subseteq \Sigma^*$ around $\operatorname{im} a$ by

$$C := \{a_1 s(a_1, a), v_1, s(a, a_1) a_2 s(a_2, a), v_2, \ldots, s(a, a_{k-1}) a_k s(a_k, a), v_k\} .$$

Since $w' \in C^* s(a, a_k)$, by Lemma 9, there exist $\ell \leq g(n) - 1$ and $u_1, \ldots, u_\ell \in C$ such that $M(aw) = M(aw') = M(au_1 u_2 \cdots u_\ell s(a, a_k))$. For all $v \in C$ we have $|v| \leq 2\binom{n}{r} + 1 \leq 2^{n+2}$, and $|s(a, a_k)| \leq \binom{n}{r} \leq 2^n$. Hence the lemma holds for $u := u_1 u_2 \cdots u_\ell s(a, a_k)$, as $|u| \leq 2^{n+2}(g(n) - 1) + 2^n \leq 2^{n+2} g(n) - 2$. ◀

We are ready to prove Proposition 5.

Proof of Proposition 5. Decompose the word w into $w = a_1 w_1 a_2 w_2 \cdots a_k w_k$ for $a_i \in \Sigma$ so that for all $i \in \{1, \ldots, k\}$ the vertices $\operatorname{im} a_i, \operatorname{im} w_i$ are in the same SCC, and for all $i \in \{1, \ldots, k-1\}$ the vertices $\operatorname{im} w_i, \operatorname{im} a_{i+1}$ are in different SCCs. By Lemma 6, we have $k \leq 2\binom{n}{r} \leq 2^{n+1}$. For all $i \in \{1, \ldots, k\}$, by Lemma 11, there is $u_i \in \Sigma^*$ with $|u_i| \leq 2^{n+2} g(n) - 2$ such that $M(a_i w_i) = M(a_i u_i)$. Hence the proposition holds for $u := a_1 u_1 a_2 u_2 \cdots a_k u_k$, as $|u| \leq 2^{n+1}(2^{n+2} g(n) - 2 + 1) \leq 2^{2n+3} - 1$. ◀

3.2 The General Case

In this subsection we prove Theorem 3. For $r \in \{0, \ldots, n\}$ let $d_r \in \mathbb{N}$ be the smallest number such that for any $w \in \Sigma^*$ with $\operatorname{rk} w \geq r$ there is $u \in \Sigma^*$ with $M(w) = M(u)$ and $|u| \leq d_r$. Also write h for the bound from Proposition 5.

▶ **Proposition 12.** *For any $r \in \{0, \ldots, n-1\}$ we have $d_r \leq d_{r+1} + (d_{r+1} + 1)h$.*

Proof. Let $w \in \Sigma^*$ with $\operatorname{rk} w \geq r$. We need to show that there is $u \in \Sigma^*$ with $M(w) = M(u)$ and $|u| \leq d_{r+1} + (d_{r+1} + 1)h$. Decompose w into $w = w_0 a_1 w_1 a_2 w_2 \cdots a_k w_k$ for $a_i \in \Sigma$ such that $\operatorname{rk} w_0 > r$ and for all $i \in \{1, \ldots, k\}$ we have $\operatorname{rk}(a_i w_i) = r$ and $\operatorname{rk} w_i > r$. (This decomposition is unique; in particular, $a_k w_k$ is the shortest suffix of w with rank r.) By the definition of d_{r+1}, for all $i \in \{0, \ldots, k\}$ there exists $u_i \in \Sigma^*$ with $M(w_i) = M(u_i)$ and $|u_i| \leq d_{r+1}$. Then $M(w) = M(u_0 a_1 u_1 a_2 u_2 \cdots a_k u_k)$.

Define a new alphabet Σ_r and a monoid morphism $M_r : \Sigma_r^* \to \mathbb{Q}^{n \times n}$ with $M_r(\Sigma_r) = \{M(a_i u_i) : i \in \{1, \ldots, k\}\}$, and note that $\operatorname{rk} M_r(b) = r$ for all $b \in \Sigma_r$. Then there is a word $y \in \Sigma_r^*$ such that $M_r(y) = M(a_1 u_1 \cdots a_k u_k)$. By Proposition 5, there is $x \in \Sigma_r^*$ with $M_r(y) = M_r(x)$ and $|x| \leq h$. Obtain the word $v \in \Sigma^*$ from x by replacing each letter $b \in \Sigma_r$ in x by $a_i u_i$ for $i \in \{1, \ldots, k\}$ such that $M_r(b) = M(a_i u_i)$. Then $M_r(x) = M(v)$, and thus $M(w) = M(u_0 a_1 u_1 \cdots a_k u_k) = M(u_0) M_r(y) = M(u_0) M_r(x) = M(u_0) M(v) = M(u_0 v)$, where $|u_0 v| = |u_0| + |v| \leq d_{r+1} + (d_{r+1} + 1)|x| \leq d_{r+1} + (d_{r+1} + 1)h$. ◀

We can now prove our main result.

Proof of Theorem 3. We prove by induction that for all $r \in \{0, \ldots, n\}$ we have $d_r \leq (h+1)^{n-r} d_n + (h+1)^{n-r} - 1$. For the base case, $r = n$, this is trivial. For the step, let $r < n$. We have:

$$\begin{aligned} d_r &\leq h + (h+1) d_{r+1} && \text{(Proposition 12)} \\ &\leq h + (h+1)\big((h+1)^{n-r-1} d_n + (h+1)^{n-r-1} - 1\big) && \text{(induction hypothesis)} \\ &= h + (h+1)^{n-r} d_n + (h+1)^{n-r} - h - 1 \end{aligned}$$

This completes the induction proof. Hence $d_0 \leq (h+1)^n (d_n + 1) = 2^{n(2n+3)} g(n)^n (d_n + 1)$. The rank-$n$ matrices in $M(\Sigma)$ generate a finite subgroup of $GL(n, \mathbb{Q})$. So it follows by Remark 4 that $d_n + 1 \leq g(n)$. Thus $d_0 \leq 2^{n(2n+3)} g(n)^{n+1}$. ◀

4 Algorithmic Applications

Theorem 1 gives an exponential-space algorithm for deciding finiteness of a finitely generated rational matrix semigroup. In fact, the following theorem shows that deciding finiteness is in the second level of the weak EXP hierarchy (see e.g. [16] for a definition).

▶ **Theorem 13.** *Given be a finite set $\mathcal{M} \subseteq \mathbb{Q}^{n \times n}$ of rational matrices, the problem of deciding finiteness of the generated semigroup $\overline{\mathcal{M}}$ is in* coNEXP$^{\text{NP}}$.

Proof. For a NEXP$^{\text{NP}}$ algorithm deciding infiniteness, non-deterministically guess in exponential time some $M = M_1 \cdots M_\ell$, $M_i \in \mathcal{M}$, with $\ell = 2^{n(2n+3)} g(n)^{n+1} + 1$ as a witness for infiniteness. Then, using a call to an NP oracle, check whether there are $M_1', \ldots, M_r' \in \mathcal{M}$ such that $M = M_1' \cdots M_r'$ for some $0 \leq r < \ell$. If the call is successful then reject, otherwise accept.

Correctness of the algorithm immediately follows from Theorem 1: if $\overline{\mathcal{M}}$ is finite, then the $M_1', \ldots, M_r' \in \mathcal{M}$ such that $M = M_1' \cdots M_r'$ are guaranteed to exist. ◀

This is the first improvement of the non-elementary algorithm of Mandel and Simon [25].

Another immediate consequence of Theorem 1 is an upper bound on the complexity of the membership problem for finite matrix semigroups:

▶ **Theorem 14.** *Given a finite set of rational matrices $\mathcal{M} \subseteq \mathbb{Q}^{n\times n}$ such that $\overline{\mathcal{M}}$ is finite and $A \in \mathbb{Q}^{n\times n}$, the problem of deciding whether $A \in \overline{\mathcal{M}}$ is in* NEXP*.*

In the remainder of this section, we discuss implications of Theorem 13 to decision problems in automata theory.

4.1 Weighted Automata

The motivation for Mandel and Simon to study the finiteness problem originated from investigating the decidability of the finiteness problem (originally called boundedness problem in [25]) for weighted automata. A *weighted automaton over* $\mathbb{Q}$ is a quintuple $\mathcal{A} = (n, \Sigma, M, \alpha, \eta)$ where $n \in \mathbb{N}$ is the number of states, Σ is the finite alphabet, $M : \Sigma \to \mathbb{Q}^{n\times n}$ maps letters to transition matrices, $\alpha \in \mathbb{Q}^n$ is the initial state vector, and $\eta \in \mathbb{Q}^n$ is the final state vector. We extend M to the monoid morphism $M : \Sigma^* \to \mathbb{Q}^{n\times n}$ as before. Such an automaton defines a function $|\mathcal{A}| : \Sigma^* \to \mathbb{Q}$ by defining $|\mathcal{A}|(w) = \alpha M(w)\eta^T$, where the superscript T denotes transpose. We say, $\mathcal{A}$ is *finite* if the image of $|\mathcal{A}|$ is finite, i.e., if $|\mathcal{A}|(\Sigma^*) \subseteq \mathbb{Q}$ is a finite set. The *finiteness problem* asks whether a given automaton is finite.

It is clear that if $M(\Sigma^*)$ is finite then $\mathcal{A}$ is finite. The converse is not generally true: e.g., any automaton $\mathcal{A}$ whose initial state vector is the zero vector satisfies $|\mathcal{A}|(\Sigma^*) = \{0\}$, hence is finite, regardless of $M(\Sigma^*)$. However, it is argued in the proof of Corollary 5.4 in [25] that, given an automaton $\mathcal{A}$, one can compute, in exponential time, a polynomial-size automaton $\mathcal{B}$ with monoid morphism $M_{\mathcal{B}}$ such that (i) $|\mathcal{A}| = |\mathcal{B}|$, and (ii) $\mathcal{A}$ (and hence $\mathcal{B}$) is finite if and only if $M_{\mathcal{B}}(\Sigma^*)$ is finite.[2] Mandel and Simon use this argument to show that the finiteness problem for weighted automata over $\mathbb{Q}$ is decidable. Theorem 13 then immediately gives:

▶ **Corollary 15.** *The finiteness problem for weighted automata over* $\mathbb{Q}$ *can be decided in* coNEXP$^{\mathrm{NP}}$.

4.2 Affine Integer Vector Addition Systems with States

We show that Theorem 1 together with Corollary 2 imply an upper bound for the reachability problem in affine integer vector addition systems with states with the finite monoid property (afmp-$\mathbb{Z}$-VASS) studied in [5]. An affine $\mathbb{Z}$-VASS in dimension $d \in \mathbb{N}$ is a tuple $\mathcal{V} = (d, Q, T)$ such that Q is a finite set of states and $T \subseteq Q \times \mathbb{Z}^{d\times d} \times \mathbb{Z}^d \times Q$ is a finite transition relation. Setting $\mathcal{M} := \{A \in \mathbb{Z}^{d\times d} : (q, A, \vec{b}, r) \in T\}$, in afmp-$\mathbb{Z}$-VASS we additionally require that $\overline{\mathcal{M}}$ is finite. A configuration of $\mathcal{V}$ is a tuple $(q, \vec{v}) \in Q \times \mathbb{Z}^d$ which we write as $q(\vec{v})$. We define the step relation $\to \subseteq (Q \times \mathbb{Z}^d)^2$ such that $q(\vec{v}) \to r(\vec{w})$ if and only if there is a transition $(q, A, \vec{b}, r) \in T$ such that $\vec{w} = A \cdot \vec{v} + \vec{b}$. Moreover, we denote by $\to^*$ the reflexive transitive closure of $\to$. For a configuration $q(\vec{v})$, we define the reachability set of $q(\vec{v})$ as $\mathcal{R}(q(\vec{v})) := \{r(\vec{w}) : q(\vec{v}) \to^* r(\vec{w})\}$. Given configurations $q(\vec{v})$ and $r(\vec{w})$, reachability is the problem of deciding whether $r(\vec{w}) \in \mathcal{R}(q(\vec{v}))$. Note that $\mathcal{R}(q(\vec{v}))$ is in general infinite despite $\overline{\mathcal{M}}$ being finite.

[2] We remark that this automaton $\mathcal{B}$ has the minimal number of states among the automata defining the function $|\mathcal{A}|$. This minimal automaton goes back to [29] and has been further studied in, e.g., [7, 10].

The reachability problem for afmp-$\mathbb{Z}$-VASS was shown decidable in [5] by a reduction to reachability in $\mathbb{Z}$-VASS. A $\mathbb{Z}$-VASS is an afmp-$\mathbb{Z}$-VASS in which every transition is of the form $(q, I_d, \vec{b}, r)$. The reachability problem for $\mathbb{Z}$-VASS is known to be NP-complete, see e.g. [15]. The size of the $\mathbb{Z}$-VASS obtained in the reduction given in [5] grows in $|\overline{\mathcal{M}}|$ and hence leads to a non-elementary upper bound for reachability in afmp-$\mathbb{Z}$-VASS assuming Mandel and Simon's bound. The results of this paper enable us to significantly improve this upper bound.

▶ **Corollary 16.** *The reachability problem for afmp-$\mathbb{Z}$-VASS can be decided in* EXPSPACE.

Proof. Let $\mathcal{V} = (d, Q, T)$ be an afmp-$\mathbb{Z}$-VASS and let $\mathcal{M}$ be defined as above. Set $||\mathcal{M}|| := |\overline{\mathcal{M}}| \cdot d^2 \cdot \max\{\log(||A|| + 1) : A \in \overline{\mathcal{M}}\}$, where $||A||$ is the largest absolute value of all entries of A. Since $||A_1 \cdots A_n|| \le d^n \cdot ||A_1|| \cdots ||A_n||$ for all $A_1, \ldots, A_n \in \mathbb{Z}^{d\times d}$ and $n \in \mathbb{N}$, by Theorem 1 and Corollary 2 we have

$$||\mathcal{M}|| \le |T|^{2^{O(d^2\cdot\log d)}} \cdot d^2 \cdot 2^{d(2d+3)} g(d)^{d+1} \cdot (\log d + ||T||) \le ||T||^{2^{O(d^2\cdot\log d)}},$$

where $||T|| := \sum_{(q,A,\vec{b},r)\in T} d^2 \cdot \lceil \log ||A|| + \log ||\vec{b}|| + 1 \rceil$. It can be deduced from the proof of [5, Thm. 7] that reachability in $\mathcal{V}$ can be decided in non-deterministic space that is polynomially bounded in the encoding of $\mathcal{V}$ and poly-logarithmically in $||\mathcal{M}||$, from which the desired exponential space upper bound follows. ◀

5 Conclusion

The main result of this paper has been to show that any element in the finite multiplicative semigroup $\overline{\mathcal{M}}$ generated by a finite set $\mathcal{M}$ of m rational $n \times n$ matrices can be obtained as a product of generators of length at most $2^{O(n^2 \log n)}$. This length bound immediately gives that $|\overline{\mathcal{M}}|$ is bounded by $m^{2^{O(n^2 \log n)}}$.

There remain two immediate questions that we did not answer in this article. The first is whether the order of growth of $|\overline{\mathcal{M}}|$ we obtained is tight. If $\overline{\mathcal{M}}$ is a group its order can be bounded by $2^n n!$ for almost all n, and this bound is attained by the group of signed permutation matrices. In contrast, in the semigroup case $|\overline{\mathcal{M}}|$ also depends on m. We conjecture that our doubly exponential upper bound is not optimal and that it is possible to establish an exponential upper bound of $|\overline{\mathcal{M}}|$ in terms of m and n. The second open question concerns the precise complexity of deciding finiteness of matrix semigroups. We have been unable to establish any non-trivial lower bounds on this problem and conjecture that our coNEXP$^{\text{NP}}$ upper bound can significantly be improved, possibly by adapting techniques of Babai et al. [2].

References

1 J. Almeida and B. Steinberg. Matrix mortality and the Černý-Pin conjecture. In *Proceedings of Developments in Language Theory (DLT)*, pages 67–80, 2009.

2 L. Babai, R. Beals, and D. Rockmore. Deciding finiteness of matrix groups in deterministic polynomial time. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation (ISSAC)*, pages 117–126, 1993.

3 N. Berry, A. Dubickas, N. D. Elkies, B. Poonen, and C. Smyth. The conjugate dimension of algebraic numbers. *The Quarterly Journal of Mathematics*, 55(3):237–252, 2004.

4 J. Berstel and C. Reutenauer. *Rational Series and Their Languages*, volume 12 of *Monographs in Theoretical Computer Science. An EATCS Series.* Springer-Verlag, 1988.

5 M. Blondin, C. Haase, and F. Mazowiecki. Affine extensions of integer vector addition systems with states. In *Proceedings of the International Conference on Concurrency Theory (CONCUR)*, pages 14:1–14:17, 2018.

6 W. Burnside. On an unsettled question in the theory of discontinuous groups. *The Quarterly Journal of Pure and Applied Mathematics*, 33:230–238, 1902.

7 J.W. Carlyle and A. Paz. Realizations by stochastic finite automata. *Journal of Computer and System Sciences*, 5(1):26–40, 1971.

8 A. de Luca and S. Varricchio. *Finiteness and Regularity in Semigroups and Formal Languages.* Monographs in Theoretical Computer Science. An EATCS Series. Springer-Verlag, 1999.

9 W. Feit. The orders of finite linear groups. Unpublished preprint.

10 M. Fliess. Matrices de Hankel. *Journal de Mathématiques Pures et Appliquées*, 53:197–222, 1974.

11 A. Freedman, R. N. Gupta, and R. M. Guralnick. Shirshov's theorem and representations of semigroups. *Pacific Journal of Mathematics*, 181(3):159–176, 1997.

12 S. Friedland. The maximal orders of finite subgroups in $GL_n(\mathbb{Q})$. *Proceedings of the American Mathematical Society*, 125(12):3519–3526, 1997.

13 E. S. Golod. On nil-algebras and finitely approximable p-groups. *Izv. Akad. Nauk SSSR Ser. Mat.*, 28:273–276, 1964.

14 E. S. Golod and I. R. Shafarevich. On the class field tower. *Izv. Akad. Nauk SSSR Ser. Mat.*, 28:261–272, 1964.

15 C. Haase and S. Halfon. Integer vector addition systems with states. In *Proceedings of Reachability Problems (RP)*, pages 112–124, 2014.

16 L. A. Hemachandra. The strong exponential hierarchy collapses. *Journal of Computer and System Sciences*, 39(3):299–322, 1989. doi:10.1016/0022-0000(89)90025-1.

17 E. Hrushovski, J. Ouaknine, A. Pouly, and J. Worrell. Polynomial invariants for affine programs. In *Proceedings of the Symposium on Logic in Computer Science (LICS)*, pages 530–539, 2018.

18 G. Jacob. Un algorithme calculant le cardinal, fini ou infini, des demi-groupes de matrices. *Theoretical Computer Science*, 5(2):183–204, 1977.

19 R. Kannan and R. J. Lipton. The orbit problem is decidable. In *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 252–261, 1980.

20 R. Kannan and R. J. Lipton. Polynomial-time algorithm for the orbit problem. *Journal of the ACM*, 33(4):808–821, 1986.

21 I. Kaplansky. *Fields and Rings.* University of Chicago Press, second edition, 1972.

22 S. Kiefer and C. Mascle. On finite monoids over nonnegative integer matrices and short killing words. In *Proceedings of the International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 43:1–43:13, 2019.

23 J. Kuzmanovich and A. Pavlichenkov. Finite groups of matrices whose entries are integers. *The American Mathematical Monthly*, 109(2):173–186, 2002.

24 G. Lallement. *Semigroups and combinatorial applications.* John Wiley & Sons, 1979.

25 A. Mandel and I. Simon. On finite semigroups of matrices. *Theoretical Computer Science*, 5(2):101–111, 1977.

26 R. McNaughton and Y. Zalcstein. The Burnside problem for semigroups. *Journal of Algebra*, 34:292–299, 1975.

27 M. Newman. *Integral Matrices.* Academic Press, 1972.

28 I. Schur. Über Gruppen periodischer Substitutionen. In *Sitzungsbericht Preuss. Akad. Wiss.*, pages 619–627, 1911.

29 M. P. Schützenberger. On the definition of a family of automata. *Information and Control*, 4(2–3):245–270, 1961.

30 B. Steinberg. Yet another solution to the Burnside problem for matrix semigroups. *Canadian Mathematical Bulletin*, 55(1):188–192, 2012.

31 H. Straubing. The Burnside problem for semigroups of matrices. In L. J. Cummings, editor, *Combinatorics on Words*, pages 279–295. Academic Press, 1983.

32 A. Weber and H. Seidl. On finitely generated monoids of matrices with entries in ℕ. *Informatique théorique et Applications/Theoretical Informaties and Applications*, 25:19–38, 1991.

33 B. Weisfeiler. On the size and structure of finite linear groups. Unpublished preprint. `arXiv:1203.1960`.

34 B. Weisfeiler. Post-classification version of Jordan's theorem on finite linear groups. *Proceedings of the National Academy of Sciences of the United States of America*, 81:5278–5279, 1984.

Rational Subsets of Baumslag-Solitar Groups

Michaël Cadilhac
DePaul University, Chicago, IL, USA
michael@cadilhac.name

Dmitry Chistikov
Centre for Discrete Mathematics and its Applications (DIMAP) & Department of Computer Science, University of Warwick, Coventry, UK
d.chistikov@warwick.ac.uk

Georg Zetzsche
Max Planck Institute for Software Systems (MPI-SWS), Kaiserslautern, Germany
georg@mpi-sws.org

Abstract

We consider the rational subset membership problem for Baumslag-Solitar groups. These groups form a prominent class in the area of algorithmic group theory, and they were recently identified as an obstacle for understanding the rational subsets of $\mathrm{GL}(2,\mathbb{Q})$.

We show that rational subset membership for Baumslag-Solitar groups $\mathrm{BS}(1,q)$ with $q \geq 2$ is decidable and PSPACE-complete. To this end, we introduce a word representation of the elements of $\mathrm{BS}(1,q)$: their pointed expansion (PE), an annotated q-ary expansion. Seeing subsets of $\mathrm{BS}(1,q)$ as word languages, this leads to a natural notion of PE-regular subsets of $\mathrm{BS}(1,q)$: these are the subsets of $\mathrm{BS}(1,q)$ whose sets of PE are regular languages. Our proof shows that every rational subset of $\mathrm{BS}(1,q)$ is PE-regular.

Since the class of PE-regular subsets of $\mathrm{BS}(1,q)$ is well-equipped with closure properties, we obtain further applications of these results. Our results imply that (i) emptiness of Boolean combinations of rational subsets is decidable, (ii) membership to each fixed rational subset of $\mathrm{BS}(1,q)$ is decidable in logarithmic space, and (iii) it is decidable whether a given rational subset is recognizable. In particular, it is decidable whether a given finitely generated subgroup of $\mathrm{BS}(1,q)$ has finite index.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness; Theory of computation → Formal languages and automata theory; Theory of computation → Models of computation

Keywords and phrases Rational subsets, Baumslag-Solitar groups, decidability, regular languages, pointed expansion

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.116

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

Related Version `https://arxiv.org/abs/2006.11898`

Acknowledgements The research for this work was carried out in part at the Autobóz Research Camp in 2019 in Firbush, Scotland. The authors are very grateful to Piotr Hofman for comments that led to important insights and to Michael Blondin as well as to all other participants for discussions.

1 Introduction

Subsets of groups. Regular languages are an extremely versatile tool in algorithmics on sets of finite words. This is mainly due to two reasons. First, they are robust in terms of representations and closure properties: They can be described by finite automata, by recognizing morphisms, and by monadic second-order logic and they are closed under Boolean and an abundance of other operations. Second, many properties (such as emptiness) are easily decidable using finite automata.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 116; pp. 116:1–116:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Given this success, there have been several attempts to develop an analogous notion for subsets of (infinite, finitely generated) groups. Adapting the notion of recognizing morphism yields *recognizable subsets* of a group G. They are closed under Boolean operations, and problems such as membership or emptiness are decidable. However, since they are merely unions of cosets of finite-index normal subgroups, their expressiveness is severely limited.

Another notion is that of *rational subsets*, which transfer (non-deterministic) finite automata to groups. Starting with pioneering work by Benois [6] in 1969, they have matured into an important tool in group theory. Rational subsets are quite expressive: They include finitely generated submonoids and are closed under (finite) union, pointwise product, and Kleene star. Moreover, they have been applied successfully to solving equations in groups [11, 9], as well as in other settings [2, 34].

The high expressiveness of rational subsets comes at the cost of undecidability of decision problems for many groups. The most fundamental one is the *membership problem* for rational subsets: Given a rational subset R of a group G and an element $g \in G$, does g belong to R? Understanding for which groups this problem is decidable received significant attention over the last two decades, see [24] for a survey. Unfortunately, the rational subsets do not quite reach the level of robustness of regular languages. In general, the class of rational subsets of a group is not closed under Boolean operations, and the papers [25, 4] study for which groups the rational subsets form a Boolean algebra.

Baumslag-Solitar groups. A prominent class of groups is that of *Baumslag-Solitar groups* $\mathrm{BS}(p,q)$. For each $p, q \in \mathbb{N}$, the group is defined as $\mathrm{BS}(p,q) = \langle a, t \mid ta^pt^{-1} = a^q\rangle$. They were introduced in 1962 by Baumslag and Solitar to provide an example of a two-generator one-relator group that is non-Hopfian. They recently came into focus from the algorithmic perspective in a paper by Kharlampovich, López, and Miasnikov [22], which shows that solvability of equations is decidable in $\mathrm{BS}(1,q)$. They have also been studied from several other perspectives, such as the decidability and complexity of the word problem [28, 14, 35], the conjugacy problem [14, 35], tiling problems [1], and computing normal forms [13, 18, 17].

More specifically to our setting, the Baumslag-Solitar groups have recently been identified by Diekert, Potapov, and Semukhin [15] as a stumbling block in solving rational subset membership in the group $\mathrm{GL}(2,\mathbb{Q})$, that is, the group of invertible 2×2 matrices over $\mathbb{Q}$. They show that any subgroup of $\mathrm{GL}(2,\mathbb{Q})$ containing $\mathrm{GL}(2,\mathbb{Z})$ is either of the form $\mathrm{GL}(2,\mathbb{Z}) \times \mathbb{Z}^k$ for $k \geq 1$ or contains $\mathrm{BS}(1,q)$ as a subgroup for some $q \geq 2$. Rational subset membership for $\mathrm{GL}(2,\mathbb{Z}) \times \mathbb{Z}^k$ is today a matter of standard arguments [24], because $\mathrm{GL}(2,\mathbb{Z})$ is virtually free. Therefore, making significant progress towards decidability in larger subgroups requires understanding rational subsets of $\mathrm{BS}(1,q)$.

One can represent the elements of $\mathrm{BS}(1,q)$ as pairs (r,m), where r is a number in $\mathbb{Z}[\frac{1}{q}]$, say $r = \pm\sum_{i=-n}^{n} a_iq^i$ for $a_{-n}, a_{-n+1}, \ldots, a_n \in \{0,\ldots,q-1\}$,[1] and $m \in \mathbb{Z}$. Here, one can think of m as a *cursor* pointing to a position in the q-ary expansion $a_nq^n + \cdots + a_{-n}q^{-n}$. Then the action of the generators of $\mathrm{BS}(1,q)$ is as follows. Multiplication by t or t^{-1} moves the cursor to the left or the right, respectively. Multiplication by a adds q^m; likewise, multiplication by a^{-1} subtracts q^m. Thus, from an automata-theoretic perspective, one can view the rational subset membership problem as the reachability problem for an extended version of one-counter automata. Instead of storing a natural number, such an automaton stores a number $r \in \mathbb{Z}[\frac{1}{q}]$. Moreover, instead of instructions "increment by 1" and "decrement

[1] $\mathbb{Z}[\frac{1}{q}]$ denotes (the additive group of) the smallest subring of $(\mathbb{Q},+,\cdot)$ containing $\mathbb{Z}$ and $1/q$; as a set, it consists of all rational numbers of the form $n \cdot q^j$, $n, j \in \mathbb{Z}$.

by 1”, it has an additional $\mathbb{Z}$-counter m that determines the value to be added in the next update. Then, performing “increment” on r will add q^m and “decrement” on r will subtract q^m. The $\mathbb{Z}$-counter m supports the classical “increment” and “decrement” instructions.

Contribution. Our *first main contribution* is to show is that for each group $\mathrm{BS}(1, q)$, the rational subset membership problem is decidable and PSPACE-complete. To this end, we show that each rational subset can be represented by a regular language of finite words that encode elements of $\mathrm{BS}(1, q)$ in the natural way: For (r, m) as above, we encode each digit a_i by a letter; and we decorate the digits at position 0 and at position m. We call this encoding the *pointed expansion* (PE) of (r, m). This leads to a natural notion of subsets of $\mathrm{BS}(1, q)$, which we call *PE-regular*. We regard the introduction of this notion as the *second main contribution* of this work.

The class of PE-regular subsets of $\mathrm{BS}(1, q)$ has several properties that make them a promising tool for decision procedures for $\mathrm{BS}(1, q)$: First, our proof shows that it effectively includes the large class of rational subsets, in particular any finitely generated submonoid. Second, they form an effective Boolean algebra. Third, due to them being regular languages of words, they inherit many algorithmic tools from the setting of free monoids. We apply these properties to obtain *three applications of our main results.*

1. Membership in each fixed rational subset can be decided in logarithmic space.
2. We show that it is decidable whether a given PE-regular subset (and thus a given rational subset) is recognizable. Recognizability of rational subsets is rarely known to be decidable for groups: The only examples known to the authors are free groups, for which decidability was shown by Sénizergues [31] (and simplified by Silva [33]) and free abelian groups (this follows from [19, Theorem 3.1]). Since (i) finitely generated subgroups are rational subsets and (ii) a subgroup of any group G is recognizable if and only if it has finite index in G, our result implies that it is decidable whether a given finitely generated subgroup of $\mathrm{BS}(1, q)$ has finite index. Studying decidability of this finite index problem in groups was recently proposed by Kapovich [12, Section 4.3].
3. Our results imply that emptiness of Boolean combinations (hence inclusion, equality, etc.) of rational subsets is decidable. (We also show that the rational subsets of $\mathrm{BS}(1, q)$ are not closed under intersection.) This is a strong decidability property that already fails for groups as simple as $F_2 \times \mathbb{Z}$ (this follows from [20, Theorem 6.3]), where F_2 is the free group over two generators, and hence for $\mathrm{GL}(2, \mathbb{Z}) \times \mathbb{Z}^k$, $k \geq 1$.

Finally, we remark that since $\mathrm{BS}(1, q)$ is isomorphic to the group of all matrices $\left(\begin{smallmatrix} q^m & r \\ 0 & 1 \end{smallmatrix}\right)$ for $m \in \mathbb{Z}$ and $r \in \mathbb{Z}[\frac{1}{q}]$, our results can be interpreted as solving the rational subset membership problem for this subgroup of $\mathrm{GL}(2, \mathbb{Q})$.

Related work. It is well-known that membership in a given finitely generated subgroup, called the *generalized word problem* of $\mathrm{BS}(1, q)$, is decidable. This is due to a general result of Romanovskiĭ, who showed in [29] and [30] that solvable groups of derived length two have a decidable generalized word problem (it is an easy exercise to show that $\mathrm{BS}(1, q)$ is solvable of derived length two for each $q \in \mathbb{N}$).

Another restricted version of rational subset membership is the *knapsack problem*, which was introduced by Myasnikov, Nikolaev, and Ushakov [27]. Here, one is given group elements $g_1, \ldots, g_k, g$ and is asked whether there exist $x_1, \ldots, x_k \in \mathbb{N}$ with $g_1^{x_1} \cdots g_k^{x_k} = g$. A recent paper on the knapsack problem in Baumslag-Solitar groups by Dudkin and Treyer [16] left open whether the knapsack problem is decidable in $\mathrm{BS}(1, q)$ for $q \geq 2$. This was settled very recently in [26], where one expresses solvability of $g_1^{x_1} \cdots g_k^{x_k} = g$ in a variant of Büchi arithmetic. A slight extension of that proof yields a regular language as above for the set $S = \{g_1^{x_1} \cdots g_k^{x_k} \mid x_1, \ldots, x_k \in \mathbb{N}\}$. Note that each element g_i moves the cursor either to the left (i.e. increases m), to the right (i.e. decreases m), or not at all. Thus, in a product $g_1^{x_1} \cdots g_k^{x_k}$, the cursor direction is reversed at most $k-1$ times. The challenge of our translation from rational subsets to PE-regular subsets is to capture products where the cursor changes direction an unbounded number of times.

Finally, closely related to rational subsets, there is another approach to group-theoretic problems via automata: One can represent finitely generated subgroups of free groups using *Stallings graphs*. Due to the special setting of free groups, they behave in many ways similar to automata over words and are thus useful for decision procedures [21]. Stallings graphs have recently been extended to semidirect products of free groups and free abelian groups by Delgado [10]. However, this does not include products $\mathbb{Z}[\frac{1}{q}] \rtimes \mathbb{Z}$ and is restricted to subgroups.

2 Basic notions

Automata, rational subsets, and regular languages. Since we work with automata over finite words and over groups, we define automata over a general monoid M. A subset $S \subseteq M$ is *recognizable* if there is a finite monoid F and a morphism $\varphi\colon M \to F$ such that $S = \varphi^{-1}(\varphi(S))$. If M is a group, one can equivalently require F to be a finite group.

For a subset $S \subseteq M$, we write $\langle S \rangle$ or S^* for the submonoid *generated by* S, i.e. the set of elements that can be written as a (possibly empty) product of elements of S. In particular, the neutral element $1 \in M$ always belongs to $\langle S \rangle = S^*$. A *generating set* is a subset $\Sigma \subseteq M$ such that $M = \langle \Sigma \rangle$. We say that M is *finitely generated (f.g.)* if it has a finite generating set. Suppose M is finitely generated and fix a finite generating set Σ. An *automaton over* M is a tuple $\mathcal{A} = (Q, \Sigma, E, q_0, q_f)$, where Q is a finite set of *states*, $E \subseteq Q \times \Sigma \times Q$ is a finite set of *edges*, $q_0 \in Q$ is its *initial state*, and $q_f \in Q$ is its *final state*. A *run (in $\mathcal{A}$)* is a sequence $\rho = (p_0, a_1, p_1) \cdots (p_{m-1}, a_m, p_m)$, where $(p_{i-1}, a_i, p_i) \in E$ for $i \in [1, m]$. It is *accepting* if $p_0 = q_0$ and $p_m = q_f$. By $[\rho]$, we denote the *production* of ρ, that is, the element $a_1 \cdots a_m \in M$. Two runs are *equivalent* if they start in the same state, end in the same state, and have the same production. For a set of runs P, we denote $[P] = \{[\rho] \mid \rho \in P\}$.

The subset *accepted by* $\mathcal{A}$ is $\mathsf{L}(\mathcal{A}) = \{[\rho] \mid \rho \text{ is an accepting run in } \mathcal{A}\}$. A subset $R \subseteq M$ is called *rational* if it is accepted by some automaton over M. It is a standard fact that the family of rational subsets of M does not depend on the chosen generating set Σ. Rational subsets of a free monoid Γ^* for some alphabet Γ are also called *regular languages*. If $M = \Gamma^* \times \Delta^*$ for alphabets Γ, Δ, then rational subsets of M are also called *rational transductions*. If $T \subseteq \Gamma^* \times \Delta^*$ and $L \subseteq \Gamma^*$, then we set $TL = \{v \in \Delta^* \mid \exists u \in L\colon (u, v) \in T\}$. It is well-known that if $L \subseteq \Gamma^*$ is regular and $T \subseteq \Gamma^* \times \Delta^*$ is rational, then TL is regular as well [7].

Baumslag-Solitar groups. The *Baumslag-Solitar groups* are the groups $\mathrm{BS}(p, q)$ for $p, q \in \mathbb{N}$, where $\mathrm{BS}(p, q) = \langle a, t \mid ta^pt^{-1} = a^q \rangle$. They were introduced in 1962 by Baumslag and Solitar [3] to provide an example of a non-Hopfian group with two generators and one defining relation. In this paper, we focus on the case $p = 1$. In this case, there is a well-known isomorphism $\mathrm{BS}(1, q) \cong \mathbb{Z}[\frac{1}{q}] \rtimes \mathbb{Z}$ and we will identify the two groups. Here, $\mathbb{Z}[\frac{1}{q}]$ is the additive group of number nq^i with $n, i \in \mathbb{Z}$, and $\rtimes$ denotes semidirect product. Building this semidirect product requires us to specify an automorphism φ_m of $\mathbb{Z}[\frac{1}{q}]$ for each $m \in \mathbb{Z}$, which is given by $\varphi_m(nq^i) = q^m \cdot nq^i$.

For readers not familiar with semidirect products, we give an alternative self-contained definition of $\mathbb{Z}[\frac{1}{q}] \rtimes \mathbb{Z}$. The elements of this group are pairs (r, m), where $r \in \mathbb{Z}[\frac{1}{q}]$ and $m \in \mathbb{Z}$. The multiplication is defined as

$$(r, m)(r', m') = (r + q^m \cdot r', m + m').$$

We think of an element (r, m) as representing a number r in $\mathbb{Z}[\frac{1}{q}]$ together with a cursor m to a position in the q-ary expansion of r. Multiplying an element (r, m) by the pair $(1, 0)$ from the right means adding 1 at the position in r given by m, hence adding q^m to r and leaving the cursor unchanged: we have $(r, m)(1, 0) = (r + q^m, m)$. Multiplying by $(0, 1)$ moves the cursor one position to the left: $(r, m)(0, 1) = (r, m + 1)$. It is easy to see that $\mathbb{Z}[\frac{1}{q}] \rtimes \mathbb{Z}$ is generated by the set $\{(1, 0), (-1, 0), (0, 1), (0, -1)\}$. The isomorphism $\mathrm{BS}(1, q) \xrightarrow{\sim} \mathbb{Z}[\frac{1}{q}] \rtimes \mathbb{Z}$ mentioned above maps a to $(1, 0)$ and t to $(0, 1)$. Since we identify $\mathrm{BS}(1, q)$ and $\mathbb{Z}[\frac{1}{q}] \rtimes \mathbb{Z}$, we will have $a = (1, 0)$ and $t = (0, 1)$. In particular, a can be thought of as "add"/"increment", and t as "move". We regard elements of the subgroup $\mathbb{Z}[\frac{1}{q}] \times \{0\}$ of $\mathrm{BS}(1, q)$ as elements of $\mathbb{Z}[\frac{1}{q}]$, i.e., integers or rational fractions with denominator q^i, $i \geq 1$.

Rational subset membership. Unless specified otherwise, automata over $\mathrm{BS}(1, q)$ will use the generating set $\Sigma = \{a, a^{-1}, t, t^{-1}\} = \{(1, 0), (-1, 0), (0, 1), (0, -1)\}$. The central decision problem of this work is the *rational subset membership problem for* $\mathrm{BS}(1, q)$:
Given An automaton $\mathcal{A}$ over $\mathrm{BS}(1, q)$ and an element $g \in \mathrm{BS}(1, q)$ as a word over Σ.
Question Does g belong to $\mathsf{L}(\mathcal{A})$?

Automata over $\mathbf{BS(1, q)}$. In the following definitions, let $\mathcal{A} = (Q, \Sigma, E, q_0, q_f)$ be an automaton over $\mathrm{BS}(1, q)$. For a run ρ of $\mathcal{A}$, recall that $[\rho] \in \mathbb{Z}[\frac{1}{q}] \rtimes \mathbb{Z}$ is the *production* of ρ. Moreover, if $[\rho] = (r, m)$ with $r \in \mathbb{Z}[\frac{1}{q}]$ and $m \in \mathbb{Z}$, then we define $\mathsf{pos}(\rho) = m$, and call this the *final position* of ρ. More generally, the *position* at a particular point in ρ is the final position of the corresponding prefix of ρ. By $\mathsf{pmax}(\rho)$, we denote the maximal value of $\mathsf{pos}(\pi)$ where π is a prefix of ρ. Analogously, $\mathsf{pmin}(\rho)$ is the minimal value of $\mathsf{pos}(\pi)$ where π is a prefix of ρ. A run ρ is *returning* if $\mathsf{pos}(\rho) = 0$. It is *returning-left* if in addition $\mathsf{pmin}(\rho) = 0$. Note that for a returning run ρ, we have $[\rho] \in \mathbb{Z}[\frac{1}{q}]$ and if ρ is returning-left, we have $[\rho] \in \mathbb{Z}$. Let $|\rho|$ be the length of the run ρ as a word over E. We will often write ρ_i assuming $\rho = \rho_1\rho_2 \ldots \rho_\ell$ where each $\rho_i \in E$ and $\ell = |\rho|$. A run is a *cycle* if it is returning and starts and ends in the same state. The *thickness* of a run ρ is defined as the greatest number of times a position is seen:

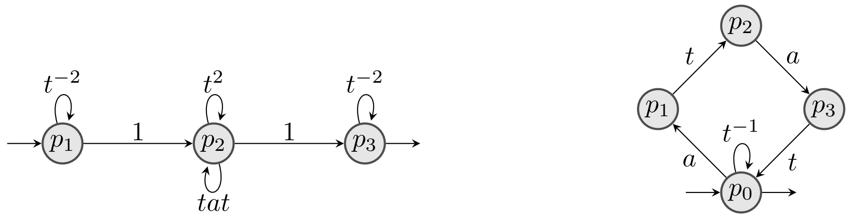
$$\mathsf{thickness}(\rho) = \max_{n \in \mathbb{Z}} |\{i \mid \mathsf{pos}(\rho_1 \cdots \rho_i) = n\}| \ .$$

We call a run *k-thin* if its thickness is at most k.

We let $\mathsf{Runs}(\mathcal{A})$ (resp. $\mathsf{Ret}(\mathcal{A})$, $\mathsf{RetL}(\mathcal{A})$) be the set of all accepting runs (resp. accepting returning runs, accepting returning-left runs) of $\mathcal{A}$. We add k in subscript to restrict the set to k-thin runs; for instance, $\mathsf{Ret}_k(\mathcal{A})$ is the set of k-thin returning runs. Further, we write $\mathsf{Runs}_k^{p \to p'}(\mathcal{A})$ for k-thin runs that start in p and end in p', and use the similar notations $\mathsf{Ret}_k^{p \to p'}(\mathcal{A})$ and $\mathsf{RetL}_k^{p \to p'}(\mathcal{A})$.

Seeing $\{0, \ldots, q-1\}$ as an alphabet, write Φ_q for letters from this alphabet with possibly a $\bullet$ subscript (e.g., $0_\bullet$), a $\triangleleft$ superscript (e.g., $0^\triangleleft$), or both (e.g., $0_\bullet^\triangleleft$). For $v = (r, n) \in \mathrm{BS}(1, q)$, we write $\mathsf{pe}(v)$ for its base-q *pointed expansion* (or just *expansion*) as a word in $\pm\Phi_q^*$, where the subscript $\bullet$ and the superscript $\triangleleft$ appear only once, the former representing the radix point, the latter indicating the value of n. That is, if $r = \sum_{i=-k_2}^{k_1} a_i q^i$, with $k_1, k_2 \geq 0$, $\mathsf{pe}(v)$ is the following word:

$$\pm a_{k_1} \cdots a_1 (a_0)_\bullet a_{-1} \cdots a_{-k_2} \ ,$$



(a) Automaton over $\mathrm{BS}(1,q)$ from Example 3.7.

(b) Automaton over $\mathrm{BS}(1,2)$ from Example 4.1.

Figure 1 Example automata over $\mathrm{BS}(1,q)$.

where $\triangleleft$ is added to a_n. We tacitly assume a uniqueness condition: the expansion $\mathsf{pe}(v)$ of an element $v \in \mathrm{BS}(1,q)$ is the shortest that abides by the definition. Expansions are read by automata in the left to right direction, i.e., from most to least significant digit.

▶ **Definition 2.1.** *We say that a subset of $R \subseteq \mathrm{BS}(1,q)$ is* PE-regular*, where PE stands for pointed expansion, if the word language $\{\mathsf{pe}(v) \mid v \in R\}$ is regular.*

We remark that basic properties of regular languages support the transformation of noncanonical expansions of elements $\mathrm{BS}(1,q)$, i.e., those with zeros on the left or right, into canonical ones, $\mathsf{pe}(v)$. Finally, recall that we identify each $r \in \mathbb{Z}[\frac{1}{q}]$ with $(r,0) \in \mathbb{Z}[\frac{1}{q}] \rtimes \mathbb{Z}$. Hence, for $r \in \mathbb{Z}[\frac{1}{q}]$, $\mathsf{pe}(r)$ is the q-ary expansion of r (with $\triangleleft$ as an additional decoration at the radix point).

3 Main results

Our first main result is that one can translate rational subsets into PE-regular subsets.

▶ **Theorem 3.1.** *Every rational subset of $\mathrm{BS}(1,q)$ is effectively PE-regular.*

This will be shown in Section 4. Since membership is decidable for regular languages and given $g \in \mathrm{BS}(1,q)$ as a word over $\{a, a^{-1}, t, t^{-1}\}$, one can compute $\mathsf{pe}(g)$, Theorem 3.1 implies that rational subset membership is decidable. Our next main result is that the problem is PSPACE-complete.

▶ **Theorem 3.2.** *The rational subset membership problem for $\mathrm{BS}(1,q)$ is* PSPACE*-complete.*

This is shown in Section 5. We shall also conclude that membership to each fixed rational subset is decidable in logspace.

▶ **Theorem 3.3.** *For each fixed rational subset of $\mathrm{BS}(1,q)$, membership is decidable in logarithmic space.*

The proof can also be found in Section 5. Note that, in particular, membership to each fixed subgroup of $\mathrm{BS}(1,q)$ is decidable in logarithmic space. Another application of Theorem 3.1 is that one can decide whether a given rational subset of $\mathrm{BS}(1,q)$ is recognizable.

▶ **Theorem 3.4.** *Given a PE-regular subset R of $\mathrm{BS}(1,q)$, it is decidable whether R is recognizable.*

This is shown in Section 6. Since a subgroup of any group H is recognizable if and only if it has finite index in H (see, e.g. [2, Prop. 3.2]), we obtain:

▶ **Corollary 3.5.** *Given a f.g. subgroup of* $\mathrm{BS}(1,q)$*, it is decidable whether it has finite index.*

We close this section by showing that regular subsets of $\mathrm{BS}(1,q)$ are robust in terms of closure properties.

▶ **Proposition 3.6.** *The PE-regular subsets of* $\mathrm{BS}(1,q)$ *form an effective Boolean algebra. Moreover, for PE-regular subsets* $R, S \subseteq \mathrm{BS}(1,q)$*, the sets* $RS = \{rs \mid r \in R,\ s \in S\}$ *and* $R^{-1} = \{r^{-1} \mid r \in R\}$ *are PE-regular as well.*

The proof is straightforward. Together with Theorem 3.1, this implies that emptiness of Boolean combinations (hence inclusion, equality) is decidable for rational subsets. To further highlight the advantages of PE-regular subsets, we also show that the rational subsets of $\mathrm{BS}(1,q)$ are not closed under intersection.

▶ **Example 3.7** (Intersection of rational subsets)**.** Let R be the set accepted by the automaton in Figure 1a. The automaton first moves an even number of positions to the right (p_1) and then an even number of positions to the left while adding 1 in a subset of the odd positions (p_2). Finally, it goes an even number of positions to the left again. Note that $(r,m) \in R$ if and only if $r = \sum_{i \in A} q^{2i+1}$ for some finite $A \subseteq \mathbb{Z}$ and $m \in 2\mathbb{Z}$. Now consider the rational sets aR and Ra and their intersection $I = aR \cap Ra$. Note that $(r,m) \in aR$ if and only if $r = 1 + \sum_{i \in A} q^{2i+1}$ and $m \in 2\mathbb{Z}$ for some finite $A \subseteq \mathbb{Z}$. Moreover, $(r,m) \in Ra$ if and only if $r = q^m + \sum_{i \in A} q^{2i+1}$ and $m \in 2\mathbb{Z}$ for some finite $A \subseteq \mathbb{Z}$. Therefore, we have $(r,m) \in I$ if and only if $r = 1 + \sum_{i \in A} q^{2i+1}$ and $m = 0$ for some finite $A \subseteq \mathbb{Z}$. Since I only contains elements with cursor 0, but carries non-zero digits in positions that are arbitrarily far to the right, it follows that I is not rational.

However, the PE-regular subsets of $\mathrm{BS}(1,q)$ are not closed under iteration.

▶ **Example 3.8** (Iteration of PE-regular subsets)**.** The subset $A = \{(1 + 2^{-i}, 0) \mid i \geq 1\}$ of $\mathrm{BS}(1,2)$ is PE-regular, because $\mathsf{pe}(A) = 1\overset{\triangleleft}{\bullet}0^*1$ is a regular language. However, the set A^* is not PE-regular: one can show that for each $n \geq 1$, we have $n = \min\{m \in \mathbb{N} \mid (m + 2^{-1} + \cdots + 2^{-n}, 0) \in A^*\}$.[2] Therefore, for each $n \geq 1$, a word in $\mathsf{pe}(A^*)$ with 1^n to the right of the radix point can have an integer part of n and cannot have a smaller integer part. This implies that $\mathsf{pe}(A^*)$ is not regular.

4 Every rational subset of BS(1, q) is effectively PE-regular

In this section, we prove Theorem 3.1. We first illustrate our approach on an example.

▶ **Example 4.1.** Consider the automaton over $\mathrm{BS}(1,2)$ in Figure 1b. In its only initial and final state p_0, it has a choice of two operations: (i) move the cursor one position to the right (i.e. multiplication by t^{-1}) or (ii) perform the increment on two neighbouring cells and stop one position left of them (i.e. multiplication by $atat$). The automaton can perform these operations arbitrarily many times in any order.

We shall prove that the automaton accepts

$$R = \{(3n \cdot 2^{m-2k}, m) \mid n \in \mathbb{N},\ k \in \mathbb{N},\ m \in \mathbb{Z},\ 0 \geq m - 2k,\ 3n \cdot 2^{m-2k} \geq f(m,k)\}\ ,$$

where

$$f(m,k) = \sum_{i=1}^{k} 3 \cdot 2^{m-2i} = \sum_{j=m-2k}^{m-1} 2^j = 2^m - 2^{m-2k}\ .$$

[2] We denote $\mathbb{N} = \{0, 1, 2, \ldots\}$.

The language $\mathsf{pe}(R)$ is regular. Indeed, note that the number $f(m,k)$ has a particularly simple binary representation. A pointed expansion of (r,m) belongs to $\mathsf{pe}(R)$ if there is a position $m-2k \le 0$ such that reading the digits left of position $m-2k$ yields a number (namely $3n$) that (a) is divisible by 3 and (b) lies above a bound with a simple binary expansion.

Let us now prove that the automaton accepts R. Let ρ be an accepting run producing (r,m). Choose $k \in \mathbb{N}$ so that $\mathsf{pmin}(\rho) = m-2k$ or $\mathsf{pmin}(\rho) = m-2k+1$ (depending on whether $m - \mathsf{pmin}(\rho)$ is even or odd). Then $0 \ge \mathsf{pmin}(\rho) \ge m-2k$. Each time operation (ii) is performed from position $\ell \in \mathbb{Z}$, the update is $(r,m) \to (r + 3 \cdot 2^{\ell}, m+2)$.

Now, once ρ visits position $\mathsf{pmin}(\rho)$, in order to eventually reach a position $\ell > \mathsf{pmin}(\rho)$, the operation (ii) must be performed on some position $\ge \ell - 2$. In particular, to reach position m, it must be performed at some position $m_1 \ge m-2$. If $m_1 > \mathsf{pmin}(\rho)$, to reach m_1, it must also be performed at some position $m_2 \ge m-4$, etc. Therefore, ρ has to perform (ii) at positions $m_i \ge m-2i$ for each i with $m > m-2i \ge \mathsf{pmin}(\rho)-1$. In other words, it has to do this for each $i = 1, \ldots, k$. Each time ρ performs (ii) at m_i, it adds $3 \cdot 2^{m_i}$. Moreover, each extra time ρ performs (ii), it adds a multiple of $3 \cdot 2^{m-2k}$, because $\mathsf{pmin}(\rho) \ge m-2k$. Thus, the number produced in total is some $3n \cdot 2^{m-2k}$ where

$$3n \cdot 2^{m-2k} \ge \sum_{i=1}^{k} 3 \cdot 2^{m_i} \ge \sum_{i=1}^{k} 3 \cdot 2^{m-2i} = f(m,k) \ .$$

Conversely, suppose $n \in \mathbb{N}$ and $k \in \mathbb{N}$, $m \in \mathbb{Z}$, $0 \ge m-2k$, and $3n \cdot 2^{m-2k} \ge f(m,k)$. The automaton first moves to position $m-2k$ using operation (i). Then, it performs operations (ii), (i), and (i) again, ℓ times in a loop (we specify ℓ later). That way, it adds $3\ell \cdot 2^{m-2k}$. Then, it moves to position m by applying operation (ii) exactly k times. Hence, it applies (ii) at positions $m-2i$ for $i = 1, \ldots, k$ and each time, it adds $3 \cdot 2^{m-2i}$. In total, the effect is

$$3\ell \cdot 2^{m-2k} + \sum_{i=1}^{k} 3 \cdot 2^{m-2i} = 3\ell \cdot 2^{m-2k} + f(m,k) \ .$$

Since $3n \cdot 2^{m-2k} \ge f(m,k)$ and $f(m,k)$ is an integer multiple of $3 \cdot 2^{m-2k}$, we can choose $\ell \in \mathbb{N}$ so as to produce $3n \cdot 2^{m-2k}$.

Following this example, we first show that any run has the same production as a thin (i.e. bounded thickness) run in which thin returning-left cycles are inserted (p. 9); in the example, such a cycle applies operations (ii), (i), and (i). We then prove that the productions of thin runs form a PE-regular set (p. 10); in the example, the thin run moves to the right to position $\mathsf{pmin}(\rho)$ using operation (i) and then left to $m \ge \mathsf{pmin}(\rho)$ using operations (i) and (ii). Finally, we show that iterating returning-left thin cycles also leads to a PE-regular set (p. 11); in the example, this is how we get all numbers divisible by 3 above a particular bound. We combine these three statements to prove Theorem 3.1.

In combining the thin run with cycles, we will need to ensure that the cycles are anchored on the correct state. To this end, we introduce an annotated version of $\mathsf{pe}([\rho])$ as follows. Let $\mathcal{A}$ be an automaton over $\mathrm{BS}(1,q)$ with state set Q. Let ρ be a run in $\mathcal{A}$ starting and ending in arbitrary states and with $[\rho] = (r,m)$. Letting $\bar{Q} = \{\bar{p} \mid p \in Q\}$ be a copy of Q, we define $\mathsf{sv}(\rho)$, the *state view* of ρ, to be the word over the alphabet $\Phi_q \cup Q \cup \bar{Q} \cup \{\pm\}$ built as follows. First, write: $\mathsf{pe}([\rho]) = \pm a_{k_1} \cdots a_1 a_0 a_{-1} \cdots a_{-k_2}$,where a_0 has subscript $\bullet$. Second, let $P_i \in (Q \cup \bar{Q})^{|Q|}$, for $i \in \{-k_2, \ldots, k_1\}$, be a word that contains all the states of Q once in a fixed ordering of Q, either with a bar or not; the states without a bar are exactly those that visit position i in ρ. That is, p appears in P_i iff there is a prefix of ρ ending in p whose final position is i. The state view of ρ is then:

$$\mathsf{sv}(\rho) = \pm a_{k_1} \cdot P_{k_1} \cdots a_0 \cdot P_0 \cdot a_{-1} \cdot P_{-1} \cdots a_{-k_2} \cdot P_{-k_2} \enspace .$$

We naturally extend sv to sets of runs.

Any run is equivalent to a thin run augmented with thin returning-left cycles. We now focus on two properties of runs: the states they visit in the automaton and the final position of their prefixes. To that end, we introduce the following notions. For Q a finite set, a *position path* is a word $\pi \in (Q \times \mathbb{Z})^*$. We extend the analogy with graphs calling elements of $Q \times \mathbb{Z}$ *vertices*, talking of the vertices *visited* by a position path, and using the notion of (position) *subpaths* and *cycles*. The *thickness* of a position path π is defined as:

$$\mathsf{thickness}(\pi) = \max_{n \in \mathbb{Z}} |\{i \mid \pi_i = (q, n) \text{ for some } q\}| \enspace .$$

▶ **Lemma 4.2.** *Let Q be a finite set and $\pi \in (Q \times \mathbb{Z})^*$ be a position path. For any subset V' of the vertices visited by π, there exists a subpath π' of π such that:*

1. *π' starts and ends with the same vertices as π,*
2. *π' visits all the vertices in V',*
3. $\mathsf{thickness}(\pi') \leq |Q| \cdot (1 + 2|V'|)$,
4. *$\pi - \pi'$ consists only of cycles.*

Proof (sketch). We first consider a shortest subpath π' of π from the initial to the final vertices of π – this implies that π' has thickness at most $|Q|$. We then treat each missing vertex from V' in turn, and add to π' a subpath from π that is a cycle and includes that vertex. Each of these iterations can augment the thickness of π' by at most $2|Q|$. ◀

▶ **Corollary 4.3.** *Let $\mathcal{A}$ be an automaton over $\mathrm{BS}(1, q)$ with state set Q, and let $k = |Q| + 2|Q|^2$. Any run of $\mathcal{A}$ is equivalent to a run in $\mathsf{Runs}_k(\mathcal{A})$ on which, for each state p appearing in the run, cycles from $\mathsf{RetL}_k^{p \to p}(\mathcal{A})$ are inserted at an occurrence of p with smallest position.*

Conversely, any run built by taking a run in $\mathsf{Runs}_k(\mathcal{A})$ and inserting cycles from $\mathsf{RetL}_k^{p \to p}(\mathcal{A})$ at an occurrence of p is a run of $\mathcal{A}$.

Proof. The converse is clear, we thus focus on the first direction.

(Step 1: Decomposing a run into a thin run and cycles.) Let $\rho \in \mathsf{Runs}(\mathcal{A})$, and extract from it a position path $\pi = \pi_0 \cdots \pi_{|\rho|}$ as follows. We let, $\pi_0 = (q_0, 0)$ and for all $i \geq 1$:

$$\pi_i = (p, n) \text{ where } \rho_i = (\cdot, \cdot, p) \text{ and } n = \mathsf{pos}(\rho_1 \cdots \rho_i) \enspace .$$

For each state p visited by ρ, let $n_p = \min\{n \mid \text{there exists } i \text{ such that } \pi_i = (p, n)\}$; in words, n_p is the smallest final position of a prefix of ρ ending in p. Using $V' = \{(p, n_p) \mid \rho \text{ visits } p\}$, Lemma 4.2 provides a position path π' of thickness $\leq k = |Q| + 2|Q|^2$ visiting all of V'.

From π', we can obtain the corresponding subpath ρ' of ρ that has the same starting and ending state and positions as ρ, and such that ρ is made of ρ' onto which cycles are added. The thickness of ρ' is bounded by k, but the cycles can be of any thickness.

(Step 2: Thinning the cycles.) Consider a cycle β that gets added to ρ' to form ρ, say at position i (after initial i moves, $\rho'_1 \cdots \rho'_i$), and assume that $\mathsf{thickness}(\beta) > k$. Since a position is repeated more than $k > |Q|$ times, there is a cycle β' *within* β with $\mathsf{thickness}(\beta') \leq k$;

write then $\beta = \alpha \cdot \beta' \cdot \alpha'$. Let p be the state in β' that has the smallest position, that is, p is the ending state of the prefix γ of β' with final position $\mathsf{pmin}(\beta')$; write $\beta' = \gamma \cdot \gamma'$. By definition, we have $\mathsf{pos}(\rho'_1 \cdots \rho'_i \alpha\gamma) \geq n_p$. Note that $\gamma' \cdot \gamma$ is in $\mathsf{RetL}_k^{p\to p}(\mathcal{A})$. We now remove β' from β and then insert $\gamma' \cdot \gamma$ at the position j in ρ' that is such that $\rho'_1 \cdots \rho'_j$ ends in p with final position n_p. For the contribution of $\gamma' \cdot \gamma$ to be the same as that of β' in the original path, we insert it q^d times, where $d = \mathsf{pos}(\rho'_1 \cdots \rho'_i \alpha\gamma) - n_p$.

This shows that if any cycle added to ρ' is of thickness $> k$, then a subcycle of it can be moved to another position of ρ' as a returning-left cycle. Iterating this process, all the cycles added to ρ' will thus be of thickness $\leq k$. Moreover, if an added cycle β is not returning-left after these operations, or if it does not sit at an occurrence of its initial state with *smallest* position, this means that we can decompose it just as above as $\gamma \cdot \gamma'$, with γ reaching $\mathsf{pmin}(\beta)$, and move $\gamma' \cdot \gamma$, a returning-left cycle, to an appropriate position in ρ' as before. ◀

Intermezzo: reflecting on Corollary 4.3. Before we continue with the proof, we want to illustrate how crucial the previous corollary is. Lemma 4.2 tells us that we can obtain every run from a thin run by then adding cycles. This already simplifies the structure of $\mathsf{Runs}(\mathcal{A})$: indeed, inserting cycles at a certain position in a run $\rho \in \mathsf{Runs}(\mathcal{A})$ corresponds (in algebraic terms) to adding to $[\rho]$ a subset of $\mathbb{Z}[\frac{1}{q}]$ closed under addition, i.e., a submonoid. (Closure under addition follows from the observation that any two returning cycles from each $\mathsf{Ret}_k^{p\to p}(\mathcal{A})$ can be concatenated.)

Sometimes one can conclude that every submonoid of a monoid has a simple structure. For example, every submonoid M of $\mathbb{Z}$ is semilinear and hence a PE-regular subset of $\mathbb{Z}[\frac{1}{q}]$. Unfortunately, the situation in $\mathbb{Z}[\frac{1}{q}]$ is not as simple as in $\mathbb{Z}$: One can show that $\mathbb{Z}[\frac{1}{q}]$ has uncountably many submonoids. Thus, $\mathbb{Z}[\frac{1}{q}]$ has submonoids with undecidable membership problem; moreover, there is no hope for a finite description for every submonoid as in $\mathbb{Z}$. Thus, we need to look at our specific submonoids. A simple observation similar to Lemma 4.2 allows us to obtain every run from a thin part by adding *thin* cycles. Hence, the submonoids that we add are of the form $[\mathsf{Ret}_k^{p\to p}(\mathcal{A})]^*$. It is not hard to show (see Lemma 4.4) that $[\mathsf{Ret}_k^{p\to p}(\mathcal{A})]$ is always a PE-regular set. Thus, one may hope to prove that the regularity of $[\mathsf{Ret}_k^{p\to p}(\mathcal{A})]$ implies regularity of $[\mathsf{Ret}_k^{p\to p}(\mathcal{A})]^*$. (This was an approach to rational subset membership proposed by the third author of this work in [12, Section 4.7].) However, Example 3.8 tells us that even for PE-regular $R \subseteq \mathrm{BS}(1,q)$, the set R^* may not be PE-regular.

Therefore, Corollary 4.3 is the key insight of our proof. It says that a run can be decomposed into a thin part and thin *returning-left* cycles. Since returning-left cycles produce integers, this will lead us to submonoids of $\mathbb{Z}$.

Sets of thin runs are PE-regular.

▶ **Lemma 4.4.** *Let $\mathcal{A}$ be an automaton over* $\mathrm{BS}(1,q)$*, p, p' be states of $\mathcal{A}$, and $k > 0$. The sets* $\mathsf{sv}(\mathsf{Runs}_k^{p\to p'}(\mathcal{A})), \mathsf{sv}(\mathsf{Ret}_k^{p\to p'}(\mathcal{A}))$*, and* $\mathsf{sv}(\mathsf{RetL}_k^{p\to p'}(\mathcal{A}))$ *are effectively regular.*

Proof (sketch). We see $\mathcal{A}$ as a two-way automaton, and apply a construction similar to the classical proof that two-way automata are no more expressive than one-way automata [32]. This transforms $\mathcal{A}$ into a one-way automaton over the alphabet $\{-1, 0, 1\}^k$, where each component tracks a 1-thin partial run. It is a classical exercise to show that automata can compute the addition of numbers in a given base; this can be extended to *signed-digit* expansions, in which negative digits can be used [8, Section 2.2.2.2]. We thus rely on this to compute the sum, componentwise, of these partial runs. Adding state information to that construction is straightforward, so that we obtain automata for state views. ◀

Iterations of returning-left thin cycles are PE-regular. It is well-known that for every set $S \subseteq \mathbb{N}$ the generated monoid $S^* = \{s_1 + \cdots + s_m \mid s_1, \ldots, s_m \in S, m \geq 0\}$ is eventually identical with $\gcd(S) \cdot \mathbb{N}$. In other words, the set $(\gcd(S) \cdot \mathbb{N}) \setminus S^*$ is finite and we may define $F(S) = \max((\gcd(S) \cdot \mathbb{N}) \setminus S^*)$. The number $F(S)$ is called the *Frobenius number* of S. With this, we have $S^* = \{n \in S^* \mid n \leq F(S)\} \cup \{n \in \gcd(S) \cdot \mathbb{N} \mid n > F(S)\}$. If $S \subseteq -\mathbb{N}$, then we set $F(S) := F(-S)$. Now consider an arbitrary set $S \subseteq \mathbb{Z}$. If S contains both a positive and a negative number, then $S^* = \gcd(S) \cdot \mathbb{Z}$ and we set $F(S) := 0$. We shall use the following well-known fact [36].

▶ **Lemma 4.5.** *If $S = \{n_1, \ldots, n_k\}$ with $0 < n_1 < \cdots < n_k$, then $F(S) \leq n_k^2$.*

▶ **Lemma 4.6.** *For every automaton $\mathcal{A}$ over $\mathrm{BS}(1, q)$, the language $\mathsf{pe}([\mathsf{RetL}_k^{p \to p}(\mathcal{A})]^*)$ is effectively regular.*

Proof. Recall that we identify each $r \in \mathbb{Z}[\frac{1}{q}]$ with $(r, 0) \in \mathbb{Z}[\frac{1}{q}]$. In particular, for $n \in \mathbb{Z}$, $\mathsf{pe}(n)$ is the same as $\mathsf{pe}((n, 0))$.

Denote $S = [\mathsf{RetL}_k^{p \to p}(\mathcal{A})]$. We first consider the case $S \subseteq \mathbb{N}$ and $S \neq \emptyset$. Suppose we can compute $\gcd(S)$ and a bound $B \in \mathbb{N}$ with $B \geq F(S)$. Then we have

$$S^* = \underbrace{\{n \in S^* \mid n \leq B\}}_{=:X} \cup \underbrace{\{n \in \gcd(S) \cdot \mathbb{N} \mid n > B\}}_{=:Y} \tag{1}$$

and it suffices to show that $\mathsf{pe}(X)$ and $\mathsf{pe}(Y)$ are effectively regular. Note that X is finite and can be computed by finding all $n \leq B$ with $n \in S$ (recall that membership in S is decidable because $\mathsf{sv}(\mathsf{RetL}_k^{p \to p}(\mathcal{A}))$ is effectively regular by Lemma 4.4) and building sums. Moreover, $\mathsf{pe}(Y)$ is regular because the set $L_0 = \mathsf{pe}(\gcd(S) \cdot \mathbb{N})$ is effectively regular and so is $L_1 = \{\mathsf{pe}(n) \mid n \in \mathbb{N},\ n > B\}$, and hence $\mathsf{pe}(Y) = L_0 \cap L_1$.

Thus, it remains to compute $\gcd(S)$ and some $B \geq F(S)$. For the former, find any $r \in S$ and consider its decomposition $r = p_1^{e_1} \cdots p_m^{e_m}$ into prime powers. For each $i \in [1, m]$, we compute $d_i \in [0, e_i]$ and $n_i \in S$ such that (i) $S \subseteq p_i^{d_i} \cdot \mathbb{N}$, and (ii) $n_i \in S \setminus p_i^{d_i+1} \cdot \mathbb{N}$. Since for $d \in \mathbb{N}$, we can construct an automaton for $\mathsf{pe}(S \cap d \cdot \mathbb{N})$, these d_i and n_i can be computed. Observe that $\gcd(S) = p_1^{d_1} \cdots p_m^{d_m}$. Let $T = \{r, n_1, \ldots, n_k\}$. Observe that $\gcd(T) = \gcd(S)$, and hence T^* and S^* are ultimately identical. Since $T \subseteq S$, this means $F(S) \leq F(T)$. By Lemma 4.5, we have $F(T) \leq (\max\{r, n_1, \ldots, n_k\})^2$, which yields our bound B.

The case $S \subseteq -\mathbb{N}$ is analogous to $S \subseteq \mathbb{N}$. If S contains a positive and a negative number, then $S^* = \gcd(S) \cdot \mathbb{Z}$, so it suffices to just compute $\gcd(S)$. This is done as above. Finally, deciding between these three cases is easy. This completes the proof. ◀

Wrapping up: Proof of Theorem 3.1. Let $\mathcal{A}$ be an automaton over $\mathrm{BS}(1, q)$ with state set Q. Corollary 4.3 indicates that the set of productions of accepting runs is the same as the set of productions of k-thin runs in which thin cycles are introduced.

By Lemma 4.4, $\mathsf{sv}(\mathsf{Runs}_k(\mathcal{A}))$ is a regular language L. For any state p of $\mathcal{A}$, let $L_p = \mathsf{pe}\big([\mathsf{RetL}_k^{p \to p}(\mathcal{A})]^*\big)$, a regular language by Lemma 4.6. For padding purposes, let $s \in Q$ be some state, and let h be the morphism from $(\Phi_q \cup \{\pm\})^*$ to $(\Phi_q \cup Q \cup \{\pm\})^*$ defined, for any $a \in \Phi_q$, by $h(a) = as^{|Q|}$, and $h(+) = +$, $h(-) = -$. Define now L'_p to be the image by h of the version of L_p where arbitrary 0's are added after the sign, and at the end of the number (these 0's do not change the value represented).

Consider now the language R over the alphabet $(\Phi_q \cup Q \cup \bar{Q} \cup \{\pm\})^{|Q|+1}$ whose projection on the first component is the language L, and the other components correspond to the languages L'_p, for each $p \in Q$. The first component indicates in particular the states of $\mathcal{A}$

that visited that location; to synchronize the different components of R, we ensure that the letter annotated with $\bullet$ in L'_p is aligned with a letter from L that is followed by p – that is, the starting position of L'_p is at a position in L that is seen while being in the state p.

Finally, an automaton can do the componentwise addition in base q, collapsing the $|Q|+1$ components into a single one. The radix point is given by the digit with $\bullet$ of L, i.e., in the first component; and similarly for $\lhd$. The resulting language, thanks to Corollary 4.3, is the language of the pointed expansions of all runs in $\mathsf{Runs}(\mathcal{A})$. ◀

5 Complexity

Computing pointed expansions. In this section, we prove Theorems 3.2 and 3.3. For the upper bounds in Theorems 3.2 and 3.3, we shall rely on the fact that, given an element $g \in \mathbb{Z}[\frac{1}{q}] \rtimes \mathbb{Z}$ as a word over $\Sigma = \{a, a^{-1}, t, t^{-1}\}$, one can compute the pointed expansion $\mathsf{pe}(g)$ in logarithmic space. This is a direct consequence of a result of Elder, Elston, and Ostheimer [18, Proposition 32]. They show that given a word w over Σ, one can compute in logarithmic space an equivalent word of one of the forms (i) t^i, (ii) $(a^{\eta_0})^{t^{\alpha_0}}(a^{\eta_1})^{t^{\alpha_1}}\cdots(a^{\eta_k})^{t^{\alpha_k}}t^i$ or (iii) $(a^{-\eta_0})^{t^{\alpha_0}}(a^{-\eta_1})^{t^{\alpha_1}}\cdots(a^{-\eta_k})^{t^{\alpha_k}}t^i$, where $i \in \mathbb{Z}$, $k \in \mathbb{N}$, $0 < \eta_j < q$ for $j \in [0,k]$, and $\alpha_0 > \cdots > \alpha_k$. Here, x^y stands for $y^{-1}xy$ in the group. Since these normal forms denote the elements (i) $(0,i)$, (ii) $(\sum_{j=0}^{k}\eta_j q^{-\alpha_j}, i)$ and (iii) $(-\sum_{j=0}^{k}\eta_j q^{-\alpha_j}, i)$, respectively, it is easy to turn these normal forms into $\mathsf{pe}(w)$ using logarithmic space.

This allows us to prove Theorem 3.3: For every rational subset $R \subseteq \mathrm{BS}(1,q)$, the language $\mathsf{pe}(R)$ is a regular language. In particular, there exists a deterministic automaton $\mathcal{B}$ for $\mathsf{pe}(R)$. Therefore, given $g \in \mathrm{BS}(1,q)$ as a word over $\{a, a^{-1}, t, t^{-1}\}$, we compute $\mathsf{pe}(g)$ in logspace and then check membership of $\mathsf{pe}(g)$ in $\mathsf{L}(\mathcal{B})$, which is decidable in logarithmic space.

PSPACE-completeness. The PSPACE lower bound in Theorem 3.2 is a reduction from the intersection nonemptiness of finite-state automata, a well-known PSPACE-complete problem [23]. For the PSPACE upper bound, we strengthen Theorem 3.1 by constructing a polynomial-size representation of an exponential size automaton for the resulting regular language. A *succinct finite automaton* is a tuple $\mathcal{S} = (n, \Gamma, (\varphi_x)_{x\in\Gamma\cup\{\varepsilon\}}, p_0, p_f\})$, where $n \in \mathbb{N}$ is its *bit length*, Γ is its *input alphabet*, $\varphi_x(\mathsf{v}_1, \ldots, \mathsf{v}_n, \mathsf{v}'_1, \ldots, \mathsf{v}'_n)$ is a formula from propositional logic with free variables $\mathsf{v}_1, \ldots, \mathsf{v}_n, \mathsf{v}'_1, \ldots, \mathsf{v}'_n$ for each $x \in \Gamma \cup \{\varepsilon\}$, $p_0 \in \{0,1\}^n$ is its *initial state*, and $p_f \in \{0,1\}^n$ is its *final state*. The *size* of $\mathcal{S}$ is defined as $|\mathcal{S}| = n + \sum_{x\in\Gamma\cup\{\varepsilon\}} |\varphi_x|$, where $|\varphi|$ denotes the length of the formula φ.

Moreover, $\mathcal{S}$ *represents* the automaton $\mathcal{A}(\mathcal{S})$, which is defined as follows. It has the state set $\{0,1\}^n$, initial state p_0, and final state p_f. For states $p = (b_1, \ldots, b_n), p' = (b'_1, \ldots, b'_n) \in \{0,1\}^n$ and $x \in \Gamma \cup \{\varepsilon\}$, there is an edge (p, x, q) in $\mathcal{A}(\mathcal{S})$ if and only if $\varphi_x(b_1, \ldots, b_n, b'_1, \ldots, b'_n)$ holds. We define the *language accepted by* $\mathcal{S}$ as $\mathsf{L}(\mathcal{S}) = \mathsf{L}(\mathcal{A}(\mathcal{S}))$.

We allow ε-edges in succinct automata, and with Boolean formulas, one can encode steps in a Turing machine. Thus, a succinct automaton of polynomial size can simulate a polynomial space Turing machine with a one-way read-only input tape. Our descriptions of succinct automata will therefore be in the style of polynomial space algorithms. We show:

▶ **Theorem 5.1.** *Given a rational subset* $R \subseteq \mathrm{BS}(1,q)$*, one can construct in polynomial space a polynomial-size succinct automaton accepting* $\mathsf{pe}(R)$*.*

This allows us to decide rational subset membership in PSPACE: Given an automaton $\mathcal{A}$ over $\mathrm{BS}(1,q)$ and an element g as a word over $\{a, a^{-1}, t, t^{-1}\}$, we construct a succinct automaton $\mathcal{B}$ for $\mathsf{pe}(\mathsf{L}(\mathcal{A}))$ and the pointed expansion $\mathsf{pe}(g)$ in logarithmic space. Since membership in succinct automata is well-known to be in PSPACE, we can check whether $\mathsf{pe}(g) \in \mathsf{L}(\mathcal{B})$.

Constructing succinct automata. It remains to prove Theorem 5.1. The construction of a succinct automaton for $\mathsf{pe}(R)$ proceeds with the same steps as in Section 4. For most of these steps, our constructions already yield small succinct automata (e.g., one for $\mathsf{pe}([\mathsf{RetL}_k^{p\to p'}(\mathcal{A})])$ in Lemma 4.4). The exception is Lemma 4.6 – in which case the key ingredient is as follows.

▶ **Proposition 5.2.** *Given an automaton $\mathcal{A}$ over $\mathrm{BS}(1,q)$, a state p of $\mathcal{A}$, and $k \in \mathbb{N}$ in unary, one can compute in polynomial space the number $\gcd([\mathsf{RetL}_k^{p\to p}(\mathcal{A})])$ and a bound $B \geq F([\mathsf{RetL}_k^{p\to p}(\mathcal{A})])$. Both are at most exponential in k and the size of $\mathcal{A}$.*

Our bound on F extends the bound for automatic sets in $\mathbb{N}$ [5, Lemma 4.5] to thin two-way computations. Before proving Proposition 5.2, let us show how it implies Theorem 5.1.

Proof of Theorem 5.1. The constructions in Lemma 4.4 and Theorem 3.1, immediately yield a polynomial-size succinct automaton for $\mathsf{pe}(R)$ once a succinct automaton for each $\mathsf{pe}([\mathsf{RetL}_k^{p\to p}(\mathcal{A})]^*)$ is found. For the latter, we proceed as in Lemma 4.6. Let $S = [\mathsf{RetL}_k^{p\to p}(\mathcal{A})]$ and compute $\gcd(S)$ and a bound $B \geq F(S)$ using Proposition 5.2. Then, by Equation (1) on page 11, it suffices to construct a succinct automaton for $\mathsf{pe}(X)$ and one for $\mathsf{pe}(Y)$. For $\mathsf{pe}(X)$, we use the fact that we can construct a succinct automaton $\mathcal{B}$ for $\mathsf{pe}(S)$. Our automaton for $\mathsf{pe}(X)$ proceeds as follows. With ε-transitions, it runs $\mathcal{B}$ to successively guess numbers $\leq B$ from S and stores each of them temporarily in its state. Such a number requires $O(\log(B))$ bits. In another $O(\log(B))$ bits, it stores the sum of the numbers guessed so far. This continues as long as the sum is at most B. Then, our automaton reads the resulting sum from the input. This automaton clearly accepts $\mathsf{pe}(X)$.

For $\mathsf{pe}(Y)$, we have to construct a succinct automaton that accepts any number $> B$ that is divisible by $\gcd(S)$. Since $\gcd(S)$ is available as a number with polynomially many digits, we can construct a succinct automaton accepting $\mathsf{pe}(\gcd(S)\cdot\mathbb{N})$: It keeps the remainder modulo $\gcd(S)$ of the currently read prefix. This requires $O(\log(\gcd(S))$ many bits. Since B also has polynomially many digits, we can construct a succinct automaton for $\{n \in \mathbb{N} \mid n > B\}$. An automaton for the intersection then accepts $\mathsf{pe}(Y)$. ◀

It is easy to see that the number produced by a returning-left run is at most exponential in the length of the run. The exact bound will not be important.

▶ **Lemma 5.3.** *If ρ is a run in $\mathsf{RetL}_k(\mathcal{A})$ of length ℓ, then $|[\rho]| \leq q^{2\ell}$.*

The main ingredient for Proposition 5.2 will be Lemma 5.4. We write $\rho \ll \rho'$ if $|\rho| < |\rho'|$. Moreover, for $d \in \mathbb{Z}$, we write $\rho \ll_d \rho'$ if $\rho \ll \rho'$ and for some $\ell \in \mathbb{Z}$, we have $[\rho'] = \ell \cdot [\rho] + d$.

▶ **Lemma 5.4.** *There is a polynomial f such that the following holds. Let $\mathcal{A}$ be an n-state automaton over $\mathrm{BS}(1,q)$ and let p, p' be two states of $\mathcal{A}$. Let $\rho_{11} \in \mathsf{RetL}_k^{p\to p'}(\mathcal{A})$ with $|\rho_{11}| > f(n,k)$. There exist runs $\rho_{00}, \rho_{10}, \rho_{01} \in \mathsf{RetL}_k^{p\to p'}(\mathcal{A})$ and $d \in \mathbb{Z}$ so that:*

$$\begin{tikzcd}
\rho_{01} \arrow[r, phantom, "\ll_d"] \arrow[d, phantom, "\rotatebox{90}{$\ll$}"] & \rho_{11} \arrow[d, phantom, "\rotatebox{90}{$\ll$}"] \\
\rho_{00} \arrow[r, phantom, "\ll_d"] & \rho_{10}
\end{tikzcd} \tag{2}$$

Here, one shows that a long run can be shortened independently in two ways: Going left in the diagram (2), and going down. Shortening the run by "going left" changes the production of the run by the same difference, up to a factor ℓ that may differ in the two rows. Lemma 5.5 applies Lemma 5.4 to construct small numbers in $[\mathsf{RetL}_k(\mathcal{A})]$ that are not divisible by a given m. Later, these numbers allow us to compute $\gcd([\mathsf{RetL}_k^{p\to p}(\mathcal{A})])$ and bound $F([\mathsf{RetL}_k^{p\to p}(\mathcal{A})])$.

▶ **Lemma 5.5.** *There is a polynomial f such that the following holds. Let $m \in \mathbb{Z}$. Let $\mathcal{A}$ be an n-state automaton over $\mathrm{BS}(1,q)$ and let p, p' be two states of $\mathcal{A}$. Suppose there is a number in $[\mathsf{RetL}_k^{p\to p'}(\mathcal{A})]$ not divisible by m; then there is also an $s \in [\mathsf{RetL}_k^{p\to p'}(\mathcal{A})]$ not divisible by m such that $|s| \le q^{f(n,k)}$.*

Proof. Let f be the polynomial from Lemma 5.4. Let $\rho \in \mathsf{RetL}_k^{p\to p'}(\mathcal{A})$ be of minimal length such that m does not divide $[\rho]$. Suppose $|\rho| > f(n,k)$. Write $\rho_{11} = \rho$ and apply Lemma 5.4. By minimality of ρ_{11}, we get $[\rho_{00}] \equiv [\rho_{10}] \equiv [\rho_{01}] \equiv 0 \bmod m$. In particular, $\rho_{00} \ll_d \rho_{10}$ implies $d \equiv 0 \bmod m$. However, since $\rho_{01} \ll_d \rho_{11}$ and $[\rho_{11}] \not\equiv 0 \bmod m$, we get $d \not\equiv 0 \bmod m$, a contradiction. Hence, $|\rho| \le f(n,k)$ and thus $|[\rho]| \le q^{2f(n,k)}$ by Lemma 5.3. ◀

With Lemma 5.5 in hand, one can show Proposition 5.2 similarly to Lemma 4.6.

6 Recognizability

In this section, we prove Theorem 3.4. We first present a characterization of recognizability that is easily checkable for PE-regular subsets. It is well-known that a subset S of $\mathbb{Z}$ is recognizable if and only if there is a $k \in \mathbb{Z} \setminus \{0\}$ such that for every $s \in \mathbb{Z}$, we have $s \in S$ if and only if $s + k \in S$. Our characterization is an analog for Baumslag-Solitar groups.

A subset $S \subseteq \mathbb{Z}[\frac{1}{q}] \rtimes \mathbb{Z}$ is called *k-periodic* if for every $s \in \mathbb{Z}[\frac{1}{q}] \rtimes \mathbb{Z}$, we have (i) $s \in S$ if and only if $s(0,k) \in S$ and (ii) for every $\ell \in \mathbb{Z}$, we have $s \in S$ if and only if $s(q^\ell - q^{\ell+k}, 0) \in S$. In other words, membership in S is insensitive to (i) moving the cursor k positions and (ii) replacing a power of q by another power of q whose exponent differs by k. The set S is *periodic* if it is k-periodic for some $k \ge 1$. We show the following:

▶ **Proposition 6.1.** *A subset $S \subseteq \mathbb{Z}[\frac{1}{q}] \rtimes \mathbb{Z}$ is recognizable if and only if S is periodic.*

The fact that recognizable sets are periodic is an easy exercise. For the converse, we show that the subgroup H of $G = \mathbb{Z}[\frac{1}{q}] \rtimes \mathbb{Z}$ generated by $(0,k)$ and all $(q^\ell - q^{\ell+k}, 0)$ for $\ell \in \mathbb{Z}$ is normal and the quotient G/H is finite. Then, S is recognized by the projection $G \to G/H$.

To decide whether a PE-regular $R \subseteq \mathrm{BS}(1,q)$ is recognizable, we show effective regularity of the set $N \subseteq \{\mathtt{a}\}^*$ of all words $\mathtt{a}^k$ such that R is *not* k-periodic. Then, we just have to check whether N contains all words $\mathtt{a}^k$ with $k \ge 1$, which is clearly decidable. Since R is PE-regular, the set $D = R(G \setminus R)^{-1} \cup (G \setminus R)R^{-1}$ is effectively PE-regular (Proposition 3.6). Then R is not k-periodic if and only if $(0,k) \in D$ or $(q^\ell - q^{\ell+k}, 0) \in D$ for some $\ell \in \mathbb{Z}$. The element $(0,k)$ has the pointed expansion $0^{\triangleleft}0^{k-1}0_{\bullet}$. The pointed expansions of $(q^\ell - q^{\ell+k}, 0)$ for $\ell \in \mathbb{Z}$ are exactly those words obtained from words $-0^r(q-1)^{k-1}0^s$ for $r, s \in \mathbb{N}$ by decorating one of the digits with $^{\triangleleft}$ and with $_{\bullet}$, and removing leading or trailing 0's. Therefore, it is easy to see that $T_1 = \{(0^{\triangleleft}0^{k-1}0_{\bullet}, \mathtt{a}^k) \mid k \ge 1\}$ and $T_2 = \{(\mathsf{pe}((q^\ell - q^{\ell+k}, 0)), \mathtt{a}^k) \mid \ell \in \mathbb{Z},\ k \ge 1\}$ are rational transductions. This implies that $N = T_1(\mathsf{pe}(D)) \cup T_2(\mathsf{pe}(D)) \subseteq \mathtt{a}^*$ is effectively regular. Then clearly, R is not k-periodic if and only if $\mathtt{a}^k \in N$.

References

1 Nathalie Aubrun and Jarkko Kari. Tiling problems on Baumslag-Solitar groups. In *Proceedings of Machines, Computations and Universality 2013 (MCU 2013)*, pages 35–46, 2013. `doi:10.4204/EPTCS.128.12`.

2 Laurent Bartholdi and Pedro V. Silva. Rational subsets of groups. *CoRR*, abs/1012.1532, 2010. Chapter 23 of the handbook AutoMathA (to appear). `arXiv:1012.1532`.

3 Gilbert Baumslag and Donald Solitar. Some two-generator one-relator non-Hopfian groups. *Bulletin of the American Mathematical Society*, 68(3):199–201, 1962. doi:10.1090/S0002-9904-1962-10745-9.

4 Galina Aleksandrovna Bazhenova. Rational sets in finitely generated nilpotent groups. *Algebra and Logic*, 39(4):215–223, 2000. doi:10.1007/BF02681647.

5 Jason P. Bell, Kathryn Hare, and Jeffrey Shallit. When is an automatic set an additive basis? *Proceedings of the American Mathematical Society, Series B*, 5(6):50–63, 2018. doi:10.1090/bproc/37.

6 Michèle Benois. Parties rationnelles du groupe libre. *CR Acad. Sci. Paris*, 269:1188–1190, 1969.

7 Jean Berstel. *Transductions and Context-Free Languages*. Teubner, 1979.

8 Valérie Berthé and Michel Rigo, editors. *Combinatorics, automata, and number theory*, volume 135 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 2010.

9 Laura Ciobanu and Murray Elder. Solutions sets to systems of equations in hyperbolic groups are EDT0L in PSPACE. In *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, pages 110:1–110:15, 2019. doi:10.4230/LIPIcs.ICALP.2019.110.

10 Jordi Delgado Rodríguez. *Extensions of free groups: algebraic, geometric, and algorithmic aspects*. PhD thesis, Universitat Politècnica de Catalunya. Facultat de Matemàtiques i Estadística, 2017.

11 Volker Diekert, Claudio Gutierrez, and Christian Hagenah. The existential theory of equations with rational constraints in free groups is PSPACE -complete. *Information and Computation*, 202(2):105–140, 2005. doi:10.1016/j.ic.2005.04.002.

12 Volker Diekert, Olga Kharlampovich, Markus Lohrey, and Alexei G. Myasnikov. Algorithmic Problems in Group Theory (Dagstuhl Seminar 19131). *Dagstuhl Reports*, 9(3):83–110, 2019. doi:10.4230/DagRep.9.3.83.

13 Volker Diekert and Jürn Laun. On computing geodesics in Baumslag-Solitar groups. *International Journal on Algebra and Computation*, 21(1-2):119–145, 2011. doi:10.1142/S0218196711006108.

14 Volker Diekert, Alexei G. Myasnikov, and Armin Weiß. Conjugacy in Baumslag's group, generic case complexity, and division in power circuits. In *Proceedings of 11th Latin American Symposium on Theoretical Informatics (LATIN 2014)*, pages 1–12, 2014. doi:10.1007/978-3-642-54423-1_1.

15 Volker Diekert, Igor Potapov, and Pavel Semukhin. Decidability of membership problems for flat rational subsets of $\mathrm{GL}(2,\mathbb{Q})$ and singular matrices, 2019. arXiv:1910.02302.

16 F. A. Dudkin and A. V. Treyer. Knapsack problem for Baumslag–Solitar groups. *Siberian Journal of Pure and Applied Mathematics*, 18:43–55, 2018. doi:10.33048/pam.2018.18.404.

17 Murray Elder. A linear-time algorithm to compute geodesics in solvable Baumslag–Solitar groups. *Illinois Journal of Mathematics*, 54(1):109–128, 2010. doi:10.1215/ijm/1299679740.

18 Murray Elder, Gillian Elston, and Gretchen Ostheimer. On groups that have normal forms computable in logspace. *Journal of Algebra*, 381:260–281, 2013. doi:10.1016/j.jalgebra.2013.01.036.

19 Seymour Ginsburg and Edwin H. Spanier. Bounded regular sets. *Proceedings of the American Mathematical Society*, 17(5):1043–1049, 1966. doi:10.2307/2036087.

20 Oscar H. Ibarra. Reversal-bounded multicounter machines and their decision problems. *Journal of the ACM*, 25(1):116–133, 1978. doi:10.1145/322047.322058.

21 Ilya Kapovich and Alexei Myasnikov. Stallings foldings and subgroups of free groups. *Journal of Algebra*, 248(2):608–668, 2002. doi:10.1006/jabr.2001.9033.

22 Olga Kharlampovich, Laura López, and Alexei Miasnikov. Diophantine problem in some metabelian groups, 2019. arXiv:1903.10068.

23 Dexter Kozen. Lower bounds for natural proof systems. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FOCS 1977)*, pages 254–266, 1977. doi:10.1109/SFCS.1977.16.

24 Markus Lohrey. The rational subset membership problem for groups: a survey. In C. M. Campbell, M. R. Quick, E. F. Robertson, and C. M. Roney-Dougal, editors, *Groups St Andrews 2013*, volume 422 of *Lond. Math. S.*, pages 368–389, Cambridge, United Kingdom, 2016. Cambridge University Press. doi:10.1017/CBO9781316227343.024.

25 Markus Lohrey and Géraud Sénizergues. Rational subsets in HNN-extensions and amalgamated products. *International Journal on Algebra and Computation*, 18(1):111–163, 2008. doi:10.1142/S021819670800438X.

26 Markus Lohrey and Georg Zetzsche. Knapsack in metabelian Baumslag-Solitar groups, 2020. arXiv:2002.03837.

27 Alexei Myasnikov, Andrey Nikolaev, and Alexander Ushakov. Knapsack problems in groups. *Mathematics of Computation*, 84:987–1016, 2015. doi:10.1090/S0025-5718-2014-02880-9.

28 David Robinson. *Parallel Algorithms for Group Word Problems*. PhD thesis, Department of Mathematics, University of Califoria, San Diego, 1993.

29 N. S. Romanovskiĭ. Some algorithmic problems for solvable groups. *Algebra and Logic*, 13:13–16, 1974. doi:10.1007/BF01462922.

30 N. S. Romanovskiĭ. The occurrence problem for extensions of abelian groups by nilpotent groups. *Siberian Mathematical Journal*, 21:273–276, 1980. doi:10.1007/BF00968275.

31 Géraud Sénizergues. On the rational subsets of the free group. *Acta Informatica*, 33(3):281–296, 1996. doi:10.1007/s002360050045.

32 John C Shepherdson. The reduction of two-way automata to one-way automata. *IBM Journal of Research and Development*, 3(2):198–200, 1959. doi:10.1147/rd.32.0198.

33 Pedro V. Silva. Free group languages: Rational versus recognizable. *RAIRO—Theoretical Informatics and Applications*, 38(1):49–67, 2004. doi:10.1051/ita:2004003.

34 Pedro V. Silva. An automata-theoretic approach to the study of fixed points of endomorphisms. In Ventura E. González-Meneses J., Lustig M., editor, *Algorithmic and Geometric Topics Around Free Groups and Automorphisms*, Advanced Courses in Mathematics—CRM Barcelona, pages 1–42. Birkhäuser, 2017. doi:10.1007/978-3-319-60940-9_1.

35 Armin Weiß. *On the Complexity of Conjugacy in Amalgamated Products and HNN Extensions*. PhD thesis, Institut für Formale Methoden der Informatik, Universität Stuttgart, 2015. doi:10.18419/opus-3538.

36 Herbert S. Wilf. A circle-of-lights algorithm for the “money-changing problem”. *The American Mathematical Monthly*, 85(7):562–565, 1978. doi:10.2307/2320864.

On Polynomial Recursive Sequences

Michaël Cadilhac
DePaul University, Chicago, IL, USA
michael@cadilhac.name

Filip Mazowiecki
Max Planck Institute for Software Systems, Saarland Informatics Campus, Saarbücken, Germany
filipm@mpi-sws.org

Charles Paperman
Université de Lille, Villeneuve d'Ascq, France
charles.paperman@univ-lille.fr

Michał Pilipczuk
University of Warsaw, Poland
michal.pilipczuk@mimuw.edu.pl

Géraud Sénizergues
Université de Bordeaux, France
geraud.senizergues@u-bordeaux.fr

Abstract

We study the expressive power of *polynomial recursive sequences*, a nonlinear extension of the well-known class of linear recursive sequences. These sequences arise naturally in the study of nonlinear extensions of weighted automata, where (non)expressiveness results translate to class separations. A typical example of a polynomial recursive sequence is $b_n = n!$. Our main result is that the sequence $u_n = n^n$ is not polynomial recursive.

2012 ACM Subject Classification Theory of computation → Models of computation

Keywords and phrases recursive sequences, expressive power, weighted automata, higher-order pushdown automata

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.117

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

Related Version `https://arxiv.org/abs/2002.08630`

Funding *Michał Pilipczuk*: This work is a part of project TOTAL that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme, grant agreement No. 677651.

Acknowledgements We thank Maria Donten-Bury for suggesting the proof of Theorem 11 presented here. This proof replaced our previous more elaborate and less transparent argument. We also thank James Worrell, David Purser and Markus Whiteland for helpful comments. The research for this work was carried out in part at the Autobóz Research Camp in 2019 in Firbush, Scotland. Finally, we thank the participants of the automata seminar at the University of Warsaw for an insightful discussion on the class of rational recursive sequences (considered in Section 7).

1 Introduction

Sequences defined recursively arise naturally in many areas, particularly in mathematics and computer science. One of the most studied classes is that of *linear recursive sequences*. Such sequences are defined by fixing the values of the first k elements, while every subsequent

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 117; pp. 117:1–117:17
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

element can be obtained as a linear combination of the k elements preceding it. The most famous example is the Fibonacci sequence, defined by setting $f_0 = 0$, $f_1 = 1$, and the recurrence relation $f_{n+2} = f_{n+1} + f_n$.

It is well known that every linear recursive sequence can be defined by a system of k jointly recursive sequences, where for every sequence we fix the initial value and provide a recurrence relation expressing the $(n+1)$st element as a linear combination of the nth elements of all the sequences [15]. For example, to define the Fibonacci sequence f_n in this way, one needs one auxiliary sequence: we set $f_0 = 0$, $g_0 = 1$, and postulate

$$\begin{cases} f_{n+1} = g_n, \\ g_{n+1} = f_n + g_n. \end{cases} \tag{1}$$

In this paper we study *polynomial recursive sequences* over rational numbers that generalise linear recursive sequences. They are defined by systems of sequences like (1), but on the right hand side we allow arbitrary polynomial expressions, rather than just linear combinations. For example, the sequence $b_n = n!$ can be defined in this way using one auxiliary sequence: we may set $b_0 = c_0 = 1$ and write

$$\begin{cases} b_{n+1} = b_n \cdot c_n, \\ c_{n+1} = c_n + 1. \end{cases} \tag{2}$$

Thus, the recurrence relation uses two polynomials: $P_1(x_1, x_2) = x_1 x_2$ and $P_2(x_1, x_2) = x_2 + 1$.

The two classes of linear and polynomial recursive sequences appear naturally in automata theory, and in particular in connection with weighted automata and higher-order pushdown automata. *Weighted automata* over the rational semiring are a quantitative variant of finite automata that assign rational numbers to words [10]. In the special case of a 1-letter alphabet, each word can be identified with its length. Then a weighted automaton defines a mapping from natural numbers (possible lengths) to rationals, and this can be seen as a sequence. It is known that sequences definable in this way by weighted automata are exactly the linear recursive sequences [6]. *Pushdown automata of order k* can be used for defining mappings from words to words [21]; in particular, for $k = 2$ and 1-letter alphabets, such automata compute exactly the linear recursive sequences of natural integers [11].

Thus, nonlinear extensions of linear recursive sequences may correspond to nonlinear extensions of weighted automata. For the latter, consider three examples:

- *polynomial recurrent relations* that generalise pushdown automata of order 3 [12, 21];
- *cost-register automata* which arose as a variant of streaming transducers [3, 4];
- *polynomial automata*, connected to reachability problems for vector addition systems [7].

Surprisingly, these three models, although introduced in different contexts, are all equivalent.[1] Moreover, over unary alphabets they define exactly polynomial recursive sequences, in the same fashion as weighted automata (respectively order 2 pushdown automata) over unary alphabets define linear recursive sequences.

The goal of this paper is to study the expressive power of polynomial recursive sequences. Clearly, this expressive power extends that of linear recursive sequences: it is easy to see that every linear recursive sequence has growth bounded by $2^{\mathcal{O}(n)}$, while already the sequence $b_n = n!$ grows faster. In fact, already the recurrence relation $a_0 = 2$, $a_{n+1} = (a_n)^2$ defines

[1] This is a simple but technical observation as the three models are essentially syntactically equivalent. Throughout the paper we will use the name *cost-register automata* to refer to all three models.

the sequence 2^{2^n}, whose growth is doubly-exponential. However, there are well-known integer sequences related to these examples for which definability as a polynomial recursive sequence seems much less clear. The first example is the sequence $u_n = n^n$. The second example is the sequence of Catalan numbers $C_n = \frac{1}{n+1}\binom{2n}{n}$. Note that by Stirling's approximation, n^n is asymptotically very close to $n!$, while C_n is, up to factors polynomial in n, roughly equal to 4^n. For these reasons, simple asymptotic considerations cannot prove the sequences $u_n = n^n$ and C_n to be not polynomial recursive. Recall that the Catalan numbers admit multiple combinatorial interpretations, which can be used to derive the recurrence formulas $C_{n+1} = \sum_{i=0}^{n} C_i C_{n-i}$ and $(n+2)C_{n+1} = (4n+2)C_n$. Note that these formulas are *not* of the form of recurrence formulas considered in this work. Additionally, it is known that Catalan numbers C_n are *not* linear recursive (see e.g. [8]), despite having growth $2^{\mathcal{O}(n)}$.

Our results. We show that both the sequence of Catalan numbers C_n and the sequence $u_n = n^n$ are not polynomial recursive. For this, we present two techniques for proving that a sequence is *not* polynomial recursive. The first technique for Catalan numbers is number-theoretical: we show that a polynomial recursive sequence of integers is ultimately periodic modulo any large enough prime. The second technique for n^n is more algebraic in nature: we show that for every polynomial recursive sequence there exists $k \in \mathbb{N}$ such that every k consecutive elements of the sequence satisfy a nontrivial polynomial equation. The fact that $u_n = n^n$ is not polynomial recursive is our main result. These inexpressibility results were announced without proofs by the fifth coauthor in an invited talk in 2007 [21]. The present paper contains proofs and extensions of these results.

Applications. The discussed models of cost-register automata [12, 4, 7] are not the only nonlinear extensions of weighted automata that appear in the literature. We are aware of at least two more extensions: weighted context-free grammars [5, 8] and weighted MSO logic [9, 17]. As it happens, over the 1-letter alphabet, weighted context-free grammars can define Catalan numbers, and weighted MSO logic can define n^n. Therefore, as a corollary of our results we show that functions expressible in pushdown-automata of level 4, weighted context-free grammars and weighted MSO logic are not always expressible in the class of cost-register automata.

The class of holonomic sequences is another extension of linear recursive sequences [16]. These sequences are defined recursively with one sequence, but the coefficients in the recursion are polynomials of the element's index. For example, $b_n = 1$ and $b_{n+1} = (n+1)b_n$ defines $b_n = n!$. The expressiveness of this class has also been studied and in particular the sequence n^n is known to be not in the class of holonomic sequences [14]. As a corollary of our results one can show that there are no inclusions between the classes of holonomic sequences and polynomial recursive sequences. On the one hand every holonomic sequence is asymptotically bounded by $2^{p(n)}$ for some polynomial p [16], and the sequence $a_n = 2^{2^n}$ is polynomial recursive. On the other hand, Catalan numbers admit a definition as a holonomic sequence: $C_0 = 1$ and $(n+2)C_{n+1} = (4n+2)C_n$. In Section 7 we discuss the class of *rational recurrence sequences* that generalises both holonomic and polynomial recursive sequences.

Organisation. In Section 2 we give basic definitions and examples of linear and polynomial recursive sequences. In Section 3 we show that the definition of polynomial recursive sequences requires a system of sequences and, unlike linear recursive sequences, cannot be equivalently defined using only one sequence. Then in Sections 4 and 5 we show that the sequence of Catalan numbers C_n and the sequence $u_n = n^n$ are not polynomial recursive. In Section 6 we explain in details our corollaries for weighted automata. We conclude in Section 7.

2 Preliminaries

By $\mathbb{N}$ we denote the set of nonnegative integers. A *sequence* over a set $\mathbb{D}$ is a function $u\colon \mathbb{N} \to \mathbb{D}$; all the sequences considered in this work are over the field of rationals $\mathbb{Q}$. We use the notation $\langle u_n \rangle_{n\in\mathbb{N}}$ for elements of sequences, where $u_n = u(n)$. Also, we use bold-face letters as a short-hand for sequences, e.g., $\mathbf{u} = \langle u_n \rangle_{n\in\mathbb{N}}$.

We now introduce the two main formalisms for describing sequences: linear recursive sequences and polynomial recursive sequences.

Linear recursive sequences. A *k-variate linear form* (or *linear form* if k is irrelevant) over $\mathbb{Q}$ is a function $L\colon \mathbb{Q}^k \to \mathbb{Q}$ of the form

$$L(x_1, \dots, x_k) = a_1x_1 + \dots + a_kx_k$$

for some $a_1, \dots, a_k \in \mathbb{Q}$. A sequence of rationals $\mathbf{u}$ is a *linear recursive sequence* if there exist $k \in \mathbb{N}$ and a k-variate linear form L such that $\mathbf{u}$ satisfies the recurrence relation

$$u_{n+k} = L(u_n, \dots, u_{n+k-1}) \qquad \text{for all } n \in \mathbb{N}. \tag{3}$$

Observe that such a sequence is uniquely determined by the form L and its first k elements: $u_0, \dots, u_{k-1} \in \mathbb{Q}$. The minimal k for which a description of $\mathbf{u}$ as in (3) can be given is called the *order* of $\mathbf{u}$. For example, Fibonacci numbers are uniquely defined by the recurrence relation $f_{n+2} = f_{n+1} + f_n$ and starting elements $f_0 = 0$, $f_1 = 1$. Note that this recurrence relation corresponds to the linear form $L(x_1, x_2) = x_1 + x_2$.

We now present a second definition of linear recursive sequences which, as we will explain, is equivalent to the first definition. Suppose $\mathbf{u}^1, \mathbf{u}^2, \dots, \mathbf{u}^k$ are sequences of rationals. We say that these sequences *satisfy a system of linear recurrence equations* if there are k-variate linear forms $L_1, \dots, L_k$ such that:

$$\begin{cases} u^1_{n+1} = L_1(u^1_n, \dots, u^k_n), \\ \vdots \\ u^k_{n+1} = L_k(u^1_n, \dots, u^k_n). \end{cases} \tag{4}$$

for all $n \in \mathbb{N}$. Note that such a system can be equivalently rewritten in the matrix form

$$\vec{u}_{n+1} = M\vec{u}_n$$

where $\vec{u}_n = (u^1_n, \dots, u^k_n)^\mathsf{T}$ and M is the $k \times k$ matrix over $\mathbb{Q}$ such that $M\vec{x} = (L_1(\vec{x}), \dots, L_k(\vec{x}))^\mathsf{T}$ for all $\vec{x} \in \mathbb{Q}^k$. Note that then $\vec{u}_n = M^n\vec{u}_0$ for all $n \in \mathbb{N}$.

It is well known that systems of linear recurrence equations can be equivalently used to define linear recursive sequences, as explained in the following result.

▶ **Proposition 1** ([15]). *A sequence $\mathbf{u}$ is a linear recursive sequence if and only if there exists $k \in \mathbb{N}$ and sequences $\mathbf{u}^1, \dots, \mathbf{u}^k$ that satisfy a system of linear recurrence equations, where $\mathbf{u}^1 = \mathbf{u}$. Moreover, the smallest k for which this holds is the order of $\mathbf{u}$.*

To get more accustomed with this equivalent definition, let us consider the sequence $a_n = n^2$. Since $(n+1)^2 = n^2 + 2n + 1$, we consider two auxiliary sequences $b_n = n$ and $c_n = 1$. The initial values of these sequences are $a_0 = b_0 = 0$ and $c_0 = 1$. Thus, a_n can be defined by providing these initial values together with a system of linear equations

$$\begin{cases} a_{n+1} = a_n + 2b_n + c_n, \\ b_{n+1} = b_n + c_n, \\ c_{n+1} = c_n. \end{cases} \tag{5}$$

In the matrix form, we could equivalently write that $(a_n, b_n, c_n)^\intercal = M^n\vec{e}$, where

$$M = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}, \; \vec{e} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}.$$

It can be readily verified that a_n is also defined by the recurrence $a_{n+3} = 3a_{n+2} - 3a_{n+1} + a_n$.

The difference between the two definitions is that in (3) we have only one sequence, but the depth of the recursion can be any k. Conversely, in (4) we are allowed to have k sequences, but the depth of recursion is 1. The equivalence provided by Proposition 1 is quite convenient and is often used in the literature, see e.g. [20].

We give a short proof of Proposition 1, different from the proof in [15]. The reason is that this proof provides us with intuition that will turn out to be useful later on.

Proof of Proposition 1. For the left-to-right implication, suppose $\mathbf{u}$ is a linear recursive sequence of order k; say it is defined by the recursive formula $u_{n+k} = L(u_n, \ldots, u_{n+k-1})$, where L is a k-variate linear form. Define the sequences $\mathbf{u}^1, \ldots, \mathbf{u}^k$ by setting

$$u^i_n := u_{n+i-1} \qquad \text{for all } i \in \{1, \ldots, k\} \text{ and } n \in \mathbb{N}.$$

Then $\mathbf{u}^1 = \mathbf{u}$ and the sequences $\mathbf{u}^1, \ldots, \mathbf{u}^k$ satisfy the system of equations as in (4), where $L_k = L$ and $L_i(x_1, \ldots, x_k) = x_{i+1}$ for $i \in \{1, \ldots, k-1\}$.

For the right-to-left implication, suppose that there exist $k \in \mathbb{N}$ and sequences $\mathbf{u}^1, \ldots, \mathbf{u}^k$ that satisfy the system of equations (4) for some linear forms $L_1, \ldots, L_k$, such that $\mathbf{u} = \mathbf{u}^1$. Let M be a $k \times k$ matrix over $\mathbb{Q}$ that encodes the linear forms $L_1, \ldots, L_k$; that is, $\vec{u}_n = M^n\vec{u}_0$, where $\vec{u}_n = (u^1_n, \ldots, u^k_n)^\intercal \in \mathbb{Q}^k$. Consider the linear map $R\colon \mathbb{Q}^k \to \mathbb{Q}^{k+1}$ defined as

$$R(\vec{x}) = (\ \vec{e}M^0\vec{x}\,,\ \vec{e}M^1\vec{x}\,,\ \ldots\ ,\ \vec{e}M^k\vec{x}\ )^\intercal,$$

where $\vec{e} = (1, 0, \ldots, 0) \in \mathbb{Q}^k$. Note that

$$R(\vec{u}_n) = (u^1_n, u^1_{n+1}, \ldots, u^1_{n+k}) = (u_n, u_{n+1}, \ldots, u_{n+k}) \qquad \text{for all } n \in \mathbb{N}. \tag{6}$$

Observe that R is a linear map from $\mathbb{Q}^k$ to $\mathbb{Q}^{k+1}$, hence the image of R is a linear subspace of $\mathbb{Q}^{k+1}$ of co-dimension at least 1. Hence, there exists a nonzero linear form $K\colon \mathbb{Q}^{k+1} \to \mathbb{Q}$ such that $\operatorname{im} R \subseteq \ker K$, or equivalently $K(R(\vec{x})) = 0$ for all $\vec{x} \in \mathbb{Q}^k$. By (6), we have

$$K(u_n, u_{n+1}, \ldots, u_{n+k}) = 0 \qquad \text{for all } n \in \mathbb{N}. \tag{7}$$

Let $a_0, a_1, \ldots, a_k \in \mathbb{Q}$ be such that

$$K(x_0, \ldots, x_k) = a_0x_0 + \ldots + a_kx_k.$$

Since K is nonzero there exists the largest index t such that $a_t \neq 0$. From (7) we infer that

$$u_{n+t} = -\frac{a_{t-1}}{a_t} \cdot u_{n+t-1} - \frac{a_{t-2}}{a_t} \cdot u_{n+t-2} - \ldots - \frac{a_0}{a_t} \cdot u_n \qquad \text{for all } n \in \mathbb{N},$$

so $\mathbf{u}$ is a linear recursive sequence of order at most t. ◀

▶ Remark 2. One could imagine setting up all the definitions presented above using *affine forms* instead of linear forms, that is, functions $A\colon \mathbb{Q}^k \to \mathbb{Q}$ of the form

$$A(x_1, \ldots, x_k) = a_1x_1 + \ldots + a_kx_2 + c,$$

where $a_1, \ldots, a_k, c \in \mathbb{Q}$. However, as we may always add constant sequences to the system of recurrence equations defining a sequence, considering affine forms does not increase the expressive power. In fact, from Proposition 1 it can be easily derived that we obtain exactly the same class of linear recursive sequences, regardless of whether we use linear or affine forms in both definitions.

Poly-recursive sequences. We now generalise the concept of linear recursive sequences by allowing polynomial functions instead of only linear forms. The starting point of the generalisation is the definition via a system of recurrence equations, as in (4).

▶ **Definition 3.** *A sequence of rationals* $\mathbf{u}$ *is* polynomial recursive *(or* poly-recursive *for short) if there exist* $k \in \mathbb{N}$*, sequences of rationals* $\mathbf{u}^1, \ldots, \mathbf{u}^k$ *satisfying* $\mathbf{u} = \mathbf{u}^1$*, and polynomials* $P_1, \ldots, P_k \in \mathbb{Q}[x_1, \ldots, x_k]$ *such that for all* $n \in \mathbb{N}$*, we have*

$$\begin{cases} u^1_{n+1} = P_1(u^1_n, \ldots, u^k_n), \\ \vdots \\ u^k_{n+1} = P_k(u^1_n, \ldots, u^k_n). \end{cases} \tag{8}$$

Again, notice that polynomials $P_1, \ldots, P_k$ and the initial values $u^1_0, \ldots u^k_0$ uniquely determine the sequences $\mathbf{u}^1, \ldots, \mathbf{u}^k$, hence in particular the sequence $\mathbf{u} = \mathbf{u}^1$.

Let us examine a few examples. First, recall the sequences $a_n = 2^{2^n}$ and $b_n = n!$ defined in Section 1. Another example is the sequence $d_n = 2^{n^2}$. Since $2^{(n+1)^2} = 2^{n^2+2n+1}$, we define $d_0 = e_0 = 1$ and let

$$\begin{cases} d_{n+1} = d_n \cdot (e_n)^2 \cdot 2, \\ e_{n+1} = e_n \cdot 2. \end{cases}$$

The polynomials used in the last definition are $P_1(x_1, x_2) = 2x_1(x_2)^2$ and $P_2(x_1, x_2) = 2x_2$. Notice that this idea can be easily generalised to define any sequence of the form $r^{Q(n)}$, where r is a rational number and Q is a polynomial with rational coefficients. We remark that all three sequences $a_n = 2^{2^n}$, $b_n = n!$, $d_n = 2^{n^2}$ are not linear recursive for simple asymptotic reasons (from the discussion in Section 1).

3 Simple poly-recursive sequences

The following notion is a natural generalisation of the definition (3) of linear recursive sequences to the setting of recurrences defined using polynomials.

▶ **Definition 4.** *A sequence of rationals* $\mathbf{u}$ *is* simple poly-recursive *if there exists* $k \in \mathbb{N}$ *and a polynomial* $P \in \mathbb{Q}[x_1, x_2, \ldots, x_k]$ *such that*

$$u_{n+k} = P(u_n, u_{n+1}, \ldots, u_{n+k-1}) \qquad \textit{for all } n \in \mathbb{N}. \tag{9}$$

Again, note that if $\mathbf{u}$ is simple poly-recursive as above, then the polynomial P and the first k values $u_0, \ldots, u_{k-1}$ uniquely determine the sequence $\mathbf{u}$.

Clearly, every linear recursive sequence is a simple poly-recursive sequence. In fact, by Proposition 1 and Remark 2, the two notions would coincide if we required that the polynomial P in the definition above has degree at most 1. On the other hand, observe that the same construction as in the first paragraph of the proof of Proposition 1 shows that every simple poly-recursive sequence is poly-recursive. We now prove that this inclusion is strict.

▶ **Theorem 5.** *The sequence* $b_n = n!$ *is not simple poly-recursive.*

Proof. Towards a contradiction, suppose there is $k \in \mathbb{N}$ and a polynomial $P \in \mathbb{Q}[x_1, \ldots, x_k]$ such that

$$b_{n+k} = P(b_n, b_{n+1}, \ldots, b_{n+k-1}) \qquad \text{for all } n \in \mathbb{N}. \tag{10}$$

Let us write

$$P = Q + A,$$

where $Q, A \in \mathbb{Q}[x_1, \ldots, x_k]$ are such that A is the sum of all the monomials in the expansion of P that have degree at most 1, while Q is the sum of all the remaining monomials in the expansion of P. Thus, A is an affine form, while every monomial in the expansion of Q has total degree at least 2.

Since A is an affine form, there exists a number $c \in \mathbb{N}$ such that

$$|A(q_1, \ldots, q_k)| < c + c \cdot \max_{1 \leqslant i \leqslant k} |q_i| \qquad \text{for all } q_1, \ldots, q_k \in \mathbb{Q}.$$

Thus, for all $n > 2c$ we have

$$|A(b_n, b_{n+1}, \ldots, b_{n+k-1})| \leqslant c + c \cdot (n + k - 1)! < (n + k)! = b_{n+k}. \tag{11}$$

Since by (10) it follows that

$$Q(b_n, b_{n+1}, \ldots, b_{n+k-1}) = b_{n+k} - A(b_n, b_{n+1}, \ldots, b_{n+k-1}),$$

using (11) we may conclude that for all $n > 2c$ the following inequality holds:

$$0 < Q(b_n, b_{n+1}, \ldots, b_{n+k-1}) < 2b_{n+k}. \tag{12}$$

Let m be the product of all denominators of all the coefficients appearing in the expansion of P into a sum of monomials. Note that for all $n > m$, the number $\tilde{b}_n := \frac{b_n}{m} = \frac{n!}{m}$ is an integer. Furthermore, we have that $\tilde{b}_n$ divides $\tilde{b}_{n'}$ for all $n' \geqslant n$. Since every monomial in the expansion of Q has total degree at least 2, we infer that for all $n > m$, we have

$$\left(\tilde{b}_n\right)^2 \mid Q(b_n, b_{n+1}, \ldots, b_{n+k-1}). \tag{13}$$

By combining (13) with the left inequality of (12), we conclude that for all $n > \max(2c, m)$,

$$Q(b_n, b_{n+1}, \ldots, b_{n+k-1}) \geqslant \left(\tilde{b}_n\right)^2.$$

This bound together with the right inequality of (12) implies that

$$\left(\frac{n!}{m}\right)^2 = \left(\tilde{b}_n\right)^2 < 2b_{n+k} = 2 \cdot (n + k)!.$$

This inequality, however, is not true for every sufficiently large n, a contradiction. ◀

4 Modular periodicity

Recall that a sequence of numbers $\mathbf{r}$ is *ultimately periodic* if there exist $N, k \in \mathbb{N}$ such that for all $n \geqslant N$, we have $r_n = r_{n+k}$. In this section we prove the following periodicity property of poly-recursive sequences, which, by means of contradiction, provides a basic technique for proving that a given sequence is not poly-recursive.

▶ **Theorem 6.** *Suppose $\mathbf{u}$ is a poly-recursive sequence of integers. Then there exists $a \in \mathbb{N}$ such that for every prime $p > a$, the sequence $r_n := u_n \bmod p$ is ultimately periodic.*

Proof. Let $\mathbf{u}$ be defined by the system of recursive equations

$$\begin{cases} u^1_{n+1} = P_1(u^1_n, \ldots, u^k_n), \\ \vdots \\ u^k_{n+1} = P_k(u^1_n, \ldots, u^k_n), \end{cases} \tag{14}$$

where $\mathbf{u}^1, \ldots, \mathbf{u}^k$ are sequences such that $\mathbf{u}^1 = \mathbf{u}$ and $P_1, \ldots, P_k \in \mathbb{Q}[x_1, \ldots, x_k]$.

Without loss of generality we may assume that the initial values $u_0^1, \ldots, u_0^k$ are integers. Indeed, this is certainly the case for $u_0^1 = u_0$, while for every $i > 1$, we may rewrite the system so that it uses the sequence $\widetilde{\mathbf{u}}^i = q_i \cdot \mathbf{u}^i$ instead of $\mathbf{u}^i$, where q_i is the denominator of u_0^i. For this, the starting condition for $\widetilde{\mathbf{u}}^i$ can be set as $\widetilde{u}_0^i = q_i \cdot u_0^i$, which is an integer, in all polynomials $P_1, \ldots, P_k$ we may substitute x_i with x_i/q_i, and the polynomial P_i can be replaced with $q_i \cdot P_i$.

Further, without loss of generality we may assume that all the monomials present in the expansions of all the polynomials $P_1, \ldots, P_k$ have the same total degree $d > 1$. Such polynomials are called homogeneous of degree d and they have the property that $P_i(ax_1, \ldots, ax_k) = a^d P_i(x_1, \ldots, x_k)$ for all $a \in \mathbb{Q}$. Indeed, let $d > 1$ be any integer that is not smaller than the degrees of all the polynomials $P_1, \ldots, P_k$. To the system (14) we add a new sequence $\mathbf{u}^{k+1}$, defined by setting

$$u_0^{k+1} = 1 \qquad \text{and} \qquad u_{n+1}^{k+1} = \left(u_n^{k+1}\right)^d \text{ for } n \in \mathbb{N}.$$

Thus $\mathbf{u}^{k+1}$ is constantly equal to 1. Then each monomial $M(x_1, \ldots, x_k)$ appearing in the expansion of any of the polynomials $P_i(x_1, \ldots, x_k)$ can be replaced by the monomial $M(x_1, \ldots, x_k) \cdot x_{k+1}^{d-t} \in \mathbb{Q}[x_1, \ldots, x_k, x_{k+1}]$, where t is the total degree of M. It is straightforward to see that the modified system of recursive equations still defines $\mathbf{u} = \mathbf{u}^1$, while all monomials appearing in all the polynomials used in it have the same degree d.

After establishing these two assumptions, we proceed to the main proof. Let $a \in \mathbb{N}$ be a positive integer such that the polynomials

$$\tilde{P}_i \coloneqq a \cdot P_i$$

all belong to $\mathbb{Z}[x_1, \ldots, x_k]$, that is, have integer coefficients. For instance, one can take a to be product of all the denominators of all the rational coefficients appearing in the polynomials $P_1, \ldots, P_k$. For all $i \in \{1, \ldots, k\}$ and $n \in \mathbb{N}$, let us define

$$\tilde{u}_n^i \coloneqq a^{\frac{d^n-1}{d-1}} \cdot u_n^i.$$

By a straightforward induction we show that the sequences $\tilde{\mathbf{u}}^1, \ldots, \tilde{\mathbf{u}}^k$ satisfy the system of recursive equations

$$\begin{cases} \tilde{u}_{n+1}^1 = \tilde{P}_1(\tilde{u}_n^1, \ldots, \tilde{u}_n^k), \\ \vdots \\ \tilde{u}_{n+1}^k = \tilde{P}_k(\tilde{u}_n^1, \ldots, \tilde{u}_n^k). \end{cases} \tag{15}$$

Indeed, the induction base is trivial and for the induction step recall that all monomials have the same degree d, hence

$$\tilde{P}_i(\tilde{u}_n^1, \ldots, \tilde{u}_n^k) = a \cdot P_i(a^{\frac{d^n-1}{d-1}} \cdot u_n^1, \ldots, a^{\frac{d^n-1}{d-1}} \cdot u_n^k) = a \cdot a^{\frac{d^{n+1}-d}{d-1}} \cdot u_{n+1}^i = a^{\frac{d^{n+1}-1}{d-1}} \cdot u_{n+1}^i = \tilde{u}_{n+1}^i.$$

Observe that since the initial values $\tilde{u}_0^i = u_0^i$ are integers, and the polynomials $\tilde{P}_i$ have integer coefficients, we can infer that all entries of the sequences $\tilde{\mathbf{u}}^1, \ldots, \tilde{\mathbf{u}}^k$ are integers.

We now show that for every prime $p > a$, the sequence $\mathbf{r}$ defined as $r_n = u_n \bmod p$ is ultimately periodic; this will conclude the proof. For every $i \in \{1, \ldots, k\}$ and $n \in \mathbb{N}$, let

$$\tilde{r}_n^i \coloneqq \tilde{u}_n^i \bmod p.$$

By (15) and the fact that the polynomials $\tilde{P}_i$ have integer coefficients, for every $n \in \mathbb{N}$ the vector of entries $(\tilde{r}^1_{n+1}, \ldots, \tilde{r}^k_{n+1})$ is uniquely determined by the vector $(\tilde{r}^1_n, \ldots, \tilde{r}^k_n)$. Since this vector may take only at most p^k different values, it follows that the sequences $\tilde{\mathbf{r}}^1, \ldots, \tilde{\mathbf{r}}^k$ are ultimately periodic.

Now note that for every $n \in \mathbb{N}$, we have

$$a^{\frac{d^n-1}{d-1}} \cdot r_n \equiv a^{\frac{d^n-1}{d-1}} \cdot u_n = \tilde{u}^1_n \equiv \tilde{r}^1_n \mod p.$$

Since $p > a$ and p is a prime, we have that a and p are coprime. Therefore, there exists an integer b such that $ab \equiv 1 \bmod p$. By multiplying the above congruence by $b^{\frac{d^n-1}{d-1}}$, we have

$$r_n \equiv b^{\frac{d^n-1}{d-1}} \cdot \tilde{r}^1_n \mod p. \tag{16}$$

Observe that the sequence $b_n = b^{\frac{d^n-1}{d-1}}$ satisfies the recursive equation $b_{n+1} = b \cdot (b_n)^d$, hence the sequence $(b_n \bmod p)$ is ultimately periodic. Since $\tilde{\mathbf{r}}^1$ is ultimately periodic as well, from (16) we conclude that the sequence $\mathbf{r}$ is ultimately periodic. ◀

We use Theorem 6 to prove that the Catalan numbers are not poly-recursive. Recall that the nth Catalan number C_n is given by the formula $C_n = \frac{1}{n+1}\binom{2n}{n}$.

Alter and Kubota [2] studied the behaviour of the Catalan numbers modulo primes. It is easy to see (and proved in [2]) that for every prime p, the sequence C_n contains infinitely many numbers divisible by p, and infinitely many numbers not divisible by p. Let a *p-block* be a maximal contiguous subsequence of the sequence C_n consisting of entries divisible by p. The p-blocks can be naturally ordered along the sequence C_n, so let L^p_k be the length of the kth p-block. Then Alter and Kubota proved the following.

▶ **Theorem 7** ([2]). *For every prime $p > 3$ and $k \geqslant 1$, we have*

$$L^p_k = \frac{p^{m+1} - 3}{2},$$

where m is the largest integer such that $\left(\frac{p+1}{2}\right)^m$ divides k.

Note that Theorem 7 in particular implies that for every prime $p > 3$, the sequence C_n contains arbitrary long p-blocks. This means that C_n taken modulo p cannot be ultimately periodic. By combining this with Theorem 6, we conclude the following.

▶ **Corollary 8.** *Catalan numbers are not poly-recursive.*

5 Cancelling polynomials

Consider the following definition, which can be seen as a variation of the definition of simple poly-recursive sequences, which we discussed in Section 3.

▶ **Definition 9.** *A sequence of rationals* $\mathbf{u}$ *admits a* cancelling polynomial *if there exist $k \in \mathbb{N}$ and a nonzero polynomial $P \in \mathbb{Q}[x_0, \ldots, x_k]$ such that*

$$P(u_n, u_{n+1}, \ldots, u_{n+k}) = 0 \qquad \textit{for all } n \in \mathbb{N}.$$

▶ Remark 10. A cancelling polynomial P can be always assumed to have integer coefficients, i.e. to belong to $\mathbb{Z}[x_0, \ldots, x_k]$, because one may multiply P by the product of all denominators that occur in its coefficients.

Observe that the notion of a cancelling polynomial extends the definition of simple poly-recursive sequences (Definition 4) in the following sense: a sequence is simple poly-recursive if and only if it admits a cancelling polynomial $P(x_0, \ldots, x_k)$ whose expansion into a sum of monomials involves only one term containing x_k, namely the monomial x_k itself. This particular form of the considered algebraic constraint was vitally used in the proof of Proposition 5, where we showed that the sequence $b_n = n!$ is not simple poly-recursive. In fact, if one drops this restriction, then it is easy to see that the sequence $b_n = n!$ actually admits a cancelling polynomial: for instance $P(x_0, x_1, x_2) = x_0x_2 - (x_1)^2 - x_0x_1$.

We now prove that the above example is not a coincidence.

▶ **Theorem 11.** *Every poly-recursive sequence admits a cancelling polynomial.*

Proof. The proof follows the same basic idea as the proof of Proposition 1 that we gave in Section 2. The difference is that instead of linear maps we work with maps defined by polynomial functions, hence instead of linear independence we shall work with the notion of algebraic independence.

Recall that if $\mathbb{K} \subseteq \mathbb{L}$ is a field extension, then elements $a_1, \ldots, a_k \in \mathbb{L}$ are *algebraically dependent* over $\mathbb{K}$ if there is a nonzero polynomial $P \in \mathbb{K}[x_1, \ldots, x_k]$ such that $P(a_1, \ldots, a_k) = 0$ in $\mathbb{L}$. We will use the following well-known fact; see e.g. [18, Chapter VIII, Theorem 1.1].

▷ Claim 12. If $\mathbb{K}$ is a field and $k \in \mathbb{N}$, then in the field of rational expressions $\mathbb{K}(x_1, \ldots, x_k)$ every $k + 1$ elements are algebraically dependent over $\mathbb{K}$.

We proceed to the proof of the theorem. Let $\mathbf{u}$ be the poly-recursive sequence in question. By definition, for some $k \in \mathbb{N}$ there are sequences $\mathbf{u}^1, \ldots, \mathbf{u}^k$ and polynomials $P_1, \ldots, P_k \in \mathbb{Q}[x_1, \ldots, x_k]$ such that for all $n \in \mathbb{N}$,

$$\begin{cases} u^1_{n+1} = P_1(u^1_n, \ldots, u^k_n), \\ \vdots \\ u^k_{n+1} = P_k(u^1_n, \ldots, u^k_n). \end{cases}$$

We inductively define polynomials $P^{(t)}_1, \ldots, P^{(t)}_k \in \mathbb{Q}[x_1, \ldots, x_k]$ as follows. For $t = 0$, set

$$P^{(0)}_i(x_1, \ldots, x_k) = x_i \qquad \text{for all } i \in \{1, \ldots, k\},$$

and for $t \geqslant 1$, set

$$P^{(t)}_i(x_1, \ldots, x_k) = P_i(P^{(t-1)}_1(x_1, \ldots, x_k), \ldots, P^{(t-1)}_k(x_1, \ldots, x_k)) \quad \text{for all } i \in \{1, \ldots, k\}.$$

The following claim follows from the construction by a straightforward induction.

▷ Claim 13. For all $n, t \in \mathbb{N}$ and $i \in \{1, \ldots, k\}$, we have $P^{(t)}_i(u^1_n, \ldots, u^k_n) = u^i_{n+t}$.

Consider the polynomials

$$P^{(0)}_1, P^{(1)}_1, \ldots, P^{(k)}_1 \in \mathbb{Q}[x_1, \ldots, x_k].$$

By Claim 12, these polynomials (treated as elements of $\mathbb{Q}(x_1, \ldots, x_k)$) are algebraically dependent over $\mathbb{Q}$, so there exists a nonzero polynomial $Q \in \mathbb{Q}[y_0, y_1, \ldots, y_k]$ such that the polynomial

$$R(x_1,\dots,x_k) = Q(P_1^{(0)}(x_1,\dots,x_k), P_1^{(1)}(x_1,\dots,x_k),\dots,P_1^{(k)}(x_1,\dots,x_k))$$

is identically zero. It now remains to observe that by Claim 13, for every $n \in \mathbb{N}$ we have

$$0 = R(u_n^1,\dots,u_n^k) = Q(u_n^1, u_{n+1}^1,\dots,u_{n+k}^1) = Q(u_n, u_{n+1},\dots,u_{n+k}),$$

hence Q is a cancelling polynomial for $\mathbf{u}$. ◀

▶ Remark 14. Notice that a given polynomial can be the cancelling polynomial of many different sequences. For example, the polynomial $(x_0)^2 - 1$ is a cancelling polynomial of any sequence over $\{-1, 1\}$. In particular, some of those sequences are not ultimately periodic modulo p, for any prime numbers p, and thus are not poly-recursive by Theorem 6. Hence, the converse direction of Theorem 11 does not hold.

We now present an application of Theorem 11 by showing that the sequence $u_n = n^n$ is not poly-recursive. By Theorem 11, it suffices to show that there is no cancelling polynomial for this sequence. Contrary to the reasoning presented in Section 4, where we used off-the-shelf results about modular (non)periodicity of Catalan numbers, proving the nonexistence of a cancelling polynomial for the n^n sequence turns out to be a somewhat challenging task.

We first observe that when we apply a multivariate polynomial to consecutive entries of u_n, we can rewrite the result in another form:

▶ **Lemma 15.** *Let $Z \in \mathbb{Z}[x_0, x_1,\dots,x_k]$ be a nonzero polynomial. Then there exist nonzero polynomials $P_1,\dots,P_m, Q_1,\dots,Q_m \in \mathbb{Z}[x]$ such that the polynomials $P_1,\dots,P_m$ are pairwise different and for every $n \in \mathbb{N}$,*

$$Z\left(n^n, (n+1)^{n+1},\dots,(n+k)^{n+k}\right) = \sum_{i=1}^{m} P_i(n)^n \cdot Q_i(n).$$

Proof. By expanding Z as a sum of monomials, we may write

$$Z(x_0,\dots,x_k) = \sum_{i=1}^{m} c_i \cdot M_i(x_0,\dots,x_j), \tag{17}$$

where for all $i \in \{1,\dots,m\}$, $c_i \neq 0$ and

$$M_i(x_0,\dots,x_k) = \prod_{j=0}^{k} x_j^{d_{i,j}}$$

are pairwise different monomials. Now observe that for every $n \in \mathbb{N}$, we have

$$\begin{aligned} M_i\left(n^n, (n+1)^{n+1},\dots,(n+k)^{n+k}\right) &= \prod_{j=0}^{k} (n+j)^{d_{i,j}\cdot(n+j)} \\ &= \left(\prod_{j=0}^{k} (n+j)^{d_{i,j}}\right)^n \cdot \prod_{j=0}^{k} (n+j)^{d_{i,j}\cdot j}. \end{aligned} \tag{18}$$

Hence, if we define

$$P_i(x) = \prod_{j=0}^{k} (x+j)^{d_{i,j}} \qquad \text{and} \qquad Q_i(x) = c_i \cdot \prod_{j=0}^{k} (x+j)^{d_{i,j}\cdot j},$$

then, by (17) and (18), we conclude that

$$Z\left(n^n,(n+1)^{n+1},\ldots,(n+k)^{n+k}\right)=\sum_{i=1}^{m}P_i(n)^n\cdot Q_i(n)\qquad\text{for all } n\in\mathbb{N},$$

as required. It now suffices to observe that (1) all polynomials P_i and Q_i are nonzero, because $c_i \neq 0$ and the monomial M_i is nonzero, and (2) the polynomials P_i are pairwise different, because they have pairwise different multisets of roots. ◀

With Lemma 15 established, we move to the main result of this section.

▶ **Theorem 16.** *The sequence $u_n = n^n$ is not poly-recursive.*

Proof. Suppose, for the sake of contradiction, that the sequence $u_n = n^n$ is poly-recursive. By Theorem 11 and Remark 10, there exists a nonzero polynomial $Z \in \mathbb{Z}[x_0, x_1, \ldots, x_k]$ that is cancelling for u_n. By Lemma 15, we can find nonzero polynomials $P_1, \ldots, P_m, Q_1, \ldots, Q_m \in \mathbb{Z}[x]$, where $P_1, \ldots, P_m$ are pairwise different, such that

$$\sum_{i=1}^{m}P_i(n)^n\cdot Q_i(n)=0\qquad\text{for all } n\in\mathbb{N}. \tag{19}$$

This system of equations seems somewhat unwieldy due to the presence of the term $P_i(n)^n$, where n is involved both in the base and in the exponent. The following claim formulates the key idea of the proof: if we consider the equations (19) modulo any prime, then the bases and the exponents of these terms can be made independent.

▷ Claim 17. For every prime p and all $a, b \in \mathbb{Z}$ where $b > 0$, it holds that

$$\sum_{i=1}^{m}P_i(a)^b\cdot Q_i(a)\equiv 0 \mod p\ .$$

Proof. Since p and $p-1$ are coprime, there is an $n \in \mathbb{N}$ such that $n > b$, $n \equiv a \bmod p$ and $n \equiv b \bmod p-1$. Thus for any $1 \leqslant i \leqslant m$:

$$Q_i(n)\equiv Q_i(a) \mod p \qquad\text{and}\qquad P_i(n)^n\equiv P_i(a)^n\equiv P_i(a)^b \mod p\ ,$$

the second part holding by Fermat's Little Theorem. The claim now follows by considering equality (19) modulo p. ◁

Let $a \in \mathbb{N}$ and let $D_a = [d_{ij}]_{1\leqslant i,j\leqslant m}$ be the $m \times m$ matrix defined by $d_{ij} = P_j(a)^i$. Since this is essentially a Vandermonde matrix, its determinant has a simple expression, as expressed in the following claim.

▷ Claim 18. Let $S \in \mathbb{Z}[x]$ be defined as

$$S(x)=\prod_{i=1}^{m}P_i(x)\cdot\prod_{1\leqslant i<j\leqslant m}(P_i(x)-P_j(x))\ .$$

Then S is nonzero and $\det(D_a) = S(a)$.

Proof. That S is nonzero follows from the fact that the polynomials P_i are all nonzero and pairwise different.

Now observe that D_a is a Vandermonde matrix with columns consisting of consecutive powers of $P_j(a)$, for $1 \leqslant j \leqslant m$ with columns consisting of consecutive powers of $P_j(a)$, starting with $P_j(a)^1$ (whereas the Vandermonde matrix starts with $P_j(a)^0$). It is well known that the determinant of the Vandermonde matrix $[P_j(a)^{i-1}]_{1\leqslant i,j\leqslant m}$ is

$$\prod_{1\leqslant i<j\leqslant m} (P_i(a) - P_j(a)) \ .$$

Further, multiplying the jth column by $P_j(a)$, for all j, results in the determinant being multiplied by $\prod_{i=1}^{m} P_i(a)$. This proves the claim. ◁

We will need the following classical definition.

▶ **Definition 19.** *Let R be a ring and M be a $m \times m$ matrix over R. The* adjugate matrix *$\widehat{M}$ of M is an $m \times m$ matrix over R that satisfies $\widehat{M}M = \det(M) \cdot I$, where I is the $m \times m$ identity matrix.*

It is well known that the adjugate matrix always exists. Now let $u_a = (Q_1(a), \dots, Q_m(a))^\intercal$. Claim 17 implies that for every prime p,

$$D_a u_a \equiv \vec{0} \mod p,$$

where $\vec{0}$ is the m-dimensional zero vector. By multiplying both sides of this equation by the adjugate matrix of D_a taken over $\mathbb{Z}_p$, we conclude that for every prime p, we have

$$\det(D_a) \cdot u_a \equiv \vec{0} \mod p \qquad \text{for all } a \in \mathbb{N}.$$

This is equivalent to

$$S(a) \cdot Q_i(a) \equiv 0 \mod p \qquad \text{for all } a \in \mathbb{N} \text{ and } 1 \leqslant i \leqslant m. \tag{20}$$

This means that for every prime p and every $1 \leqslant i \leqslant m$, the following assertion holds: every $a \in \mathbb{F}_p$ is a zero of the polynomial $S \cdot Q_i$ considered as a polynomial over $\mathbb{F}_p$.

Recall that the polynomials $S, Q_1, \dots, Q_m \in \mathbb{Z}[x]$ are nonzero. Consider a prime p that is larger than every coefficient occurring in the expansion of the polynomials S, $Q_1, \dots, Q_m$ into sums of monomials, and that is further larger than $\deg(S) + \max_{j\in\{1,\dots,m\}} \deg(Q_j)$. Then the polynomials $S, Q_1, \dots, Q_m$ are nonzero even when regarded as polynomials over $\mathbb{F}_p$, hence the same can be said also about the polynomials $S \cdot Q_i$, for all $1 \leqslant i \leqslant m$. However, by (20), for every $1 \leqslant i \leqslant m$ the polynomial $S \cdot Q_i$ has at least $p > \deg(S) + \deg(Q_i)$ roots over $\mathbb{F}_p$. This is a contradiction. ◀

6 Applications in weighted automata

In this section we discuss the implications of the results we presented in the previous sections for various questions regarding the expressive power of extensions of weighted automata. We will briefly describe the model of weighted automata and focus only on its expressive power. We refer an interested reader to e.g. [1, 10] for an introduction to the area.

Given a semiring $\mathbb{S}$, a *weighted automaton* $\mathcal{A}$ is a tuple $(d, \Sigma, \{M_a\}_{a\in\Sigma}, I, F)$, where:

- $d \in \mathbb{N}$ is the dimension;
- Σ is a finite alphabet;
- every M_a is a $d \times d$ matrix over $\mathbb{S}$; and
- I and F are the initial and the final vector in $\mathbb{S}^d$, respectively.

In this paper we only consider the semiring $\mathbb{S} = \mathbb{Q}$. A weighted automaton defines a function $[\![\mathcal{A}]\!] : \Sigma^* \to \mathbb{S}$ as follows: if $w = a_1 \dots a_n \in \Sigma^*$, then

$$[\![\mathcal{A}]\!](w) \;=\; I^{\mathsf{T}} \cdot M_{a_1} M_{a_2} \dots M_{a_n} \cdot F. \tag{21}$$

Note that when $|\Sigma| = 1$, this definition coincides with (the matrix form of) the definition (4) of linear recursive sequences. Assuming $|\Sigma| = 1$, one can identify each word with its length, which means that a weighted automaton defines a sequence $[\![\mathcal{A}]\!] : \mathbb{N} \to \mathbb{S}$. Therefore, weighted automata recognise exactly linear recursive sequences. See [6] for a broader discussion of the connection between linear recursive sequences and weighted automata.

We now discuss three nonlinear extensions of weighted automata that can be found in the literature. These extensions are studied in different areas and, as far as we are aware, they have never been compared in terms of expressive power before. We show that the results we presented in Sections 4 and 5 can be used to prove separation results, in terms of the expressive power, for some of these classes.

Like in the case of weighted automata, any automaton within the considered classes defines a function $f \colon \Sigma^* \to \mathbb{Q}$, where Σ is the working alphabet. For our purposes, we restrict attention to the case of unary alphabets, that is, $|\Sigma| = 1$. Thus, the three considered classes of extended weighted automata correspond to three separate classes of sequences $f \colon \mathbb{N} \to \mathbb{Q}$, similarly as standard weighted automata correspond to the class of linear recursive sequences.

Cost-register automata (CRA). Cost-register automata (CRA) were introduced in at least three contexts [21, 4, 7]. To avoid technical details, we simply observe that CRAs over unary alphabets recognize exactly poly-recursive sequences, as defined in Definition 3. Since [21, 4, 7] discuss several variants of CRAs, to avoid ambiguity we refer to the definition of a CRA that can be found in [19][2].

Weighted context-free grammars (WCFG). Weighted automata can be equivalently defined as an extension of finite automata, where each translation is labelled by an element of the semiring $\mathbb{S}$ (see e.g. [1]). In short, each run is assigned a value: the semiring product of the labels of all the transitions used in the run. Given a word w, the automaton outputs the semiring sum of the values assigned to all runs accepting w.

Weighted context-free grammars are an extension of context-free grammars in the same way weighted automata are an extension of finite automata. Every grammar rule is assigned a label from $\mathbb{S}$. Then every derivation tree is assigned the semiring product of the labels of all the rules used in the tree. The output for a word w is defined as the semiring sum of all values assigned to derivation trees of w. See e.g. [13] for more details. Here we present only one example from [13] over the semiring $\mathbb{Q}$.

Consider the grammar with one nonterminal X (which is also the starting nonterminal) and one terminal a with the following rules: $X \to a$, $X \to XX$. Both rules are assigned weight 1. Therefore, for every word a^n the output is the number of derivation trees. It is easy to see that if we denote the output on the word a^n by D_n, then D_n is the number of full binary trees with n leaves, which is the sequence of Catalan numbers shifted by one, i.e. $D_0 = 0$ and $D_{n+1} = C_n$. By Corollary 8 and since it is easy to see that poly-recursive sequences are closed under shifts, we conclude the following.

▶ **Corollary 20.** *The class of sequences definable by unary-alphabet WCFGs over $\mathbb{Q}$ is not contained in the class of sequences recognizable by unary-alphabet CRAs over $\mathbb{Q}$.*

[2] The equivalence of CRAs and poly-recursive sequences over a unary alphabet is basically a syntactic translation, if one assumes that CRAs have only one state. Proving that every CRA can be defined by a one state CRA is a simple encoding of states into the registers.

Weighted MSO (WMSO). Weighted MSO logic [9, 17] was introduced as a logic involving weights that intended to capture the expressive power of weighted automata, similarly as finite automata are characterized by MSO. In general, WMSO turns out to be strictly more expressive than weighted automata. We will not define the whole syntax of WMSO, only a simple fragment that does not even use variables. See [9, 17] for the full definition.

Fix the semiring $\mathbb{S} = \mathbb{Q}$. Similarly as for weighted automata, every WMSO formula φ over $\mathbb{Q}$ defines a function $[\![\varphi]\!] : \Sigma^* \to \mathbb{Q}$. As for atomic formulas, every $c \in \mathbb{Q}$ is an atomic formula that defines the constant function $[\![c]\!](w) = c$. Instead of the boolean connectives $\vee$ and $\wedge$, WMSO formulas can be added using $+$ and multiplied using $\cdot$, with the obvious semantics. Instead of having the existential quantifier $\exists_x$ and the universal quantifier $\forall_x$, we have the sum quantifier $\sum_x$ and the product quantifier $\prod_x$. Then

$$\left[\!\!\left[\sum_x \varphi\right]\!\!\right](w) = \sum_{i=1}^{n} [\![\varphi[x \to a_i]]\!](w) \qquad \text{for all } w = a_1 \dots a_n \in \Sigma^*,$$

and similarly for $[\![\prod_x \varphi]\!](w)$. For example, $[\![\sum_x 1]\!](a^n) = n$. It follows that

$$\left[\!\!\left[\prod_x \sum_y 1\right]\!\!\right](a^n) = n^n.$$

This proves that the sequence n^n can be defined in unary-alphabet WMSO over $\mathbb{Q}$, so by Theorem 16 we may conclude the following.

▶ **Corollary 21.** *The class of sequences definable in unary-alphabet WMSO over $\mathbb{Q}$ is not contained in the class of sequences recognizable by unary-alphabet CRAs over $\mathbb{Q}$.*

7 Conclusion

We proved that two sequences, the Catalan numbers C_n and $u_n = n^n$, are not polynomial recursive. For this, we exhibited two properties that poly-recursive sequences always satisfy: ultimate periodicity modulo large prime numbers and admitting a cancelling polynomial.

Going further than poly-recursive sequences, one can consider the class of *rational recursive sequences*. These are specified like polynomial recursive sequences (Definition 3) but on the right hand side of the system of equations (8) we allow the P_i's to be taken from the field of fractions of the polynomial ring. That is, each P_i is of the form $P_i(x_1, \dots, x_k) = \frac{Q_i(x_1,\dots,x_k)}{R_i(x_1,\dots,x_k)}$, where $Q_i, R_i \in \mathbb{Q}[x_1, \dots, x_k]$ and $R_i \neq 0$.

This class extends both poly-recursive sequences and holonomic sequences (see Section 1). For example one can express the sequence of Catalan numbers, since $C_{n+1} = \frac{4n+2}{n+2} \cdot C_n$ and an ancillary sequence can hold the value n. On the other hand, the proof of the existence of cancelling polynomials for poly-recursive sequences (Theorem 11) carries over to rational recursive sequences. In particular, $u_n = n^n$ is not even rational recursive.

This discussion points to the notion of rational recursive sequences as a natural object for future research.

References

1 Shaull Almagor, Udi Boker, and Orna Kupferman. What's decidable about weighted automata? In *Automated Technology for Verification and Analysis, 9th International Symposium, ATVA 2011, Taipei, Taiwan, October 11-14, 2011. Proceedings*, pages 482–491, 2011. doi:10.1007/978-3-642-24372-1_37.

2 Ronald Alter and K. K. Kubota. Prime and prime power divisibility of Catalan numbers. *Journal of Combinatorial Theory, Series A*, 15(3):243–256, 1973. doi:10.1016/0097-3165(73)90072-1.

3 Rajeev Alur and Pavol Cerný. Expressiveness of streaming string transducers. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2010, December 15-18, 2010, Chennai, India*, pages 1–12, 2010. doi:10.4230/LIPIcs.FSTTCS.2010.1.

4 Rajeev Alur, Loris D'Antoni, Jyotirmoy V. Deshmukh, Mukund Raghothaman, and Yifei Yuan. Regular functions and cost register automata. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013*, pages 13–22, 2013. doi:10.1109/LICS.2013.65.

5 James K Baker. Trainable grammars for speech recognition. *The Journal of the Acoustical Society of America*, 65(S1):S132–S132, 1979.

6 Corentin Barloy, Nathanaël Fijalkow, Nathan Lhote, and Filip Mazowiecki. A robust class of linear recurrence sequences. In *28th EACSL Annual Conference on Computer Science Logic, CSL 2020, January 13-16, 2020, Barcelona, Spain*, pages 9:1–9:16, 2020. doi:10.4230/LIPIcs.CSL.2020.9.

7 Michael Benedikt, Timothy Duff, Aditya Sharad, and James Worrell. Polynomial automata: Zeroness and applications. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12, 2017. doi:10.1109/LICS.2017.8005101.

8 Vijay Bhattiprolu, Spencer Gordon, and Mahesh Viswanathan. Extending Parikh's theorem to weighted and probabilistic context-free grammars. In *Quantitative Evaluation of Systems - 14th International Conference, QEST 2017, Berlin, Germany, September 5-7, 2017, Proceedings*, pages 3–19, 2017. doi:10.1007/978-3-319-66335-7_1.

9 Manfred Droste and Paul Gastin. Weighted automata and weighted logics. *Theor. Comput. Sci.*, 380(1-2):69–86, 2007. doi:10.1016/j.tcs.2007.02.055.

10 Manfred Droste, Werner Kuich, and Heiko Vogler. *Handbook of Weighted Automata.* Springer, 1st edition, 2009.

11 Julien Ferté, Nathalie Marin, and Géraud Sénizergues. Word-mappings of level 2. *Theory Comput. Syst.*, 54(1):111–148, 2014. doi:10.1007/s00224-013-9489-5.

12 S. Fratani and Géraud Sénizergues. Iterated pushdown automata and sequences of rational numbers. *Ann. Pure Appl. Logic*, 141(3):363–411, 2006. doi:10.1016/j.apal.2005.12.004.

13 Pierre Ganty and Elena Gutiérrez. The Parikh property for weighted context-free grammars. In *38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2018, December 11-13, 2018, Ahmedabad, India*, pages 32:1–32:20, 2018. doi:10.4230/LIPIcs.FSTTCS.2018.32.

14 Stefan Gerhold. On some non-holonomic sequences. *Electr. J. Comb.*, 11(1), 2004. URL: http://www.combinatorics.org/Volume_11/Abstracts/v11i1r87.html.

15 Vesa Halava, Tero Harju, Mika Hirvensalo, and Juhani Karhumäki. Skolem's problem-on the border between decidability and undecidability. Technical report, Technical Report 683, Turku Centre for Computer Science, 2005.

16 Manuel Kauers and Peter Paule. *The Concrete Tetrahedron - Symbolic Sums, Recurrence Equations, Generating Functions, Asymptotic Estimates.* Texts & Monographs in Symbolic Computation. Springer, 2011. doi:10.1007/978-3-7091-0445-3.

17 Stephan Kreutzer and Cristian Riveros. Quantitative monadic second-order logic. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013*, pages 113–122, 2013. doi:10.1109/LICS.2013.16.

18 Serge Lang. *Algebra*. Graduate Texts in Mathematics. Springer, 2002.

19 Filip Mazowiecki and Cristian Riveros. Copyless cost-register automata: Structure, expressiveness, and closure properties. *J. Comput. Syst. Sci.*, 100:1–29, 2019. doi:10.1016/j.jcss.2018.07.002.

20 Joël Ouaknine and James Worrell. On linear recurrence sequences and loop termination. *SIGLOG News*, 2(2):4–13, 2015. URL: https://dl.acm.org/citation.cfm?id=2766191.

21 Géraud Sénizergues. Sequences of level 1, 2, 3, ..., k , . In *Computer Science - Theory and Applications, Second International Symposium on Computer Science in Russia, CSR 2007, Ekaterinburg, Russia, September 3-7, 2007, Proceedings*, pages 24–32, 2007. doi:10.1007/978-3-540-74510-5_6.

A Recipe for Quantum Graphical Languages

Titouan Carette
Université de Lorraine, CNRS, Inria, LORIA, Nancy, France
titouan.carette@loria.fr

Emmanuel Jeandel
Université de Lorraine, CNRS, Inria, LORIA, Nancy, France
emmanuel.jeandel@loria.fr

Abstract

Different graphical calculi have been proposed to represent quantum computation. First the ZX-calculus [4], followed by the ZW-calculus [12] and then the ZH-calculus [1]. We can wonder if new ZX-like calculi will continue to be proposed forever. This article answers negatively. All those language share a common core structure we call Z^*-algebras. We classify Z^*-algebras up to isomorphism in two dimensional Hilbert spaces and show that they are all variations of the aforementioned calculi. We do the same for linear relations and show that the calculus of [2] is essentially the unique one.

2012 ACM Subject Classification Theory of computation → Quantum computation theory; Theory of computation → Semantics and reasoning; Mathematics of computing; Theory of computation → Equational logic and rewriting

Keywords and phrases Categorical Quantum Mechanics, Quantum Computing, Category Theory

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.118

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

Related Version A full version of this paper is available at `https://members.loria.fr/TCarette/wp-content/blogs.dir/196/files/sites/196/2020/04/supparxiv.pdf`.

Funding ANR-17-CE25-0009, PIA-GDN/Quantex.

1 Introduction

The most common formalization of quantum computing is the circuit model, a diagrammatical language representing unitary matrices in a two dimensional Hilbert space, see [20] for an introduction. Verification of quantum processes requires a sound and complete equational theory for quantum circuits, i.e. a complete presentation of unitaries by generators and relations. This is known to be a difficult open problem.

By relaxing the unitarity condition and allowing all linear maps, at least three different complete equational theories were found. The *ZX-calculus* was introduced in [4] and was designed as a part of the categorical quantum mechanics program. It relies on the interaction between two complementary observables. The ZX-calculus has proven to be a good language to reason about quantum processes [7, 11]. However, finding a set of rules to make it complete has been open for a long time, and part of the solution [15] involved a secondary graphical language: the ZW-calculus [12, 5]. This calculus is built on two tripartite entanglement classes (GHZ and W-states) unraveling new structures. Yet another complete graphical language was later introduced, the ZH-calculus [1], inspired by hyper-graph states.

Compared to quantum circuits, these three languages share an important advantage. Processes and matrices are not represented merely by diagrams, but by *graphs* (hence the term *graph*ical language). Isomorphic graphs represent the same quantum evolution. This peculiarity is embedded in the *only topology matters* paradigm. This is a subtle feature: a usual diagrammatic language (like quantum circuits) starts with a given set of primitives (usually quantum gates) for which the notion of inputs and outputs is significant. When *only topology matters*, one can readily switch an input into an output, and conversely.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 118; pp. 118:1–118:17
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

This property follows from some specificities of the building blocks of those languages. One goal of this article is to give a formal definition of these specificities.

Then, we will be able to prove that the three existing graphical calculi for quantum computing, ZX, ZH and ZW, are *essentially* the only possible graphical calculi of their kind.

To do this, we identify in this paper a common structure underlying the already defined calculi, that we call a Z^*-algebra. Formally, the structure consists of two Frobenius algebras interacting *via* a bialgebra rule. To this, we add one additional property, called *compatibility*, to ensure the *only topology matters* paradigm. We then describe all the Z^*-algebras in two dimensional Hilbert spaces and show that they all happen to be phase-shifted versions of four structures we call ZX, ZH, ZW and ZZ. The first three appear respectively in the ZX, ZH and ZW-calculus. The last one corresponds to a degenerate calculus arising from only one self-interacting special Frobenius algebra.

It is important to note that languages for quantum computing are not the only known to enjoy these nice properties. In particular Bonchi and his coauthors [2] gave in 2017 a graphical language for linear relations, with striking similarities to the ZX, ZH and ZW calculi. In fact, we will prove that their language is the only Z^*-algebra for linear relations.

There exist some other formalisms trying to unify ZX-like languages, in particular in the context of interacting Frobenius algebras [10] or Hopf-Frobenius algebras [6]. However, these formalisms usually require too much structures, and fail to capture all three examples simultaneously. Typically they do not capture the ZW-calculus.

Some of our work indirectly has to do with the classification of finite dimensional algebras, bialgebras and Frobenius algebras. In the general case, an exact classification of algebras is not known, even in the commutative case. It is known that there is an infinite number of algebras up to isomorphism of dimension d for any $d > 6$. All of them are known for $d \leq 6$ [22]. We find a classification of low dimension bialgebras in [8]. We can find some constructions related to Z^*-algebras in [17] and [9].

This paper starts by introducing the *prop formalism* for graphical languages. The second section introduces various algebraic structures culminating in the definition of Z^*-algebras. The third section provides a classification of Z^*-algebras up to isomorphism for qubits. The last section gives some hint towards classification in higher dimension and provides a classification of Z^*-algebras in the context of linear relations. All the proofs are in the complete version of the paper.

2 Diagrammatical quantum computing

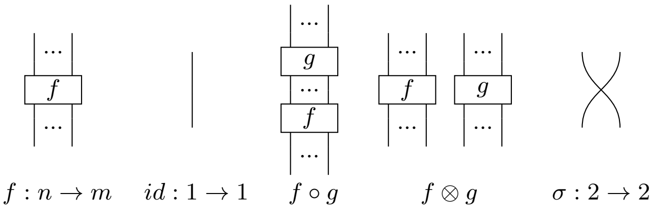
In this paper all processes are represented by a combinatorial structure called a prop [26].

▶ **Definition 1** (prop). *A* ***prop*** $\boldsymbol{P}$ *is a collection of sets* $\boldsymbol{P}[n,m]$*, indexed by* $\mathbb{N}^2$*. An element* $f \in \boldsymbol{P}[n,m]$ *is called a morphism and is usually written* $f : n \to m$*. These sets are linked by the following operators:*

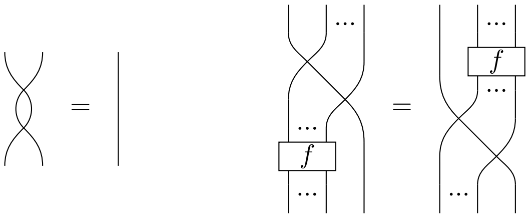
- *A* ***composition*** $\circ : \boldsymbol{P}[b,c] \times \boldsymbol{P}[a,b] \to \boldsymbol{P}[a,c]$ *satisfying:* $(f \circ g) \circ h = f \circ (g \circ h)$*.*
- *A* ***tensor product*** $\otimes : \boldsymbol{P}[a,b] \times \boldsymbol{P}[c,d] \to \boldsymbol{P}[a+c, b+d]$*, satisfying:* $(f \otimes g) \otimes h = f \otimes (g \otimes h)$ *and* $(f \circ g) \otimes (h \circ k) = (f \otimes h) \circ (g \otimes k)$*.*
- *An* ***empty morphism*** $1 : 0 \to 0$ *such that* $f \otimes 1 = 1 \otimes f = f$ *for all* $f : a \to b$*.*
- *An* ***identity*** $id : 1 \to 1$ *such that* $f \circ id^{\otimes a} = id^{\otimes b} \circ f = f$ *for all* $f : a \to b$*. With the convention* $id^{\otimes 0} = 1$*.*
- *A* ***symmetry*** $\sigma : 2 \to 2$ *satisfying:* $\sigma^2 = id^{\otimes 2}$ *and such that,* $\sigma_a \circ (f \otimes id) = (id \otimes f) \circ \sigma_b$*, for all* $f : a \to b$*, where* $\sigma_{n+1} = (1^{\otimes n} \otimes \sigma) \circ (1 \otimes \sigma_n)$*, with* $\sigma_0 = id$*.*

In the language of categories [18], a prop is a small strict symmetric monoidal category whose monoid of object is spanned by a unique object. They can be seen as resource sensitive Lawvere theories where multiple outputs are allowed [3].

Props admit a nice diagrammatical representation that gives a topological interpretation to the axioms [23]. A morphism $f : n \to m$ is represented as a box with n inputs and m outputs. Composition is represented by plugging the boxes. The tensor product by drawing the boxes side by side. The identity is represented by a single wire, the empty morphism by an empty diagram and the symmetry by wire crossing.



That choice of notation fits nicely with the axioms of props. The corresponding equations are natural in the diagrammatic notation. In particular the symmetry axioms express that the boxes can move through wires:



This diagrammatical language is sound [16]. So we can equivalently work with equations or diagrams.

▶ **Example 2** (sets and functions). Let X be a set. In the prop $\mathbf{Fun}_X$, the set $\mathbf{Fun}_X[n, m]$ is exactly the set of functions from X^n to X^m, with composition being the usual composition, and tensor product being the cartesian product.

▶ **Example 3** (matrices). For an integer d and a field $\mathbb{K}$, the prop $\mathbf{Mat}_{\mathbb{K}^d}$ is defined by $\mathbf{Mat}_{\mathbb{K}^d}[n, m] := \mathcal{M}_{d^m \times d^n}(\mathbb{K})$, the matrices of size d^m by d^n over the field $\mathbb{K}$. The composition is the matrix product and the tensor is the Kronecker product. Keeping with quantum computing traditions, we will denote by $(|e_i\rangle)_{1 \leq i \leq d}$ a basis of $\mathbb{K}^d$.

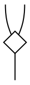
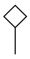
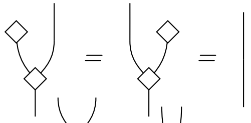
The main prop of interest for quantum computing is $\mathbf{Qubits} := \mathbf{Mat}_{\mathbb{C}^2}$. The quantum analog of bits, the qubits, are described by vectors in $\mathbb{C}^2$. A register of n qubits is then a vector in the tensor product $\mathbb{C}^{2^n}$.

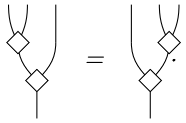
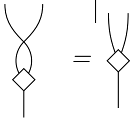
3 Graphical structures

While the diagrammatical languages presented in the previous section make reasoning about props easier, it is still somewhat strict: inputs come to the top of the box representing f, outputs goes out at the bottom. *Graphical languages* do not have this restriction, and we will explain here what additional properties should be satisfied to obtain a better framework.

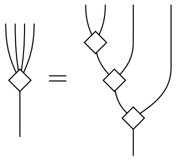
3.1 Half a spider

We start by studying elementary associative binary operations with units: monoids.

▶ **Definition 4.** *A monoid is a morphism $\mu : 2 \to 1$ (the product) and a morphism $\eta : 0 \to 1$ (the unit) that satisfy the equations: $\mu \circ (\eta \otimes \mathrm{id}) = \mu \circ (\mathrm{id} \otimes \eta) = \mathrm{id}$ and $\mu \circ (\mu \otimes \mathrm{id}) = \mu \circ (\mathrm{id} \otimes \mu)$. If we depict the product*  *and the unit*  *, the equations becomes:* 

and  *The monoid is commutative if $\mu \circ \sigma = \mu$. In pictures,*  *.*

All the monoids in this paper are supposed to be commutative.

Once we have a monoid (μ, η), we can define an n-ary product inductively by $\mu_0 = \eta$, $\mu_1 = id$ and $\mu_{n+1} = \mu \circ (\mu_n \otimes id)$. As an example, here is μ_4:  .

Using the equations, we have more generally $\mu_{n+p} = \mu \circ (\mu_n \otimes \mu_p)$, so how to transform the operator μ_n into compositions of μ and η doesn't matter.

▶ **Example 5.** A monoid in $\mathbf{Fun}_X$ is exactly what is usually called a monoid on X. Monoid in $\mathbf{Mat}_{\mathbb{K}^d}$ are exactly the d-dimensional $\mathbb{K}$-algebras.

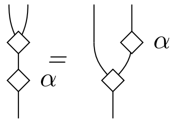
In the following we will be mainly interested in the two following examples:

▶ **Example 6** (co-copy)**.** Given a basis $(|i\rangle)_{1 \le i \le d}$ of $\mathbb{K}^d$, the **co-copy** monoid is define in $\mathbf{Mat}_{\mathbb{K}^d}$ by $\eta : 1 \mapsto \sum_{i=1}^{d} |i\rangle$ and $\mu : |i\rangle|j\rangle \mapsto \begin{cases} |i\rangle \text{ if } i = j \\ \text{else } 0 \end{cases}$

▶ **Example 7** (monoid algebra [21])**.** Given a monoid $M = (X, *, e)$ in $\mathbf{Fun}_X$ with X of cardinality d, we can define a monoid $\mathbb{K}[M]$ in $\mathbf{Mat}_{\mathbb{K}^d}$ by indexing each element of a basis by the elements of M. We then take: $\eta : 1 \mapsto |e\rangle$ and $\mu : |a\rangle|b\rangle \mapsto |a * b\rangle$. If M is a group, we will speak of a *group algebra.*

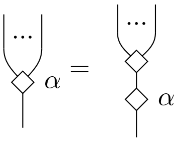
Starting from a monoid M in $\mathbf{Fun}_X$ with X of cardinality $d + 1$ that contains a zero element (that we note $\perp$), we can build a *contracted algebra* $\mathbb{K}M$ in $\mathbf{Mat}_{\mathbb{K}^d}$ by essentially the same construction, but identifying the element $\perp$ with the matrix 0. One can see that the previous example of the co-copy actually fits in this framework: it is exactly $\mathbb{K}M$ for the monoid in $\mathbf{Fun}_{\{\perp,1,\ldots,n\}}$ defined by $i * j = i$ if $i = j$ and $\perp$ otherwise.

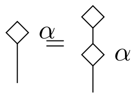
Any commutative monoid defines a group of phases:

▶ **Definition 8** (phase)**.** *Given a commutative monoid (μ, η), a **phase** is an invertible morphism $\alpha : 1 \to 1$ such that: $\alpha \circ \mu = \mu \circ (\mathrm{id} \otimes \alpha)$. Pictorially:*  .

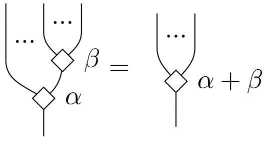
The phases form an abelian group. In general we will write this group multiplicatively and write $\alpha\beta$ instead of $\alpha \circ \beta$. *In the following, the notations α and β will be reserved for elements of the phase group.*

▶ **Discussion 9.** *An invertible scalar (a* $0 \to 0$ *morphism) is obviously a phase. Therefore the group of invertible scalars* S *is always a subgroup of the phase group* G*. If it is a direct summand, i.e. if* $G = S \times H$ *for some group* H*, then one can simplify the presentation by "dropping out" the scalars and only consider nontrivial phases. This will be the case later on for the Qubit prop, but there are examples for which such a simplification cannot be made.*

Once we have phases, we can introduce a new sequence of operators $\mu_n(\alpha)$ and $\eta(\alpha)$ defined by $\mu_n(\alpha) = \alpha \circ \mu_n$ and $\eta(\alpha) = \alpha \circ \eta$. Pictorially 

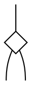
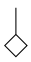
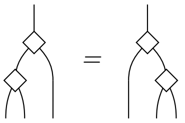
and 

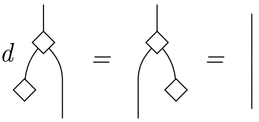
.

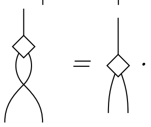
These operators satisfy the following equation: 

It is interesting to note that $\mu(\alpha)$ itself defines a monoid, by taking as unit $\eta(\alpha^{-1})$. We will call it a **phase-shifted** monoid of the original monoid.

Monoids dualize to co-monoids:

▶ **Definition 10.** *A co-monoid in a prop is a morphism* $\Delta : 1 \to 2$ *(the co-product) and a morphism* $\epsilon : 1 \to 0$ *(the co-unit) that satisfies the equations:* $(\Delta \otimes \mathrm{id}) \circ \Delta = (\mathrm{id} \otimes \Delta) \circ \Delta$ *and* $(\epsilon \otimes \mathrm{id}) \circ \Delta = (\mathrm{id} \otimes \epsilon) \circ \Delta = \mathrm{id}$*. If we depict the co-product*  *and the co-unit*  *, the equations become:* 

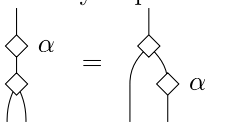
and 

. A co-monoid is co-commutative if it satisfies: $\sigma \circ \Delta = \Delta$*. In pictures,* 

.

All the co-monoids in this paper are supposed to be cocommutative.

Again we can inductively define Δ_n by $\Delta_0 = \epsilon, \Delta_1 = id$ and $\Delta_{n+1} = (\Delta_n \otimes id) \circ \Delta$.

We define phases for co-commutative co-monoids in the same way as phases for commutative monoids, they are the invertible morphisms satisfying: 

.

We can also define the morphisms $\Delta(\alpha)$ and $\epsilon(\alpha)$ as well as the phase-shifted co-monoid $(\Delta(\alpha), \epsilon(1/\alpha))$.

▶ **Example 11** (copy in $\mathbf{Fin}_X$)**.** The functions $\Delta : x \mapsto (x, x)$, with ϵ the only function from X to X^0, defines a co-monoid in $\mathbf{Fin}_X$. This is the only co-monoid in this prop.

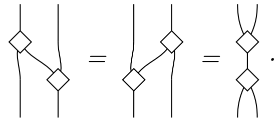
▶ **Example 12** (copy in $\mathbf{Mat}_{\mathbb{K}^d}$)**.** Given a basis of $(|i\rangle)_{1 \leq i \leq d}$, the **copy** co-monoid is defined in $\mathbf{Mat}_{\mathbb{K}^d}$ by $\epsilon : |i\rangle \mapsto 1$ and $\Delta : |i\rangle \mapsto |i\rangle|i\rangle$.

▶ **Example 13** (group co-algebra)**.** Given a finite group G of size d we can define a co-monoid in $\mathbf{Mat}_{\mathbb{K}^d}$ by $|x\rangle \mapsto \frac{1}{d} \sum_{a*b=x} |a\rangle|b\rangle$, the co-unit is $|x\rangle \mapsto \begin{cases} 1 \text{ if } x = e \\ \text{else } 0 \end{cases}$.

3.2 One spider

A monoid and a co-monoid can interact forming a Frobenius algebra.

▶ **Definition 14.** *A monoid* (μ, η) *and a co-monoid* (Δ, ϵ) *form a Frobenius algebra iff they satisfy:* $(\mathrm{id} \otimes \mu) \circ (\Delta \otimes \mathrm{id}) = (\mu \otimes \mathrm{id}) \circ (\mathrm{id} \otimes \Delta) = \Delta \circ \mu$. *Pictorially:*



A Frobenius algebra is commutative if the monoid is commutative and the co-monoid is cocommutative. ***All the Frobenius algebras in this paper are commutative.***

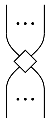
In a Frobenius algebra the phases of the monoid coincide with the phases of the co-monoid. Thus we can speak without ambiguity of the phases of a Frobenius algebra.

▶ **Example 15** (In $\mathbf{Fin}_X$). There are no Frobenius algebras in $\mathbf{Fin}_X$ (unless $|X| = 1$).

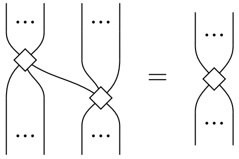
▶ **Example 16** (copy and cocopy). Given a basis $(|i\rangle)_{1 \leq i \leq d}$ of $\mathbb{K}^d$, the co-copy monoid and copy co-monoid form a Frobenius algebra in $\mathbf{Mat}_{\mathbb{K}^d}$.

▶ **Example 17** (group Frobenius algebra). Given a group G of size d, the group algebra and the group co-algebra form a Frobenius algebra in $\mathbf{Mat}_{\mathbb{K}^d}$.

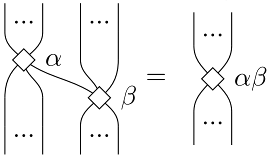
When we have a Frobenius algebra $(\mu, \eta, \Delta, \epsilon)$ we can define a family of morphisms $S_{n,m} : n \to m$ by $S_{n,m} := \mu_n \circ \Delta_m$. We call them *spiders* and depict them



. They satisfy the following equation:



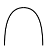
. As we have done for monoids and co-monoids, provided a phase α, we define decorated spiders $S_{n,m}(\alpha)$ by $S_n = m_n \circ \alpha \circ \Delta_p$. These new morphisms satisfy the equation:



3.2.1 Compact structure

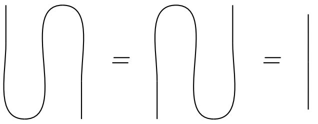
The symmetry in a prop allows various topological moves involving the wires. We can go further by providing a way to bend them. This is done by compact structures.

▶ **Definition 18** (compact structure). *A compact structure is given by two morphisms* $\delta : 0 \to 2$ *and* $\nu : 2 \to 0$ *depicted as*

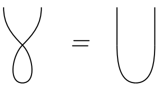


and

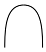
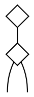
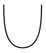
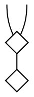
satisfying the snake equation $(\nu \otimes \mathrm{id}) \circ (\mathrm{id} \otimes \delta) = (\mathrm{id} \otimes \nu) \circ (\delta \otimes \mathrm{id}) = \mathrm{id}$. *Pictorially:*



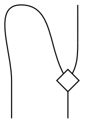
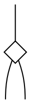
. *A compact structure is* symmetric *if* $\nu \circ \sigma = \nu \circ (\mathrm{id} \otimes \mathrm{id})$*: pictorially,*



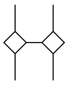
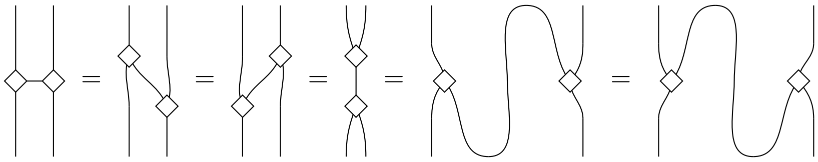
. *(This implies a similar statement on* δ*).* **All compact structures in this paper are symmetric.**

A compact structure allows to bend the wire leading to new topological properties. This extends the diagrammatical language [23]. Any Frobenius algebra directly provides a compact structure given by $\delta = \Delta \circ \eta$ and $\nu = \epsilon \circ \mu$, pictorially:  =  and  =  .

If the Frobenius algebra is commutative then this compact structure is symmetric.

This compact structure behaves well with the Frobenius algebra, we have: $(\mathrm{id} \otimes \mu) \circ (\delta \otimes \mathrm{id}) = \Delta$, pictorially:  = 

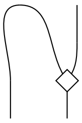
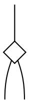
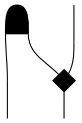
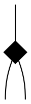
This equation is interesting from a topological point of view. Bending the wires of a diagram gives a diagram representing the same morphism. This has been referred to as the *only topology matters* paradigm [4]. **For us, the only topology matter paradigm is the key property of a graphical language**.

In particular, we can by abuse of notation write:  which may represent any of the following diagrams : 

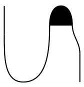
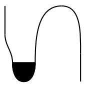
In general, we can give an unambiguous meaning to any multi-graph with input and outputs. We emphasize that this property plays a central role in the elegance of the Z^* calculi.

3.3 Two spiders

The ZX, ZW and ZH-calculii all have two Frobenius algebras. The next step is therefore to have two of them.

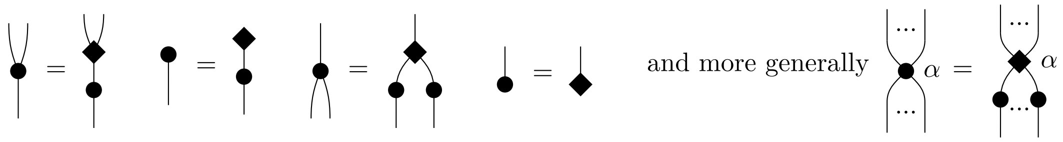
In this setting, the *only topology matters* paradigm doesn't apply anymore. Indeed, coloring the two algebras in white and black, we have:  =  and  =  using the two compact structures corresponding to the two algebras, but in general ≠ .

So we cannot hope for the two compact structures to be equal, but we can hope for some sort of *compatibility*:

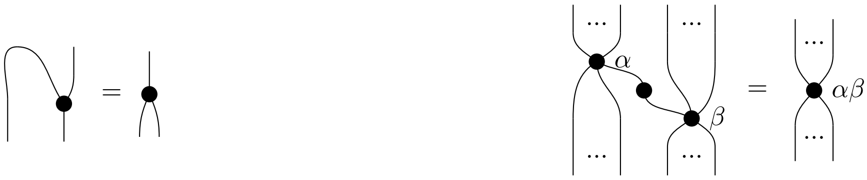
▶ **Definition 19** (compatibility). *Two Frobenius algebras are* ***compatible*** *if their compact structure satisfy:*  =  *. We call the left hand side the* ***dualizer***.

Note that the snake equation(s) implies that the left hand side is always the inverse of the right hand side. In compatible case the dualizer is an involution.

When the two Frobenius algebras are compatible, we can adjust the language so that we can bend wires on both structures. This is done at the price of a slight modification of the second algebra. We introduce now a new generator (represented by a black node) that represents the dualizer, and introduce four new generators in place of the original structures:



With the new generators, we succeed in obtaining a new language: Indeed: we can now bend the wires of the new generator, and we keep a form of the spider rule:

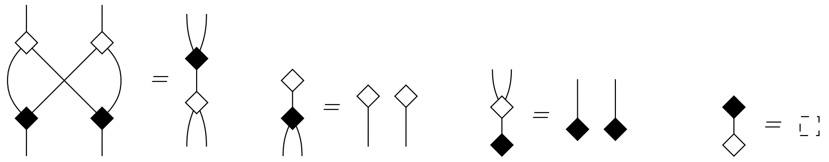


▶ **Discussion 20.** *One could decide similarly to change the first Frobenius algebra rather than the second one. In fact, if there is a preexisting compact structure, it also make sense to search for a compatibility between the preexisting compact structure and the two algebras. This is somehow what has been done in [12].*

3.4 Two spiders interacting

We now require the two spiders to interact in a precise way.

▶ **Definition 21** (Bialgebra)**.** *A co-monoid and a monoid form a bialgebra iff they satisfy the three following equations:*



Bigebra (B) *Copy (C1)* *Cocopy (C2)* *Identity (Id)*

The four bialgebra laws enforces some kind of commutation property between the co-monoid and the monoid. There are conflicting definitions in the literature on which properties one should impose on a bialgebra. The one we take is from Sweedler[25].

We now come to our main definition:

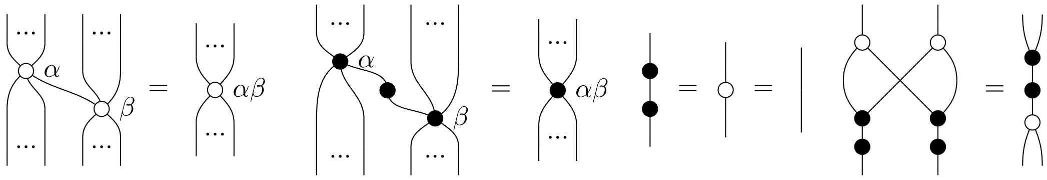
▶ **Definition 22** (Z^*-algebra)**.** *A* ***Z^*-algebra*** *is formed by two compatible Frobenius algebras such that the co-monoid of the first one satisfies the bigebra rule (B) with the monoid of the second one.*

A Z^*-algebra formed by two Frobenius algebras F and G will be denoted FG.

▶ **Discussion 23.** *One could give a different definition of a Z^*-algebra, by imposing all four conditions of the bialgebra law, or even impose it to both monoid/co-monoid pairs. However it turns out that the most important examples (esp. the ZW-calculus) do not satisfy all equations. We isolate the bigebra law as being central.*

Using the notations from the previous section, we see that a Z^*-algebra leads to a *graphical*-calculus, formed by two spiders that are subject to the following rules[1]:

[1] The white node is the same as the white lozenge, but represented differently to emphasize that the whole calculus is different



Together with the *only topology matters* paradigm, which means we can bend the wires of any node, changing an input into an output.

The rules we obtain are a common subset of the rules of the Z^*-calculi [2, 4, 12, 1].

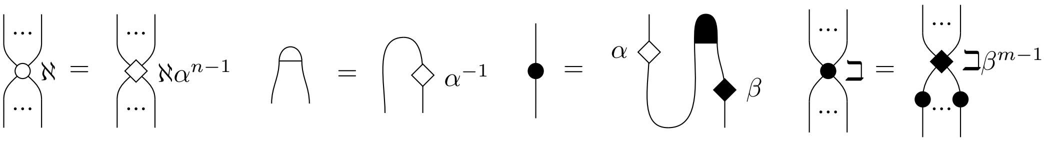
4 Classification of Z^*-algebras in Qubits and LinRel

Now that we have defined what we think is a ZX-like calculus, we can proceed to the main theorem: there are essentially only four possible such calculi for quantum computing up to isomorphism: the ZX-calculus, the ZW-calculus, the ZW-calculus, and the (trivial) ZZ-calculus. Before we can give a formal statement of the theorem, we need to explain what we mean by "essentially".

Consider a Z^*-algebra formed of two Frobenius algebras named A and B. Suppose that λ is a invertible scalar (i.e. a $0 \to 0$ morphism). If we multiply, say, the generators of the monoid of A by λ and the generators of the co-monoid of B by $1/\lambda$, then we obtain a new Z^*-algebra. This new algebra is usually not isomorphic to the first one, but for all practical purposes, they behave the same.

More generally, suppose we add a phase α to the monoid of A (replacing μ, η by $\mu(\alpha)$ and $\eta(\alpha)$) and we add similarly a phase β to the co-monoid of B. Then we obtain two new Frobenius algebras that we will call A^α and B_β which satisfies all axioms of a Z^*-algebra, *except possibly the compatibility relations.* We call this a *phase-shifted* versions of the original Z^*-algebra. We will show that all possible Z^*-algebra for quantum computing are phase-shifted version of four basic ones.

Phase-shifted algebras are a bit subtle. To ease the understanding, we provide here the graphical calculus that corresponds to $A^\alpha B_\beta$ in terms of the original generators, with the caveat that it is a graphical calculus only if the compatibility relation is satisfied (equivalently, the black node below is an involution). n and m denote respectively the number of inputs and outputs and we represent the compact structure of the white algebra differently in both calculi to avoid confusion:



4.1 Z^*-algebras in Qubits

We now investigate the particular case of graphical calculi for quantum computing. This corresponds to the special case **Qubits** = $\mathbf{Mat}_{\mathbb{C}^2}$.

A monoid in Qubit is exactly the same as a $\mathbb{C}$-algebra of dimension 2. Algebras in dimension 2 have been classified [24]: there are only two algebras up to isomorphism.

For our purpose however, we will introduce four algebras (the first three being isomorphic), that we call Z, X, H and W. Working in the basis $(|0\rangle, |1\rangle)$. They correspond to contracted algebras $\mathbb{C}M$, see 7.

Z	$\|0\rangle$	$\|1\rangle$
$\|0\rangle$	$\|0\rangle$	0
$\|1\rangle$	0	$\|1\rangle$

X	$\|0\rangle$	$\|1\rangle$
$\|0\rangle$	$\|0\rangle$	$\|1\rangle$
$\|1\rangle$	$\|1\rangle$	$\|0\rangle$

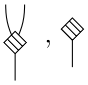
H	$\|0\rangle$	$\|1\rangle$
$\|0\rangle$	$\|0\rangle$	$\|0\rangle$
$\|1\rangle$	$\|0\rangle$	$\|1\rangle$

W	$\|0\rangle$	$\|1\rangle$
$\|0\rangle$	$\|0\rangle$	$\|1\rangle$
$\|1\rangle$	$\|1\rangle$	0

Those multiplication tables describe the behavior of the algebras on $|0\rangle$ and $|1\rangle$.

We see that Z behaves like a Kronecker delta ensuring equality, X is the XOR gate, H the AND gate and W is the effect algebra on two elements.

The matricial representation in the computational basis are:

 :=

Z	
μ_Z	η_Z
$\begin{pmatrix}1&0&0&0\\0&0&0&1\end{pmatrix}$	$\begin{pmatrix}1\\1\end{pmatrix}$

X	
μ_X	η_X
$\begin{pmatrix}1&0&0&1\\0&1&1&0\end{pmatrix}$	$\begin{pmatrix}1\\0\end{pmatrix}$

H	
μ_H	η_H
$\begin{pmatrix}1&1&1&0\\0&0&0&1\end{pmatrix}$	$\begin{pmatrix}0\\1\end{pmatrix}$

W	
μ_W	η_W
$\begin{pmatrix}1&0&0&0\\0&1&1&0\end{pmatrix}$	$\begin{pmatrix}1\\0\end{pmatrix}$

The phase group of Z is ${\mathbb{C}^*_\times}^2$. The phase group of W is $\mathbb{C}^*_\times \times \mathbb{C}_+$.

If we write the phases for our four favorite monoids, they read:

(a,b) :=

Z
$a,b \in \mathbb{C}^*$
$a\begin{pmatrix}1&0\\0&b\end{pmatrix}$

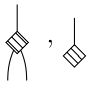
X
$a,b \in \mathbb{C}^*$
$\frac{a}{2}\begin{pmatrix}1+b&1-b\\1-b&1+b\end{pmatrix}$

H
$a,b \in \mathbb{C}^*$
$a\begin{pmatrix}1&1-b\\0&b\end{pmatrix}$

W
$a \in \mathbb{C}^*\ b \in \mathbb{C}$
$a\begin{pmatrix}1&0\\b&1\end{pmatrix}$

As explained in Discussion 9, in the case of Qubit, we can write[2] the phase group $G = \mathbb{C}^\star \times H$ where the first component $\mathbb{C}^\star$ corresponds to the invertible scalars, and H is some commutative group ($H = \mathbb{C}^\star_\times$ in the first three cases, and $H = \mathbb{C}_+$ in the last case). One could then index the phases only by this subgroup H (i.e. always take $a = 1$), introducing scalars when necessary. This is what has been done in the literature.

All four monoids form Frobenius algebras, with the following co-monoids[3], co-units, and compact structures:

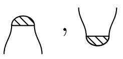
 :=

Z	
Δ_Z	ϵ_Z
$\begin{pmatrix}1&0\\0&0\\0&0\\0&1\end{pmatrix}$	$\begin{pmatrix}1&1\end{pmatrix}$

X	
Δ_X	ϵ_X
$\frac{1}{2}\begin{pmatrix}1&0\\0&1\\0&1\\1&0\end{pmatrix}$	$\begin{pmatrix}2&0\end{pmatrix}$

H	
Δ_H	ϵ_H
$\begin{pmatrix}1&2\\0&-1\\0&-1\\0&1\end{pmatrix}$	$\begin{pmatrix}1&2\end{pmatrix}$

W	
Δ_W	ϵ_W
$\begin{pmatrix}0&0\\1&0\\1&0\\0&1\end{pmatrix}$	$\begin{pmatrix}0&1\end{pmatrix}$

 :=

Z	
δ_Z	ν_Z
$\begin{pmatrix}1\\0\\0\\1\end{pmatrix}$	$\begin{pmatrix}1&0&0&1\end{pmatrix}$

X	
δ_X	ν_X
$\frac{1}{2}\begin{pmatrix}1\\0\\0\\1\end{pmatrix}$	$\begin{pmatrix}2&0&0&2\end{pmatrix}$

H	
δ_H	ν_H
$\begin{pmatrix}2\\-1\\-1\\1\end{pmatrix}$	$\begin{pmatrix}1&1&1&2\end{pmatrix}$

W	
δ_W	ν_W
$\begin{pmatrix}0\\1\\1\\0\end{pmatrix}$	$\begin{pmatrix}0&1&1&0\end{pmatrix}$

We can now state our main theorem:

▶ **Theorem 24.** *The only Z^*-algebras up to isomorphism in **Qubits** are, with $a, b \in \mathbb{C}^*$: $Z^{(a,\frac{b}{a})}Z_{(\frac{1}{a},\frac{a}{b})}$, $Z^{(a,\frac{b}{a})}Z_{(-\frac{1}{a},\frac{a}{b})}$, $Z^{(a,\frac{b}{a})}Z_{(\frac{1}{a},-\frac{a}{b})}$, $Z^{(a,\frac{b}{a})}Z_{(-\frac{1}{a},-\frac{a}{b})}$, $Z^{(a,1)}X_{(\frac{2}{a},1)}$, $Z^{(a,1)}X_{(-\frac{2}{a},1)}$, $Z^{(a,-1)}X_{(\frac{2}{a},1)}$, $Z^{(a,-1)}X_{(-\frac{2}{a},1)}$, $Z^{(\frac{a}{b},b^2)}X_{(\frac{2}{a},-1)}$, $Z^{(a,-1)}X_{(\frac{2b}{a},\frac{1}{b^2})}$, $Z^{(a,\frac{1}{b^2-1})}H_{(\frac{b}{a},\frac{1-b^2}{b^2})}$ with $b^2 \neq 1$, $Z^{(a,\frac{1}{b^2})}W_{(\frac{b}{a},0)}$ and $W^{(a,0)}Z_{(\frac{b}{a},\frac{1}{b^2})}$.*

[2] This can be done more generally in any prop if the group of scalars is divisible.

[3] The co-monoids have been choosen such that the three first Frobenius algebras are isomorphic, hence the weird $\frac{1}{2}$ factor in X.

The idea is to show that, up to isomorphism, there are only five possible monoids/comonoid pairs satisfying the bigebra rule, and then show how they can possibly extend to Z^*-algebras.

We will now compare the calculi we obtain with the literature.

- The ZZ-calculus never has been really considered, as having two spiders that are identical is not useful. However, its existence is not happenstance: in general a Frobenius algebra would not make a bigebra with itself. In this case, it works as Z is a *special* Frobenius algebra.
- The ZX-calculus [4] corresponds to what we call $ZX_{(2,2)}$. This is a particular calculus as the dualizer is trivial: both algebras have the same compact structure (up to scalars). We say that the two algebras are *coflexible*. There are a few substantial differences between our calculus and the ZX-calculus. Instead of using all possibles phases in $\mathbb{C}^\star$, the authors use phases in the unit circle. Subsequent work [19] introduced so-called lambda-boxes to restore all phases. Second, the $ZX_{(2,2)}$-calculus is a bit awkward as the two Frobenius algebras Z and $X_{(2,2)}$ are not isomorphic, but only isomorphic up to a scalar. By rescaling the X algebra, we can obtain a calculus where both algebras are dual, at the price of a slightly different bigebra rule. The isomorphism corresponds to the Hadamard matrix; as this matrix is symmetric, we can add it to our language without changing the *only topology matters* paradigm, and we obtain this way the ZX-calculus defined in [4].
- The fact that other calculi of the form $Z^\alpha X_\beta$ exist corresponds to some commutation properties between phases of the two algebras. In fact, they correspond to what is called *the π-commutation rule*: $(1,\lambda)_Z \circ (1,-1)_X = \lambda(1,-1)_X \circ (1,\frac{1}{\lambda})_Z$ where $(a,b)_Z$ is a phase of Z and $(a,b)_X$ is a phase of X.
- The original rules [4] of the ZX-calculus correspond exactly to:
  - The *only topology matters* paradigm (rule T)
  - The rules above valid on any graphical calculi (rules S1, S2, B2), including in this case the copy rule (B1) and some form of the identity rule (rule D1)
  - one rule relative to the π-commutation (rule K2)
  - one rule relative to Hadamard, the isomorphism between Z and X (rule C)
  - one rule stating that Z is a special Frobenius algebra (hidden in rule S1)
  - one rule called π-copying (K1), and one rule related to the scalars (rules D2). The second rule is anecdotal. The first one relates to what are the automorphisms of Z.

  Therefore, with one omission, all original rules of the ZX-calculus could be rediscovered again in a systematic way using our definition.
- The ZW-calculus as discussed in [13, 14] is exactly what we call ZW. The calculus however do not use phases on the black nodes. The original ZW-calculus introduced in [12] by the same author is slightly different. Intuitively it corresponds to a different kind of graphical languages where the two Frobenius algebras have been made compatible with a third compact structure. The fact that other calculi of the form $Z^\alpha W_\beta$ exist essentially amounts to the same π-commutation rule as before.
- The ZH-calculus as discussed in [1] is exactly what we call $ZH_{(\sqrt{2},-\frac{1}{2})}$. However the authors do not use phases on the white node, and use a different parametrizations of the phases on the black node. The phase they call x is what we would call the phase $(1,1-2x)_H$. This makes the spider rule more awkward in their calculus. The fact that other calculi of the form $Z^\alpha H_\beta$ exist is linked to the following rule: $2\frac{\lambda+1}{\lambda}(1,\lambda)_Z \circ H \circ (1,\frac{1}{2}\frac{1}{\lambda+1})_H = (1,2(\lambda+1))_H \circ H \circ (1,\frac{1}{\lambda})_Z$ where $(a,b)_Z$ is a phase of Z, $(a,b)_H$ is a phase of H and H is the Hadamard gate.

4.2 Generalization for qudits

A similar classification could be theoretically done for other dimensions, i.e. for the prop $\mathbf{Mat}_{\mathbb{C}^d}$. However difficulties arise. Indeed, all possible algebras have been classified only in dimension $d \leq 6$ [22] (in fact there are an infinite number of non isomorphic commutative algebras of dimension 7), and the work is even more terse on bigebras (some work [8] has been done for bi*algebras* in dimension 2 and 3) or Frobenius algebras (although a theoretical characterization exist).

We will therefore focus here on generalizations of the existing structures of dimension 2 to higher dimension.

The ZX-structure corresponds to an interaction between the two algebras $\mathbb{C}^2$ and $\mathbb{C}[\mathbb{Z}/2\mathbb{Z}]$. One could readily generalize this to higher dimensions replacing $\mathbb{Z}/2\mathbb{Z}$ by $\mathbb{Z}/d\mathbb{Z}$, or actually any other commutative group of size d. In higher dimension, the dualizer actually becomes non trivial: it corresponds to the morphism $x \mapsto -x$ in $\mathbb{Z}/d\mathbb{Z}$, which is trivial only if $d = 2$.

The ZW-structure can also be generalized easily. We again replace the algebra $\mathbb{C}^2$ by $\mathbb{C}^d$. and we can generalize the W algebra in dimension 2 to the algebra $\mathbb{C}[X] \big/ \left(X^d\right)$.

We did not find any generalization of ZH that work in arbitrary dimension. The obvious d-dimensional generalization of H would be as a contracted monoid algebra $\mathbb{C}M$ where M is the monoid $(\mathcal{F}, \cap)$ for a family of subsets $\mathcal{F}$ closed under intersection (the 2-dimensional version corresponding to $\mathcal{F} = \{\{1\}, \emptyset\}$). This generalization gives indeed two Frobenius algebra that satisfy a bigebra law, but they usually are not compatible (unless $\mathcal{F} = 2^X$ for some set X).

4.3 In $\mathbf{LinRel}_{\mathbb{K}}$

Quantum computing is not the only place where ZX-like calculi appear: another Z^*-algebra that occurs in the literature is *Graphical linear algebra* [2] in the prop $\mathbf{LinRel}_{\mathbb{K}}$. In $\mathbf{LinRel}_{\mathbb{K}}$ a map $n \to m$ is a linear subspace of $\mathbb{K}^{n+m}$.

It turns out that there are only two monoids in $\mathbf{LinRel}_{\mathbb{K}}$, and they are not isomorphic: the monoid given by the subspace $\{(x, x, x), x \in \mathbb{K}\}$ and the monoid given by $\{(x, y, x+y), x, y \in \mathbb{K}\}$. Their respective phase groups are both trivial. Both these monoids, that we call B and N, actually happen to have Frobenius algebra structures.

▶ **Proposition 25.** *There are only four Z^*-algebras in $\mathbf{LinRel}_{\mathbb{K}}$: BB, NN, BN and NB.*

As these are the only potential candidates, we just have to check that they indeed give Z^*-algebras. BB and NN are trivial, and BN is just a dual version of NB, therefore the graphical calculi of [2] is THE only possible ZX-like calculus for this prop.

5 Future works

We have classified Z^*-algebras in $\mathbf{Mat}_{\mathbb{C}^2}$ and $\mathbf{LinRel}_{\mathbb{K}}$. Further investigations will concern other categories. In the case of $\mathbf{Mat}_{\mathbf{R}}$ for a semiring $\mathbf{R}$, generalizations of ZW and ZX exist. A natural question is which other Z^*-algebras exist in this setting. All the monoids and co-monoids we considered were commutative, the non commutative case is also of interest, leading to a more general notion of graphical language involving *port graphs* or *rotation systems*. An other direction would be to drop the unit and the compact structure and find what defines a graphical language in this case. This is necessary in infinite dimensional Hilbert spaces for example.

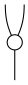
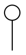
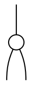
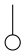
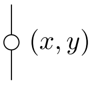
6 All graphical calculi for quantum computing

In this last section we provide a complete description of the Z^*-algebras in **Qubits**.

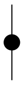
6.1 The ZZ-calculis

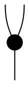
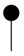
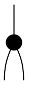
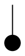
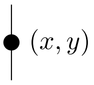
6.1.1 $Z^{(a,b/a)}Z_{(1/a,a/b)}$

This is the first calculus presented in the Theorem, up to a re-parametrization that makes it slightly better looking:

 $= \begin{pmatrix} a & 0 & 0 & 0 \\ 0 & 0 & 0 & b \end{pmatrix}$  $= \begin{pmatrix} \frac{1}{a} \\ \frac{1}{b} \end{pmatrix}$  $= \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix}$  $= \begin{pmatrix} 1 & 1 \end{pmatrix}$ 

$= x \begin{pmatrix} 1 & 0 \\ 0 & y \end{pmatrix}$

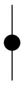
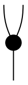
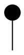
$= \begin{pmatrix} a & 0 & 0 & b \end{pmatrix}$ $= \begin{pmatrix} \frac{1}{a} \\ 0 \\ 0 \\ \frac{1}{b} \end{pmatrix}$  $= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$

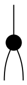
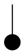
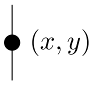
 $= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$  $= \begin{pmatrix} 1 \\ 1 \end{pmatrix}$  $= \begin{pmatrix} \frac{1}{a} & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & \frac{1}{b} \end{pmatrix}$  $= \begin{pmatrix} a & b \end{pmatrix}$ 

$= x \begin{pmatrix} 1 & 0 \\ 0 & y \end{pmatrix}$

6.1.2 $Z^{(a,b/a)}Z_{(1/a,-a/b)}$

The only difference with the previous calculus is in the following generators:

 $= \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$  $= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$  $= \begin{pmatrix} 1 \\ -1 \end{pmatrix}$

 $= \begin{pmatrix} \frac{1}{a} & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & -\frac{1}{b} \end{pmatrix}$  $= \begin{pmatrix} a & -b \end{pmatrix}$ 

$= x \begin{pmatrix} 1 & 0 \\ 0 & -y \end{pmatrix}$

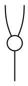
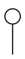
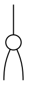
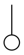
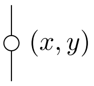
6.1.3 $Z^{(a,b/a)}Z_{(-1/a,a/b)}$ and $Z^{(a,b/a)}Z_{(-1/a,-a/b)}$

These calculi differ from the previous ones only by the presence of a global scalar “-1” in all matrices corresponding to the black nodes.

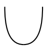
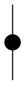
6.2 The ZX-calculi

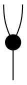
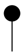
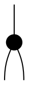
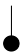
6.2.1 $Z^{(a,1)}X_{(2/a,1)}$

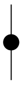
In the case $a = 1$, this is almost the ZX-calculus of [4]:

 $= a \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$  $= \frac{1}{a} \begin{pmatrix} 1 \\ 1 \end{pmatrix}$  $= \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix}$  $= \begin{pmatrix} 1 & 1 \end{pmatrix}$ 

$= x \begin{pmatrix} 1 & 0 \\ 0 & y \end{pmatrix}$

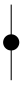
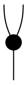
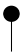
 $= a\begin{pmatrix}1 & 0 & 0 & 1\end{pmatrix}$ $= \frac{1}{a}\begin{pmatrix}1\\0\\0\\1\end{pmatrix}$  $= \begin{pmatrix}1 & 0\\0 & 1\end{pmatrix}$

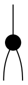
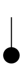
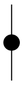
 $= \begin{pmatrix}1 & 0 & 0 & 1\\0 & 1 & 1 & 0\end{pmatrix}$  $= \begin{pmatrix}1\\0\end{pmatrix}$  $= \frac{1}{a}\begin{pmatrix}1 & 0\\0 & 1\\0 & 1\\1 & 0\end{pmatrix}$  $= a\begin{pmatrix}1 & 0\end{pmatrix}$

 $(x,y) \;= \frac{1}{2}\,x\begin{pmatrix}y+1 & -y+1\\-y+1 & y+1\end{pmatrix}$

6.2.2 $Z^{(a,1)}X_{(2/a,-1)}$

The only difference with the previous calculus is in the following generators:

 $= \begin{pmatrix}0 & 1\\1 & 0\end{pmatrix}$  $= \begin{pmatrix}0 & 1 & 1 & 0\\1 & 0 & 0 & 1\end{pmatrix}$  $= \begin{pmatrix}0\\1\end{pmatrix}$

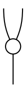
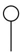
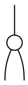
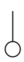
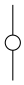
 $= \frac{1}{a}\begin{pmatrix}0 & 1\\1 & 0\\1 & 0\\0 & 1\end{pmatrix}$  $= a\begin{pmatrix}0 & 1\end{pmatrix}$  $(x,y) \;= \frac{1}{2}\,x\begin{pmatrix}-y+1 & y+1\\y+1 & -y+1\end{pmatrix}$

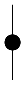
6.2.3 $Z^{(a,1)}X_{(-2/a,-1)}$ and $Z^{(a,1)}X_{(-2/a,1)}$

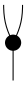
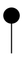
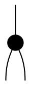
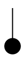
These calculi differ from the previous ones only by the presence of a global scalar "-1" in all matrices corresponding to the black nodes.

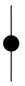
6.2.4 $Z^{(a/b,b^2)}X_{(2/a,-1)}$

This is a quite different calculus:

 $= a\begin{pmatrix}\frac{1}{b} & 0 & 0 & 0\\0 & 0 & 0 & b\end{pmatrix}$  $= \frac{1}{a}\begin{pmatrix}b\\\frac{1}{b}\end{pmatrix}$  $= \begin{pmatrix}1 & 0\\0 & 0\\0 & 0\\0 & 1\end{pmatrix}$  $= \begin{pmatrix}1 & 1\end{pmatrix}$  $(x,y) \;= x\begin{pmatrix}1 & 0\\0 & y\end{pmatrix}$

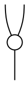
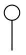
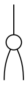
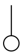
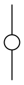
$= a\begin{pmatrix}\frac{1}{b} & 0 & 0 & b\end{pmatrix}$ $= \frac{1}{a}\begin{pmatrix}b\\0\\0\\\frac{1}{b}\end{pmatrix}$  $= \begin{pmatrix}0 & b\\\frac{1}{b} & 0\end{pmatrix}$

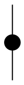
 $= \begin{pmatrix}0 & b & b & 0\\\frac{1}{b} & 0 & 0 & \frac{1}{b}\end{pmatrix}$  $= \begin{pmatrix}0\\\frac{1}{b}\end{pmatrix}$  $= \frac{1}{a}\begin{pmatrix}0 & b^2\\1 & 0\\1 & 0\\0 & \frac{1}{b^2}\end{pmatrix}$  $= a\begin{pmatrix}0 & 1\end{pmatrix}$

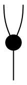
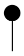
 $(x,y) \;= \frac{1}{2}\,x\begin{pmatrix}-by+b & by+b\\\frac{y+1}{b} & -\frac{y-1}{b}\end{pmatrix}$

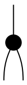
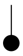
6.2.5 $Z^{(a,-1)}X_{(2b/a,1/b^2)}$

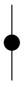
This is a calculus dual to the previous one, but the equations look more intricate:

 $= a\begin{pmatrix}1&0&0&0\\0&0&0&-1\end{pmatrix}$  $= \frac{1}{a}\begin{pmatrix}1\\-1\end{pmatrix}$  $= \begin{pmatrix}1&0\\0&0\\0&0\\0&1\end{pmatrix}$  $= \begin{pmatrix}1&1\end{pmatrix}$  $(x,y) \quad = x\begin{pmatrix}1&0\\0&y\end{pmatrix}$

$= a\begin{pmatrix}1&0&0&-1\end{pmatrix}$ $= \frac{1}{a}\begin{pmatrix}1\\0\\0\\-1\end{pmatrix}$  $= \frac{1}{2b}\begin{pmatrix}b^2+1&1-b^2\\b^2-1&-1-b^2\end{pmatrix}$

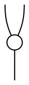
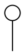
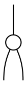
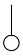
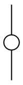
 $= \frac{1}{2b}\begin{pmatrix}b^2+1&1-b^2&1-b^2&b^2+1\\b^2-1&-1-b^2&-1-b^2&b^2-1\end{pmatrix}$  $= \frac{1}{2b}\begin{pmatrix}b^2+1\\b^2-1\end{pmatrix}$

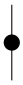
 $= \frac{1}{2ab}\begin{pmatrix}b^2+1&1-b^2\\b^2-1&-1-b^2\\b^2-1&-1-b^2\\b^2+1&1-b^2\end{pmatrix}$  $= \frac{a}{2b}\begin{pmatrix}b^2+1&1-b^2\end{pmatrix}$

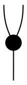
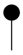
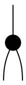
 $(x,y) \quad = \frac{x}{2b}\begin{pmatrix}b^2y+1&1-b^2y\\b^2y-1&-1-b^2y\end{pmatrix}$

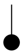
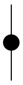
6.3 The ZH-calculi

6.3.1 The $Z^{(a,1/(b^2-1))}H_{(b/a,(1-b^2)/b^2)}$ calculus

 $= a\begin{pmatrix}1&0&0&0\\0&0&0&\frac{1}{b^2-1}\end{pmatrix}$  $= \frac{1}{a}\begin{pmatrix}1\\b^2-1\end{pmatrix}$  $= \begin{pmatrix}1&0\\0&0\\0&0\\0&1\end{pmatrix}$  $= \begin{pmatrix}1&1\end{pmatrix}$  $(x,y) \quad = x\begin{pmatrix}1&0\\0&y\end{pmatrix}$

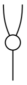
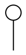
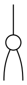
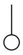
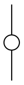
$= a\begin{pmatrix}1&0&0&\frac{1}{b^2-1}\end{pmatrix}$ $= \frac{1}{a}\begin{pmatrix}1\\0\\0\\b^2-1\end{pmatrix}$  $= \frac{1}{b}\begin{pmatrix}1&1\\b^2-1&-1\end{pmatrix}$

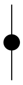
 $= \frac{1}{b}\begin{pmatrix}1&1&1&1\\b^2-1&b^2-1&b^2-1&-1\end{pmatrix}$  $= \frac{1}{b}\begin{pmatrix}1\\-1\end{pmatrix}$  $= \frac{1}{ab}\begin{pmatrix}1&1\\b^2-1&b^2-1\\b^2-1&b^2-1\\(b^2-1)^2&1-b^2\end{pmatrix}$

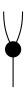
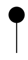
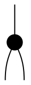
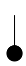
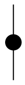
 $= \frac{a}{b}\begin{pmatrix}1&\frac{1}{1-b^2}\end{pmatrix}$  $(x,y) \quad = \frac{x}{b}\begin{pmatrix}1&1\\b^2-1&b^2-1-b^2y\end{pmatrix}$

6.4 The ZW-calculi

6.4.1 The $Z^{(a,1/c^2)}W_{(c/a,0)}$

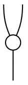
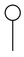
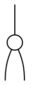
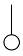
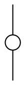
 $= a\begin{pmatrix}1&0&0&0\\0&0&0&\frac{1}{c^2}\end{pmatrix}$  $= \frac{1}{a}\begin{pmatrix}1\\c^2\end{pmatrix}$  $= \begin{pmatrix}1&0\\0&0\\0&0\\0&1\end{pmatrix}$  $= \begin{pmatrix}1&1\end{pmatrix}$  $(x,y) \quad = x\begin{pmatrix}1&0\\0&y\end{pmatrix}$

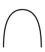
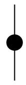
$= a\begin{pmatrix}1&0&0&\frac{1}{c^2}\end{pmatrix}$ $= \frac{1}{a}\begin{pmatrix}1\\0\\0\\c^2\end{pmatrix}$  $= \begin{pmatrix}0&\frac{1}{c}\\c&0\end{pmatrix}$

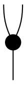
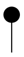
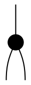
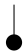
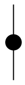
 $= \begin{pmatrix} 0 & \frac{1}{c} & \frac{1}{c} & 0 \\ c & 0 & 0 & 0 \end{pmatrix}$  $= \begin{pmatrix} 0 \\ c \end{pmatrix}$  $= \frac{1}{a} \begin{pmatrix} 0 & \frac{1}{c} \\ c & 0 \\ c & 0 \\ 0 & 0 \end{pmatrix}$  $= a \begin{pmatrix} 0 & \frac{1}{c} \end{pmatrix}$  $(x, y) \quad = x \begin{pmatrix} cy & \frac{1}{c} \\ c & 0 \end{pmatrix}$

6.4.2 The $W^{(a,0)}Z_{(b/a,1/b^2)}$ calculus

This is very similar to the previous calculus, except that W is now chosen as the white node, meaning that the black node is actually Z, up to the dualizer.

 $= a \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix}$  $= \frac{1}{a} \begin{pmatrix} 1 \\ 0 \end{pmatrix}$  $= \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}$  $= \begin{pmatrix} 0 & 1 \end{pmatrix}$  $(x, y) \quad = x \begin{pmatrix} 1 & 0 \\ y & 1 \end{pmatrix}$

$= a \begin{pmatrix} 0 & 1 & 1 & 0 \end{pmatrix}$  $= \frac{1}{a} \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$  $= \begin{pmatrix} 0 & b \\ \frac{1}{b} & 0 \end{pmatrix}$

 $= \begin{pmatrix} 0 & 0 & 0 & b \\ \frac{1}{b} & 0 & 0 & 0 \end{pmatrix}$  $= \begin{pmatrix} b \\ \frac{1}{b} \end{pmatrix}$  $= \frac{1}{a} \begin{pmatrix} 0 & b \\ 0 & 0 \\ 0 & 0 \\ \frac{1}{b} & 0 \end{pmatrix}$  $= a \begin{pmatrix} \frac{1}{b} & b \end{pmatrix}$  $(x, y) \quad = x \begin{pmatrix} 0 & by \\ \frac{1}{b} & 0 \end{pmatrix}$

References

1 Miriam Backens and Aleks Kissinger. ZH: A complete graphical calculus for quantum computations involving classical non-linearity. *arXiv preprint*, 2018. arXiv:1805.02175.

2 Filippo Bonchi, Paweł Sobociński, and Fabio Zanasi. Interacting Hopf algebras. *Journal of Pure and Applied Algebra*, 221(1):144–184, 2017.

3 Filippo Bonchi, Paweł Sobociński, and Fabio Zanasi. Deconstructing Lawvere with distributive laws. *Journal of logical and algebraic methods in programming*, 95:128–146, 2018.

4 Bob Coecke and Ross Duncan. Interacting quantum observables: categorical algebra and diagrammatics. *New Journal of Physics*, 13(4):043016, 2011.

5 Bob Coecke and Aleks Kissinger. The compositional structure of multipartite quantum entanglement. In *International Colloquium on Automata, Languages, and Programming*, pages 297–308. Springer, 2010.

6 Joseph Collins and Ross Duncan. Hopf-Frobenius algebras and a simpler Drinfeld double. *Electronic Proceedings in Theoretical Computer Science*, 2019.

7 Niel de Beaudrap and Dominic Horsman. The ZX calculus is a language for surface code lattice surgery. *Quantum*, 4:218, 2020.

8 Khadra Dekkar and Abdenacer Makhlouf. Bialgebra structures of 2-associative algebras. *arXiv preprint*, 2008. arXiv:0809.1144.

9 Yukio Doi and Mitsuhiro Takeuchi. BiFrobenius algebras. *Contemporary Mathematics*, 267:67–98, 2000.

10 Ross Duncan and Kevin Dunne. Interacting Frobenius Algebras are Hopf. In *2016 31st Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–10. IEEE, 2016.

11 Ross Duncan, Aleks Kissinger, Simon Perdrix, and John Van De Wetering. Graph-theoretic simplification of quantum circuits with the ZX-calculus. *arXiv preprint*, 2019. arXiv:1902.03178.

12 Amar Hadzihasanovic. A diagrammatic axiomatisation for qubit entanglement. In *2015 30th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 573–584. IEEE, 2015.

13 Amar Hadzihasanovic. *The algebra of entanglement and the geometry of composition.* PhD thesis, University of Oxford, 2017. arXiv:1709.08086.

14 Amar Hadzihasanovic, Kang Feng Ng, and Quanlong Wang. Two complete axiomatisations of pure-state qubit quantum computing. In *2018 33rd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 502–511. ACM, 2018. doi:10.1145/3209108.3209128.

15 Emmanuel Jeandel, Simon Perdrix, and Renaud Vilmart. A complete axiomatisation of the ZX-calculus for Clifford+ T quantum mechanics. In *2018 33rd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 559–568, 2018.

16 André Joyal and Ross Street. The geometry of tensor calculus, I. *Advances in mathematics*, 88(1):55–112, 1991.

17 M Koppinen. On algebras with two multiplications, including Hopf algebras and Bose–Mesner algebras. *Journal of Algebra*, 182(1):256–273, 1996.

18 Saunders Mac Lane. *Categories for the Working Mathematician.* Springer, 1971.

19 Kang Feng Ng and Quanlong Wang. Completeness of the zx-calculus for pure qubit clifford+ t quantum mechanics. *arXiv preprint*, 2018. arXiv:1801.07993.

20 Michael A Nielsen and Isaac Chuang. Quantum computation and quantum information, 2002.

21 J.S. Ponizovskii. Semigroup rings. *Semigroup Forum*, 36:1–46, 1987.

22 Bjorn Poonen. Isomorphism types of commutative algebras of finite rank. *Computational arithmetic geometry*, 463:111–120, 2008.

23 Peter Selinger. A survey of graphical languages for monoidal categories. In *New structures for physics*, pages 289–355. Springer, 2010.

24 E Study. Über systeme complexer zahlen und ihre anwendung in der theorie der transformationsgruppen. *Monatshefte für Mathematik und Physik*, 1:283–354, 1890.

25 Moss E. Sweedler. *Hopf Algebras.* W.A. Benjamin, Inc., 1969.

26 Fabio Zanasi. Interacting hopf algebras: the theory of linear systems. *arXiv preprint*, 2018. arXiv:1805.03032.

On the Power of Ordering in Linear Arithmetic Theories

Dmitry Chistikov
Centre for Discrete Mathematics and its Applications (DIMAP) & Department of Computer Science, University of Warwick, Coventry, UK
d.chistikov@warwick.ac.uk

Christoph Haase
Department of Computer Science, University College London, UK
c.haase@ucl.ac.uk

Abstract

We study the problems of deciding whether a relation definable by a first-order formula in linear rational or linear integer arithmetic with an order relation is definable in absence of the order relation. Over the integers, this problem was shown decidable by Choffrut and Frigeri [*Discret. Math. Theor. C.*, 12(1), pp. 21–38, 2010], albeit with non-elementary time complexity. Our contribution is to establish a full geometric characterisation of those sets definable without order which in turn enables us to prove coNP-completeness of this problem over the rationals and to establish an elementary upper bound over the integers. We also provide a complementary Π_2^{P} lower bound for the integer case that holds even in a fixed dimension. This lower bound is obtained by showing that universality for ultimately periodic sets, i.e., semilinear sets in dimension one, is Π_2^{P}-hard, which resolves an open problem of Huynh [*Elektron. Inf.verarb. Kybern.*, 18(6), pp. 291–338, 1982].

2012 ACM Subject Classification Theory of computation → Automated reasoning

Keywords and phrases logical definability, linear arithmetic theories, semi linear sets, ultimately periodic sets, numerical semigroups

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.119

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

Related Version An extended version of the paper including full proofs is available from the authors' webpages.

Funding *Christoph Haase*: This work is part of a project that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant agreement No. 852769, ARiAT).

1 Introduction

A central topic in mathematical and computational logic is to investigate the expressive power of first-order formulas in a given structure. Notable results in this context include Robinson's seminal work showing that the integers are first-order definable in the structure $\langle \mathbb{Q}, +, \cdot \rangle$, thus rendering its first-order theory undecidable [13]. Another example is the celebrated theorem of Muchnik [10] showing decidability of the problem of determining whether a relation first-order definable in the structure $\langle \mathbb{Z}, +, V_k, < \rangle$ (whose first-order theory is known as *Büchi arithmetic*) is definable in the weaker substructure $\langle \mathbb{Z}, +, < \rangle$ (known as *linear integer arithmetic* or *Presburger arithmetic*). Here, V_k is the function mapping an integer to the highest power of k dividing it without remainder. Muchnik's approach yields a quadruply exponential time algorithm for this problem when the relation is given as a deterministic finite-state automaton; a polynomial-time algorithm was later claimed by Leroux [9]. It has recently been shown that this problem can be solved in quasi-linear time for unary relations [1].

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 119; pp. 119:1–119:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In this paper, we investigate the computational complexity of the *weak definability problem*, which is the problem of deciding whether a relation first-order definable in the structures $\langle \mathbb{Q}, +, < \rangle$ or $\langle \mathbb{Z}, +, < \rangle$ is definable in its weak counterpart, which is obtained from replacing the order relation with the equality relation. It follows from elementary model-theoretic arguments that such *weak linear arithmetic theories* are strictly less expressive compared to their counterparts that include the order relation, since $h(x) := -x$ is an automorphism in structures without order but fails to be one in the presence of the order relation. For Presburger arithmetic, the weak definability problem was shown decidable by Choffrut and Frigeri, albeit with non-elementary time complexity [5]. To the best of the authors' knowledge, no decidability results on weak definability for the structure $\langle \mathbb{Q}, +, < \rangle$ are known. Following [5], *weak Presburger arithmetic* (alternatively *weak linear integer arithmetic)* refers to the first-order theory of $\langle \mathbb{Z}, +, = \rangle$ whereas *weak linear rational arithmetic* refers to the theory of $\langle \mathbb{Q}, +, = \rangle$.

The main contribution of this paper is to significantly improve existing algorithmic upper bounds for the weak definability problem by establishing elementary upper bounds over both $\mathbb{Q}$ and $\mathbb{Z}$: we show that weak definability is in coNP for rational linear arithmetic, and decidable in elementary time for Presburger arithmetic. To this end, we develop geometric criteria that precisely characterize when a relation is definable without order. We also establish complementary lower bounds. While a matching coNP lower bound in the rational case is easy to obtain, we furthermore establish a Π_2^{P} lower bound for the weak definability problem for Presburger arithmetic that holds even in a fixed dimension. This lower bound is obtained by establishing Π_2^{P}-completeness of the universality problem for ultimately periodic sets, i.e., semi-linear sets in dimension one. This positively answers one of the longest-standing open problems in the theory of semi-linear sets posed by Huynh in 1982 [8], who asked whether the universality problem for semi-linear sets in dimension one is Π_2^{P}-complete. Our lower bound moreover strengthens previously known Π_2^{P} lower bounds for the inclusion problems for linear sets which have only been obtained in recent years, see [4, Thm. 12] and [16].

2 The weak definability problem

Everywhere in this paper, we denote by $\mathbb{Q}$, $\mathbb{Z}$, $\mathbb{N}$ and $\mathbb{N}_+$ the rational numbers, integers, natural numbers including zero and the positive integers, respectively. Given $a, b \in \mathbb{Z}$, we write $[a, b]$ for $\{a, a+1, \ldots, b\}$ and $[a]$ as a shortcut for $[1, a]$. Throughout this paper, numbers are assumed to be encoded in binary.

Linear arithmetic theories. Quantifier-free formulas ψ of the linear arithmetic theories we consider in this paper are obtained as Boolean combinations of linear constraints of the form $a_1 \cdot x_1 + \cdots + a_n \cdot x_n \sim b$, where all a_i and b are integer constants, the x_i are first-order variables, and $\sim$ is a relation symbol $<$ or $=$. If ψ only contains equality symbols it is a formula of a weak linear arithmetic theory, in which case we call ψ *weak*. We can always without loss of generality assume that ψ is in negation normal form, and if ψ is not weak, we can furthermore assume that ψ is negation-free. Formulas Φ of linear arithmetic theories additionally allow for quantification over first-order variables and are of the form $Q_1 x_1 \cdots Q_n x_n\, \psi$, where each Q_i is $\exists$ or $\forall$, and ψ is a quantifier-free formula. We write $\|\Phi\|$ to denote the maximum of two and the maximum of the absolute values of all constants occurring in Φ, and $|\Phi|$ to denote the length of Φ which is the number of symbols required to write down Φ.

The semantics of such formulas is given with respect to the structures of linear rational arithmetic $\langle \mathbb{Q}, +, < \rangle$ and linear integer arithmetic $\langle \mathbb{Z}, +, < \rangle$. We call Φ weak whenever ψ is weak. Let $x_1, \ldots, x_n$ be the free variables of Φ, and let $\mathbb{D} \subseteq \mathbb{Q}$; often $\mathbb{D}$ will be $\mathbb{Q}$ or $\mathbb{Z}$. We write $[\![\Phi]\!]_{\mathbb{D}} \subseteq \mathbb{D}^n$ for the set of all variable assignments making Φ a true sentence in the structure $\langle \mathbb{D}, +, < \rangle$. We may drop the subscript $\mathbb{D}$ when $\mathbb{D}$ is clear from the context.

Definable relations and weak definability. Fix $\mathbb{D}$ to be either $\mathbb{Q}$ or $\mathbb{Z}$. A relation $R \subseteq \mathbb{D}^n$ is called $\mathbb{D}$-definable whenever there is a linear arithmetic formula $\Phi(x_1, \ldots, x_n)$ such that $(m_1, \ldots, m_n) \in R$ if and only if $(m_1, \ldots, m_n) \in [\![\Phi]\!]_{\mathbb{D}}$. In particular, we call R *weakly* $\mathbb{D}$*-definable* if the witnessing formula Φ is weak. The weak definability problem, the main decision problem considered in this paper, asks whether a relation definable by an arbitrary linear arithmetic formula is definable without order: *Given a linear arithmetic formula* Φ*, is the set* $[\![\Phi]\!]_{\mathbb{D}}$ *weakly* $\mathbb{D}$*-definable?*

Obviously, for a relation to be definable without order, it has to also be definable with order. In other words, weak definability, somewhat ironically, is a stronger property than definability.

▶ **Example 1.** Consider $\Phi(x) := x < 0 \vee x \geq 1$. It is not difficult to check that $[\![\Phi]\!]_{\mathbb{Q}}$ is not weakly $\mathbb{Q}$-definable. However, $[\![\Phi]\!]_{\mathbb{Z}}$ is weakly $\mathbb{Z}$-definable by $\Phi'(x) := \neg(x = 0)$, demonstrating that weak $\mathbb{Z}$-definability does not imply weak $\mathbb{Q}$-definability. Conversely, weak $\mathbb{Q}$-definability does not imply weak $\mathbb{Z}$-definability either. Let $\Phi(x) := \exists y\,(x > 0 \wedge x = 2 \cdot y) \vee (x < 0 \wedge x = 3 \cdot y)$. Then $[\![\Phi]\!]_{\mathbb{Z}}$ is not weakly $\mathbb{Z}$-definable, but $[\![\Phi]\!]_{\mathbb{Q}}$ is weakly $\mathbb{Q}$-definable via the same $\Phi'(x)$ as above.

When studying the complexity of weak definability problems, it makes sense to restrict the input formulas we consider. On the one hand, due to quantifier elimination (Fourier-Motzkin and Presburger's quantifier elimination procedures, respectively), quantifier-free formulas of linear rational arithmetic can define all sets definable in linear rational arithmetic, and existential formulas those in linear integer arithmetic. On the other hand, satisfiability can be reduced to deciding weak definability, meaning that if we allow arbitrary formulas as input, deciding weak definability is at least as hard as deciding linear rational and integer arithmetic, respectively. This does, however, blur the inherent complexity of deciding weak definability. For these reasons, we restrict formulas in instances of the weak definability problem to the most restricted fragments that are still expressively complete:

Weak $\mathbb{Q}$-definability: Given a *quantifier-free* formula Φ of linear arithmetic, decide whether $[\![\Phi]\!]_{\mathbb{Q}}$ is weakly $\mathbb{Q}$-definable.

Weak $\mathbb{Z}$-definability: Given an *existential* quantifier-free formula Φ of linear arithmetic, decide whether $[\![\Phi]\!]_{\mathbb{Z}}$ is weakly $\mathbb{Z}$-definable.[1]

By establishing an analogue of semilinear sets characterizing sets definable in weak integer arithmetic, Choffrut and Frigeri [5] have shown that weak $\mathbb{Z}$-definability is decidable. As the main result of the present paper, we show that deciding weak $\mathbb{Q}$-definability is coNP-complete, and deciding weak $\mathbb{Z}$-definability is elementary and Π_2^{P}-hard.

[1] One could alternatively take quantifier-free formulas with additional divisibility predicates $c \,|\, \cdot$ for every $c \in \mathbb{N}_+$.

3 Preliminaries

Linear algebra. We denote by $\boldsymbol{e}_i$ the i-th unit vector in any dimension. For $\boldsymbol{v} = (v_1,\dots,v_d) \in \mathbb{Z}^d$, we denote $\|\boldsymbol{v}\| := \max\{2, \max_{1\leq i\leq d}|v_i|\}$, and for $V \subseteq \mathbb{Q}^d$ we set $\|V\| := \max_{\boldsymbol{v}\in V}\|\boldsymbol{v}\|$. Likewise, for a matrix $\boldsymbol{A}$, $\|\boldsymbol{A}\|$ is defined as the maximum over $\|\boldsymbol{v}\|$ for every column vector $\boldsymbol{v}$ of $\boldsymbol{A}$. We sometimes treat finite sets $V \subseteq \mathbb{Q}^m$ with n elements as $m \times n$ matrices (e.g., by ordering the vectors in V lexicographically), which we denote by $\boldsymbol{V}$. Given $A, B \subseteq \mathbb{Q}^d$, the Minkowski sum of A and B is defined as $A + B := \{a + b : a \in A,\ b \in B\}$. If A or B is a singletons, we omit set brackets and write, e.g., $a + B$ instead of $\{a\} + B$. Analogously, we define $A \cdot B := \{a \cdot b : a \in A,\ b \in B\}$. We write $A \triangle B$ for the symmetric difference of A and B, i.e., $A \triangle B := (A \setminus B) \cup (B \setminus A)$.

A set $A \subseteq \mathbb{Q}^d$ is an *affine subspace* if $A = \boldsymbol{a} + V$ for some $\boldsymbol{a} \in \mathbb{Q}^d$ and a linear subspace $V \subseteq \mathbb{Q}^d$. Affine subspaces are sometimes called flats. The affine hull of any set $V \subseteq \mathbb{Q}^d$ is the smallest affine subspace containing V and is equal to

$$\operatorname{aff} V := \left\{\sum_{i=1}^{n} \lambda_i \cdot \boldsymbol{v}_i : n \in \mathbb{N},\ \boldsymbol{v}_i \in V,\ \lambda_i \in \mathbb{Q},\ i \in [n],\ \sum_{i=1}^{n} \lambda_i = 1\right\}.$$

The *dimension* of an affine subspace A, denoted $\dim A$, is defined as the dimension of the associated linear space A_0 such that $A = \boldsymbol{v} + A_0$ for some $\boldsymbol{v} \in \mathbb{Q}^d$. It is standard that this is well-defined. If affine subspaces A_1, A_2 are such that $A_1 \subseteq A_2$, then $\dim A_1 \leq \dim A_2$ (and moreover $A_1 = A_2$ iff $\dim A_1 = \dim A_2$).

A set S in $\mathbb{Q}^d$ (resp. $\mathbb{Z}^d$) *is vanishing (is zero, has measure zero) with respect to an affine subspace* $A \subseteq \mathbb{Q}^d$ if the set S is contained in a finite union of affine subpaces of A of dimension strictly less than A. Note that $S \subseteq A$ for any such S. For example, all finite subsets of $\mathbb{Q}^d$ have measure zero with respect to $\mathbb{Q}^d$ unless $d = 0$. If we do not specify the affine subspace A explicitly, $A = \mathbb{Q}^d$ is implicitly assumed.

The geometry of linear arithmetic. We recall some definitions and results from the literature on the geometry of solutions to systems of linear constraints.

It is well known that the set of solutions of a system of linear equations $\boldsymbol{B} \cdot \boldsymbol{x} = \boldsymbol{c}$ over the rationals is an affine subspace. Conversely, every affine subspace of $\mathbb{Q}^d$ is the set of solutions to a system of (linearly independent) equations with integer coefficients. For a representation $A = \{\boldsymbol{x} \in \mathbb{Q}^d : \boldsymbol{B} \cdot \boldsymbol{x} = \boldsymbol{c}\}$, we write $\|A\|$ to denote $\max\{\|\boldsymbol{B}\|, \|\boldsymbol{c}\|\}$, when $\boldsymbol{B}$ and $\boldsymbol{c}$ are determined by the context.

The Minkowski–Weyl theorem states that the set of points in a rational polyhedron $\boldsymbol{A} \cdot \boldsymbol{x} \geq \boldsymbol{b}$ can be represented as the sum of a bounded polytope and a convex cone, and vice versa. Given a finite set of vectors $V = \{\boldsymbol{v}_1, \dots, \boldsymbol{v}_n\} \subseteq \mathbb{Q}^d$, we define

$$\operatorname{conv} V := \left\{\sum_{i=1}^{n} \lambda_i \cdot \boldsymbol{v}_i : \sum_{i=1}^{n} \lambda_i = 1,\ \lambda_i \in \mathbb{Q}_{\geq 0},\ i \in [n]\right\} \quad \text{and}$$

$$\operatorname{cone} V := \left\{\sum_{i=1}^{n} \lambda_i \cdot \boldsymbol{v}_i : \lambda_i \in \mathbb{Q}_{\geq 0},\ i \in [n]\right\},$$

the *convex hull* of V and the *convex cone* generated by V, respectively. It follows that sets in $\mathbb{Q}^d$ definable in linear rational arithmetic can be obtained as finite unions of sets of the form $(\operatorname{conv} B + \operatorname{cone} P) \setminus A$, where A is a finite union of affine subspaces of $\mathbb{Q}^d$.

An effective version of the Minkowski–Weyl theorem over $\mathbb{Q}$ states that the two representations (as the intersection of half-spaces and as $\operatorname{conv} F + \operatorname{cone} G$ for finite sets F, G of generators) can be translated from one to another with a single exponential blowup (see, e.g., [15, Chapter 10]).

In the discrete setting, semilinear sets [11] characterize the sets of integer vectors definable in linear integer arithmetic [6]. For technical purposes, we give a slightly more generic definition. Let $\boldsymbol{b} \in \mathbb{Z}^d$ and $P = \{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_n\} \subseteq \mathbb{Z}^d$, we call $(\boldsymbol{b}, P)$ a *generator tuple* with *base* $\boldsymbol{b}$ and *periods* P. Fix $\mathbb{D}$ to be some subset of $\mathbb{Q}^d$, then the $\mathbb{D}$-*linear set* $L_{\mathbb{D}}(\boldsymbol{b}, P)$ generated by the tuple $(\boldsymbol{b}, P)$ is defined as

$$L_{\mathbb{D}}(\boldsymbol{b}, P) = \boldsymbol{b} + \mathbb{D} \cdot \boldsymbol{p}_1 + \cdots + \mathbb{D} \cdot \boldsymbol{p}_n \,.$$

Thus, $\mathbb{N}$-linear sets recover standard linear sets [11] as defined in the literature. Given a set $M \subseteq \mathbb{Z}^d$, we say that M is $\mathbb{D}$-*linear* if there is a generator tuple $(\boldsymbol{b}, P)$ such that $M = L_{\mathbb{D}}(\boldsymbol{b}, P)$. Clearly, $\mathbb{N}$-linear sets contain all $\mathbb{Z}$-linear sets, since $L_{\mathbb{Z}}(\boldsymbol{b}, P) = L_{\mathbb{N}}(\boldsymbol{b}, P \cup -P)$, but the converse does not hold: it is easy to prove that $\mathbb{N}$ is $\mathbb{N}$-linear but not $\mathbb{Z}$-linear.

We say that M is a *hybrid* $\mathbb{D}$-*linear set* if it is a union of the form

$$L_{\mathbb{D}}(B, P) = \bigcup_{\boldsymbol{b} \in B} L_{\mathbb{D}}(\boldsymbol{b}, P)$$

where both sets $B, P \subseteq \mathbb{Z}^d$ are finite. A $\mathbb{D}$-linear or hybrid $\mathbb{D}$-linear set, as defined above, is k-*dimensional* if the linear span of the set P has dimension k; it is *full-dimensional* if $k = d$. One may think of k-dimensional $\mathbb{Z}$-linear sets as of shifted lattices (or cosets of lattices) inside k-dimensional affine subspaces.

We say that M is $\mathbb{D}$-*semilinear* if it is a finite union of $\mathbb{D}$-linear sets. We call a semilinear set *proper* if all sets of period vectors of those linear sets are linearly independent.

Hybrid $\mathbb{N}$-linear sets (often just *hybrid linear sets* for short) are discrete analogues of convex polyhedra; they are exactly sets of integer solutions to systems of linear inequalities $\boldsymbol{A} \cdot \boldsymbol{x} \geq \boldsymbol{c}$ [17, 2], which implies that sets definable in linear integer arithmetic are $\mathbb{N}$-semilinear [6]. Similarly, integer solutions of systems of linear equations are $\mathbb{Z}$-linear:

▶ **Proposition 2.** *Suppose $\boldsymbol{A} \cdot \boldsymbol{x} = \boldsymbol{c}$ has a solution in $\mathbb{Z}^d$. Then its set of solutions in $\mathbb{Z}^d$ is a proper $\mathbb{Z}$-linear set $L_{\mathbb{Z}}(\boldsymbol{b}, P)$, with $\|\boldsymbol{b}\| \leq 2^{O(d^4 \log d)} \cdot \|\boldsymbol{A}\|^{O(d^4)} \cdot \|\boldsymbol{c}\|$ and $\|P\| \leq 2^{O(d^4 \log d)} \cdot \|\boldsymbol{A}\|^{O(d^4)}$. Moreover, $|P| = d - \operatorname{rank} \boldsymbol{A}$ and the vectors in P are fully determined by $\boldsymbol{A}$ (i.e., independent of $\boldsymbol{c}$).*

It should be noted, however, that $\mathbb{Z}$-semilinear sets do not fully characterize sets definable in weak linear integer arithmetic. Choffrut and Frigeri [5] have shown that $M \subseteq \mathbb{Z}^d$ is definable in weak linear arithmetic if and only if $M = \bigcup_{i \in I} S_i \setminus T_i$, where the S_i are proper $\mathbb{Z}$-linear sets, T_i proper $\mathbb{Z}$-semilinear sets, and $T_i \subseteq S_i$ for all $i \in I$.

Descriptional complexity. The complexity upper bounds we obtain in this paper rely on bounds on the constants in the generator representation of sets definable in linear arithmetic theories. Given a $\mathbb{D}$-semilinear set $S \subseteq \mathbb{Z}^d$ in generator representation, $S = \bigcup_{i \in I} L(B_i, P_i)$, we define $\|S\| := \max_{i \in I} \max \|B_i \cup P_i\|$.

▶ **Proposition 3.** *Let Φ be an existential formula of linear integer arithmetic and $S = [\![\Phi]\!]_{\mathbb{Z}}$. Then $S = \bigcup_{i \in I} L(B_i, P_i)$ such that $\log\|S\|, \log|I| \leq \operatorname{poly}(|\Phi|)$.*

Proof. For a system of linear inequalities $\boldsymbol{A} \cdot \boldsymbol{x} \geq \boldsymbol{c}$, the statement follows, e.g., from [17]. Moreover, the disjunctive normal form of Φ consists of at most $2^{|\Phi|}$ conjunctions, each of which is a system of linear inequalities, from which the statement follows. ◀

4 A characterisation of weak $\mathbb{Q}$- and $\mathbb{Z}$-definability

We now establish properties that fully characterize when a subset of $\mathbb{Q}^d$ is weakly $\mathbb{Q}$- and weakly $\mathbb{Z}$-definable, respectively.

▶ **Definition 4.** *Suppose $X \subseteq \mathbb{Q}^d$. We say that X:*

- *has* 0–1 property *with respect to an affine subspace A if either $X \cap A$ or $A \setminus X$ is contained in a finite union of affine subspaces of dimension* $\dim A - 1$*,*
- *has* hierarchical 0–1 property *with respect to an affine subspace A if it has 0–1 property with respect to A, where the subspaces of lower dimension are some $H_1, \ldots, H_m$, and, if* $\dim A > 1$*, it has hierarchical 0–1 property with respect to each H_i, $1 \leq i \leq m$,*
- *has* (hierarchical) 0–1 property *if it has (hierarchical) 0–1 property with respect to $\mathbb{Q}^d$,*
- *has* global 0–1 property *if, for every affine subspace $A \subseteq \mathbb{Q}^d$, it has 0–1 property with respect to A,*
- *has* ℓ-bounded 0–1 property *if, for every affine subspace $A = \{\boldsymbol{x} \in \mathbb{Q}^d : \boldsymbol{B} \cdot \boldsymbol{x} = \boldsymbol{c}\}$ with $\|\boldsymbol{B}\|, \|\boldsymbol{c}\| \leq \ell$, it has 0–1 property with respect to A.*

The term “0–1 property” refers to the intuition that a (weakly $\mathbb{Q}$-definable) set must either vanish (“zero”) or fill almost all the space (“one”). The hierarchical version of the property basically says, “and the same holds recursively for these subspaces of lower dimension.”

The following theorem, whose proof is deferred to the full version of the paper, shows how these properties relate to weak $\mathbb{Q}$-definability.

▶ **Theorem 5.** *For all sets $X \subseteq \mathbb{Q}^d$ the following statements are equivalent:*

- *X is weakly $\mathbb{Q}$-definable,*
- *X has hierarchical 0–1 property,*
- *X has global 0–1 property, and*
- *X has $\|\Phi\|$-bounded 0–1 property, where Φ is a quantifier-free formula of linear rational arithmetic such that $X = [\![\Phi]\!]$.*

▶ **Example 6.** To illustrate the global version of the property, we demonstrate how from Theorem 5 we can derive that $\Phi(x, y) := x \geq 0 \wedge y = 0$ is not weakly $\mathbb{Q}$-definable. Observe that although $[\![\Phi]\!]$ satisfies the 0–1 property, it fails the global 0–1 property with $A = \{(x, 0) : x \in \mathbb{Q}\}$. Indeed, both $[\![\Phi]\!] \cap A$ and $A \setminus [\![\Phi]\!]$ are infinite sets, whereas every affine subspace of A of dimension lower than A, i.e., of dimension zero, is a single point. As a consequence, $[\![\Phi]\!] \cap A$ is not contained in a finite union of such subspaces, and neither is $A \setminus [\![\Phi]\!]$.

In comparison, $\Psi(x, y) := y = 0$ satisfies the global 0–1 property. We observe, for example, that for $A = \{(x, 0) : x \in \mathbb{Q}\}$ the set $A \setminus [\![\Psi]\!]$ is empty and thus contained in a finite union of subspaces of A of dimension $0 < \dim A$.

Note that the ℓ-bounded version of the property will later give a decision procedure for weak $\mathbb{Q}$-definability.

Weakly $\mathbb{Z}$-definable sets do not have to satisfy an immediate analogue of the 0–1 property: $2 \cdot \mathbb{Z} \subseteq \mathbb{Q}$ is an example of a weakly $\mathbb{Z}$-definable set failing the property. The following definition bridges this gap and requires instead that almost all the space is “tiled” by the set X in a periodic manner. Empty tiling (almost all space is empty, i.e., X vanishes) and full tiling ($\mathbb{Z}^d \setminus X$ vanishes in $\mathbb{Q}^d$) are special cases of this.

▶ **Definition 7.** *Suppose $X \subseteq \mathbb{Z}^d$. We say that X:*

- *has* mosaic property *with respect to an affine subspace A if there exists a hybrid $\mathbb{Z}$-linear set $F \subseteq A$ of dimension* $\dim A$ *such that $(X \cap A) \bigtriangleup F$ is contained in a finite union of affine subspaces inside A of dimension* $\dim A - 1$*,*

- *has* hierarchical mosaic property with respect to an affine subspace A *if it has mosaic property with respect to A, where the subspaces of lower dimension are some $H_1, \dots, H_m$, and, if* $\dim A > 1$*, it has hierarchical mosaic property with respect to each H_i, $1 \le i \le m$,*
- *has* (hierarchical) mosaic property *if it has (hierarchical) mosaic property with respect to $\mathbb{Q}^d$,*
- *has* global mosaic property *if, for every affine subspace $A \subseteq \mathbb{Q}^d$, it has mosaic property with respect to A,*
- *has* ℓ-bounded mosaic property *if, for every affine subspace $A = \{\boldsymbol{x} \in \mathbb{Q}^d : \boldsymbol{B} \cdot \boldsymbol{x} = \boldsymbol{c}\}$ with $\|\boldsymbol{B}\|, \|\boldsymbol{c}\| \le \ell$, it has mosaic property with respect to A.*

The intuition behind the variants of the mosaic property is the same as for the 0–1 property.

▶ **Example 8.** We show how the sets defined by the following logical formulae over the integers satisfy the hierarchical mosaic property.

1. $\Phi_1(x, y) := \exists u \exists v.(x = 3u \lor x = 3u + 1) \land y = 2v$. The mosaic property w.r.t. $\mathbb{Q}^2$ holds with $F = L_{\mathbb{Z}}(\{(0, 0), (1, 0)\}, \{(3, 0), (0, 2)\})$, because $[\![\Phi_1]\!]_{\mathbb{Z}} = F$. So the hierarchical mosaic property (w.r.t. $\mathbb{Q}^2$) holds too, with $m = 0$.
2. $\Phi_2(x, y) := (x = 0) \lor (y > 0) \lor (y < 0)$. The mosaic property w.r.t. $\mathbb{Q}^2$ holds with $F = \mathbb{Z}^2 = L_{\mathbb{Z}}(\mathbf{0}, \{\boldsymbol{e}_1, \boldsymbol{e}_2\})$, because we can pick an "exceptional" subspace $H_1 = \{(x, y) \in \mathbb{Q}^2 : y = 0\}$. Going inside H_1, we notice that $[\![\Phi_2]\!]_{\mathbb{Z}}$ also has the mosaic property w.r.t. H_1, because its intersection with H_1 is the singleton $\{\mathbf{0}\}$, so we can pick $F' = \emptyset$ and zero-dimensional subspace $H_{1,1} = \{\mathbf{0}\}$. Therefore, $[\![\Phi_2]\!]_{\mathbb{Z}}$ satisfies the hierarchical mosaic property (w.r.t. $\mathbb{Q}^2$).
3. $\Phi_3(x, y, z) := (y = 0 \land z = 0) \lor (x = 0 \land (\exists t. y = 2t) \land z = 0) \lor (x = 0 \land y = 0 \land (\exists t. z = 3t + 1))$. For the mosaic property w.r.t. $\mathbb{Q}^3$, we can pick $F = \emptyset$ with $H_1 = [\![x = 0]\!]_{\mathbb{Q}}$ and $H_2 = [\![y = 0]\!]_{\mathbb{Q}}$. The mosaic property w.r.t. H_1 holds with $F_1 = \emptyset$ and $H_{1,1} = [\![x = 0 \land z = 0]\!]_{\mathbb{Q}}$, $H_{1,2} = [\![x = 0 \land y = 0]\!]_{\mathbb{Q}}$. Similarly, the mosaic property w.r.t. H_2 holds with $F_2 = \emptyset$ and $H_{2,1} = [\![y = 0 \land z = 0]\!]_{\mathbb{Q}}$, $H_{2,2} = H_{1,2}$. We leave it as an exercise to the reader to check that $[\![\Phi_3]\!]_{\mathbb{Z}}$ has the mosaic property with respect to each of $H_{1,1}$, $H_{1,2}$, and $H_{2,1}$. In conclusion, $[\![\Phi_3]\!]_{\mathbb{Z}}$ has the hierarchical mosaic property (w.r.t. $\mathbb{Q}^3$).

The proof of the following theorem is deferred to the full version of this paper:

▶ **Theorem 9.** *There exists a function $f(s, d) \colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, $f(n, d) = s^{d^{O(d)}}$, such that for all sets $X \subseteq \mathbb{Z}^d$ the following statements are equivalent:*

- *X is weakly $\mathbb{Z}$-definable,*
- *X has hierarchical mosaic property,*
- *X has global mosaic property, and*
- *X has $f(\|\Phi\|, d)$-bounded mosaic property, where Φ is an existential formula of linear integer arithmetic such that $X = [\![\Phi]\!]$.*

The function f gives an upper bound on the magnitude of coefficients in systems of linear equations that specify relevant affine subspaces. We can see from the theorem that, for fixed dimension d, the function f is at most polynomial; and in arbitrary dimension, the bit size of required coefficients is at most $d^{O(d)} \cdot \log\|\Phi\|$, a single exponential in d.

4.1 More on mosaic property

We introduce the following problem.

<u>Piece of Mosaic</u>

INPUT: Semilinear set $S \subseteq \mathbb{Z}^d$ in generator representation, system of equations $\boldsymbol{B}\cdot\boldsymbol{x} = \boldsymbol{c}$ with integer coefficients defining an affine subspace $A \subseteq \mathbb{Q}^d$.

OUTPUT: Hybrid $\mathbb{Z}$-linear set F represented as $F = \bigcup_{j\in J} L_{\mathbb{Z}}(C_j, Q_j)$ and such that, if S has mosaic property w.r.t. A, then $(S \triangle F) \cap A$ vanishes w.r.t. A.

Note that this is a promise problem: if S does *not* satisfy the mosaic property w.r.t. A, then no restriction is imposed on F.

The intuition for the name of this problem is that an algorithm for it "finds a piece of mosaic", F, in the set S: if S as a whole satisfies the mosaic property (w.r.t. A), then $S \cap A$ should really look like F everywhere in A (except for maybe lower-dimensional exceptions). But it is also possible that some parts of S are unlike F, and then S does not satisfy the mosaic property (in which case the algorithm may output any hybrid $\mathbb{Z}$-linear set).

Also note that, in the case that S satisfies the mosaic property, although the set F must be hybrid $\mathbb{Z}$-linear, the algorithm is permitted to output a $\mathbb{Z}$-semilinear representation of it. This is because the bit size and the norm can grow significantly if a hybrid $\mathbb{Z}$-linear representation is required. (For example, in dimension one, the set $\bigcup_{i=1}^{n} L_{\mathbb{Z}}(0, i)$ is hybrid $\mathbb{Z}$-linear, but all its representations as $L_{\mathbb{Z}}(C, Q)$ have bit size superpolynomial in n.)

▶ **Lemma 10.** *Let $t := (\|S\| + \|A\|)^{\mathrm{poly}(d)}$. The following statements hold:*

1. *If a semilinear set S satisfies the mosaic property with respect to an affine subspace A, then*
   - (a) *the set F in the definition of the property is determined uniquely and*
   - (b) *$(S \cap A) \triangle F$ is contained in a finite collection of affine subspaces of A of dimension $\dim A - 1$ each, defined by linear equations with integer coefficients of absolute value at most t.*
2. *There is a t-time algorithm that solves the Piece of Mosaic problem, which always produces an $F = \bigcup_{j\in J} L_{\mathbb{Z}}(C_j, Q_j)$ such that $\|C_j\|, \|Q_j\| \le t$.*

The proof is given in the full version of this paper.

▶ **Lemma 11.** *There is an algorithm with running time $2^{(\|A\|+\|S\|)^{\mathrm{poly}(d)}}$ that, given a semilinear set $S = \bigcup_{i\in I} L(B_i, P_i)$ and a system of equations $\boldsymbol{B}\cdot\boldsymbol{x} = \boldsymbol{c}$ with integer coefficients defining an affine subspace $A \subseteq \mathbb{Q}^d$, decides whether S has the mosaic property with respect to A.*

Proof. Run the algorithm from Lemma 10(2) to obtain a hybrid $\mathbb{Z}$-linear set F such that $\|F\| \le t$ with t defined as in the lemma. To check whether S has mosaic property, we determine whether $(S \cap A) \triangle F$ is vanishing. By Proposition 2, $A \cap \mathbb{Z}^d = L(C, Q)$ such that $\|C\|, \|Q\| \le \|A\|^{\mathrm{poly}(d)}$. Then from [2, Thm. 6], we get that $S \cap A = \bigcup_{k\in K} L(D_k, E_k)$ such that $\|D_k\|, \|E_k\| \le (\|S\| + \|A\|)^{\mathrm{poly}(d)}$. We compute the semilinear representation M of $((S \cap A) \setminus F) \cup (F \setminus (S \cap A))$. It follows the bounds on the description size of (each) set difference [2, Cor. 22] that $\log\|M\| \le (\|A\| + \|S\|)^{\mathrm{poly}(d)}$. In order to check whether $(S \cap A) \triangle F$ vanishes w.r.t. to A, it remains to iterate over all hybrid linear sets defining M and to check whether the dimension of the affine hull of the period vectors in each set is strictly less than $\dim A$. We remark that the upper bound on the running time holds, because $\|A\|$ and $\|S\|$ are at most exponential in the bit size of the encoding. ◀

▶ **Lemma 12.** *There is a doubly exponential algorithm that, given $\Phi \subseteq \mathbb{Z}^d$, decides whether $[\![\Phi]\!]$ has the $f(\|\Phi\|, d)$-bounded mosaic property, where f is defined as in Theorem 9.*

Proof. It follows from Proposition 3 that we can compute in time $2^{\text{poly}(|\Phi|)}$ the semilinear representation of $[\![\Phi]\!]$. The lemma then follows by an application of Lemma 11; note that the running time of the algorithm of that lemma has only a single exponential dependency on $\|S\| + \|A\|$. ◀

5 Computational complexity of weak $\mathbb{Q}$- and $\mathbb{Z}$-definability

Building upon the results in the previous sections, we now prove the main theorem of this paper.

▶ **Theorem 13.** *The weak $\mathbb{Q}$-definability problem is coNP-complete, and the weak $\mathbb{Z}$-definability problem is Π_2^{P}-hard and can be decided by an algorithm with elementary running time.*

We first outline the upper bounds of this theorem and then the lower bounds. The lower bound for weak $\mathbb{Z}$-definability already holds in a fixed dimension and is obtained from showing that the universality problem for one-dimensional semi-linear sets is Π_2^{P}-hard. We give the proof of Π_2^{P}-hardness of this universality problem in a separate Section 6.

5.1 Upper bounds for deciding weak $\mathbb{Q}$- and $\mathbb{Z}$-definability

By Theorem 5, a set $X \subseteq \mathbb{Q}^d$ given by a quantifier-free formula Φ of linear rational arithmetic is weakly $\mathbb{Q}$-definable if and only if for every affine subspace A, given by a system of linear equations $\boldsymbol{A} \cdot \boldsymbol{x} = \boldsymbol{c}$ such that $\|\boldsymbol{A}\|, \|\boldsymbol{c}\| \leq \|\Phi\|$, either $X \cap A$ or $A \setminus X = A \cap \overline{X}$ is contained in a finite union of affine subspaces of dimension $\dim A - 1$. To prove that Φ is not weakly $\mathbb{Q}$-definable, we attempt to find an affine subspace A such that neither $X \cap A$ nor $A \setminus X$ is contained in a finite union of affine subspaces of dimension $\dim A - 1$. Note that due to Φ being quantifier-free, $\neg\Phi$ is also a quantifier-free formula of linear rational arithmetic. Hence it will suffice to only discuss the case $X \cap A$; and, in this case, we can also assume without loss of generality that Φ is negation-free. Let Ψ be the disjunctive normal form of Φ. We clearly have $[\![\Psi]\!] = X$, and $[\![\Psi]\!] \cap A$ is not contained in a finite union of hyperplanes of dimension $\dim A - 1$ if and only if for some polyhedron P, defined by a conjunction of Ψ, the polyhedron $P \cap A$ has dimension $\dim A$. We now outline how to decide in polynomial time whether $\dim(P \cap A)$ equals $\dim A$.

Both P and A are given as systems of linear constraints, and hence we immediately obtain $P \cap A$ as a system of linear constraints. The dimension $\dim(P \cap A)$ is, by definition, equal to $\dim \operatorname{aff}(P \cap A)$. When $P \cap A$ is given by a system of non-strict linear inequalities, one can obtain in polynomial time a representation of $\operatorname{aff}(P \cap A)$ as the intersection of at most d implicit equalities of $P \cap A$, each obtained from a row of the system defining $P \cap A$ [15, p. 100]. One can then compute $\dim \operatorname{aff}(P \cap A)$, by (a variant of) Gaussian elimination [15, Section 3.3]. In general, the system of constraints defining P may contain strict inequalities though. However, this does not cause any problems, since for a polyhedron P, $\dim(P) = \dim(\operatorname{cl} P)$, where $\operatorname{cl} P$ is the closure of P. If $P \neq \emptyset$ and given as a system of linear constraints, $\operatorname{cl} P$ can be obtained by making all strict inequalities in the defining system of P non-strict.

From this line of reasoning, we obtain a coNP upper bound for deciding weak $\mathbb{Q}$-definability as follows. We guess $\boldsymbol{A}$ and $\boldsymbol{c}$ above in non-deterministic polynomial time. While the disjunctive normal forms of Φ and $\neg\Phi$ (both assumed negation-free, with no loss of generality) can be exponentially long, we can – given the formulas Φ and $\neg\Phi$ – guess a single conjunction of their disjunctive normal forms in non-deterministic polynomial time, by inspecting the structure of Φ and $\neg\Phi$. These two conjunctions induce polyhedra P and P'. We then decide in polynomial time whether the dimension of the polyhedra $P \cap A$ and $P' \cap A$ equals $\dim A$. We reject if and only if this is the case for both conjunctions.

We now turn towards a sketch of the upper bound for weak $\mathbb{Z}$-definability. To this end, let $X \subseteq \mathbb{Z}^d$ be defined by an existential formula $\Phi(\boldsymbol{x})$ of linear integer arithmetic. For an elementary upper bound, following Theorem 9 we iterate over all affine subspaces A given by systems of equations $\boldsymbol{B} \cdot \boldsymbol{x} = \boldsymbol{c}$ such that $\|\boldsymbol{B}\|, \|\boldsymbol{c}\|$ are at most $f(\|\Phi\|, d)$ and check whether Φ has mosaic property with respect to A. By Lemma 12, there is a doubly exponential algorithm that achieves this.

5.2 Lower bounds for deciding weak $\mathbb{Q}$-definability and $\mathbb{Z}$-definability

Lower bound for weak $\mathbb{Q}$-definability. We show a matching coNP-lower bound for deciding weak $\mathbb{Q}$-definability by a reduction from the problem of deciding whether a Boolean 3-DNF formula is a tautology. Let $\psi = \psi_1 \vee \cdots \vee \psi_k$ be in 3-DNF over Boolean variables $X_1, \ldots, X_d$. Let ϕ_1 be the formula of rational arithmetic obtained from ψ by applying the function h to every literal, with h defined as $h(X_i) := x_i = 1$ and $h(\neg X_i) := x_i = 0$. In addition, define

$$\phi_2 := \bigvee_{1 \le i \le d} x_i > 1 \vee x_i < 0 \vee (0 < x_i < 1)$$

Observe that $[\![\phi_1 \vee \phi_2]\!]$ is the whole of $\mathbb{Q}^d$ except possibly a finite set of points corresponding to those truth assignments that do not make ψ evaluate to true, i.e., $[\![\phi_1 \vee \phi_2]\!] = \mathbb{Q}^d$ if and only if ψ is a tautology. Now define $\Phi := \phi_1 \vee \phi_2 \vee u > 0$, where u is a fresh variable. Note that $[\![\Phi]\!]$ is the whole of $\mathbb{Q}^{d+1}$ except possibly several half-lines that correspond to assignments falsifying ψ. If a half-line is missing then $[\![\Phi]\!]$ does not satisfy the global 0–1 property and is therefore not weakly $\mathbb{Q}$-definable, by Theorem 5. Otherwise no half-line is missing, ψ is a tautology, $[\![\Phi]\!]$ is equal to $\mathbb{Q}^d$ and is weakly $\mathbb{Q}$-definable.

Lower bound for weak $\mathbb{Z}$-definability. We show a Π_2^{P}-lower bound for weak $\mathbb{Z}$-definability via a reduction from the universality problem for semilinear sets. Given a semilinear set $M \subseteq \mathbb{N}^d$ in the generator representation, the *universality problem* is to decide whether $M = \mathbb{N}^d$. It was asked by Huynh [8] whether this problem is Π_2^{P}-hard when he established a Π_2^{P}-upper bound for this problem. We show in the next section that this is the case, even in dimension one, and assume hardness for now to show our lower bound for weak $\mathbb{Z}$-definability.

To this end, let $M = \bigcup_{i \in I} L(\boldsymbol{b}_i, P_i) \subseteq \mathbb{N}^d$. One easily constructs an existential formula $\psi(x_1, \ldots, x_d)$ of linear integer arithmetic such that $M = [\![\psi]\!]$. Now consider $\Phi(u, x_1, \ldots, x_d) := \psi \vee u > 0 \vee \bigvee_{i \in [d]} x_i < 0$, where u is a fresh variable. Analogously to what we showed above, $[\![\Phi]\!]$ is the whole of $\mathbb{Z}^{d+1}$ except possibly several "discrete half-lines" corresponding to elements $\boldsymbol{v} \in \mathbb{N}^d \setminus M$. By the global mosaic property (Theorem 9), $[\![\Phi]\!]$ is weakly $\mathbb{Z}$-definable if and only if no half-line is missing.

6 A lower bound for deciding universality of semilinear sets

We exclusively deal with $\mathbb{N}$-semilinear sets in this section and so, when presenting such semilinear sets in generator presentation, for readability, instead of writing, e.g., $L_{\mathbb{N}}(\boldsymbol{b}, P)$ we subsequently drop the subscript $\mathbb{N}$ and simply write $L(\boldsymbol{b}, P)$. The main result of this section is a Π_2^{P} lower bound for the universality problem for ultimately periodic sets, which are semilinear sets in dimension one:

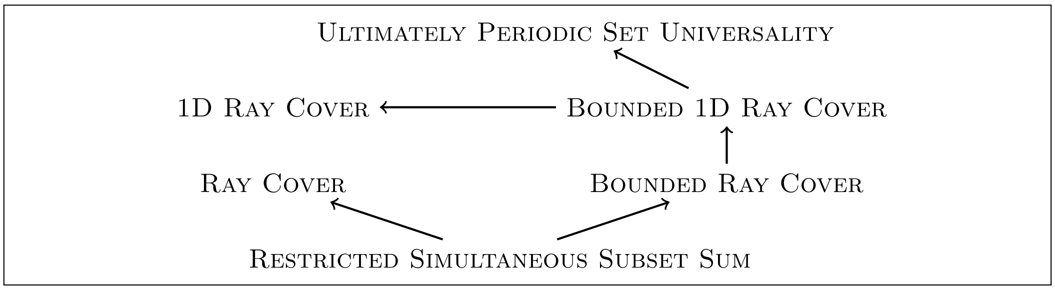

Ultimately Periodic Set Universality
1D Ray Cover
Bounded 1D Ray Cover
Ray Cover
Bounded Ray Cover
Restricted Simultaneous Subset Sum

Figure 1 Reductions in the proof of Theorem 14. $X \to Y$ denotes a logarithmic-space reduction from X to Y.

Ultimately Periodic Set Universality

INPUT: Finite set I and, for each $i \in I$, a number $b_i \in \mathbb{N}$ and a finite set $P_i \subseteq \mathbb{N}$.
QUESTION: Is $\bigcup_{i \in I} L(b_i, P_i) = \mathbb{N}$?

Since a set $M \subseteq \mathbb{N}^d$ is universal if and only if $L(\mathbf{0}, \{\boldsymbol{e}_1, \ldots, \boldsymbol{e}_d\}) \subseteq M$, this universality problem is a special case of *inclusion* for semilinear sets, which asks to decide if $N \subseteq M$ for semilinear sets M, N. While inclusion for semilinear sets has been known to be Π_2^{P}-complete since the 1980s [8], the lower bound has only been strengthened to hold for inclusion of *linear* sets as recently as 2018 [4, Thm. 12], and then also to linear sets in dimension one [16]. It has also recently been shown that universality for linear sets is decidable in polynomial time, even for hybrid linear sets of the form $L(B, P) := \bigcup_{\boldsymbol{b} \in B} L(\boldsymbol{b}, P)$ [3].

Our new Π_2^{P} lower bound is based on a chain of reductions between intermediate problems that we now introduce. The overall reduction chain is displayed in Figure 1. Our starting point is the Simultaneous Subset Sum problem introduced in [4] from which we derive a slightly restricted version. Both problems are defined in Figure 2 and are variants of the classical subset sum problem. They ask whether all elements in a finite arithmetic progression can be obtained as sums of subsets of a given set $W \subseteq \mathbb{N}$.

Via a reduction from Restricted Simultaneous Subset Sum, we prove Π_2^{P}-hardness of two special cases of the semilinear set inclusion problem, Bounded Ray Cover and Ray Cover, which are defined in Figure 3. The problem Ray Cover asks whether a discrete ray in $\mathbb{N}^d$ (basically an arithmetic progression) is contained in an integer cone (linear set) – this is a restricted variant of linear set inclusion. The problem Bounded Ray Cover is the same but only concerns a finite segment of the ray. The Π_2^{P}-hardness of Ray Cover follows from that of Bounded Ray Cover.

Next, by a reduction from Bounded Ray Cover, we show Π_2^{P}-hardness of the one-dimensional versions of the ray cover problems, formally defined in Figure 4. A reduction from Bounded 1D Ray Cover will then give the desired Π_2^{P}-lower bound of Ultimately Periodic Set Universality.

▶ **Theorem 14.** *All six problems in Figure 1 are* Π_2^{P}*-complete.*

All upper bounds in Theorem 14 are easily obtained from the observation that the respective problems either reduce to semi-linear set inclusion, which is in Π_2^{P} [8], or involve sets of numbers of polynomial bit size.

To prove the lower bounds, once the intermediate problems have been identified, one of the key insights is in the reduction from Bounded Ray Cover to Bounded 1D Ray Cover, which maps a problem from $\mathbb{N}^d$ into $\mathbb{N}$. In the remainder of this section, we focus on the techniques that enable us to overcome this challenge.

SIMULTANEOUS SUBSET SUM

INPUT: Finite set $W \subseteq \mathbb{N}$, and $t, h, 2^m \in \mathbb{N}$ such that $t < h$.

QUESTION: For every $i \in [0, 2^m - 1]$, does there exist a $W' \subseteq W$ such that $\sum W' = t + i \cdot h$?

RESTRICTED SIMULTANEOUS SUBSET SUM

INPUT: Finite set $W \subseteq \mathbb{N}$ and $t, 2^k, 2^m \in \mathbb{N}$ such that $t < 2^k$.

QUESTION: For every $i \in [0, 2^m - 1]$, does there exist a $W' \subseteq W$ such that $\sum W' = t + i \cdot 2^k$?

Figure 2 The simultaneous subset sum problem introduced in [4] and its restricted version.

RAY COVER

INPUT: Finite set $P \subseteq \mathbb{N}^d$ and $a, 2^k \in \mathbb{N}$ such that $a < 2^k$.

QUESTION: Does $L(\mathbf{0}, P) \supseteq L(\boldsymbol{a}, \boldsymbol{b})$, where $\boldsymbol{a} = (a, \mathbf{1})$ and $\boldsymbol{b} = (2^k, \mathbf{0})$?

BOUNDED RAY COVER

INPUT: Finite set $P \subseteq \mathbb{N}^d$ and $a, 2^k, 2^m \in \mathbb{N}$ such that $a < 2^k$.

QUESTION: Does $L(\mathbf{0}, P) \supseteq L(\boldsymbol{a}, \boldsymbol{b}) \cap [0, B]^d$, where $\boldsymbol{a} = (a, \mathbf{1})$, $\boldsymbol{b} = (2^k, \mathbf{0})$, and $B = a + (2^m - 1) \cdot 2^k$?

Figure 3 Ray cover problems in arbitrary dimensions.

1D RAY COVER

INPUT: Finite set $P \subseteq \mathbb{N}$ and $a, 2^k \in \mathbb{N}$ such that $a < 2^k$.

QUESTION: Does $L(0, P) \supseteq L(a, b)$, where $b = 2^k$?

BOUNDED 1D RAY COVER

INPUT: Finite set $P \subseteq \mathbb{N}$ and $a, 2^k, 2^m \in \mathbb{N}$ such that $a < 2^k$.

QUESTION: Does $L(0, P) \supseteq L(a, b) \cap [0, B]$, where $b = 2^k$ and $B = a + (2^m - 1) \cdot b$?

Figure 4 Ray cover problems in dimension one.

6.1 Aggregation of several dimensions into one

Aggregation is an important technique in the theory of integer programming (see, e.g., Schrijver's book [15, Sections 16.6 and 18.2]). It has been used by Huynh [8] and Simon [16] for showing lower bounds for the inclusion problem for (semi)linear sets in dimension one. Aggregation can be achieved in the setting of linear Diophantine equations over nonnegative (integer) variables, following a classic result of Glover and Woolsey [7], which we extend.

As we already mentioned, we apply aggregation in order to reduce BOUNDED RAY COVER to BOUNDED 1D RAY COVER. In the former problem, think of vectors $\boldsymbol{v} \in L(\boldsymbol{a}, \boldsymbol{b}) \cap [0, B]^d$ as targets to hit. Each of them is hit if and only if $L(\mathbf{0}, P)$ contains it, i.e., if the system of equations $\boldsymbol{P} \cdot \boldsymbol{x} = \boldsymbol{v}$ has a solution in $\mathbb{N}^d$. The instance is a yes-instance if and only if all 2^m targets are hit. While Glover and Woolsey's result would allow us to aggregate *one* system of equations like this (for a single target) into a single equation, the key technical challenge in our setting is that – in contrast to [7, Thm. 3] – we need to aggregate *several* such systems, for 2^m different targets (each into its own one equation). That is, these systems have identical coefficients, but different constant terms.

For the following lemma, we will explain the rationale behind the constraints “α is a power of two” and “$\alpha/\beta > \max B$” subsequently.

▶ **Lemma 15.** *Let A, B be finite subsets of $\mathbb{N}_+$. Then there exist $\alpha, \beta \in \mathbb{N}$ such that α is a power of two, $\alpha/\beta > \max B$, and, for each pair $(a, b) \in A \times B$, every system of equations*

$$\begin{cases} \boldsymbol{c} \cdot \boldsymbol{x} = a \\ \boldsymbol{d} \cdot \boldsymbol{x} = b \end{cases}$$

with $\boldsymbol{c}, \boldsymbol{d} \in \mathbb{N}^d$ and $(a, b) \in A \times B$ has the same set of solutions in $\mathbb{N}^d$ as the single equation

$$(\alpha \boldsymbol{c} + \beta \boldsymbol{d}) \cdot \boldsymbol{x} = \alpha a + \beta b\,.$$

Moreover, α and β are polynomial-time computable from $\max A$ and $\max B$ and are independent of $\boldsymbol{c}$ and $\boldsymbol{d}$.

Proof. It follows from [7, Thm. 3] of Glover and Woolsey that when A and B are singletons $A = \{a\}$ and $B = \{b\}$ the statement holds if

$$\alpha > b, \qquad \beta > a, \quad \text{and} \quad \gcd(\alpha, \beta) = 1\,. \tag{1}$$

We show how to find a *single pair* of coefficients $\alpha, \beta \in \mathbb{N}$ that make Glover and Woolsey’s result applicable to the system in question for *all* pairs $(a, b) \in A \times B$. Indeed, choose r as the smallest power of two such that $r - 1$ strictly exceeds both $\max A$ and $\max B$. Pick $\beta = r - 1 > \max A$ and $\alpha = r^2$. We now check that α and β satisfy conditions (1). Observe that

$$\begin{aligned} \alpha &= r^2 > r - 1 > b, && \text{for all } b \in B, \\ \beta &= r - 1 > a, && \text{for all } a \in A, \end{aligned}$$

and $\gcd(\alpha, \beta) = \gcd\left((r - 1)(r + 1) + 1, r - 1\right) = 1$. It remains to note that $\alpha/\beta \geq r > \max B$. ◀

Note that Lemma 15 aggregates several systems using *the same* α and β. It guarantees that the numbers α and β stay small (have polynomial size) and satisfy additional constraints, to make the subsequent reduction from Bounded 1D Ray Cover to Ultimately Periodic Set Universality possible. We now discuss how these constraints arise.

6.2 Additional constraints and the final reduction

Let us take a step forward in the chain of reductions in Figure 1. We illustrate the significance of the additional requirements on the input of Bounded 1D Ray Cover using the following simple observation on the representation of the complement of a segment of a linear progression.

▶ **Lemma 16.** *Let $a, 2^k, 2^m \in \mathbb{N}$. Denote $b = 2^k$ and $B = a + (2^m - 1) \cdot b$, as in the input of Bounded 1D Ray Cover. Then the set $\mathbb{N} \setminus (L(a, b) \cap [0, B])$ is semilinear with a generator representation of size $O(\lfloor a/b \rfloor \log a + k^2 + m)$.*

Proof. Observe that $\mathbb{N} \setminus (L(a, b) \cap [0, B]) = \{n \in \mathbb{N} : n < a,\ n \equiv a \mod b\} \cup \{n \in \mathbb{N} : n \not\equiv a \mod b\} \cup \{n \in \mathbb{N} : n > B\}$. The third set on the right-hand side is $L(B + 1, 1)$, and the first set is a finite set with $\lfloor a/b \rfloor$ elements, i.e., a union of $\lfloor a/b \rfloor$ linear sets of the form $L(x, 0)$, $x < a$. For the second set, we will rely on the assumption that $b = 2^k$. Suppose that in binary we have $a \bmod b = a_{k-1} \cdot 2^{k-1} + \cdots + a_1 \cdot 2^1 + a_0 \cdot 2^0$, $a_i \in \{0, 1\}$. Notice that an arbitrary $n \in \mathbb{N}$ is not congruent to a modulo 2^k if and only if, for some $j \in \{0, \dots, k-1\}$, it is congruent to $c_j := \overline{a}_j \cdot 2^j + a_{j-1} \cdot 2^{j-1} \cdots a_1 \cdot 2^1 + a_0 \cdot 2^0$ modulo 2^{j+1}, where $\overline{a}_j = 1 - a_j$. Therefore, the second set in the equation above is equal to $\bigcup_{j=0}^{k-1} L(c_j, 2^{j+1})$. ◀

Given Lemma 16, the final reduction from Bounded 1D Ray Cover to Ultimately Periodic Set Universality is simple. Indeed, $L(0, P) \supseteq L(a, b) \cap [0, B]$ if and only if $L(0, P) \cup (\mathbb{N} \setminus (L(a, b) \cap [0, B])) = \mathbb{N}$, where the second set in the union is described by a semilinear set in generator representation of polynomial size. The bound on the size holds, because k and m cannot exceed the bit size of the input instance and $\lfloor a/b \rfloor = \lfloor a/2^k \rfloor = 0$ by the promise that $a < 2^k$ in the definition of Bounded 1D Ray Cover.

Note that the proof of Lemma 16 would work just as well if we had b equal to a power of 3, or a power of any other fixed number (other forms are also possible). For the input of Bounded 1D Ray Cover, any of these constraints is not difficult to satisfy on its own. In comparison, the dependence on $\lfloor a/b \rfloor$ in Lemma 16 is more important and more difficult to handle. We ensure in our chain of reductions from Restricted Simultaneous Subset Sum that $a/b < 1$; in particular, in our aggregation procedure (Lemma 15) we require that the coefficients α and β satisfy $\alpha/\beta > b$ The condition that α is a power of two comes from our preference to use an arithmetic progression with a difference of the form 2^v, $v \in \mathbb{N}$, in Lemma 16: $L(a, b)$, $b = 2^k$.

Detailed Π_2^{P}-hardness proofs for the lower bounds in Theorem 14 can be found in the full version of this paper.

7 Conclusion

Choffrut and Frigeri left open the question whether there is an algorithm with an elementary running time deciding weak $\mathbb{Z}$-definability [5]. We have shown in this article that this is the case. There still remains a significant gap between our Π_2^{P} lower bound and the rather crude upper bound that we obtained. Our algorithm is based on a geometric characterisation of sets definable in weak Presburger arithmetic that complements the generator characterisation obtained by Choffrut and Frigeri [5].

While weak definability is an interesting problem in its own right, another motivation for our work stems from the fact it is an open problem whether deciding weak Presburger arithmetic is computationally easier than deciding full Presburger arithmetic. In fact, in his original article [12], Presburger only showed decidability of weak Presburger arithmetic and remarked that his proof could be adapted to also work for Presburger arithmetic. We are unable to give an answer to this open problem at the present stage, but we believe that the geometric insights that enable us to show the elementary upper bounds of the weak definability problems may eventually help settling the computational complexity of weak Presburger arithmetic. Note that deciding weak linear rational arithmetic has essentially the same complexity as linear rational arithmetic [14].

References

1 B. Boigelot, I. Mainz, V. Marsault, and M. Rigo. An efficient algorithm to decide periodicity of b-recognisable sets using MSDF convention. In *International Colloquium on Automata, Languages, and Programming, ICALP*, volume 80 of *LIPIcs*, pages 118:1–118:14, 2017. doi:10.4230/LIPIcs.ICALP.2017.118.

2 D. Chistikov and C. Haase. The taming of the semi-linear set. In *International Colloquium on Automata, Languages, and Programming, ICALP*, volume 55 of *LIPIcs*, pages 128:1–128:13, 2016. doi:10.4230/LIPIcs.ICALP.2016.128.

3 D. Chistikov and C. Haase. On the complexity of quantified integer programming. In *International Colloquium on Automata, Languages, and Programming, ICALP*, volume 80 of *LIPIcs*, pages 94:1–94:13, 2017. doi:10.4230/LIPIcs.ICALP.2017.94.

4 D. Chistikov, C. Haase, and S. Halfon. Context-free commutative grammars with integer counters and resets. *Theor. Comput. Sci.*, 735:147–161, 2018. doi:10.1016/j.tcs.2016.06.017.

5 C. Choffrut and A. Frigeri. Deciding whether the ordering is necessary in a Presburger formula. *Discret. Math. Theor. C.*, 12(1):21–38, 2010. URL: http://dmtcs.episciences.org/510.

6 S. Ginsburg and E.H. Spanier. Bounded ALGOL-like languages. *T. Am. Math. Soc.*, pages 333–368, 1964. doi:10.2307/1994067.

7 F.W. Glover and R.E.D. Woolsey. Aggregating diophantine equations. *Zeitschr. für OR*, 16(1):1–10, 1972. doi:10.1007/BF01917186.

8 T.-D. Huynh. The complexity of semilinear sets. *Elektron. Inf.verarb. Kybern.*, 18(6):291–338, 1982.

9 J. Leroux. A polynomial time Presburger criterion and synthesis for number decision diagrams. In *Symposium on Logic in Computer Science, LICS*, pages 147–156, 2005. doi:10.1109/LICS.2005.2.

10 A.A. Muchnik. The definable criterion for definability in Presburger arithmetic and its applications. *Theor. Comput. Sci.*, 290(3):1433–1444, 2003. doi:10.1016/S0304-3975(02)00047-6.

11 R. Parikh. On context-free languages. *J. ACM*, 13(4):570–581, 1966. doi:10.1145/321356.321364.

12 M. Presburger. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In *Comptes Rendus du I congres de Mathematiciens des Pays Slaves*, pages 92–101. Warszawa, 1929.

13 J. Robinson. Definability and decision problems in arithmetic. *J. Symb. Log.*, 14(2):98–114, 1949. doi:10.2307/2266510.

14 A. Ronquist. A lower bound on the complexity of real addition without order. Master's thesis, University of Oxford, United Kingdom, 2019.

15 A. Schrijver. *Theory of linear and integer programming.* John Wiley & Sons, 1986.

16 H.U. Simon. On the containment problem for linear sets. In *Symposium on Theoretical Aspects of Computer Science, STACS*, volume 96 of *LIPIcs*, pages 55:1–55:12, 2018. doi:10.4230/LIPIcs.STACS.2018.55.

17 J. von zur Gathen and M. Sieveking. A bound on solutions of linear integer equalities and inequalities. *P. Am. Math. Soc.*, 72(1):155–158, 1978. doi:10.1090/S0002-9939-1978-0500555-0.

The Post Correspondence Problem and Equalisers for Certain Free Group and Monoid Morphisms

Laura Ciobanu
Heriot-Watt University, Edinburgh, Scotland, UK
l.ciobanu@hw.ac.uk

Alan D. Logan
Heriot-Watt University, Edinburgh, Scotland, UK
a.logan@hw.ac.uk

Abstract

A marked free monoid morphism is a morphism for which the image of each generator starts with a different letter, and immersions are the analogous maps in free groups. We show that the (simultaneous) PCP is decidable for immersions of free groups, and provide an algorithm to compute bases for the sets, called equalisers, on which the immersions take the same values. We also answer a question of Stallings about the rank of the equaliser.

Analogous results are proven for marked morphisms of free monoids.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory; Theory of computation → Complexity classes; Mathematics of computing → Combinatorics on words; Mathematics of computing → Combinatorial algorithms

Keywords and phrases Post Correspondence Problem, marked map, immersion, free group, free monoid

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.120

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

Related Version https://arxiv.org/abs/2002.07574

Funding Research supported by EPSRC grant EP/R035814/1.

1 Introduction

In this paper we prove results about the classical Post Correspondence Problem ($\mathrm{PCP}_{\mathrm{FM}}$), which we state in terms of equalisers of free monoid morphisms, and the analogue problem $\mathrm{PCP}_{\mathrm{FG}}$ for free groups ([5], [19]), and we describe the solutions to $\mathrm{PCP}_{\mathrm{FM}}$ and $\mathrm{PCP}_{\mathrm{FG}}$ for certain classes of morphisms. While the classical $\mathrm{PCP}_{\mathrm{FM}}$ is famously undecidable for arbitrary maps of free monoids [21] (see also the survey [12] and the recent result of Neary [20]), $\mathrm{PCP}_{\mathrm{FG}}$ for free groups is an important open question [8, Problem 5.1.4]. Additionally, for both free monoids and free groups there are only few results describing algebraically the solutions to classes of instances known to have decidable $\mathrm{PCP}_{\mathrm{FM}}$ or $\mathrm{PCP}_{\mathrm{FG}}$. Our results apply to *marked* morphisms in the monoid case, and to their counterparts in free groups, called *immersions*. Marked morphisms are the key tool used in resolving the $\mathrm{PCP}_{\mathrm{FM}}$ for the free monoid of rank two [9], and therefore understanding the solutions to the $\mathrm{PCP}_{\mathrm{FG}}$ for immersions is an important step towards resolving the $\mathrm{PCP}_{\mathrm{FG}}$ for the free group of rank two. The density of marked morphisms and immersions among all the free monoid or group maps is strictly positive (Section 10), so our results concern a significant proportion of instances.

An *instance* of the $\mathrm{PCP}_{\mathrm{FM}}$ is a tuple $I = (\Sigma, \Delta, g, h)$, where Σ, Δ are finite alphabets, Σ^*, Δ^* are the respective free monoids, and $g, h : \Sigma^* \to \Delta^*$ are morphisms. The *equaliser* of g, h is $\mathrm{Eq}(g, h) = \{x \in \Sigma^* \mid g(x) = h(x)\}$. The $\mathrm{PCP}_{\mathrm{FM}}$ is the decision problem:

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 120; pp. 120:1–120:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

$\mathrm{PCP}_{\mathbf{FM}}$: Given $I = (\Sigma, \Delta, g, h)$, is the equaliser $\mathrm{Eq}(g, h)$ trivial?

Analogously, an instance of the $\mathrm{PCP}_{\mathrm{FG}}$ is a four-tuple $I = (\Sigma, \Delta, g, h)$ with $g, h : F(\Sigma) \to F(\Delta)$ morphisms between the free groups $F(\Sigma)$ and $F(\Delta)$, and $\mathrm{PCP}_{\mathrm{FG}}$ is the decision problem pertaining to the similarly defined $\mathrm{Eq}(g, h)$ in free groups.

Beyond $\mathrm{PCP}_{\mathrm{FM}}$, in this paper we also consider the *Algorithmic Equaliser Problem*, denoted $\mathrm{AEP}_{\mathrm{FM}}$ (or $\mathrm{AEP}_{\mathrm{FG}}$ in the group case), which for an instance $I = (\Sigma, \Delta, g, h)$ with g, h free monoid morphisms (or free group morphisms for $\mathrm{AEP}_{\mathrm{FG}}$), says:

$\mathrm{AEP}_{\mathbf{FM}}$: Given $I = (\Sigma, \Delta, g, h)$, output

(a) a finite basis for $\mathrm{Eq}(g, h)$, or

(b) a finite automaton recognising the set $\mathrm{Eq}(g, h)$.

If a finite basis or finite automaton for $\mathrm{Eq}(g, h)$ does not exist then Part (a) or (b), respectively, of the problem is insoluble. Note that (a) and (b) are connected: for free groups these two problems are in fact the same when $\mathrm{Eq}(g, h)$ is finitely generated, while for free monoids (a) implies (b). Part (a) of the $\mathrm{AEP}_{\mathrm{FM}}$ is known to be soluble when $|\Sigma| = 2$ and one of g or h is non-periodic, and insoluble otherwise [13] [12, Corollary 6].

Sets of morphisms. We are particularly interested in sets S of morphisms (not just two morphisms f, g) and their equalisers $\mathrm{Eq}(S) = \bigcap_{g,h\in S} \mathrm{Eq}(g, h)$, and we prove structural results for arbitrary sets and algorithmic results for finite sets. Our results resolve the *simultaneous* $\mathrm{PCP}_{\mathrm{FG}}$ *and* $\mathrm{PCP}_{\mathrm{FM}}$ for immersions and marked morphisms; these problems take as input a finite set S of maps and ask the same questions about equalisers as in the classical setting. Analogously, one could further define the "simultaneous $\mathrm{AEP}_{\mathrm{FG}}$ and $\mathrm{AEP}_{\mathrm{FM}}$". However, the simultaneous $\mathrm{AEP}_{\mathrm{FG}}$ is equivalent to the $\mathrm{AEP}_{\mathrm{FG}}$, and Part (b) of the simultaneous $\mathrm{AEP}_{\mathrm{FM}}$ is equivalent to Part (b) of the $\mathrm{AEP}_{\mathrm{FM}}$, as follows. As bases of intersections of finitely generated subgroups of free groups are computable (and as Parts (a) and (b) of the $\mathrm{AEP}_{\mathrm{FG}}$ are equivalent), if the $\mathrm{AEP}_{\mathrm{FG}}$ is soluble for a class $\mathcal{C}$ of maps then there exists an algorithm with input a finite set S of morphisms from $F(\Sigma)$ to $F(\Delta)$, $S \subseteq \mathcal{C}$, and output a basis for $\mathrm{Eq}(S)$. Similarly, automata accepting intersections of regular languages are computable, and so if Part (b) of the $\mathrm{AEP}_{\mathrm{FM}}$ is soluble for a class $\mathcal{C}$ of maps then there exists an algorithm with input a finite set S of morphisms from Σ^* to Δ^*, $S \subseteq \mathcal{C}$, and output a finite automaton whose language is $\mathrm{Eq}(S) = \bigcap_{g,h\in S} \mathrm{Eq}(g, h)$.

Main results. A set of words $\mathbf{s} \subseteq \Delta^*$ is *marked* if any two distinct $u, v \in \mathbf{s}$ start with a different letter of Δ, which implies $|\mathbf{s}| \leqslant |\Delta|$. A free monoid morphism $f : \Sigma^* \to \Delta^*$ is *marked* if the set $f(\Sigma)$ is marked. An *immersion of free groups* is a morphism $f : F(\Sigma) \to F(\Delta)$ where the set $f(\Sigma \cup \Sigma^{-1})$ is marked (see Section 3 for equivalent formulations). Halava, Hirvensalo and de Wolf [11] showed that $\mathrm{PCP}_{\mathrm{FM}}$ is decidable for marked morphisms; inspired by their methods we were able to obtain stronger results (Theorem A) for this kind of map, as well as expand to the world of free groups (Theorem C), where we employ "finite state automata"-like objects called *Stallings graphs.*

▶ **Theorem A.** *If S is a set of marked morphisms from Σ^* to Δ^*, then there exists a finite alphabet Σ_S and a marked morphism $\psi_S : \Sigma_S^* \to \Sigma^*$ such that* $\mathrm{Image}(\psi_S) = \mathrm{Eq}(S)$*. Moreover, for S finite, there exists an algorithm with input S and output the marked morphism ψ_S.*

▶ **Corollary B.** *The simultaneous* $\mathrm{PCP}_{\mathrm{FM}}$ *is decidable for marked morphisms of free monoids.*

▶ **Theorem C.** *If S is a set of immersions from $F(\Sigma)$ to $F(\Delta)$, then there exists a finite alphabet Σ_S and an immersion $\psi_S : F(\Sigma_S) \to F(\Sigma)$ such that* $\mathrm{Image}(\psi_S) = \mathrm{Eq}(S)$*. Moreover, when S is finite, there exists an algorithm with input S and output the immersion ψ_S.*

▶ **Corollary D.** *The simultaneous* $\mathrm{PCP}_{\mathrm{FG}}$ *is decidable for immersions of free groups.*

The Equaliser Conjecture. Our work was partially motivated by Stallings' Equaliser Conjecture for free groups, which dates from 1984 [22, Problems P1 & 5] (also [7, Problem 6] [24, Conjecture 8.3] [1, Problem F31]). Here $\mathrm{rk}(H)$ stands for the *rank*, or minimum number of generators, of a subgroup H:

▶ **Conjecture 1** (The Equalizer Conjecture, 1984)**.** *If $g, h : F(\Sigma) \to F(\Delta)$ are injective morphisms then* $\mathrm{rk}(\mathrm{Eq}(g, h)) \leqslant |\Sigma|$.

This conjecture has its roots in "fixed subgroups" $\mathrm{Fix}(\phi)$ of free group endomorphisms $\phi : F(\Sigma) \to F(\Sigma)$ (if $\Sigma = \Delta$ then $\mathrm{Fix}(\phi) = \mathrm{Eq}(\phi, \mathrm{id})$), where Bestvina and Handel proved that $\mathrm{rk}(\mathrm{Fix}(\phi)) \leqslant |\Sigma|$ for ϕ an automorphism [3], and Imrich and Turner extended this bound to all endomorphisms [14]. Bergman further extended this bound to all sets of endomorphisms [2]. Like Bergman's result, our first corollary of Theorem C considers sets of immersions, which are injective, and answers Conjecture 1 for immersions.

▶ **Corollary E.** *If S is a set of immersions from $F(\Sigma)$ to $F(\Delta)$ then* $\mathrm{rk}(\mathrm{Eq}(S)) \leqslant |\Sigma|$.

In free monoids, equalisers of injections are free [12, Corollary 4] but they are not necessarily regular languages (and hence not necessarily finitely generated) [12, Example 6]. In order to understand equalisers $\mathrm{Eq}(S)$ of sets of maps we need to understand intersections in free monoids. Recall that the intersection $A^* \cap B^*$ of two finitely generated free submonoids of a free monoid Σ^* is free [23] and one can find a regular expression that represents a basis of $A^* \cap B^*$ [4]. However, the intersection is not necessarily finitely generated [17]. The following result is surprising because we have finite generation, even for the intersection $\mathrm{Eq}(S) = \bigcap_{g,h \in S} \mathrm{Eq}(g, h)$.

▶ **Corollary F.** *If S is a set of marked morphisms from Σ^* to Δ^* then* $\mathrm{Eq}(S)$ *is a free monoid with* $\mathrm{rk}(\mathrm{Eq}(S)) \leqslant |\Sigma|$.

The Algorithmic Equaliser Problem. The $\mathrm{AEP}_{\mathrm{FG}}$ is insoluble in general, as equalisers in free groups are not necessarily finitely generated [24, Section 3], and is an open problem of Stallings' if both maps are injective [22, Problems P3 & 5]. Our next corollary of Theorem C resolves this open problem for immersions.

▶ **Corollary G.** *The* $\mathrm{AEP}_{\mathrm{FG}}$ *is soluble for immersions of free groups.*

The $\mathrm{AEP}_{\mathrm{FM}}$ is insoluble in general, primarily as equalisers are not necessarily regular languages [10, Example 4.6]. Even for maps whose equalisers form regular languages, the problem remains insoluble [18]. Another corollary of Theorem A is the following.

▶ **Corollary H.** *The* $\mathrm{AEP}_{\mathrm{FM}}$ *is soluble for marked morphisms of free monoids.*

Outline of the article. In Section 2 we prove Theorem A and its corollaries about free monoids. The remainder of the paper focuses on free groups, where the central result is Theorem 18, which is Theorem C for $|S| = 2$. In Section 3 we reformulate immersions in terms of Stallings' graphs, and in Section 4 define the "reduction" $I' = (\Sigma', \Delta', g', h')$ of an instance $I = (\Sigma, \Delta, g, h)$ of the $\mathrm{AEP}_{\mathrm{FG}}$ for immersions. Repeatedly computing reductions is the key process in our algorithm. In Section 5 we prove the process of reduction decreases the "prefix complexity" of an instance (so the word "reduction" makes sense), and in Section 6 we prove Theorem 18, mentioned above. In Section 7 we prove Theorem C and its corollaries. In Section 9 we give a complexity analysis for both our free monoid and free group algorithms, and in Section 10 we show that the density of marked morphisms and immersions among all the free monoid or group maps is strictly positive.

2 Marked morphisms in free monoids

In this section we prove Theorem A and its corollaries. We use the following immediate fact.

▶ **Lemma 2.** *Marked morphisms of free monoids are injective.*

Proof. Let $f : \Sigma^* \to \Delta^*$ be marked and let $x \neq y$ be nontrivial. One can write $x = zax'$ and $y = zby'$, where $a, b \in \Sigma$ are the first letter where x and y differ. As f is marked, $f(a) \neq f(b)$, hence $f(x) = f(z)f(a)f(x') \neq f(z)f(b)f(y') = f(y)$, so f is injective. ◀

We may assume $\Sigma \subseteq \Delta$, as $|\Sigma| \leqslant |\Delta|$ holds whenever $f : \Sigma^* \to \Delta^*$ is marked.

Consider morphisms $g : \Sigma_1^* \to \Delta^*$ and $h : \Sigma_2^* \to \Delta^*$. The set of non-empty words over an alphabet Σ is denoted Σ^+. For $a \in \Delta$, a pair $(u, v) \in \Sigma_1^+ \times \Sigma_2^+$ is an *a-block* if (i) $g(u) = h(v)$ starts with a, and (ii) u and v are minimal, that is, the length $|g(u)| = |h(v)|$ is minimal among all such pairs. If the pair (g, h) has blocks $a_i = (u_i, v_i)$, $1 \leqslant i \leqslant m$, then let Σ' be the alphabet consisting of these blocks and define $g' : (\Sigma')^* \mapsto \Sigma_1^*$ by $g'(a_i) = u_i$ and $h' : (\Sigma')^* \mapsto \Sigma_2^*$ by $h'(a_i) = v_i$. These maps are computable and, by an identical logic to [11, Section 2], are seen to be marked. Then $gg' = hh'$, and we let $k = gg' = hh'$ (so $k : (\Sigma')^* \to \Delta^*$). Since k is the composition of marked morphisms, it is itself marked. We therefore have the following.

▶ **Lemma 3.** *If $g : \Sigma_1^* \to \Delta^*$ and $h : \Sigma_2^* \to \Delta^*$ are marked morphisms then the corresponding maps $g' : \Sigma'^* \to \Sigma_1^*$, $h' : \Sigma'^* \to \Sigma_2^*$ and $k : \Sigma'^* \to \Delta^*$, $k = gg' = hh'$, are marked and are computable.*

The *reduction* of an instance $I = (\Sigma, \Delta, g, h)$ of the marked $\mathrm{PCP}_{\mathrm{FM}}$, as defined in [11], is the instance $I' := (\Sigma', \Delta, g', h')$ where Σ' is defined as above, and where g' and h' are as above, but with codomain Δ (which we may do as $\Sigma \subseteq \Delta$). We additionally assume that $\Sigma' \subseteq \Sigma$; we can do this as $|\Sigma'| \leqslant |\Sigma|$ by Lemma 3.

The following relies on [11, Lemma 1], which we strengthen by replacing the notion of "equivalence" with that of "strong equivalence": Two instances I_1 and I_2 of the $\mathrm{PCP}_{\mathrm{FM}}$ are *strongly equivalent* if their equalisers are isomorphic, which we write as $\mathrm{Eq}(I_1) \cong \mathrm{Eq}(I_2)$.

▶ **Lemma 4.** *Let $I' = (\Sigma', \Delta', g', h')$ be the reduction of $I = (\Sigma, \Delta, g, h)$ where g and h are marked. Then I and I' are strongly equivalent, and $g'(\mathrm{Eq}(I')) = \mathrm{Eq}(I) = h'(\mathrm{Eq}(I'))$.*

Proof. Firstly, note that $g'(\mathrm{Eq}(I')) \leqslant \mathrm{Eq}(I)$ [11, Lemma 1, paragraph 2]. From [11, Lemma 1, paragraph 1] it follows that $g'(\mathrm{Eq}(I')) \geqslant \mathrm{Eq}(I)$, so $g'(\mathrm{Eq}(I')) = \mathrm{Eq}(I)$. As g' is injective, the map $g'|_{\mathrm{Eq}(I')}$ is an isomorphism. Hence, I and I' are strongly equivalent, and, by symmetry for the h' map, $g'(\mathrm{Eq}(I')) = \mathrm{Eq}(I) = h'(\mathrm{Eq}(I'))$ as required. ◀

We can now improve the existing result on the marked $\mathrm{PCP}_{\mathrm{FM}}$. We store a morphism $f : \Sigma^* \to \Delta^*$ as a list $(f(a))_{a \in \Sigma}$.

▶ **Theorem 5.** *If $I = (\Sigma, \Delta, g, h)$ is an instance of the marked* $\mathrm{PCP}_{\mathrm{FM}}$ *then there exists an alphabet $\Sigma_{g,h}$ and a marked morphism $\psi_{g,h} : \Sigma^*_{g,h} \to \Sigma^*$ such that* $\mathrm{Image}(\psi_{g,h}) = \mathrm{Eq}(I)$. *Moreover, there exists an algorithm with input I and output the marked morphism $\psi_{g,h}$.*

Proof. We explain the algorithm, and note at the end that the output is a marked morphism $\psi_{g,h} : F(\Sigma_{g,h}) \to F(\Sigma)$ with the required properties, and so the result follows.

Begin by making reductions $I_0, I_1, I_2, \ldots$, starting with $I_0 = I = (\Sigma, \Delta, g, h)$, the input instance. Then by [11, Section 5, paragraph 1] we will obtain an instance $I_j = (\Sigma_j, \Delta, g_j, h_j)$ such that one of the following will occur:

1. $|\Sigma_j| = 1$.
2. $|g_j(a)| = 1 = |h_j(a)|$ for all $a \in \Sigma_j$.
3. There exists some $i < j$ with $I_i = I_j$ (sequence starts cycling).

Keeping in mind the fact that reductions preserve equalisers (Lemma 4), we obtain in each case a subset $\Sigma_{g,h}$ (possibly empty) which forms a basis for $\mathrm{Eq}(I_j)$: For Case (1), writing $\Sigma_j = \{a\}$, the result holds as if $g(a^i) = h(a^i)$ then $g(a)^i = h(a)^i$ and so $g(a) = h(a)$ as roots are unique in a free monoid. For Case (2), suppose $g_j(x) = h_j(x)$. Then g_j and h_j agree on the first letter of $x \in \Sigma^*_j$ because the image of each letter has length one, and inductively we see that they agree on every letter of x. Hence, a subset $\Sigma_{g,h}$ of Σ_j forms a basis for $\mathrm{Eq}(I_j)$.

For Case (3), suppose there is a sequence of reductions beginning and ending at I_j:

$$I_j \to I_{j+1} \to \cdots \to I_{j+(i-1)} \to I_{j+i} = I_j$$

and write $r := j + i$. By Lemma 4, $\mathrm{Eq}(I_j) = g_{j+1}g_{j+2}\ldots g_r(\mathrm{Eq}(I_r)) = \mathrm{Eq}(I_r)$; thus $\overline{g_r} := g_{j+1}g_{j+2}\ldots g_r$ restricts to an automorphism of $\mathrm{Eq}(I_j)$, so $\overline{g_r}|_{\mathrm{Eq}(I_j)} \in \mathrm{Aut}(\mathrm{Eq}(I_j))$. The automorphism $\overline{g_r}$ is necessarily length-preserving ($|\overline{g_r}(w)| = |w|$ for all $w \in \mathrm{Eq}(I_j)$). Consider $x \in \mathrm{Eq}(I_j) = \mathrm{Eq}(I_r)$. Then $\overline{g_r}$ maps the letters occurring in x_r to letters and so $g_j(= g_r)$ and $h_j(= h_r)$ map the letters occuring in x to letters, and it follows that every letter occuring in x is a solution to $I_r = I_j$. Hence, a subset $\Sigma_{g,h}$ of Σ_j forms a basis for $\mathrm{Eq}(I_j)$ as required.

Therefore, in all three cases a subset $\Sigma_{g,h}$ of Σ_j forms a basis for $\mathrm{Eq}(I_j)$, and since Σ_j is computable, this basis is as well. In order to prove the theorem, it is sufficient to prove that there is a computable immersion $\psi_{g,h} : \Sigma^*_{g,h} \to \Sigma^*$. Consider the map $\tilde{g} = g_1g_2\cdots g_j : \Sigma^*_j \to \Sigma^*$ (and the analogous $\tilde{h}$). Now, each g_i is marked, by Lemma 3, and so $\tilde{g}$ is the composition of marked morphisms and hence is marked itself. Define $\psi_{g,h} := \tilde{g}|_{\Sigma^*_{g,h}}$. This map is computable from $\tilde{g}$, and as $\Sigma_{g,h} \subseteq \Sigma_j$, the map $\psi_{g,h}$ is marked. As $\mathrm{Image}(\psi_{g,h}) = g_1g_2\ldots g_j(\mathrm{Eq}(I_j)) = \mathrm{Eq}(I)$, by Lemma 4 and the above, the result follows. ◀

Theorem 5 together with Lemma 6 give the non-algorithmic part of Theorem A. A subsemigroup M of a free monoid Σ^* is *marked* if it is the image of a marked morphism.

▶ **Lemma 6.** *If $\{M_j\}_{j\in J}$ is a set of marked subsemigroups of Σ^* then the intersection $\bigcap_{j\in J} M_j$ is marked.*

Proof. Firstly, suppose $x, y \in M_j$ for some $j \in J$. Then there exist two words $x_0 \ldots x_l$ and $y_0 \ldots y_k$, with $x_i, y_i \in \Sigma$, such that $\phi(x_0 \ldots x_l) = x$ and $\phi(y_0 \ldots y_k) = y$, where ϕ is a marked morphism. If x and y have a nontrivial common prefix, then because ϕ is marked we get $x_0 = y_0$, and $\phi(x_0)$ is a prefix of both x and y, and in particular $\phi(x_0) \in M_j$. By continuing this argument, if z is a maximal common prefix of x and y, then $z \in M_j$.

Now, suppose $x, y \in \bigcap_{j\in J} M_j$, and suppose they both begin with some letter $a \in \Sigma \cup \Sigma^{-1}$. By the above, their maximal common prefix z_a is contained in each M_j and so is contained in $\bigcap_{j\in J} M_j$. Therefore, z_a is a prefix of every element of $\bigcap_{j\in J} M_j$ beginning with an a. It follows that $\bigcap_{j\in J} M_j$ is immersed, as required. ◀

We now prove the algorithmic part of Theorem A (this is independent of Lemma 6).

▶ **Lemma 7.** *There exists an algorithm with input a finite set of marked morphisms S from Σ^* to Δ^* and output a marked morphism $\psi_S : \Sigma_S^* \to \Sigma^*$ such that* $\mathrm{Image}(\psi_S) = \mathrm{Eq}(S)$*.*

Proof. We use induction on $|S|$. By Theorem 5, the result holds if $|S| = 2$. Suppose the result holds for all sets of n marked morphisms, $n \geqslant 2$, and let S be a set of $n+1$ marked morphisms. Take elements $g, h \in S$, and write $S_g = S \setminus \{g\}$. By hypothesis, we can algorithmically obtain marked morphisms $\psi_{S_g} : \Sigma_{S_g}^* \to \Sigma^*$ and $\psi_{g,h} : \Sigma_{g,h}^* \to \Sigma^*$ such that $\mathrm{Image}(\psi_{S_g}) = \mathrm{Eq}(S_g)$ and $\mathrm{Image}(\psi_{g,h}) = \mathrm{Eq}(g, h)$.

By Lemma 3, there exists a (computable) marked morphism $\psi_S : \Sigma_S^* \to \Sigma^*$ such that $\mathrm{Image}(\psi_S) = \mathrm{Image}(\psi_{S_g}) \cap \mathrm{Image}(\psi_{g,h})$ (the map ψ_S corresponds to the map k in Lemma 3, and Σ_S to Σ'). Then, as required: $\mathrm{Image}(\psi_S) = \mathrm{Image}(\psi_{S_g}) \cap \mathrm{Image}(\psi_{g,h}) = \mathrm{Eq}(S_g) \cap \mathrm{Eq}(g, h) = \mathrm{Eq}(S)$. ◀

We now prove Theorem A, which states that the equaliser is the image of a marked map.

Proof of Theorem A. By applying Lemma 6 to Theorem 5, there exists an alphabet Σ_S and a marked morphism $\psi_S : \Sigma_S^* \to \Sigma^*$ such that $\mathrm{Image}(\psi_S) = \mathrm{Eq}(S)$, while by Lemma 7 if S is finite then such a marked morphism can be algorithmically found. ◀

We now prove Corollary F, which says that $\mathrm{Eq}(S)$ is free of rank $\leqslant |\Sigma|$.

Proof of Corollary F. Consider the marked morphism $\psi_S : \Sigma_S^* \to \Sigma^*$ given by Theorem A. By Lemma 2, ψ_S is injective so $\mathrm{Image}(\psi_S)$ is free. As ψ_S is marked the map $\Sigma_S \to \Sigma$ taking each $a \in \Sigma_S$ to the initial letter of $\psi_S(a)$ is an injection, so $|\Sigma_S| \leqslant |\Sigma|$ as required. ◀

We now prove a strong form of the $\mathrm{AEP}_{\mathrm{FM}}$ for marked morphisms.

▶ **Corollary 8.** *There exists an algorithm with input a finite set S of marked morphisms from Σ^* to Δ^* and output a basis for* $\mathrm{Eq}(S)$*.*

Proof. To algorithmically obtain a basis for $\mathrm{Eq}\,(S)$, first use the algorithm of Theorem A to obtain the marked morphism $\overline{\psi}_S : \Sigma_S^* \to \Sigma^*$ such that $\mathrm{Image}(\psi_S) = \mathrm{Eq}(S)$. Then, recalling that we store ψ_S as a list $(\psi_S(a))_{a \in \Sigma}$, the required basis is the set of elements in this list, so the set $\{\psi_S(a)\}_{a \in \Sigma}$. ◀

Corollary H, the $\mathrm{AEP}_{\mathrm{FM}}$ for marked morphisms, follows from Corollary 8 by taking $|S| = 2$, while Corollary B, the simultaneous $\mathrm{PCP}_{\mathrm{FM}}$, also follows as $\mathrm{Eq}\,(S)$ is trivial if and only if its basis is empty.

3 Immersions of free groups

We denote the free group with finite generating set Σ by $F(\Sigma)$, and view it as the set of all *freely reduced words* over $\Sigma^{\pm 1} = \Sigma \cup \Sigma^{-1}$, that is, words not containing xx^{-1} as subwords, $x \in \Sigma^{\pm 1}$, together with the operations of concatenation and free reduction (that is, the removal of any xx^{-1} that might occur when concatenating two words).

We now begin our study of immersions of free groups, as defined in the introduction. We first state the characterising lemma, then explain the terms involved before giving the proof.

▶ **Lemma 9.** *Let $g : F(\Sigma) \to F(\Delta)$ be a free group morphism. The following are equivalent.*

1. *The map g is an immersion of free groups.*
2. *Every word in the language $L(\Gamma_g, v_g)$ is freely reduced.*
3. *For all $x, y \in \Sigma \cup \Sigma^{-1}$ such that $xy \neq 1$, the length identity $|g(xy)| = |g(x)| + |g(y)|$ holds.*

Characterisation (3) is the established definition of Kapovich [15]. Characterisation (2) is the one we shall work with in this article. It uses "Stallings graphs", which are essentially finite state automata that recognise the elements of finitely generated subgroups of free groups. We define these now, and refer the reader to [16] for background on Stallings graphs.

The *(unfolded) Stallings graph* Γ_g of the free group morphism g is the directed graph formed by taking a bouquet with $|\Sigma|$ petals attached at a central vertex we call v_g, where each petal consists of a path labeled by $g(x) \in (\Delta \cup \Delta^{-1})^*$; the elements of Δ^{-1} occur as edges traversed backwards and we denote by e^{-1} the edge e in opposite direction, and by $E\Gamma_g^{\pm 1}$ the sets of edges in both directions. A path $q = (e_1, \dots, e_n)$, $e_i \in E\Gamma_g^{\pm 1}$ edges, is *reduced* if it has no backtracking, that is, $e_i^{-1} \neq e_{i+1}$ for all $1 \leqslant i < n$. We denote by $\iota(p)$ the initial vertex of a path p and $\tau(p)$ for the terminal vertex, and call a reduced path p with $\iota(p) = u = \tau(p)$ a *closed reduced circuit.*

We shall view Γ_g as a finite state automaton (Γ_g, v_g) with start and accept states both equal to v_g. Then the *extended language accepted by* (Γ_g, v_g) is the set of words labelling reduced closed circuits at v_g in Γ_g:

$$L(\Gamma_g, v_g) = \{label(p) \mid p \text{ is a reduced path with } \iota(p) = u = \tau(p)\}.$$

Immersions are precisely those maps g such that every element of $L(\Gamma_g, v_g)$ is freely reduced; this corresponds to the automaton (Γ_g, v_g) and the "reversed" automaton $(\Gamma_g, v_g)^{-1}$, where edge directions are reversed, both being deterministic (map g in Figure 1 is not an immersion; although the automaton (Γ_g, v_g) is deterministic, $(\Gamma_g, v_g)^{-1}$ is not). For such maps, $L(\Gamma_g, v_g)$ is precisely the image of the map g [16, Proposition 3.8].

Proof of Lemma 9. (1) $\Leftrightarrow$ (2). Every element of $L(\Gamma_g, v_g)$ is freely reduced if and only if the graph Γ_g, with base vertex v_g, is such that for all $e_1, e_2 \in (E\Gamma_g)^{\pm 1}$ such that both edes start at v_g or both edges end at v_g, then e_1 and e_2 have different labels (so $\gamma_g(e_1) \neq \gamma_g(e_2)$). This condition on labels is equivalent to $g(\Sigma \cup \Sigma^{-1})$ being marked, as required.

(1) $\Leftrightarrow$ (3). Condition (3) is equivalent to the condition that for all $x, y \in \Sigma \cup \Sigma^{-1}$ such that $xy \neq 1$, free cancellation does not happen between $g(x)$ and $g(y)$, which in turn is equivalent to the condition that for all such x, y the elements $g(x^{-1})$ and $g(y)$ start with different letters of $\Delta \cup \Delta^{-1}$. This is equivalent to $g(\Sigma \cup \Sigma^{-1})$ being marked, as required. ◀

▶ **Example 10.** Let $g : F(a, b) \to F(x, y)$ be the map defined by $g(a) = x^{-2}y$ and $g(b) = y^2x$. Then the graph Γ_g, where the double arrow represents x and the single arrow y, is depicted in Figure 1. The map g is not an immersion since there are two edges labeled x entering v_g (violating Characterisation (2)). Similarly, $g(a)$ and $g(b^{-1})$ both start with x^{-1} (violating Characterisation (1)) and $|g(ba)| = 4 < 6 = |g(a)| + |g(b)|$ (violating Characterisation (3)).

Using Characterisation (2), we see that immersions are injective [16, Proposition 3.8]:

▶ **Lemma 11.** *If $g : F(\Sigma) \to F(\Delta)$ is an immersion then it is injective.*

4 The reduction of an instance in free groups

By an *immersed* instance of the PCP$_{\mathrm{FG}}$ we mean an instance $I = (\Sigma, \Delta, g, h)$ where both g and h are immersions. In this section we define the "reduction" of an immersed instance of the PCP$_{\mathrm{FG}}$, which is similar to the reduction in the free monoid case.

Let Γ be a directed, labeled graph and $u \in V\Gamma$ a vertex of Γ. The *core graph of* Γ *at* u, written $\mathrm{Core}_u(\Gamma)$, is the maximal subgraph of Γ containing u but no vertices of degree 1, except possibly u itself. Note that $L(\mathrm{Core}_u(\Gamma), u) = L(\Gamma, u)$. For Γ_1, Γ_2 directed, labeled

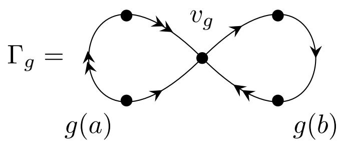


Figure 1 The graph Γ_g for the map $g : F(a,b) \to F(x,y)$ defined by $g(a) = x^{-2}y$, $g(b) = y^2x^{-1}$.

graphs, the *product graph of* Γ_1 *and* Γ_2, denoted $\Gamma_1 \otimes \Gamma_2$, is the subgraph of $\Gamma_1 \times \Gamma_2$ with vertex set $V\Gamma_1 \times V\Gamma_2$ and edge set $\{(e_1, e_2) \mid e_i \in E\Gamma_i^{\pm 1}, label(e_1) = label(e_2)\}$. One may think of the standard construction of an automaton recognising the intersection of two regular languages, each given by a finite state automaton Γ_i with start state s_i, where the core of $\Gamma_1 \otimes \Gamma_2$ at (s_1, s_2) is the automaton recognising this intersection.

Core graph of a pair of morphisms. Let $g : F(\Sigma_1) \to F(\Delta)$, $h : F(\Sigma_2) \to F(\Delta)$ be morphisms. The *core graph of the pair* (g,h), denoted $\mathrm{Core}(g,h)$, is the core graph of $\Gamma_g \otimes \Gamma_h$ at the vertex $v_{g,h} = (v_g, v_h)$, so $\mathrm{Core}(g,h) = \mathrm{Core}_{v_{g,h}}(\Gamma_g \otimes \Gamma_h)$. We shall refer to $v_{g,h}$ as the *central vertex of* $\mathrm{Core}(g,h)$. Note that $\mathrm{Core}(g,h)$ represents the intersection of the two images [16, Lemma 9.3], in the sense that

$$L(\mathrm{Core}(g,h), v_{g,h}) = \mathrm{Image}(g) \cap \mathrm{Image}(h).$$

Write $\delta_g : \mathrm{Core}(g,h) \to \Gamma_g$ and $\delta_h : \mathrm{Core}(g,h) \to \Gamma_h$ for the restriction of $\mathrm{Core}(g,h)$ to the g and h components, respectively, so $\delta_g(e_1, e_2) = e_1$, etc.

Now, let g, h be immersions. The graph $\mathrm{Core}(g,h)$ is a bouquet and every element of $L(\mathrm{Core}(g,h), v_{g,h})$ is freely reduced [16, Lemma 9.2]. We therefore have free group morphisms $g' : L(\mathrm{Core}(g,h), v_{g,h}) \to L(\Gamma_g, v_g)$ and $h' : L(\mathrm{Core}(g,h), v_{g,h}) \to L(\Gamma_h, v_h)$ induced by the maps δ_g, δ_h, where $L(\Gamma_g, v_g) = F(\Sigma_1)$ and $L(\Gamma_h, v_h) = F(\Sigma_2)$. These maps are computable [16, Corollary 9.5]. Let Σ' be the alphabet whose elements consist of the petals of $\mathrm{Core}(g,h)$. Then Σ' generates the free group $L(\mathrm{Core}(g,h), v_{g,h})$, so $F(\Sigma') = L(\mathrm{Core}(g,h), v_{g,h})$, and we see that both g' and h' are immersions with $g' : F(\Sigma') \to F(\Sigma_1)$, $h' : F(\Sigma') \to F(\Sigma_2)$. The map $gg' = hh'$, which we shall call k (so $k : F(\Sigma') \to F(\Delta)$) is the composition of immersions and hence is itself an immersion. We therefore have the following.

▶ **Lemma 12.** *If $g : F(\Sigma_1) \to F(\Delta)$ and $h : F(\Sigma_2) \to F(\Delta)$ are immersions then the corresponding maps $g' : F(\Sigma') \to F(\Sigma_1)$, $h' : F(\Sigma') \to F(\Sigma_2)$ and $k : F(\Sigma') \to F(\Delta)$, where $k = gg' = hh'$, are immersions and are computable.*

Reduction. The *reduction* of an immersed instance $I = (\Sigma, \Delta, g, h)$ of the $\mathrm{PCP}_{\mathrm{FG}}$ is the instance $I' = (\Sigma', \Delta, g', h')$ where g' and h' are as above, but with codomain Δ (which we may do as $\Sigma \subseteq \Delta$). We additionally assume that $\Sigma' \subseteq \Sigma$; we can do this as $|\Sigma'| \leqslant |\Sigma|$ by Lemma 12. As I is immersed, I' is also immersed by Lemma 12. In the next section we show that the name “reduction” makes sense, as it reduces the “prefix complexity” of instances.

▶ **Example 13.** Consider the maps $g, h : F(a,b,c) \to F(x,y,z)$ given by $g(x) = aba^2, g(b) = y^{-1}, g(c) = zxz$ and $h(a) = x, h(b) = yx^2y, h(c) = z$.

Then the graph $\operatorname{Core}(g,h)$ is a bouquet with two petals labelled xyx^2y and zxz, and $\operatorname{Image}(g)\cap\operatorname{Image}(h)=\langle xyx^2y,zxz\rangle$. Moreover, $g(ab^{-1})=h(ab)=xyx^2y$ and $g(z)=h(zxz)=zxz$. Then we can take $\Sigma'=\{a',b'\}$, and the maps given by $g'(a')=ab^{-1},g(b')=c$ and $h'(a')=ab,h'(b')=cac$ are the reduction of (g,h).

We now prove that reduction preserves equalisers. Two instances I_1 and I_2 of the $\mathrm{PCP}_{\mathrm{FG}}$ are *strongly equivalent* if the equalisers are isomorphic, which we write as $\operatorname{Eq}(I_1)\cong\operatorname{Eq}(I_2)$.

▶ **Lemma 14.** *Let $I'=(\Sigma',\Delta',g',h')$ be the reduction of $I=(\Sigma,\Delta,g,h)$ where g and h are immersions. Then I and I' are strongly equivalent, and $g'(\operatorname{Eq}(I'))=\operatorname{Eq}(I)=h'(\operatorname{Eq}(I'))$.*

Proof. It is sufficient to prove that $g'|_{\operatorname{Eq}(I')}$ is injective and $g'(\operatorname{Eq}(I'))=\operatorname{Eq}(I)$; that $h'(\operatorname{Eq}(I'))=\operatorname{Eq}(I)$ follows as $g'|_{\operatorname{Eq}(I')}=h'|_{\operatorname{Eq}(I')}$.

As g' is an immersion it is injective, by Lemma 11. Therefore, $g'|_{\operatorname{Eq}(I')}$ is injective. To see that $\operatorname{Image}(g'|_{\operatorname{Eq}(I')})\leqslant\operatorname{Eq}(I)$, suppose $x'\in\operatorname{Eq}(I')$. Writing $x=g'(x')=h'(x')$, we have $g(x)=gg'(x')=hh'(x')=h(x)$ and so $x=g'(x')\in\operatorname{Eq}(I)$, as required.

To see that $\operatorname{Image}(g'|_{\operatorname{Eq}(I')})\geqslant\operatorname{Eq}(I)$, suppose $x\in\operatorname{Eq}(I)$. Then there exists a path $p_x\in\operatorname{Core}(g,h)$, $\iota(p_x)=v_{g,h}=\tau(p_x)$, such that $\gamma_g\delta_g(p_x)=g(x)=h(x)=\gamma_h\delta_h(p_x)$ [16, Proposition 9.4], where $\gamma_g:\Gamma_g\to\Gamma_\Delta$ is the canonical morphism of directed, labeled graphs from Γ_g to the bouquet Γ_Δ with Δ petals. Hence, writing x' for the element of $F(\Sigma')$ corresponding to $p_x\in L(\operatorname{Core}(g,h),v_{g,h})$, we have that $gg'(x')=g(x)=h(x)=hh'(x)$. As h and g are injective, by Lemma 11, we have that $g'(x')=x=h'(x')$ as required. ◀

5 Prefix complexity of immersions in free groups

In this section we associate to an instance I of the $\mathrm{PCP}_{\mathrm{FG}}$ a certain complexity, called the "prefix complexity". We prove that the process of reduction does not increase this complexity, and that for all $n\in\mathbb{N}$ there are only finitely many instances with complexity $\leqslant n$.

Let $I=(\Sigma,\Delta,g,h)$ be an immersive instance of the $\mathrm{PCP}_{\mathrm{FG}}$. We define, analogously to [11, Section 4] (see also [9]), the *prefix complexity* $\sigma(I)$ as:

$$\begin{aligned}\sigma(I)=&\left|\cup_{a\in\Sigma^{\pm1}}\{x\in F(\Delta)\mid x\text{ is a proper prefix of }g(a)\}\right|\\&+\left|\cup_{a\in\Sigma^{\pm1}}\{x\in F(\Delta)\mid x\text{ is a proper prefix of }h(a)\}\right|.\end{aligned}$$

In the maps in Example 13, $\sigma(I)=10+6=16$, and $\sigma(I')=2+4=6$.

The process of reduction does not increase the prefix complexity, and we prove this by using the fact that, for any $a\in\Sigma^{\pm1}$, the proper prefixes of $g(a)$ and $h(a)$ are in bijection with the proper initial subpaths of the petals of Γ_g and Γ_h, respectively.

▶ **Lemma 15.** *Let $I=(\Sigma,\Delta,g,h)$ be an instance of the $\mathrm{PCP}_{\mathrm{FG}}$ with g and h immersions, and let I' be the reduction of I. Then $\sigma(I')\leqslant\sigma(I)$.*

Proof. We write $V_g\operatorname{Core}(g,h)=\{(v_g,v)\in V\operatorname{Core}(g,h)\mid v\in\Gamma_h\}=\delta_g^{-1}(v_g)$ for the set of vertices in the $\operatorname{Core}(g,h)$ whose first component is the central vertex v_g of Γ_g, and similarly for $V_h\operatorname{Core}(g,h)$. Note that $V_g\operatorname{Core}(g,h)\cap V_h\operatorname{Core}(g,h)=\{v_{g,h}\}$.

By construction, each petal of Γ_g and Γ_h corresponds to a letter $a\in\Sigma^{\pm1}$, and we shall denote the petal also by a. Write $P\Gamma$ for the set of reduced paths in a graph Γ. Similarly to $a\in P\Gamma_g$ and $a\in P\Gamma_h$, we map write $a\in P\operatorname{Core}(g,h)$ for the petal in $\operatorname{Core}(g,h)$ corresponding to $a\in(\Sigma')^{\pm1}$. From now on, all paths are assumed to be reduced. Define

$$\begin{aligned}G &= \cup_{a\in\Sigma^{\pm 1}}\{p \in \Gamma_g \mid p \text{ is a proper initial subpath of petal } a \in \Gamma_g\},\\ G' &= \cup_{a\in(\Sigma')^{\pm 1}}\{p \in \operatorname{Core}(g,h) \mid p \text{ is a proper initial subpath}\\ &\qquad\qquad \text{of } a \in \operatorname{Core}(g,h) \text{ s.t. its end vertex } \tau(p) \in V_g \operatorname{Core}(g,h)\},\end{aligned}$$

and define H and H' analogously. Hence, $\sigma(I) = |G|+|H|$ and analogously $\sigma(I') = |G'|+|H'|$.

For $a \in (\Sigma')^{\pm 1}$ let $q \in G'$ be a subpath of $a \in P\operatorname{Core}(g,h)$. Denote by r_q the shortest subpath of a intersecting q at only one point, their common end vertex (that is, $\tau(q) = \tau(r_q) = q \cap r_q$), such that $\iota(r_q) \in V_h \operatorname{Core}(g,h)$; the paths q and r_q can be seen as "facing" one another on a. As $V_g \operatorname{Core}(g,h) \cap V_h \operatorname{Core}(g,h) = \{v_{g,h}\}$, and as q is a proper initial subpath of a, the projection $\delta_h(r_q)$ is a non-trivial path in Γ_h. Note also that $\iota(\delta_h(r_q)) = v_h$, as $\iota(r_q) \in V_h \operatorname{Core}(g,h)$, hence there exists some $b \in \Sigma^{\pm 1}$ such that $\delta_h(r_q)$ is a proper initial subpath of the petal $b \in \Gamma_h$. Therefore, $\delta_h(r_q) \in H$. Let $\xi_H : G' \to H$ be the map given by $\xi_H(q) = \delta_h(r_q)$.

We now prove that ξ_H is injective. Suppose $p, q \in G'$ are such that $\xi_H(p) = \xi_H(q)$, and let r_p and r_q be the paths obtained from p and q, respectively, such that $\xi_H(p) = \delta_h(r_p)$ and $\xi_H(q) = \delta_h(r_q)$. Write e_p for the terminal edge of r_p, and e_q for the terminal edge of r_q, and note that these two edges have the same label and direction as $\delta_h(e_p) = \delta_h(e_q)$. Now, $\delta_g(\tau(e_p)) = v_g = \delta_g(\tau(e_q))$ as $\tau(r_p), \tau(r_q) \in V_g \operatorname{Core}(g,h)$, and as e_p and e_q have the same label and direction we have that $\delta_g(e_p) = \delta_g(e_q)$. Therefore, both δ_g and δ_h agree on e_p and e_q, and so as $\operatorname{Core}(g,h)$ is a subgraph of $\Gamma_g \times \Gamma_h$ we have that $e_p = e_q$. As $\operatorname{Core}(g,h)$ is a bouquet, there exists a unique shortest reduced path s such that $\iota(s) = v_{g,h}$ and $\tau(s) = s \cap e_p = \tau(e_p)$. Hence, $p = s = q$ as required.

Thus ξ_H is injective, and so $|G'| \leqslant |H|$. The same will hold for an analogously defined function ξ_H from H' to G, so $|H'| \leqslant |G|$. Therefore, $\sigma(I') = |G'|+|H'| \leqslant |G|+|H| = \sigma(I)$. ◀

For a fixed number $n \geqslant 1$ there are obviously only finitely many words which have $\leqslant n$ proper prefixes, and so the following is clear:

▶ **Lemma 16.** *There exist only finitely many distinct instances $I = (\Sigma, \Delta, g, h)$ of the* $\mathrm{PCP}_{\mathrm{FG}}$ *that satisfy $\sigma(I) \leqslant n$.*

As the reduction I' of an instance I gives $\sigma(I') \leqslant \sigma(I)$, and as $|\Sigma'| \subseteq |\Sigma|$, this means that the process of iteratively computing reductions will eventually cycle.

6 Solving the Algorithmic Equaliser Problem in free groups ($\mathrm{AEP}_{\mathrm{FG}}$)

The algorithm for solving the $\mathrm{AEP}_{\mathrm{FG}}$ for immersions is analogous to the algorithm for marked free monoid morphisms in Section 2. Our algorithm starts by making reductions $I_0, I_1, I_2, \ldots$, beginning with $I_0 = I$, the input instance. By Lemma 16, we will obtain an instance $I_j = (\Sigma_j, \Delta, g_j, h_j)$ such that one of the following will occur:

1. $|\Sigma_j| = 1$.
2. $\sigma(I_j) = 0$.
3. there exists some $i < j$ with $I_i = I_j$ (sequence starts cycling).

Keeping in mind the fact that reductions preserve equalisers (Lemma 14), we obtain in each case a subset $\Sigma_{g,h}$ (possibly empty) which forms a basis for $\operatorname{Eq}(I_j)$: For Case (1), writing $\Sigma_j = \{a\}$, the result holds as if $g(a^i) = h(a^i)$ then $g(a)^i = h(a)^i$ and so $g(a) = h(a)$ as roots are unique in a free group. For Case (2), $\sigma(I_j) = 0$ is equivalent to $|g(a)| = |h(a)| = 1$ for all $a \in \Sigma$. Suppose there exists some non-trivial reduced word $x = a_{i_1}^{\epsilon_1} \cdots a_{i_n}^{\epsilon_n}$ such that $g(x) = h(x)$.

Then as g and h are injective, the words $g(a_{i_1})^{\epsilon_1} \cdots g(a_{i_n})^{\epsilon_n}$ and $h(a_{i_1})^{\epsilon_1} \cdots h(a_{i_n})^{\epsilon_n}$ are freely reduced and hence are the same word, and so $g(a_{i_j}) = h(a_{i_j})$. The result then follows for Case (2). Case (3) has a more involved proof.

▶ **Lemma 17.** *Let $I = (\Sigma, \Delta, g, h)$ be an immersive instance of the* $\mathrm{PCP_{FG}}$ *that starts a cycle (i.e. starting the reduction process with I eventually gives I again). If* $\mathrm{Eq}(I)$ *is non-trivial then a subset of Σ forms a basis for* $\mathrm{Eq}(I)$*.*

Proof. There is a sequence of reductions beginning and ending at I:

$$I = I_0 \to I_1 \to \cdots \to I_{r-1} \to I_r = I$$

where $I_i = (\Sigma_i, \Delta, g_i, h_i)$. By Lemma 14, $\mathrm{Eq}(I_0) = g_1 g_2 \dots g_r(\mathrm{Eq}(I_r)) = \mathrm{Eq}(I_r)$ and so $\overline{g_r} = g_1 g_2 \dots g_r$ restricts to an automorphism of $\mathrm{Eq}(I_0)$, that is, $\overline{g_r}|_{\mathrm{Eq}(I_0)} \in \mathrm{Aut}(\mathrm{Eq}(I_0))$. For $\overline{h_r}$ defined analogously, $\overline{h_r}|_{\mathrm{Eq}(I_0)} \in \mathrm{Aut}(\mathrm{Eq}(I_0))$. Write $\mathrm{Eq}(I_k)^{(n)}$ for the set of words in $\mathrm{Eq}(I_k)$ of length precisely n, and $\mathrm{Eq}(I_k)^{(\leqslant n)}$ for the set of words in $\mathrm{Eq}(I_k)$ of length at most n. Consider some $x_0 \in \mathrm{Eq}(I_0)$ and write $x_r = \overline{g_r}^{-1}(x_0)$. Then

$$x_0 = g_1 g_2 \dots g_r(x_r) = \overline{g_r}(x_r),$$
$$x_0 = h_1 h_2 \dots h_r(x_r) = \overline{h_r}(x_r).$$

By Lemma 12, both g_i and h_i are immersions for each i, and so by Characterisation (3) of Lemma 9 we see that $|g_i(w)| \geqslant |w|$ for all $w \in F(\Sigma_i)$. Hence, $|x_0| \geqslant |x_r|$. Therefore, for all $m \geqslant 1$ the map $\overline{g_r}$ induces a map $\overline{g_r}^{(m)} : \mathrm{Eq}(I_r)^{(m)} \to \mathrm{Eq}(I_r)^{(\leqslant m)}$. Clearly $\overline{g_r}^{(1)}$ is a bijection, and so we inductively see that $\overline{g_r}^{(m)}$ has image $\mathrm{Eq}(I_r)^{(m)}$. Therefore, the automorphism $\overline{g_r}$ of $\mathrm{Eq}(I_0)$ is length-preserving ($|\overline{g_r}(w)| = |w|$ for all $w \in \mathrm{Eq}(I)$), and so maps the letters occurring in x_r to letters. Hence, $g_0 (= g_r)$ and $h_0 (= h_r)$ map the letters occuring in x_r to letters, and it follows that every letter occuring in x_r is a solution to I_0. Hence, a subset $\Sigma_{g,h}$ of Σ_r forms a basis for $\mathrm{Eq}(I_r)$. ◀

We now prove the central theorem of this article, which gives an algorithm to describe $\mathrm{Eq}(I)$ as the image of an immersion. Note that not every subgroup of a free group is the image of an immersion: for example, if $|\Sigma| = n$, then no subgroup of $F(\Sigma)$ of rank $> n$ is the image of an immersion. We store a morphism $f : F(\Sigma) \to F(\Delta)$ as a list $(f(a))_{a \in \Sigma}$.

▶ **Theorem 18.** *There exist an algorithm with input an immersive instance $I = (\Sigma, \Delta, g, h)$ of the* $\mathrm{PCP_{FG}}$ *and output an immersion $\psi_{g,h} : F(\Sigma_{g,h}) \to F(\Sigma)$ such that* $\mathrm{Image}(\psi_{g,h}) = \mathrm{Eq}(I)$*.*

Proof. Start by making reductions $I = I_0 \to I_1 \to \cdots$. By Lemma 16 we will obtain an instance $I_j = (\Sigma_j, \Delta, g_j, h_j)$ satisfying one of the Cases (1)–(3) above, and in each case a subset $\Sigma_{g,h}$ of Σ_j forms a basis for $\mathrm{Eq}(I_j)$. Since Σ_j is computable, this basis is as well.

In order to prove the theorem, it is sufficient to prove that there is a computable immersion $\psi_{g,h} : F(\Sigma_{g,h}) \to F(\Sigma)$. Consider the map $\tilde{g} = g_1 g_2 \cdots g_j : F(\Sigma_j) \to F(\Sigma)$ (and the analogous $\tilde{h}$). Now, each g_i is an immersion, so $\tilde{g}$ is the composition of immersions and hence is an immersion. Define $\psi_{g,h} = \tilde{g}|_{F(\Sigma_{g,h})}$. This map is computable from $\tilde{g}$, and as $\Sigma_{g,h} \subseteq \Sigma_j$, the map $\psi_{g,h}$ is an immersion. As $\mathrm{Image}(\psi_{g,h}) = g_1 g_2 \dots g_j(\mathrm{Eq}(I_j)) = \mathrm{Eq}(I)$, by Lemma 14 and the above, the result follows. ◀

We now prove Corollary G, which solves the $\mathrm{AEP_{FG}}$ for immersions of free groups.

Proof of Corollary G. To algorithmically obtain a basis for $\mathrm{Eq}(I)$, first obtain the immersion $\psi_{g,h} : F(\Sigma_{g,h}) \to F(\Sigma)$ given by Theorem 18. Then, recalling that we store $\psi_{g,h}$ as a list $(\psi_{g,h}(a))_{a \in \Sigma}$, the required basis is the set of elements in this list, so the set $\{\psi_{g,h}(a)\}_{a \in \Sigma}$. ◀

7 Sets of immersions

We now prove Theorem C and its corollaries. We first give a general result, from which the non-algorithmic part of Theorem C follows quickly. An *immersed subgroup* H of a free group $F(\Sigma)$ is a subgroup which is the image of an immersion. The proof of Lemma 19 is fundamentally identical to the proof of Lemma 6, via Characterisation 1 of Lemma 9.

▶ **Lemma 19.** *If $\{H_j\}_{j\in J}$ is a set of immersed subgroups of $F(\Sigma)$ then the intersection $\bigcap_{j\in J} H_j$ is immersed.*

Proof. Firstly, suppose $x, y \in H_j$ for some $j \in J$, and let z be their maximal common prefix. Then z decomposes uniquely as $z_1 z_2 \cdots z_n z'_{n+1}$ such that each $z_k \in H_j$. As H_j is immersed, and as z is a maximal common prefix of x and y, we have that $z \in H_j$.

Now, suppose $x, y \in \bigcap_{j\in J} H_j$, and suppose they both begin with some letter $a \in \Sigma \cup \Sigma^{-1}$. By the above, their maximal common prefix z_a is contained in each H_j and so is contained in $\bigcap_{j\in J} H_j$. Therefore, z_a is a prefix of every element of $\bigcap_{j\in J} H_j$ beginning with an a. It follows that $\bigcap_{j\in J} H_j$ is immersed, as required. ◀

The following lemma corresponds to the algorithmic part of Theorem C. Similar to the above, the proof of the lemma is fundamentally identical to the proof of Lemma 7.

▶ **Lemma 20.** *There exists an algorithm with input a finite set of immersions S from $F(\Sigma)$ to $F(\Delta)$ and output an immersion $\psi_S : F(\Sigma_S) \to F(\Sigma)$ such that* $\mathrm{Image}(\psi_S) = \mathrm{Eq}(S)$.

Proof. We proceed by inducting on $|S|$. By Theorem 18, the result holds if $|S| = 2$. Suppose the result holds for all sets of n immersions, $n \geqslant 2$, and let S be a set of $n+1$ immersions. Take elements $g, h \in S$, and write $S_g = S \setminus \{g\}$. By hypothesis, we can algorithmically obtain immersions $\psi_{S_g} : F(\Sigma_{S_g}) \to F(\Sigma)$ and $\psi_{g,h} : F(\Sigma_{g,h}) \to F(\Sigma)$ such that $\mathrm{Image}(\psi_{S_g}) = \mathrm{Eq}(S_g)$ and $\mathrm{Image}(\psi_{g,h}) = \mathrm{Eq}(g, h)$.

By Lemma 12, there exists a (computable) immersion $\psi_S : F(\Sigma_S) \to F(\Sigma)$ such that $\mathrm{Image}(\psi_S) = \mathrm{Image}(\psi_{S_g}) \cap \mathrm{Image}(\psi_{g,h})$ (the map ψ_S corresponds to the map k in the lemma, and Σ_S to Σ'). Then we have the required equality:

$$\begin{aligned}\mathrm{Image}(\psi_S) &= \mathrm{Image}(\psi_{S_g}) \cap \mathrm{Image}(\psi_{g,h})\\ &= \mathrm{Eq}(S_g) \cap \mathrm{Eq}(g,h)\\ &= \mathrm{Eq}(S).\end{aligned}$$ ◀

We now prove Theorem C, which states that the equaliser is the image of a computable immersion.

Proof of Theorem C. By Lemma 19, there exists an alphabet Σ_S and an immersion $\psi_S : F(\Sigma_S) \to F(\Sigma)$ such that $\mathrm{Image}(\psi_S) = \mathrm{Eq}(S)$, while by Lemma 20 if S is finite then such an immersion can be algorithmically found. ◀

We now prove Corollary D, which solves the simultaneous $\mathrm{PCP}_{\mathrm{FG}}$ for immersions.

Proof of Corollary D. First find a basis for $\mathrm{Eq}(I)$: obtain the immersion $\psi_{g,h} : F(\Sigma_{g,h}) \to F(\Sigma)$ given by Theorem C. Then, recalling that we store $\psi_{g,h}$ as a list $(\psi_{g,h}(a))_{a\in\Sigma}$, the required basis is the set of elements in this list, so the set $\{\psi_{g,h}(a)\}_{a\in\Sigma}$. Then $\mathrm{Eq}(S)$ is trivial if and only if this basis is empty. ◀

Finally, we prove Corollary E, which says that $\mathrm{Eq}(S)$ is of rank $\leqslant |\Sigma|$.

Proof of Corollary E. Consider the immersion $\psi_{g,h} : F(\Sigma_{g,h}) \to F(\Sigma)$ given by Theorem C. As $\mathrm{Image}(\psi_{g,h}) = \mathrm{Eq}(S)$ we have that $\mathrm{rk}(\mathrm{Eq}(S)) \leqslant |\Sigma_{g,h}|$, while as $\psi_{g,h}$ is an immersion we have that $|\Sigma_{g,h}| \leqslant |\Sigma|$, and the result follows. ◀

8 Algorithm to compute the equaliser

Theorems A and C produce the equaliser of a set S of morphisms as the image of a computable map ψ_S. For $S = \{g, h\}$, the structure of the algorithm that gives ψ_S (as a list of elements representing the images of the generators) is given below. The values for M in step 3 correspond to the number of instances of complexity $\leqslant \sigma(I)$, as explained in Section 9.

Algorithm 1 Structure of the algorithm that gives ψ_S.

```
1. Input $I = (\Sigma, \Delta, g, h)$.
2. Set $c =; 0$, $i := 0$, $I_0 := I$
3. Set $M := (|\Delta| + 1)^{2|\Sigma|(\sigma(I)+1)}$ (monoids) or $M := (2|\Delta|)^{2|\Sigma|(\sigma(I)+1)}$ (groups)
4. $i := i + 1$
5. Reduce instance $I_{i-1}$ to $I_i$ (as in Sections 2 and 4); store $I_i$ in memory
6. If $I_i$ has source alphabet of size 1 or $\sigma = 0$ then:
   a. Compute a basis $B$ for $\mathrm{Eq}(I_i)$
   b. Print composition($B$, $i$) (see below) and terminate.
7. If $I_i$ is simpler than $I_{i-1}$ (smaller source alphabet or $\sigma$) then set $c = 0$ and goto (4)
8. If $c > M$ then there exists a cycle which starts with $I_i$.
   a. Compute a basis $B$ for $\mathrm{Eq}(I_i)$
   b. Print composition($B$, $i$) and terminate.
```

Procedure `composition(`B`, `i`)` computes the composition of a map, stored as a list B, with the maps obtained in the reduction process, indexed from i downwards.

Algorithm 2 `composition(`B`, `i`)`.

```
1. Set $B := g_i(B)$, where $g_i$ is loaded from memory
2. $i := i - 1$
3. If $i \geqslant 0$, goto (1); else, output $B$.
```

9 Complexity analysis

The size of an instance $I = (\Sigma, \Delta, S)$, S a set of morphisms, is $|\Sigma| + |\Delta| + \sum_{g \in S} \sum_{a \in \Sigma^{\pm 1}} |g(a)|$. The algorithm underlying Theorem A can be run with $O(2^n)$ space, where n is the size of the input instance I, which gives a time bound of $O(2^{2^n})$. The space grows exponentially, unlike in [11], because the algorithm computes instances that must each be stored (as the immersion $\psi_{g,h}$ is their composition; this corresponds to the function `composition(`B`, `i`)`, above). To obtain this space complexity, first suppose $|S| = 2$ (so consider the function `pairs(`g`, `h`)`, above). There are at most $(|\Delta| + 1)^{2|\Sigma|(\sigma(I)+1)}$ instances I_j with $\sigma(I_j) \leqslant \sigma(I)$ [11, Proof of Lemma 3], which is $O(2^n)$. Every other procedure requires asymptotically less space, and hence if $|S| = 2$ we require $O(2^n)$ space. For $S = \{g_1, \dots, g_k\}$, note that we only need to compute the immersions corresponding to $\mathrm{Eq}(g_i, g_{i+1})$ for $1 \leqslant i < k$ (as these intersect to give $\mathrm{Eq}(S)$), and these can all be stored in $(k-1)\times O(2^n)=O(2^n)$ space. Intersection corresponds to reduction, and reduction can be done in PSPACE [11, Section 6]. Hence, the algorithm can be run in $O(2^n)$ space.

Similarly, the algorithm underlying Theorem C runs in $O(2^n)$ space, where n is the input size. The main difference to the above is that there are $O(2^n)$ instances I_j with $\sigma(I_j)\leqslant\sigma(I)$. To see this, write $m:=\sigma(I)$ and $d:=|\Delta|$. If $I_j=(\Sigma_j,\Delta_j,g_j,h_j)$ is such that $\sigma(I_j)\leqslant m$ then $|g(a)|\leqslant m+1$ for all $a\in\Sigma_j^{\pm 1}$, as $g(a)$ has at most m proper prefixes, and similarly $|h(a)|\leqslant m+1$. There are $2d(2d-1)^m$ freely reduced words of length $m+1$ in $F(\Sigma_j)$, and so (by using the empty word) we see that there are at most $(2d)^{m+1}$ freely reduced words of length *at most* $m+1$. As each list of $2|\Sigma_j|$ words defines an instance, there are at most $(2d)^{2|\Sigma_j|(m+1)}\leqslant(2d)^{2|\Sigma|(m+1)}$ instances that satisfy $\sigma(I)\leqslant m$. This is $O(2^n)$ as required.

10 The density of marked morphisms and immersions

Here we show that immersions and marked morphisms are not a negligible (i.e. density zero) subset of the entire set of free group and free monoid morphisms, respectively, but represent a strictly positive proportion of those.

Suppose $\Sigma=\{a_1,\ldots,a_k\}$, and $k=|\Sigma|\geqslant|\Delta|=m$. A morphism in a free monoid or free group, $\phi:\Sigma^*\to\Delta^*$ or $\phi:F(\Sigma)\to F(\Delta)$, is uniquely determined by $(\phi(a_1),\ldots,\phi(a_k))$.

We start with the monoid case. There are m^n words of length n in Δ^*, and $\sum_{1\leqslant i\leqslant n}m^i\sim cm^n$ words of length $\leqslant n$, where $c=\frac{m}{m-1}$ and we write $a_n\sim b_n$ for $\lim_{n\to\infty}\frac{a_n}{b_n}=1$. If α_n is the number of morphisms from Σ^* to Δ^* with images of length at most n, then $\alpha_n\sim(cm^n)^k$. Now let β_n be the number of marked morphisms from Σ^* to Δ^* with images of length at most n. For a marked morphism ϕ, each word in the list $(\phi(a_1),\ldots,\phi(a_k))$ must start with a different letter, followed by any word of length $\leqslant n-1$. Since there are $\binom{m}{k}k!$ options for the first letters, $\beta_n\sim\binom{m}{k}k!(cm^{n-1})^k$ and we get:

▶ **Proposition 21.** *If α_n and β_n are the numbers of morphisms and marked morphisms, respectively, from Σ^* to Δ^*, with images of length at most n, then the density of the marked morphisms among all morphisms is a positive constant:*

$$\lim_{n\to\infty}\frac{\beta_n}{\alpha_n}=\lim_{n\to\infty}\frac{\binom{m}{k}k!(cm^{n-1})^k}{(cm^n)^k}=\frac{m!}{m^k(m-k)!}.$$

In the free group case the counting is similar, but there are more restrictions on the images of an immersion: first, all images need to be reduced words, and second, not just their first letters are constrained, but also their last letters. For some ϕ, let the set of first letters of $(\phi(a_1),\ldots,\phi(a_k))$ be $F\subset\Delta^{\pm 1}$, and the set of inverses of the last letters be $L\subset\Delta^{\pm 1}$. Then $\phi:F(\Sigma)\mapsto F(\Delta)$ is an immersion if all letters in F are distinct, all the letters in L are distinct, which implies $|F|=|L|=k$, and furthermore $F\cap L=\emptyset$. An image $\phi(a_i)$ of length n has the form $\phi(a_i)=\alpha x_1x_2\ldots x_{n-2}\beta$, where $\alpha\in F$, $\beta^{-1}\in L$, $x_i\in\Delta^{\pm 1}$, and $\phi(a_i)$ is reduced, so $x_1\neq\alpha^{-1}$ and $x_{n-2}\neq\beta^{-1}$. Counting such words is more delicate than in the monoid case, but the asymptotics are similar, due to the following result ([6, Proposition 1]).

▶ **Proposition 22.** *Let A and B be subsets of $\Delta^{\pm 1}$. The number of elements of length n in $F(\Delta)$ that do not start with a letter in A and do not end with a letter in B is equal to*

$$f_{A,B}(n)=\frac{(2m-|A|)(2m-|B|)(2m-1)^{n-1}+xm+(-1)^n(|A||B|-ym)}{2m},$$

where $x=|A\cap B|-|A^{-1}\cap B|, y=|A\cap B|+|A^{-1}\cap B|$, and $m=|\Delta|$.

Let $A = \{\alpha^{-1}\}$ and $B = \{\beta^{-1}\}$; then since the number of possible $\phi(a_i)$ of length $\leqslant n$ is equal to the number of reduced words of length $\leqslant n-2$ starting with a letter different from α^{-1} and ending with a letter different from β^{-1}, this number is $\sum_{1 \leqslant j \leqslant n-2} f_{A,B}(j)$. Since $|A| = |B| = 1$, $f_{A,B}(j)$ is asymptotically $(2m-1)^j$, and the number of possible $\phi(a_i)$ is $\sim c_1(2m-1)^{n-2}$, where c_1 is a constant depending on m. Thus for fixed sets F and L the number of immersions ϕ with images in the ball of radius n is $\sim (c_1(2m-1)^{n-2})^k$. Since there are only finitely many choices for sets F and L of first and last letters, respectively, and the number of k-tuples of elements in $F(\Delta)$ of length $\leqslant n$ is $\sim (c_2(2m-1)^n)^k$ for some constant c_2, the number of immersions over the total number of maps $F(\Sigma) \mapsto F(\Delta)$ is $\sim \frac{(c_1(2m-1)^{n-2})^k}{(c_2(2m-1)^n)^k}$; so as $n \mapsto \infty$, this ratio is a positive constant depending on k and m.

References

1 Gilbert Baumslag, Alexei G. Myasnikov, and Vladimir Shpilrain. Open problems in combinatorial group theory. Second edition. In *Combinatorial and geometric group theory (New York, 2000/Hoboken, NJ, 2001)*, volume 296 of *Contemp. Math.*, pages 1–38. Amer. Math. Soc., Providence, RI, 2002. doi:10.1090/conm/296/05067.

2 George M. Bergman. Supports of derivations, free factorizations, and ranks of fixed subgroups in free groups. *Trans. Amer. Math. Soc.*, 351(4):1531–1550, 1999. doi:10.1090/S0002-9947-99-02087-5.

3 Mladen Bestvina and Michael Handel. Train tracks and automorphisms of free groups. *Ann. of Math. (2)*, 135(1):1–51, 1992. doi:10.2307/2946562.

4 Meera Blattner and Tom Head. Automata that recognize intersections of free submonoids. *Information and Control*, 35(3):173–176, 1977.

5 Laura Ciobanu, Armando Martino, and Enric Ventura. The generic Hanna Neumann Conjecture and Post Correspondence Problem, 2008. URL: http://www-eupm.upc.es/~ventura/ventura/files/31t.pdf.

6 Laura Ciobanu and Sasa Radomirovic. Restricted walks in regular trees. *Electronic J. of Combinatorics*, 13(R93), 2006. doi:10.37236/1119.

7 Warren Dicks and Enric Ventura. *The group fixed by a family of injective endomorphisms of a free group*, volume 195 of *Contemporary Mathematics*. American Mathematical Society, Providence, RI, 1996. doi:10.1090/conm/195.

8 Volker Diekert, Olga Kharlampovich, Markus Lohrey, and Alexei Myasnikov. Algorithmic problems in group theory. *Dagstuhl seminar report 19131*, 2019. URL: http://drops.dagstuhl.de/opus/volltexte/2019/11293/pdf/dagrep_v009_i003_p083_19131.pdf.

9 A. Ehrenfeucht, J. Karhumäki, and G. Rozenberg. The (generalized) Post correspondence problem with lists consisting of two words is decidable. *Theoret. Comput. Sci.*, 21(2):119–144, 1982. doi:10.1016/0304-3975(89)90080-7.

10 A. Ehrenfeucht and G. Rozenberg. Elementary homomorphisms and a solution of the D0L sequence equivalence problem. *Theoret. Comput. Sci.*, 7(2):169–183, 1978. doi:10.1016/0304-3975(78)90047-6.

11 Vesa Halava, Mika Hirvensalo, and Ronald de Wolf. Marked PCP is decidable. *Theoret. Comput. Sci.*, 255(1-2):193–204, 2001. doi:10.1016/S0304-3975(99)00163-2.

12 Tero Harju and Juhani Karhumäki. Morphisms. In *Handbook of formal languages, Vol. 1*, pages 439–510. Springer, Berlin, 1997.

13 Štěpán Holub. Binary equality sets are generated by two words. *J. Algebra*, 259(1):1–42, 2003. doi:10.1016/S0021-8693(02)00534-3.

14 W. Imrich and E. C. Turner. Endomorphisms of free groups and their fixed points. *Math. Proc. Cambridge Philos. Soc.*, 105(3):421–422, 1989. doi:10.1017/S0305004100077781.

15 Ilya Kapovich. Mapping tori of endomorphisms of free groups. *Comm. Algebra*, 28(6):2895–2917, 2000. doi:10.1080/00927870008826999.

16 Ilya Kapovich and Alexei Myasnikov. Stallings foldings and subgroups of free groups. *J. Algebra*, 248(2):608–668, 2002. doi:10.1006/jabr.2001.9033.

17 Juhani Karhumäki. A note on intersections of free submonoids of a free monoid. *Semigroup Forum*, 29(1-2):183–205, 1984. doi:10.1007/BF02573324.

18 Juhani Karhumäki and Aleksi Saarela. Noneffective regularity of equality languages and bounded delay morphisms. *Discrete Math. Theor. Comput. Sci.*, 12(4):9–17, 2010.

19 Alexei Myasnikov, Andrey Nikolaev, and Alexander Ushakov. The Post correspondence problem in groups. *J. Group Theory*, 17(6):991–1008, 2014. doi:10.1515/jgth-2014-0022.

20 Turlough Neary. Undecidability in binary tag systems and the post correspondence problem for five pairs of words. In *32nd International Symposium on Theoretical Aspects of Computer Science*, volume 30 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages 649–661. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2015.

21 Emil L. Post. A variant of a recursively unsolvable problem. *Bull. Amer. Math. Soc.*, 52:264–268, 1946. doi:10.1090/S0002-9904-1946-08555-9.

22 John R. Stallings. Graphical theory of automorphisms of free groups. In *Combinatorial group theory and topology (Alta, Utah, 1984)*, volume 111 of *Ann. of Math. Stud.*, pages 79–105. Princeton Univ. Press, Princeton, NJ, 1987.

23 Bret Tilson. The intersection of free submonoids of a free monoid is free. *Semigroup Forum*, 4:345–350, 1972. doi:10.1007/BF02570808.

24 E. Ventura. Fixed subgroups in free groups: a survey. In *Combinatorial and geometric group theory (New York, 2000/Hoboken, NJ, 2001)*, volume 296 of *Contemp. Math.*, pages 231–255. Amer. Math. Soc., Providence, RI, 2002. doi:10.1090/conm/296/05077.

Timed Games and Deterministic Separability

Lorenzo Clemente
University of Warsaw, Poland
clementelorenzo@gmail.com

Sławomir Lasota
University of Warsaw, Poland
sl@mimuw.edu.pl

Radosław Piórkowski
University of Warsaw, Poland
r.piorkowski@mimuw.edu.pl

Abstract

We study a generalisation of Büchi-Landweber games to the timed setting. The winning condition is specified by a non-deterministic timed automaton with epsilon transitions and only Player I can elapse time. We show that for fixed number of clocks and maximal numerical constant available to Player II, it is decidable whether she has a winning timed controller using these resources. More interestingly, we also show that the problem remains decidable even when the maximal numerical constant is not specified in advance, which is an important technical novelty not present in previous literature on timed games. We complement these two decidability result by showing undecidability when the number of clocks available to Player II is not fixed.

As an application of timed games, and our main motivation to study them, we show that they can be used to solve the deterministic separability problem for nondeterministic timed automata with epsilon transitions. This is a novel decision problem about timed automata which has not been studied before. We show that separability is decidable when the number of clocks of the separating automaton is fixed and the maximal constant is not. The problem whether separability is decidable without bounding the number of clocks of the separator remains an interesting open problem.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects; Theory of computation → Quantitative automata; Theory of computation → Timed and hybrid models

Keywords and phrases Timed automata, separability problems, timed games

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.121

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

Related Version A full version of the paper is available at [13], `https://arxiv.org/abs/2004.12868`.

Funding *Lorenzo Clemente*: Partially supported by the Polish NCN grant 2017/26/D/ST6/00201.
Sławomir Lasota: Partially supported by the ERC project Lipa (grant agreement No. 683080).
Radosław Piórkowski: Partially supported by the Polish NCN grant 2017/27/B/ST6/02093.

Acknowledgements We thank S. Krishna for fruitful discussions. We kindly thank the anonymous reviewers for their helpful comments.

1 Introduction

Separability. Separability is a classical problem in theoretical computer science and mathematics. A set S *separates* two sets L, M if $L \subseteq S$ and $S \cap M = \emptyset$. Intuitively, a separator S provides a certificate of disjointness, yielding information on the structure of L, M up to some granularity. There are many elegant results in computer science and mathematics showing that separators with certain properties always exist, such as Lusin's separation theorem in topology (two disjoint analytic sets are separable by a Borel set), Craig's inter-

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 121; pp. 121:1–121:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

polation theorem in logic (two contradictory first-order formulas can be separated by one containing only symbols in the shared vocabulary), in model theory (two disjoint projective classes of models are separable by an elementary class), in formal languages (two disjoint Büchi languages of infinite trees are separable by a weak language, generalising Rabin's theorem [44]), in computability (two disjoint co-recursively enumerable sets are separable by a recursive set), in the analysis of infinite-state systems (two disjoint languages recognisable by well-structured transition systems are regular separable [17]), etc.

When separability is not trivial, one may ask whether the problem is decidable. Let $\mathcal{C}$ and $\mathcal{S}$ be two classes of sets. The *$\mathcal{S}$-separability* problem for $\mathcal{C}$ amounts to decide whether, for every input sets $L, M \in \mathcal{C}$ there is a set $S \in \mathcal{S}$ separating L, M. Many results of this kind exist when $\mathcal{C}$ is the class of regular languages of finite words over finite alphabets, and $\mathcal{S}$ ranges over piecewise-testable languages [40, 18] (later generalised to context-free languages [19] and finite trees [28]), locally and locally threshold testable languages [41], first-order logic definable languages [43] (generalised to some fixed levels of the first-order hierarchy [42]). For classes of languages $\mathcal{C}$ beyond the regular ones, decidability results are more rare. For example, regular separability of context-free languages is undecidable [45, 32, 34]. Nonetheless, there are positive decidability results for separability problems on several infinite-state models, such as Petri nets [12], Parikh automata [11], one-counter automata [16], higher-order and collapsible pushdown automata [30, 14], and others.

In this paper, we go beyond languages over finite alphabets, and we study the separability problem for timed languages, which we introduce next.

Timed automata. Nondeterministic timed automata are one of the most widespread model of real-time reactive systems. They consist of finite automata extended with real-valued clocks which can be reset and compared by inequality constraints. Alur and Dill's seminal result showed PSPACE-completeness of the reachability problem [3], for which they received the 2016 Church Award [1]. This paved the way to the automatic verification of timed systems, eventually leading to mature tools such as UPPAAL [6], UPPAAL Tiga (timed games) [10], and PRISM (probabilistic timed automata) [36]. The reachability problem is still a very active research area to these days [23, 31, 2, 26, 27, 29], as well as expressive generalisations thereof, such as the binary reachability problem [15, 21, 35, 25].

Deterministic timed automata form a strict subclass of nondeterministic timed automata where the next configuration is uniquely determined from the current one and the timed input symbol. This class enjoys stronger properties, such as decidable universality/inclusion problems and complementability [3], and it is used in several applications, such as test generation [39], fault diagnosis [7], learning [49, 46]; defining winning conditions in timed games [4, 33, 8], and in a notion of recognisability of timed languages [37].

The *k, m-deterministic separability* problem asks, given two nondeterministic timed automata $\mathcal{A}$ and $\mathcal{B}$ with epsilon transitions, whether there exists a deterministic timed automaton $\mathcal{S}$ with k clocks and maximal constant bounded by m s.t. $L(\mathcal{S})$ separates $L(\mathcal{A}), L(\mathcal{B})$. Likewise one defines *$k$-deterministic separability*, where only k is fixed but not m. We can see $\mathcal{A}$ as recognising a set of good behaviours which we want to preserve and $\mathcal{B}$ recognising a set of bad behaviours which we want to exclude; a deterministic separator, when it exists, provides a compromise between these two conflicting requirements. To the best of our knowledge, separability problems for timed automata have not been investigated before. Our first main result is decidability of k, m and k-deterministic separability.

▶ **Theorem 1.1.** *The k, m and k-deterministic separability problems are decidable.*

Decidability of deterministic separability should be contrasted with undecidability of the corresponding membership problem [24, 48]. This is a rare circumstance, which is shared with languages recognised by one-counter nets [16], and conjectured to be the case for the full class of Petri net languages[1]. We solve the separability problem by reducing to an appropriate timed game (cf. Theorems 1.2 and 1.3 below). This forms the basis of our interest in defining and studying a non-trivial class of timed games, which we introduce next.

Timed games. We consider the following timed generalisation of Büchi-Landweber games [9]. There are two players, called Player I and Player II, which play taking turns in a strictly alternating fashion. At the i-th round, Player I selects a letter a_i from a finite alphabet and a nonnegative timestamp t_i from $\mathbb{R}_{\geq 0}$, and Player II replies with a letter b_i from a finite alphabet. At doomsday, the two players have built an infinite play $\pi = (a_1, b_1, t_1)\,(a_2, b_2, t_2) \cdots$, and Player I wins if, and only if, π belongs to her winning set, which is a timed langauge recognised by a nondeterministic timed automaton with ε-steps. For a fixed number of clocks $k \in \mathbb{N}$ and maximal constant $m \in \mathbb{N}$, the *k, m-timed synthesis problem* asks whether there is a finite-memory timed controller for Player II using at most k clocks and guards with maximal constant bounded by m in absolute value, ensuring that every play π conform to the controller is winning for Player II. Our second contribution is decidability of this problem.

▶ **Theorem 1.2.** *For every fixed $k, m \in \mathbb{N}$, the k, m-timed synthesis problem is decidable.*

We reduce to an untimed finite-state game with an ω-regular winning condition [9]. This should be contrasted with undecidability of the same problem when the set of winning plays for Player II is a nondeterministic timed language (cf. [22] for a similar observation). The *k-timed synthesis problem* asks whether there exists a bound $m \in \mathbb{N}$ s.t. the k, m-timed synthesis problem has a positive answer for Player II, which we also show decidable.

▶ **Theorem 1.3.** *For every fixed $k \in \mathbb{N}$, the k-timed synthesis problem is decidable.*

This requires the synthesis of the maximal constant m, which is a very interesting a technical novelty not shared with the current literature on timed games. We design a protocol whereby Player II demands Player I to be informed when clocks elapse one time unit. We require that the number of such consecutive requests be finite, yielding a bound on m (when such a value exists).

Finally, we complement the two decidability results above by showing that the synthesis problem is undecidable when the number of clocks k available to Player II is not specified in advance (cf. Theorem 6.1).

There are many variants of timed games in the literature, depending whether the players must enforce a nonzeno play, who controls the elapse of time, concurrent actions, etc. [50, 38, 5, 22, 20]. In this terminology, our timed games are asymmetric (only Player I can elapse time) and turn-based (the two players strictly alternate).

2 Preliminaries

Let $\mathbb{R}$ be the set of real numbers and $\mathbb{R}_{\geq 0}$ the set of nonnegative real numbers. For two sets A and B, let their Cartesian product be $A \cdot B$. Let $A^0 = \{\varepsilon\}$, and, for every $n \geq 0$, $A^{n+1} = A \cdot A^n$. The set of finite sequences over A is $A^* = \bigcup_{n \geq 0} A^n$, A^ω is the set of infinite

[1] All these classes of languages have a decidable disjointness problem, however regular separability is not always decidable in this case [47].

sequences, and $A^\infty = A^* \cup A^\omega$. A *(monotonic) timed word* over a finite alphabet Σ is a sequence $w = (a_1, t_1)\,(a_2, t_2) \cdots \in (\Sigma \cdot \mathbb{R}_{\geq 0})^\infty$ s.t. $0 \leq t_1 \leq t_2 \leq \cdots$, and it is *strictly monotonic* if $0 \leq t_1 < t_2 < \cdots$. A *timed language* over Σ is a set $L \subseteq (\Sigma \cdot \mathbb{R}_{\geq 0})^\infty$ of monotonic timed words; it is *strictly monotonic* if it contains only strictly monotonic timed words. The *untiming* $\mathsf{untime}(w)$ of a timed word w as above is the word $a_0 a_1 \cdots \in \Sigma^\infty$ obtained from w by removing the timestamps, which is extended to timed languages L pointwise as $\mathsf{untime}(L) = \{\mathsf{untime}(w) \mid w \in L\}$.

Clocks, constraints, and regions. Let $\mathtt{X} = \{\mathtt{x}_1, \ldots, \mathtt{x}_k\}$ be a finite set of clocks. A *clock valuation* is a function $\mu \in \mathbb{R}_{\geq 0}^{\mathtt{X}}$ assigning a nonnegative real number $\mu(\mathtt{x})$ to every clock $\mathtt{x} \in \mathtt{X}$. For a nonnegative time elapse $\delta \in \mathbb{R}_{\geq 0}$, we denote by $\mu + \delta$ the valuation assigning $\mu(\mathtt{x}) + \delta$ to every clock $\mathtt{x}$; for a set of clocks $\mathtt{Y} \subseteq \mathtt{X}$, let $\mu[\mathtt{Y} \mapsto 0]$ be the valuation which is 0 on $\mathtt{Y}$ and agrees with μ on $\mathtt{X} \setminus \mathtt{Y}$. We write μ_0 for the clock valuation mapping every clock $\mathtt{x} \in \mathtt{X}$ to $\mu_0(\mathtt{x}) = 0$. A *clock constraint* is a quantifier-free formula of the form

$$\varphi, \psi \ ::\equiv \ \mathbf{true} \mid \mathbf{false} \mid \mathtt{x}_i - \mathtt{x}_j \sim z \mid \mathtt{x}_i \sim z \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi,$$

where $\sim \in \{=, <, \leq, >, \geq\}$ and $z \in \mathbb{Z}$. A clock valuation μ satisfies a constraint φ, written $\mu \models \varphi$, if interpreting each clock $\mathtt{x}_i$ by $\mu(\mathtt{x}_i)$ makes φ true. A constraint φ defines the set $[\![\varphi]\!] = \left\{\mu \in \mathbb{R}_{\geq 0}^{\mathtt{X}} \mid \mu \models \varphi\right\}$ of all clock valuation it satisfies. When the set of clocks is fixed to $\mathtt{X}$ and the absolute value of constants is bounded by $m \in \mathbb{N}$, we speak of $\mathtt{X}, m$-constraints.Two valuations $\mu, \nu \in \mathbb{R}_{\geq 0}^{\mathtt{X}}$ are $\mathtt{X}, m$*-region equivalent*, written $\mu \sim_{\mathtt{X},m} \nu$, if they satisfy the same $\mathtt{X}, m$-constraints. An $\mathtt{X}, m$-region $[\mu]_{\mathtt{X},m} \subseteq \mathbb{R}_{\geq 0}^{\mathtt{X}}$ is an equivalence class of clock valuations w.r.t. $\sim_{\mathtt{X},m}$. For fixed finite $\mathtt{X}$ and $m \in \mathbb{N}$ there are finitely many $\mathtt{X}, m$-regions; let $\mathsf{Reg}(\mathtt{X}, m)$ denote this set. Let $\mu_0 = \lambda \mathtt{x}.0$ and $\mathtt{r}_0 = [\mu_0]_{\mathtt{X},m}$ be its region. We write $\mathtt{r} \models \varphi$ for a region $\mathtt{r} \in \mathsf{Reg}(\mathtt{X}, m)$ whenever $\mu \models \varphi$ for some $\mu \in \mathtt{r}$ (equivalently, for all such μ's). The *characteristic clock constraint* $\varphi_{\mathtt{r}}$ of a region $\mathtt{r} \in \mathsf{Reg}(\mathtt{X}, m)$ is the unique constraint (up to logical equivalence) s.t. $[\![\varphi_{\mathtt{r}}]\!] = \mathtt{r}$. When convenient, we deliberately confuse regions with their characteristic constraints. For two regions $\mathtt{r}, \mathtt{r}' \in \mathsf{Reg}(\mathtt{X}, m)$ we write $\mathtt{r} \prec \mathtt{r}'$ whenever $\mathtt{r} = [\mu]_{\mathtt{X},m}$, $\mathtt{r}' = [\mu + \delta]_{\mathtt{X},m}$ for some $\delta > 0$, and $\mathtt{r} \neq \mathtt{r}'$.

Timed automata. A (nondeterministic) *timed automaton* is a tuple $\mathcal{A} = (\Sigma, \mathtt{L}, \mathtt{X}, \mathtt{I}, \mathtt{F}, \Delta)$, where Σ is a finite input alphabet, $\mathtt{L}$ is a finite set of control locations, $\mathtt{X}$ is a finite set of clocks, $\mathtt{I}, \mathtt{F} \subseteq \mathtt{L}$ are the subsets of initial, resp., final, control locations, and Δ is a finite set of transition rules of the form $\mathsf{tr} = (p, a, \varphi, \mathtt{Y}, q) \in \Delta$, with $p, q \in \mathtt{L}$ control locations, $a \in \Sigma_\varepsilon := \Sigma \cup \{\varepsilon\}$, φ a clock constraint to be tested and $\mathtt{Y} \subseteq \mathtt{X}$ the set of clocks to be reset to 0. A *configuration* of a timed automaton $\mathcal{A}$ is a pair (p, μ) consisting of a control location $p \in \mathtt{L}$ and a clock valuation $\mu \in \mathbb{R}_{\geq 0}^{\mathtt{X}}$. It is *initial* if p is so and $\mu = \mu_0$. It is *final* if p is so. Every transition rule tr induces a discrete transition between configurations $(p, \mu) \xrightarrow{\mathsf{tr}} (q, \nu)$ when $\mu \models \varphi$ and $\nu = \mu[\mathtt{Y} \mapsto 0]$. Intuitively, a discrete transition consists of a test of the clock constraint φ, reset of clocks $\mathtt{Y}$, and step to the location q. Moreover, for every nonnegative $\delta \in \mathbb{R}_{\geq 0}$ and every configuration (p, μ) there is a time-elapse transition $(p, \mu) \xrightarrow{\delta} (p, \mu + \delta)$. The timed language ε*-recognised* by $\mathcal{A}$, denoted $L_\varepsilon(\mathcal{A})$, is the set of finite timed words $w = (a_1, t_1) \cdots (a_n, t_n) \in (\Sigma_\varepsilon \cdot \mathbb{R}_{\geq 0})^*$ s.t. there is a sequence of transitions $(p_0, \mu_0) \xrightarrow{\mathsf{tr}_1, \delta_1} \cdots \xrightarrow{\mathsf{tr}_n, \delta_n} (p_n, \mu_n)$ where $p_0 \in I$ is initial, $\mu_0(\mathtt{x}) = 0$ for every clock $\mathtt{x} \in \mathtt{X}$, $p_n \in F$ is final, and, for every $1 \leq i \leq n$, $\delta_i = t_i - t_{i-1}$ (where $t_0 = 0$) and tr_i is of the form $(p_{i-1}, a_i, \_, \_, p_i)$. The timed ω-language $L_\varepsilon^\omega(\mathcal{A}) \subseteq (\Sigma_\varepsilon \cdot \mathbb{R}_{\geq 0})^\omega$ is defined in terms of sequences as above with the condition that $p_i \in F$ infinitely often. We obtain the timed language $L(\mathcal{A}) = \pi(L_\varepsilon(\mathcal{A})) \subseteq (\Sigma \cdot \mathbb{R}_{\geq 0})^*$, resp., ω-language $L^\omega(\mathcal{A}) = \pi(L_\varepsilon^\omega(\mathcal{A})) \cap (\Sigma \cdot \mathbb{R}_{\geq 0})^\omega \subseteq (\Sigma \cdot \mathbb{R}_{\geq 0})^\omega$ *recognised* by $\mathcal{A}$, where π is the mapping that removes letters of the form $(\varepsilon, \_)$.

A timed automaton (without ε-transitions) is *deterministic* if it has exactly one initial location and, for every two rules $(p, a, \varphi, \mathtt{Y}, q)$, $(p, a, \varphi', \mathtt{Y}', q')$ with $[\![\varphi \wedge \varphi']\!] \neq \emptyset$, we have $\mathtt{Y} = \mathtt{Y}'$ and $q = q'$. We write NTA, DTA for the classes of nondeterministic, resp., deterministic timed automata without epsilon transitions. When the number of clocks in $\mathtt{X}$ is bounded by k we write k-NTA, resp., k-DTA. When the absolute value of the maximal constant is additionally bounded by $m \in \mathbb{N}$ we write k, m-NTA, resp., k, m-DTA. When epsilon transitions are allowed, we write NTA$^\varepsilon$. A timed language is called NTA language, DTA language, and so on, if it is recognized by a timed automaton in the respective class. A k, m-DTA with clocks $\mathtt{X}$ is *regionised* if each constraint is a characteristic constraint $\varphi_{\mathtt{r}}$ of some region $\mathtt{r} \in \mathsf{Reg}(\mathtt{X}, m)$ and for each location p, input $a \in \Sigma$, and $r \in \mathsf{Reg}(\mathtt{X}, m)$ there is a (necessarily unique) transition rule of the form $(p, a, \varphi_{\mathtt{r}}, \mathtt{Y}, q)$. It is well-known that a k, m-DTA can be transformed into an equivalent regionised one by adding exponentially many transitions.

▶ **Example 2.1** (NTA language which is not a DTA language)**.** Let $\Sigma = \{a\}$ be a unary alphabet and let L be the set of timed words of the form $(a, t_1) \cdots (a, t_n)$ s.t. $t_n - t_i = 1$ for some $1 \leq i < n$. $L = L(A)$ for the timed automaton $\mathcal{A} = (\Sigma, \mathtt{L}, \mathtt{X}, \mathtt{I}, \mathtt{F}, \Delta)$ with a single clock $\mathtt{X} = \{\mathtt{x}\}$ three locations $\mathtt{L} = \{p, q, r\}$, of which $\mathtt{I} = \{p\}$ is initial and $\mathtt{F} = \{r\}$ is final, and transitions rules $(p, a, \mathbf{true}, \emptyset, p)$, $(p, a, \mathbf{true}, \{\mathtt{x}\}, q)$, $(q, a, \mathtt{x} < 1, \emptyset, q)$, $(q, a, \mathtt{x} = 1, \emptyset, r) \in \Delta$. Intuitively, in p the automaton waits until it guesses that the next input will be (a, t_i), at which point it moves to q by resetting the clock (and subsequently reading a). From q, the automaton can accept by going to r only if exactly one time unit elapsed since (a, t_i). There is no DTA recognising L, since in order to recognise L deterministically one must store all timestamps in the last unit interval, and thus no bounded number of clocks suffices.

▶ **Example 2.2.** The complement of L from Example 2.1 can be recognised by an NTA with two clocks. Indeed, a timed word $(a, t_1) \cdots (a, t_n)$ is not in L if either of the following conditions hold:

1) its length n is at most 1, or

2) the total time elapsed between the first and the last letter is less than one time unit $t_n - t_1 < 1$, or

3) there is a position $1 \leq i < n$ s.t. $t_n - t_i > 1$ and $t_n - t_{i+1} < 1$.

It is easy to see that two clocks suffice to nondeterministically check the conditions above.

Since checking whether an NTA recognises a deterministic language is undecidable [24, 48], there is no recursive bound on the number of clocks sufficient to deterministically recognise an NTA language (whenever possible). Thus NTA can be non-recursively more succinct than DTA w.r.t. number of clocks. However, in general such NTA recognise timed languages whose complement is not an NTA language. The next example shows a timed language which is both NTA and co-NTA recognisable, however the number of clocks of an equivalent DTA is at least exponential in the number of clocks of the NTA.

▶ **Example 2.3.** For $k \in \mathbb{N}$, let L_k be the set of strictly monotonic timed words $(a, t_1) \cdots (a, t_n)$ s.t. $t_n - t_i = 1$ where $i = n - 2^k$. The language L_k can be recognised by a $(2 \cdot k + 2)$-clock NTA $\mathcal{A}_k$ of polynomial size. There are clocks $\mathtt{x}_0, \mathtt{x}_1, \ldots, \mathtt{x}_k$ and $\mathtt{y}_0, \mathtt{y}_1, \ldots, \mathtt{y}_k$. Clock $\mathtt{x}_0$ is used to check strict monotonicity. Clock $\mathtt{y}_0$ is reset when the automaton guesses (a, t_i). The automaton additionally keeps track of the length of the remaining input. This is achieved by implementing a k-bit binary counter, where $\mathtt{x}_j = \mathtt{y}_j$ represents that the j-th bit is one. In order to set the j-th bit to one, the automaton resets $\mathtt{x}_j, \mathtt{y}_j$; to set it to zero, it resets only $\mathtt{x}_j$. This is correct thanks to strict monotonicity. At the end the automaton checks $\mathtt{y}_0 = 1$ and that the binary counter has value 2^k. Any deterministic automaton recognising

L_k requires exponentially many clocks to store the last 2^k timestamps. The complement of L_k can be recognised by a $(2 \cdot k + 2)$-clock NTA of polynomial size. Indeed, a timed word is not in L_k if any of the following conditions hold:
1) its length n is $\leq 2^k$, or
2) $t_n - t_i < 1$ with $i = n - 2^k$, or
3) $t_n - t_i > 1$ with $i = n - 2^k$.
The automaton guesses which condition holds and uses a k-bit binary counter as above to check that position i has been guessed correctly.

3 Timed synthesis games

Let A and B be two finite alphabets of actions and let $W \subseteq (A \cdot B \cdot \mathbb{R}_{\geq 0})^\omega$ be a language of timed ω-words over the alphabet $A \cdot B$. The *timed synthesis game* $G_{A,B}(W)$ is played by Player I and Player II in rounds. At round $i \geq 0$, Player I chooses a timed action $a_i \cdot t_i \in A \cdot \mathbb{R}_{\geq 0}$ and Player II replies immediately with an untimed action $b_i \in B$. The game is played for ω rounds, and at doomsday the two players have produced an infinite play

$$\pi = a_1 b_1 t_1 a_2 b_2 t_2 \cdots \in (A \cdot B \cdot \mathbb{R}_{\geq 0})^\omega. \tag{1}$$

Player I wins the game if, and only if, $\pi \in W$.

Let $k \in \mathbb{N}$ be a bound on the number of available clocks $\mathtt{X} = \{\mathtt{x}_1, \ldots, \mathtt{x}_k\}$, and let $m \in \mathbb{N}$ be a bound on the maximal constant. A *k, m-controller* for Player II in $G_{A,B}(W)$ is a regionised k, m-DTA $\mathcal{M} = (A, B, \mathtt{L}, \ell_0, \delta)$ with input alphabet A and output alphabet B, where $\mathtt{L}$ is a set of memory locations, $\ell_0 \in \mathtt{L}$ is the initial memory location, and $\delta : \mathtt{L} \cdot A \cdot \mathsf{Reg}(k, m) \to \mathtt{L} \cdot B \cdot 2^{\mathtt{X}}$ is the update function mapping the current memory $\ell \in \mathtt{L}$, input $a \in A$, and region $\varphi \in \mathsf{Reg}(k, m)$ to $\delta(\ell, a, \varphi) = (\ell', b, \mathtt{Y})$, where $\ell' \in \mathtt{L}$ is the next memory location, $b \in B$ is an output symbol, and $\mathtt{Y} \subseteq \mathtt{X}$ is the set of clocks to be reset.

We define by mutual induction the notion of *$\mathcal{M}$-conform partial runs* $\mathsf{Run}(\mathcal{M}) \subseteq \mathtt{L} \cdot \mathbb{R}^{\mathtt{X}}_{\geq 0} \cdot (A \cdot B \cdot \mathbb{R}_{\geq 0} \cdot \mathtt{L} \cdot \mathbb{R}^{\mathtt{X}}_{\geq 0})^*$, and the strategy $[\![\mathcal{M}]\!] : \mathsf{Run}(\mathcal{M}) \cdot A \cdot \mathbb{R}_{\geq 0} \to \mathtt{L} \cdot \mathbb{R}^{\mathtt{X}}_{\geq 0} \cdot B$ induced by the controller on conform runs as follows: Initially, $(\ell_0, \mu_0) \in \mathsf{Run}(\mathcal{M})$, where $\mu_0(\mathtt{x}) = 0$ for every clock $\mathtt{x} \in \mathtt{X}$. Inductively, for every $n \geq 0$ and every $\mathcal{M}$-conform partial run

$$\rho = (\ell_0, \mu_0)\,(a_1, b_1, t_1, \ell_1, \mu_1) \cdots (a_n, b_n, t_n, \ell_n, \mu_n) \in \mathsf{Run}(\mathcal{M}), \tag{2}$$

and for every $(a_{n+1}, t_{n+1}) \in A \cdot \mathbb{R}_{\geq 0}$, we define $[\![\mathcal{M}]\!]\,(\rho \cdot a_{n+1} \cdot t_{n+1}) = (\ell_{n+1}, \mu_{n+1}, b_{n+1})$ for the unique $(\ell_{n+1}, \mu_{n+1}, b_{n+1}) \in \mathtt{L} \cdot \mathbb{R}^{\mathtt{X}}_{\geq 0} \cdot B$ s.t. $\delta(\ell_n, a_{n+1}, \varphi_{\mu_n + \delta_{n+1}}) = (\ell_{n+1}, b_{n+1}, \mathtt{Y})$ and $\mu_{n+1} = (\mu_n + \delta_{n+1})[\mathtt{Y} \mapsto 0]$, where $\delta_{n+1} = t_{n+1} - t_n$ (with $t_0 = 0$). Moreover, $\rho \cdot a_{n+1} \cdot b_{n+1} \cdot t_{n+1} \cdot \ell_{n+1} \cdot \mu_{n+1} \in \mathsf{Run}(\mathcal{M})$. An infinite $\mathcal{M}$-conform run is any sequence $\rho \in \mathtt{L} \cdot \mathbb{R}^{\mathtt{X}}_{\geq 0} \cdot (A \cdot B \cdot \mathbb{R}_{\geq 0} \cdot \mathtt{L} \cdot \mathbb{R}^{\mathtt{X}}_{\geq 0})^\omega$ such that every finite prefix thereof is $\mathcal{M}$-conform; let $\mathsf{Run}_\omega(\mathcal{M})$ be the set of such ρ's. Let $\mathsf{r2p}(\rho) \in (A \cdot B \cdot \mathbb{R}_{\geq 0})^\omega$ be the corresponding play $\pi = \mathsf{r2p}(\rho)$ as in (1) obtained by dropping locations and clocks valuations. The controller $\mathcal{M}$ is *winning* if every infinite $\mathcal{M}$-conform run ρ satisfies $\mathsf{r2p}(\rho) \notin W$. A *$k$-controller* is k, m-controller for some $m \in \mathbb{N}$. For fixed $k, m \in \mathbb{N}$, the *k, m-timed synthesis problem* asks, given A, B and an NTA$^\varepsilon$ timed language $W \subseteq (A \cdot B \cdot \mathbb{R}_{\geq 0})^\omega$, whether Player II has a winning k, m-controller in $G_{A,B}(W)$; the *k-timed synthesis problem* asks instead for a k-controller; finally, the *timed synthesis problem* asks whether there exists a controller. The $0, 0$-timed synthesis problem is equivalent to untimed synthesis problem, which is decidable by the Büchi-Landweber Theorem [9, Theorem 1′]:

▶ **Lemma 3.1** (cf. [13, Appendix A]). *The $0, 0$-synthesis problem is decidable.*

4 Deterministic separability

In this section we prove our first main result Theorem 1.1: we show that the k, m and k-deterministic separability problems are decidable. We begin with a motivating example of nonseparable languages.

▶ **Example 4.1.** Consider the NTA language L from Example 2.1. Thanks to Example 2.2 its complement is also a NTA language. Since neither L nor its complement are deterministic, they cannot be deterministically separable.

Moreover, a deterministic separator, when it exists, may need exponentially many clocks.

▶ **Example 4.2.** We have seen in Example 2.3 an $O(k)$-clock NTA language s.t. 1) its complement is also an $O(k)$-clock NTA language, and 2) any DTA recognising it requires 2^k clocks. Thus, a deterministic separator may need exponentially many clocks in the size of the input NTA.

In the rest of the section we show how to decide the separability problems. We reduce the k, m-deterministic separability to k, m-timed synthesis, and k-deterministic separability to k-timed synthesis, for every fixed $k, m \in \mathbb{N}$. Let $\mathcal{A}, \mathcal{B}$ be two NTA$^\varepsilon$ over alphabet Σ, and let $\mathtt{X}$ be a set of k clocks. We build a timed synthesis game where the two sets of actions are

$$A = \Sigma \quad \text{(Player I)}, \qquad B = \{\mathsf{acc}, \mathsf{rej}\} \quad \text{(Player II)}.$$

We define a projection function $\mathsf{proj}(a, b, t) = (a, t)$, which is extended pointwise to finite and infinite timed words $\mathsf{proj}((a_0, b_0, t_0)\,(a_1, b_1, t_1) \cdots) = (a_0, t_0)\,(a_1, t_1) \cdots$ and timed languages $\mathsf{proj}(L) = \{\mathsf{proj}(w) \mid w \in L \subseteq (A \cdot B \cdot \mathbb{R}_{\geq 0})^\omega\}$. Let $\mathsf{Acc}, \mathsf{Rej} \subseteq (A \cdot B \cdot \mathbb{R}_{\geq 0})^*$ be sets of those timed words ending in a timed letter of the form $(\_, \mathsf{acc}, \_)$, resp., $(\_, \mathsf{rej}, \_)$. The winning condition for Player I is

$$W_0 = \big(\mathsf{proj}^{-1}(L(\mathcal{A})) \cap \mathsf{Rej} \;\cup\; \mathsf{proj}^{-1}(L(\mathcal{B})) \cap \mathsf{Acc}\big) \cdot (A \cdot B \cdot \mathbb{R}_{\geq 0})^\omega. \tag{3}$$

Crucially, we observe that W_0 is a NTA$^\varepsilon$ language since $L(\mathcal{A}), L(\mathcal{B}), \mathsf{Rej}, \mathsf{Acc}$ are so, and this class is closed under inverse homomorphic images, intersections, and unions. The following lemma states the correctness of the reduction.

▶ **Lemma 4.3.** *There is a k, m-controller for Player I in $G_{A,B}(W_0)$ if, and only if, $L(\mathcal{A}), L(\mathcal{B})$ are k, m-deterministically separable.*

Proof. Let $\mathcal{M} = (A, B, \mathtt{L}, \ell_0, \delta)$ be a winning k, m-controller for Player II in $G = G_{A,B}(W_0)$. Let $\mathtt{X} = \{\mathtt{x}_1, \ldots, \mathtt{x}_k\}$ be clocks of $\mathcal{M}$. We construct a separator $\mathcal{S} = (\Sigma, \mathtt{L} \times B, \mathtt{X}, \mathtt{I}, \mathtt{F}, \Delta) \in k, m$-DTA, where $\mathtt{I} = \{(\ell_0, \mathsf{acc})\}$ if $\varepsilon \in L(\mathcal{A})$ and $\mathtt{I} = \{(\ell_0, \mathsf{rej})\}$ otherwise, $\mathtt{F} = \mathtt{L} \times \{\mathsf{acc}\}$, and

$$((\ell, b)\,, a, \varphi, \mathtt{Y}, (\ell', b')) \in \Delta \quad \text{if, and only if,} \quad \delta(\ell, a, \varphi) = (\ell', b', \mathtt{Y})\,. \tag{4}$$

We show that $L(\mathcal{S})$ separates $L(\mathcal{A}), L(\mathcal{B})$ using the fact that $\mathcal{S}$ is deterministic. In order to show $L(\mathcal{A}) \subseteq L(\mathcal{S})$, let $w = (a_1, t_1) \cdots (a_n, t_n) \in L(\mathcal{A})$ and let Player I play this timed word in G. Let the corresponding $\mathcal{M}$-conform partial play be $\pi = (a_1, b_1, t_1) \cdots (a_n, b_n, t_n)$. Since $\mathcal{M}$ is winning, π does not extend to an infinite word in W_0, and in particular $\pi \notin \mathsf{proj}^{-1}(L(\mathcal{A})) \cap \mathsf{Rej}$. But $\mathsf{proj}(\pi) = w \in L(\mathcal{A})$ by assumption, and thus $b_n = \mathsf{acc}$. The unique run of $\mathcal{S}$ on w ends up in an accepting control location of the form $(\_, b_n)$, and thus $w \in L(\mathcal{S})$, as required. The argument showing that $L(\mathcal{S}) \cap L(\mathcal{B}) = \emptyset$ is similar, using the fact that $\mathcal{S}$ is deterministic and must reach $b_n = \mathsf{rej}$ and thus reject all words $(a_1, t_1) \cdots (a_n, t_n) \in L(\mathcal{B})$.

For the other direction, let $\mathcal{S} = (\Sigma, \mathtt{L}, \mathtt{X}, \{\ell_0\}, \mathtt{F}, \Delta) \in k,m\text{-DTA}$ be a deterministic separator. We construct a winning k,m-controller for Player II in G of the form $\mathcal{M} = (A, B, \mathtt{L}, \ell_0, \delta)$ where $\delta(\ell, a, \varphi) = (\ell', b, \mathtt{Y})$ for the unique $\mathtt{Y}, \ell', b$ s.t. $(\ell, a, \varphi, \mathtt{Y}, \ell') \in \Delta$ and $b = \mathsf{acc}$ iff $\ell' \in \mathtt{F}$. In order to argue that $\mathcal{M}$ is winning in G, let $\pi = (a_1, b_1, t_1)\,(a_2, b_2, t_2) \cdots \in (A \cdot B \cdot \mathbb{R}_{\geq 0})^\omega$ be an $\mathcal{M}$-conform play. By construction of $\mathcal{M}$ we have:

▷ Claim 4.4. For every finite nonempty prefix $\pi' = (a_1, b_1, t_1) \cdots (a_n, b_n, t_n)$ of π, $\mathsf{proj}(\pi') \in L(\mathcal{S})$ if, and only if $b_n = \mathsf{acc}$.

Knowing that $L(\mathcal{A}) \subseteq \mathcal{S}$, we deduce that no prefix of π belongs to $\mathsf{proj}^{-1}(L(\mathcal{A})) \cap \mathsf{Rej}$. Similarly, knowing that $L(\mathcal{S}) \cap L(\mathcal{B}) = \emptyset$, we deduce that no prefix of π belongs to $\mathsf{proj}^{-1}(L(\mathcal{B})) \cap \mathsf{Acc}$. Thus $\pi \notin W_0$ and therefore $\mathcal{M}$ is winning. ◀

Proof of Theorem 1.1. Lemma 4.3 provides a reduction from the k,m-deterministic separability problem to the k,m-timed synthesis problem. The latter problem is decidable by Theorem 1.2. Since the construction in Lemma 4.3 is independent of m, it provides also a reduction from the k-deterministic separability problem to the k-timed synthesis problem. The latter problem is decidable by Theorem 1.3. ◀

5 Solving the timed synthesis problems

The second main result of this paper is decidability of the k,m-timed synthesis problem and of the k-synthesis problem, i.e., when the maximal constant m is not specified in advance (Theorems 1.2 and 1.3). This will be achieved in four steps. In the first two steps (see [13, Appendices B.1 and B.2]) we make certain easy simplifying assumptions that winning conditions W are strictly monotonic, and *zero-starting*: all words $(a_1, t_1)\,(a_2, t_2) \cdots \in W$ satisfy $t_1 = 0$. The main technical construction is in Section 5.1, where we prove Theorem 1.2 in such a way that we will easily obtain Theorem 1.3 as a corollary thereof in Section 5.2.

The decidability results of this section are tight, since timed synthesis is undecidable when k is not fixed (cf. Theorem 6.1).

5.1 Solving the k,m-timed synthesis problem

In this section we prove Theorem 1.2 by reducing the k,m-timed synthesis problem to a $0,0$-timed synthesis problem, which is decidable by Lemma 3.1. This is the most technically involved section. The structure of the reduction will be useful in Section 5.2 to show decidability of the k-timed synthesis problem.

Let $\mathtt{X}$ be a fixed set of clocks of size $|\mathtt{X}| = k$ and let $m \in \mathbb{N}$ be a fixed bound on constants. We reduce the k,m-synthesis problem to the $0,0$-synthesis problem by designing a protocol in which Player II, to compensate his inability to measure time elapse, can *request* certain clocks to be *tracked*. In addition, we design the Player I's winning condition that obliges her to remind whenever the value of any tracked clock is an integer, by submitting *expiry* information one time unit after a corresponding request.

Let $\mathsf{fract}(\mathtt{x})$ stand for the fractional part of the value of a clock $\mathtt{x}$. For $\mathtt{Y}_1, \mathtt{Y}_2 \subseteq \mathtt{X}$, two (partial) clock valuations $\mu \in \mathbb{R}_{\geq 0}^{\mathtt{Y}_1}, \nu \in \mathbb{R}_{\geq 0}^{\mathtt{Y}_2}$ are *fractional region equivalent* if $\mathtt{Y}_1 = \mathtt{Y}_2$ and they exhibit the same relations between fractional parts of clocks: $\mu \models \mathsf{fract}(\mathtt{x}) < \mathsf{fract}(\mathtt{x}')$ iff $\nu \models \mathsf{fract}(\mathtt{x}) < \mathsf{fract}(\mathtt{x}')$ and $\mu \models \mathsf{fract}(\mathtt{x}) = 0$ iff $\nu \models \mathsf{fract}(\mathtt{x}) = 0$, for all $\mathtt{x}, \mathtt{x}' \in \mathtt{Y}_1$. By a (partial) fractional $\mathtt{X}$-region $\mathtt{f}$ we mean an equivalence class of this equivalence relation. All elements $\mu \in \mathbb{R}_{\geq 0}^{\mathtt{Y}}$ in $\mathtt{f}$ have the same domain $\mathtt{Y}$, which we denote by $\mathsf{dom}(\mathtt{f}) = \mathtt{Y}$. Let

$\mathbf{0}(\mathtt{f}) = \{\mathtt{x} \in \mathsf{dom}(\mathtt{f}) \mid \mathtt{f} \models \mathsf{fract}(\mathtt{x}) = 0\}$. Let $\mathsf{FReg}(\mathtt{X})$ be the set of all fractional $\mathtt{X}$-regions, including the empty one $\mathtt{f}_0$ with $\mathsf{dom}(\mathtt{f}_0) = \emptyset$. For $\mathtt{r} \in \mathsf{Reg}(\mathtt{X}, m)$ and $\mathtt{f} \in \mathsf{FReg}(\mathtt{X})$, we say that $\mathtt{f}$ *agrees* with $\mathtt{r}$ if they give the same answer for clocks $\mathtt{x}, \mathtt{y} \in \mathsf{dom}(\mathtt{f})$:

- $\mathtt{f} \models \mathsf{fract}(\mathtt{x}) < \mathsf{fract}(\mathtt{y})$ if, and only if, $\mathtt{r} \models \mathsf{fract}(\mathtt{x}) < \mathsf{fract}(\mathtt{y}) \vee \mathtt{x} > m \vee \mathtt{y} > m$;
- $\mathtt{f} \models \mathsf{fract}(\mathtt{x}) = 0$ if, and only if, $\mathtt{r} \models \mathsf{fract}(\mathtt{x}) = 0 \vee \mathtt{x} > m$.

The successor relation between regions induces a corresponding relation between fractional regions: $\mathtt{f} \preceq \mathtt{f}'$ whenever $\mathsf{dom}(\mathtt{f}) = \mathsf{dom}(\mathtt{f}')$, $\mathtt{f}$ agrees with some $\mathtt{r}$, $\mathtt{f}'$ agrees with some $\mathtt{r}'$, and $\mathtt{r} \preceq \mathtt{r}'$. The immediate successor is the minimal $\mathtt{f}'$ with $\mathtt{f} \prec \mathtt{f}'$. Finally, the *successor region of* $\mathtt{r}$ *agreeing with* $\mathtt{f}$ is $\textsc{succ}_{\mathtt{X},m}(\mathtt{r}, \mathtt{f}) = \min_{\preceq} \{\mathtt{r}' \succeq \mathtt{r} \mid \mathtt{f} \text{ agrees with } \mathtt{r}'\}$. In the sequel we apply clock resets also to regions $\mathtt{r}[\mathtt{Y} \mapsto 0]$ and fractional regions.

Let the original game $G = G_{A,B}(W)$ have action alphabets A, B and Player I's winning condition $W \subseteq (A \cdot B \cdot \mathbb{R}_{\geq 0})^\omega$. Thanks to [13, Appendices B.1 and B.2] we assume that W is both strictly monotonic and zero starting. We design a new game $G' = G_{A',B'}(W'_{k,m})$ as follows. We take as the new action alphabets the sets

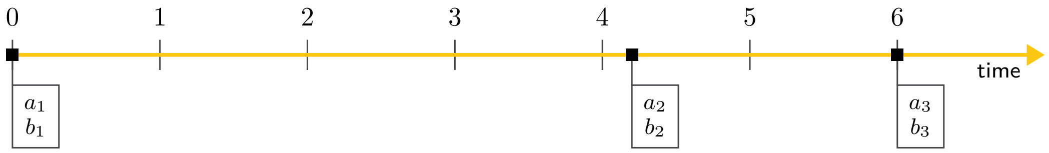
$$A' = (A \cup \{\square\}) \cdot \mathsf{FReg}(\mathtt{X}) \quad \text{and} \quad B' = (B \cup \{\square\}) \cdot 2^{\mathtt{X}}. \tag{5}$$

The players' action sets A', B' depend only on the set of clocks $\mathtt{X}$ and do not depend on the maximal constant m. Moves of the form $(\square, \_)$ are *improper* and the other ones (i.e., those involving an A or B component) are *proper*. Let an infinite play be of the form

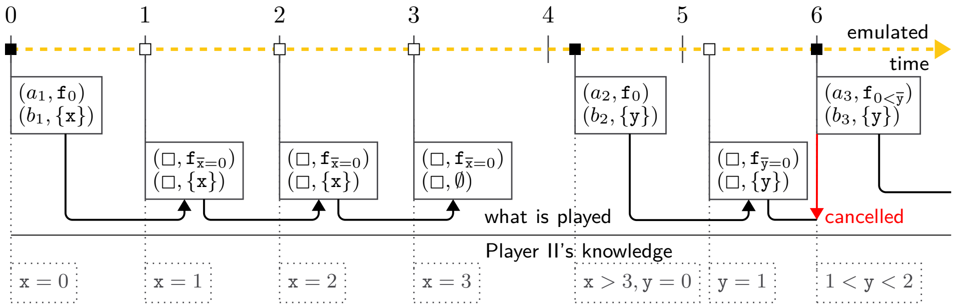
$$\pi = (a'_1, b'_1, t_1)\,(a'_2, b'_2, t_2) \cdots \in (A' \cdot B' \cdot \mathbb{R}_{\geq 0})^\omega, \text{ with } a'_i = (a_i, \mathtt{f}_i) \text{ and } b'_i = (b_i, \mathtt{Y}_i)\,. \tag{6}$$

The domain $\mathtt{T}_i = \mathsf{dom}(\mathtt{f}_i)$ of a fractional region denotes the clocks *tracked* at time t_i, i.e., those for which Player I needs to provide expiry information. Sets $\mathtt{Y}_i$'s denote clocks which Player II wants to be continued to be tracked: by an $\mathtt{x}$-*request at time* t_i we mean a Player II's move b'_i with $\mathtt{x} \in \mathtt{Y}_i$. An $\mathtt{x}$-request at time t_i is *cancelled* if there is another $\mathtt{x}$-request for the same clock at some time $t_i < u < t_i + 1$. An *improper* $\mathtt{x}$-*request chain* starting at time t_i of length $l \geq 1$ is a sequence of improper non-cancelled $\mathtt{x}$-requests at times t_i, $t_i + 1$, $\ldots$, $t_i + l - 2$, followed by an improper (but possibly cancelled) $\mathtt{x}$-request at time $t_i + l - 1$. Likewise one defines an infinite improper $\mathtt{x}$-request chain starting at time t_i.

▶ **Example 5.1.** Before defining the winning set $W'_{k,m}$ formally, we illustrate the underlying idea. Consider the following partial play $(a_1, b_1, 0)\,(a_2, b_2, 4.2)\,(a_3, b_3, 6) \in (A \cdot B \cdot \mathbb{R}_{\geq 0})^*$ in G:



In G', Player II demands Player I to provide clock expiry information. Let $\mathtt{X} = \{\mathtt{x}, \mathtt{y}\}$ and $m = 3$. Suppose Player II wants to make sure that a_2 comes at time > 3. To this end, she makes an $\mathtt{x}$-request chain of length 3 (we write $\overline{\mathtt{x}}$ instead of $\mathsf{fract}(\mathtt{x})$; $\mathtt{f}_\phi$ denotes the fractional $\mathtt{X}$-region agreeing with ϕ):



The length of an $\mathtt{x}$-chain at any given moment corresponds to the integral part of $\mathtt{x}$; the expiry information for $\mathtt{x}$ is provided by Player I precisely when the fractional part of $\mathtt{x}$ is 0.

In order to define $W'_{k,m}$ it will be convenient to have the following additional data extracted from π. Let $\delta_i = t_i - t_{i-1} \geq 0$ be the time elapsed by Player I at round i (with $t_0 = 0$). Furthermore, let $\nu_0 = \lambda \mathtt{x} \cdot 0$ be the initial clock valuation, and, for $i \geq 0$, let

$$\nu_{i+1} = (\nu_i + \delta_{i+1})[\mathtt{Y}_{i+1} \mapsto 0]. \tag{7}$$

In words, every $\mathtt{x}$-request is interpreted as reset of clock $\mathtt{x}$. The winning condition $W'_{k,m}$ in the new game will impose, in addition to W, the following further conditions to be satisfied by Player I in order to win. Let $W^{\mathrm{I}}_k \subseteq (A' \cdot B' \cdot \mathbb{R}_{\geq 0})^\omega$ be the set of plays π as in (6) which are zero-starting ($t_1 = 0$), strictly monotonic and, for every $i \geq 1$:

1. For every $\mathtt{x} \in \mathtt{X}$, $\mathtt{x}$ is expired at time t_i if, and only if, $t_i \geq 1$ and there is a non-cancelled $\mathtt{x}$-request at an earlier time $t_j = t_i - 1$.
2. Tracked clocks are consistent with requests: for every clock $\mathtt{x} \in \mathtt{X}$, $\mathtt{x}$ is tracked $\mathtt{x} \in \mathtt{T}_i$ at time t_i if, and only if, there is an $\mathtt{x}$-request at an earlier t_j with $t_i - 1 \leq t_j < t_i$.
3. The fractional regions are correct: $\mathtt{f}_i$ agrees with $[(\nu_{i-1} + \delta_i)]_{\mathtt{X},m}$.

Thus the conditions above assure that Player I provides exactly all expiry information requested by Player II in a timely manner, and the fractional regions $\mathtt{f}_i$ are consistent with the requests and time elapse. Note that any play in W^{I}_k satisfies $0 < \nu_i(\mathtt{x}) \leq 1$ for every $i \geq 1$ and $\mathtt{x} \in \mathtt{T}_i$. Indeed, positivity is due to strict monotonicity, and the upper bound due to the conditions 1–3. Provided Player I satisfies W^{I}_k, she wins whenever Player II violates any of the conditions below: Let $W^{\mathrm{II}}_{k,m} \subseteq (A' \cdot B' \cdot \mathbb{R}_{\geq 0})^\omega$ be the set of plays π as in (6) s.t.

4. Player II plays a proper move iff Player I does so.
5. Every improper Player II's $\mathtt{x}$-request b'_i is a response to Player I's expiry information for $\mathtt{x}$: $\mathtt{Y}_i \subseteq \mathbf{0}(\mathtt{f}_i)$. (Proper $\mathtt{x}$-requests are allowed unconditionally.)
6. For every clock $\mathtt{x} \in \mathtt{X}$, the length Player II's improper $\mathtt{x}$-request chains is $< m$. This is the only component in the winning condition which depends on m.

Consider the projection function $\phi : (A' \cdot B' \cdot \mathbb{R}_{\geq 0}) \to (A \cdot B \cdot \mathbb{R}_{\geq 0}) \cup \{\varepsilon\}$ s.t. $\phi((a, \_), (b, \_), t) = \varepsilon$ if $a = \Box$ or $b = \Box$, and $\phi((a, \_), (b, \_), t) = (a, b, t)$ if $a \in A$ and $b \in B$, which is extended homomorphically on finite and infinite plays. The winning condition for Player I in G' is

$$W'_{k,m} \ = \ W^{\mathrm{I}}_k \cap \left(\phi^{-1}(W) \cup (A' \cdot B' \cdot \mathbb{R}_{\geq 0})^\omega \setminus W^{\mathrm{II}}_{k,m}\right). \tag{8}$$

Since W, W^{I}_k, are NTA^ε languages, and W^{I}_k and $W^{\mathrm{II}}_{k,m}$ are k-DTA languages over $A' \cdot B'$, thanks to the closure properties DTA and NTA^ε languages the winning condition $W'_{k,m}$ is an NTA^ε language. In what follows, an *untimed controller* is a $0, 0$-controller. Then next two lemmas state the correctness of the reduction. Our assumption on strict monotonicity facilitates the correctness proof since we need not deal with simultaneous events.

▶ **Lemma 5.2.** *If there is a winning k, m-controller $\mathcal{M}$ for G, then there is a winning untimed controller $\mathcal{M}'$ for G'.*

Proof. Let $\mathcal{M} = (A, B, \mathtt{L}, \ell_0, \delta)$ be a winning k, m-controller $\mathcal{M}$ for G with clocks $\mathtt{X} = \{\mathtt{x}_1, \ldots, \mathtt{x}_k\}$ and update function $\delta : \mathtt{L} \cdot A \cdot \mathsf{Reg}(\mathtt{X}, m) \to \mathtt{L} \cdot B \cdot 2^{\mathtt{X}}$. We define a winning untimed controller $\mathcal{M}' = (A', B', \mathtt{L}', \rhd, \delta')$ for G' with memory locations $\mathtt{L}' = \{\rhd\} \cup \mathtt{L} \cdot \mathsf{Reg}(\mathtt{X}, m)$, where $\rhd$ is the initial memory location, and remaing memory locations are of the form $(\ell, \mathtt{r})$, where $\ell \in \mathtt{L}$ is the current memory location of $\mathcal{M}$ and $\mathtt{r} \in \mathsf{Reg}(\mathtt{X}, m)$ is the current region of $\mathcal{M}$'s clocks. The update function $\delta' : \mathtt{L}' \cdot A' \to \mathtt{L}' \cdot B'$ (we omit regions and clock resets because $\mathcal{M}'$ has no clocks) is defined as follows. As long as the play is in W^{I}_k, we can assume that Player I starts with $((a, \mathtt{f}_0), t)$ and $t = 0$, due to the zero-starting restriction, which allows Player II to submit requests at time 0. Consequently, let $\delta'(\rhd, (a, \mathtt{f})) = ((\ell', \mathtt{r}_0), (b, \mathtt{X}))$, where the next location ℓ' and the response b are determined by $\delta(\ell_0, a, \mathtt{r}_0) = (\ell', b)$, and the set $\mathtt{X}$ denotes a request to track all clocks. Then, for every $\ell, \mathtt{r}, a, \mathtt{f}$, let

$$\delta'((\ell, \mathtt{r}), (a, \mathtt{f})) = ((\ell', \mathtt{r}'), (b, \mathtt{Y})), \tag{9}$$

where the r.h.s. is defined as follows. Let $\mathtt{T} = \mathsf{dom}(\mathtt{f})$ be the currently tracked clocks, and $\mathtt{T}_0 = \mathbf{0}(\mathtt{f}) \subseteq \mathtt{T}$ the currently expired ones. If $\mathtt{f}$ agrees with no successor region of $\mathtt{r}$ then Player II wins immediately because Player I is violating condition 3. Therefore, assume such a successor region $\hat{\mathtt{r}} = \textsc{succ}_{\mathtt{X},m}(\mathtt{r}, \mathtt{f})$ exists. We do a case analysis based on whether Player I plays a proper or an improper move.

- Case $a \in A$ (proper move): Let $\delta(\ell, a, \hat{\mathtt{r}}) = (\ell', b, \mathtt{Y})$ thus defining ℓ' and $(b, \mathtt{Y})$ in (9). Take as the new region $\mathtt{r}' = \hat{\mathtt{r}}[\mathtt{Y} \mapsto 0]$.
- Case $a = \square$ (improper move): Let the response be also improper $b = \square$, the control location does not change $\ell' = \ell$, the new clocks to be tracked are the expired clocks with a short improper chain $\mathtt{Y} = \{\mathtt{x} \in \mathtt{T}_0 \mid \hat{\mathtt{r}} \models \mathtt{x} = 1 \vee \cdots \vee \mathtt{x} = m - 1\}$, and $\mathtt{r}' = \hat{\mathtt{r}}$.

Consider an infinite $\mathcal{M}'$-conform run in G' (omitting clock valuations since $\mathcal{M}'$ has no clocks)

$$\rho' \;=\; \rhd (a'_1, b'_1, t_1, (\ell_1, \mathtt{r}_1))\, (a'_2, b'_2, t_2, (\ell_2, \mathtt{r}_2)) \cdots \in \mathsf{Run}_\omega(\mathcal{M}'), a'_i = (a_i, \mathtt{f}_i), b'_i = (b_i, \mathtt{Y}_i).$$

If the induced play $\pi' = \mathsf{r2p}(\rho') = (a'_1, b'_1, t_1)\,(a'_2, b'_2, t_2) \cdots \in (A' \cdot B' \cdot \mathbb{R}_{\geq 0})^\omega$ is not in W^{I}_k, then Player II wins and we are done. Assume $\pi' \in W^{\mathrm{I}}_k$, and thus conditions 1–3 are satisfied. We argue that $\pi' \in W^{\mathrm{II}}_{k,m}$. The conditions 4 and 5 hold by construction. Aiming at demonstrating that 6 holds too, let $\mu_0 = \lambda \mathtt{x} \cdot 0$, and, for $i \geq 0$, let

$$\mu_{i+1} = \begin{cases} \mu_i + \delta_{i+1} & a_i = \square \quad \text{(improper round)} \\ (\mu_i + \delta_{i+1})[\mathtt{Y}_i \mapsto 0] & a_i \in A \quad \text{(proper round).} \end{cases} \tag{10}$$

Thus clock valuations μ_i are defined exactly as ν_i in (7) except that only proper requests are interpreted as clock resets. We claim that the region information $\mathtt{r}_i$ is consistent with μ_i: $\mathtt{r}_i = [\mu_i]_{\mathtt{X},m}$ (*). Indeed, this is due to $\pi' \in W^{\mathrm{I}}_k$, and the fact that $\mathcal{M}'$ updates its stored region consistently with time elapse: at every round $\mathcal{M}'$ uses the successor region agreeing with the current fractional region submitted by Player I, and resets a set of clocks $\mathtt{Y}$ exactly when she plays a proper move of the form $(a, \mathtt{Y}) \in A \cdot 2^{\mathtt{X}}$. Since an $\mathtt{x}$-request is submitted by $\mathcal{M}'$ only when $\hat{r} \models x \leq m{-}1$, condition 6 holds.

In order to show that Player II is winning, consider an $\mathcal{M}'$-conform run ρ'. It suffices to show $\pi' = \mathsf{r2p}(\rho') \notin \phi^{-1}(W)$. Let the proper moves in ρ' be at indices $1 = i_1 < i_2 < \cdots$ ($i_1 = 1$ due to zero-starting). In particular, $\ell_{i_l} = \ell_i$ for $i_l \leq i < i_{l+1}$. Consider the run $\rho = (\ell_0, \mu_0)\,(a_{i_1}, b_{i_1}, t_{i_1}, (\ell_{i_1}, \mu_{i_1}))\,(a_{i_2}, b_{i_2}, t_{i_2}, (\ell_{i_2}, \mu_{i_2})) \cdots$. Using (*) and the definition of

$\mathcal{M}'$, one can prove by induction that ρ is an $\mathcal{M}$-conform run in G. Since $\mathcal{M}$ is winning, the induced play $\pi = \mathsf{r2p}(\rho) = (a_{i_1}, b_{i_1}, t_{i_1})\,(a_{i_2}, b_{i_2}, t_{i_2}) \cdots \in (A \cdot B \cdot \mathbb{R}_{\geq 0})^\omega$, satisfies $\pi \notin W$. Again by induction one can prove that $\pi = \phi(\pi')$. Hence $\phi(\pi') \notin W$ as required. ◀

▶ **Lemma 5.3** (cf. [13, Appendix B.3]). *If there is a winning untimed controller $\mathcal{M}'$ in G', then there is a winning k, m-controller $\mathcal{M}$ in G.*

5.2 Solving the k-timed synthesis problem

In this section we prove Theorem 1.3, stating that the k-timed synthesis problem is decidable, by reducing it to the $0, 0$-synthesis problem, which is decidable by Lemma 5.2. We build on the game defined in Section 5.1. Starting from a timed game $G = G_{A,B}(W)$ we define the timed game $G'' = G_{A',B'}(W''_k)$, where the sets of actions A' and B' are as in (5), and the winning condition W''_k is defined as follows. Let $W^{\mathrm{II}}_k \subseteq (A' \cdot B' \cdot \mathbb{R}_{\geq 0})^\omega$ be the set of plays where, for every clock $\mathtt{x} \in \mathtt{X}$, improper $\mathtt{x}$-request chains have finite lengths: $W^{\mathrm{II}}_k = \bigcup_{m \in \mathbb{N}} W^{\mathrm{II}}_{k,m}$. (In other words, $(A' \cdot B' \cdot \mathbb{R}_{\geq 0})^\omega \setminus W^{\mathrm{II}}_k$ contains plays with an infinite improper $\mathtt{x}$-request chain, for some clock $\mathtt{x} \in \mathtt{X}$.) Then, W''_k is defined as $W'_{k,m}$ from (8), except that $W^{\mathrm{II}}_{k,m}$ is replaced by the weaker condition W^{II}_k (notice W''_k does not depend on m):

$$W''_k \;=\; W^{\mathrm{I}}_k \cap \left(\phi^{-1}(W) \cup (A' \cdot B' \cdot \mathbb{R}_{\geq 0})^\omega \setminus W^{\mathrm{II}}_k\right). \tag{11}$$

▶ **Lemma 5.4.** *There is a winning untimed controller for G'' if, and only if, there is some $m \in \mathbb{N}$ and a winning untimed controller for $G' = G_{A',B'}(W'_{k,m})$.*

Proof. For the "if" direction, we observe that $W''_k \subseteq W'_{k,m}$, for every $m \in \mathbb{N}$. Hence every winning untimed controller for G' is also winning for G''. For the "only if" direction, let $\mathcal{M}'' = (A', B', \mathtt{L}, \ell_0, \delta)$ be an untimed winning controller in G''. Let $m = |A'| \cdot |\mathtt{L}| + 1$. We claim that $\mathcal{M}''$ is also winning in $G' = G_{A',B'}(W'_{k,m})$ for this choice of m. Towards reaching a contradiction, suppose $\mathcal{M}''$ is losing in G'. An $\mathcal{M}''$-conform run ρ in G' (or in G'') and its associated play π are of the form

$$\begin{aligned}\rho &= \ell_0\,(a'_1, b'_1, t_1, \ell_1)\,(a'_2, b'_2, t_2, \ell_2) \cdots \in \mathsf{Run}_\omega(\mathcal{M}''), \text{ with } a'_i = (a_i, \mathtt{f}_i) \text{ and } b'_i = (b_i, \mathtt{Y}_i),\\ \pi &= \mathsf{r2p}(\rho) = (a'_1, b'_1, t_1)\,(a'_2, b'_2, t_2) \cdots \in \mathsf{Play}(\mathcal{M}'').\end{aligned}$$

Let $\rho_i \in \mathsf{Run}(\mathcal{M}'')$ be the finite prefix of ρ ending at $(a'_i, b'_i, t_i, \ell_i)$. Since $\mathcal{M}''$ is losing in G', some $\mathcal{M}''$-conform play π above is in $W'_{k,m}$. Since $\mathcal{M}''$ is winning in G'', $\pi \notin \phi^{-1}(W)$, and thus $\pi \in W^{\mathrm{I}}_k \setminus W^{\mathrm{II}}_{k,m}$. This means that π contains an improper $\mathtt{x}$-request chain C of length m, for some clock $\mathtt{x} \in \mathtt{X}$. By the definition of m, there are indices $i < j$ s.t. the the same controller memory repeats together with Player I's action $(a'_i, \ell_i) = \left(a'_j, \ell_j\right)$. In particular $\mathtt{f}_i = \mathtt{f}_j$. Since $\mathcal{M}''$ is deterministic and its action depends only on Player I's action a'_i and control location ℓ_i, a posteriori we have $b'_i = b'_j$ as well. Moreover, as consecutive timestamps in C are equal to the first one plus consecutive nonnegative integers, $\Delta = t_i - t_j \in \{1, \ldots, m-1\}$. Consider the corresponding infix $\sigma = \left(a'_{i+1}, b'_{i+1}, t_{i+1}, \ell_{i+1}\right) \cdots \left(a'_j, b'_j, t_j, \ell_j\right)$ of the run ρ. Since $\pi \in W'_{k,m}$, thanks to conditions 2 and 3 the fractional regions $\mathtt{f}_i = \mathtt{f}_j$ contain all tracked clocks, and they agree with the clock valuations ν_i and ν_j, respectively, as defined in (7). Let $\{t_i - 1 \leq t_{i_1} < t_{i_2} < \cdots < t_{i_l} < t_i\} = \{t_i - \nu_i(\mathtt{x}) \mid \mathtt{x} \in \mathsf{dom}(\mathtt{f}_i)\}$ be the timestamps corresponding to the last request of the clocks tracked at time t_i, and likewise let $\left\{t_j - 1 \leq t_{j_1} < t_{j_2} < \cdots < t_{j_{l'}} < t_j\right\} = \{t_j - \nu_j(\mathtt{x}) \mid \mathtt{x} \in \mathsf{dom}(\mathtt{f}_j)\}$. By assumption, $\mathtt{f}_i = \mathtt{f}_j$, and hence $l = l'$ and for $\mathtt{x} \in \mathsf{dom}(\mathtt{f}_i) = \mathsf{dom}(\mathtt{f}_j)$ and $1 \leq h \leq l$, $t_{i_h} = t_i - \nu_i(\mathtt{x})$ if, and only if, $t_{j_h} = t_j - \nu_j(\mathtt{x})$ (*). Moreover, since $\mathbf{0}(\mathtt{f}_i) = \mathbf{0}(\mathtt{f}_j)$, we have $t_{i_1} = t_i - 1$ if, and only if, $t_{j_1} = t_j - 1$ (**). Player I will win in G' by forcing a repetition of the infix σ *ad libitum*. In order to do so, we need to modify its timestamps. An *automorphism* of the structure $(\mathbb{R}, \leq, +1)$ is a monotonic bijection preserving integer differences, in the sense that $f(x+1) = f(x) + 1$ for every $x \in \mathbb{R}$. Note that such an automorphism is uniquely defined by its action on any unit-length interval. We claim that there exists such an automorphism $f : \mathbb{R} \to \mathbb{R}$ mapping $t_i - 1$ to $t_j - 1$ (and hence forcedly also t_i to t_j), and each t_{i_h} with $1 \leq h \leq l$ to $f(t_{i_h}) = t_{j_h}$. This is indeed the case, by (*) and (**) all timestamps t_{i_h}'s belong to the unit half-open interval $[t_i - 1, t_i)$ and likewise all timestamps t_{j_h}'s belong to $[t_j - 1, t_j)$. We apply f to a timed word $\sigma \mapsto f(\sigma)$ by acting pointwise on timestamps. Consider the infinite run $\rho' = \rho_i \cdot \sigma \cdot f(\sigma) \cdot f(f(\sigma)) \cdots$; it is $\mathcal{M}''$-conform since the controller $\mathcal{M}''$ is deterministic. By construction, ρ' contains an infinite $\mathtt{x}$-request chain, and thus $\rho' \notin W_k^{\mathrm{II}}$. It remains to argue that $\rho \in W_k^{\mathrm{I}}$ implies $\rho' \in W_k^{\mathrm{I}}$ as well. Let there be a non-cancelled $\mathtt{x}$-request at time t_s in ρ'. If $t_s < t_j - 1$, then this request must be satisfied at time $t_{s'} = t_s + 1 < t_j$, and thus already in $\rho_i \cdot \sigma$, which is the case since the latter is a prefix of $\rho \in W_k^{\mathrm{I}}$. Now assume $t_j - 1 \leq t_s < t_j$. Thus $t_s = t_{j_h}$ for some $1 \leq h \leq l$. By the definition of f, $f^{-1}(t_s) = t_{i_h} < t_j - 1$ and, thanks to the previous case, the request at t_{i_h} is satisfied at $t_{i_h} + 1$ due to (*).By applying f we obtain $f(t_{i_h} + 1) = f(t_{i_h}) + 1 = t_s + 1$, and thus the request at time t_s is satisfied at time $t_s + 1$ in $f(\sigma)$, as required. The general argument for $t_j + n\Delta + d - 1 \leq t_s < t_j + n\Delta + d$, where $n \geq 0$ and $0 \leq d < \Delta$, is similar, using induction on n. ◀

Proof of Theorem 1.3. Due to Lemmas 5.2 to 5.4, there is a winning untimed controller $\mathcal{M}''$ for G'' if, and only if there is some $m \in \mathbb{N}$ and a winning k, m-controller $\mathcal{M}$ for G. Thus the k-synthesis problem reduces to the $0, 0$-synthesis problem, and the latter is decidable thanks to Lemma 3.1. ◀

6 Future work

While deterministic separators may need exponentially many clocks (cf. Example 4.2), we do not have a computable upper bound on the number of clocks of the separating automaton (if one exists). We leave the DTA separability problem when the number of clocks is not fixed in advance as a challenging open problem. In this case, we cannot reduce the separability problem to a timed synthesis problem, since the latter is undecidable (cf. [13, Appendix C]).

▶ **Theorem 6.1.** *The timed synthesis problem is undecidable, and this holds already when Player I's winning condition is a* 1-NTA *language.*

We leave the computational complexity of separability as future work.

Deterministic separability can be considered also over infinite timed words. We chose to present the case of finite words because it allows us to focus on the essential ingredients of this problem. When going to infinite words, new phenomena appear already in the untimed setting; for instance, deterministic Büchi automata are less expressive than deterministic parity automata, and thus one should additionally specify in the input which priorities can be used by the separator; or leave them unspecified and solve a more difficult problem.

Analogous results about separability of register automata can be obtained with techniques similar to the one presented in this paper. We leave such developments for further work.

References

1 https://siglog.org/the-2016-alonzo-church-award-for-outstanding-contributions-to-logic-and-computation/, 2016.

2 S. Akshay, Paul Gastin, and Shankara Narayanan Krishna. Analyzing Timed Systems Using Tree Automata. *Logical Methods in Computer Science*, Volume 14, Issue 2, May 2018. doi:10.23638/LMCS-14(2:8)2018.

3 Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126:183–235, 1994.

4 Eugene Asarin and Oded Maler. As soon as possible: Time optimal control for timed automata. In *Proc. of HSCC'99*, HSCC '99, pages 19–30, London, UK, UK, 1999. Springer-Verlag. URL: http://dl.acm.org/citation.cfm?id=646879.710314.

5 Eugene Asarin, Oded Maler, Amir Pnueli, and Joseph Sifakis. Controller synthesis for timed automata. In *Proc. of SSSC'98*, volume 31 (18) of *5th IFAC Conference on System Structure and Control*, pages 447–452, 1998. doi:10.1016/S1474-6670(17)42032-5.

6 Gerd Behrmann, Alexandre David, Kim G. Larsen, John Hakansson, Paul Petterson, Wang Yi, and Martijn Hendriks. Uppaal 4.0. In *Proceedings of the 3rd International Conference on the Quantitative Evaluation of Systems*, QEST '06, pages 125–126, Washington, DC, USA, 2006. IEEE Computer Society. doi:10.1109/QEST.2006.59.

7 Patricia Bouyer, Fabrice Chevalier, and Deepak D'Souza. Fault diagnosis using timed automata. In *Proc. of FOSSACS'05*, FOSSACS'05, pages 219–233, Berlin, Heidelberg, 2005. Springer-Verlag. doi:10.1007/978-3-540-31982-5_14.

8 Thomas Brihaye, Thomas A. Henzinger, Vinayak S. Prabhu, and Jean-François Raskin. Minimum-time reachability in timed games. In Lars Arge, Christian Cachin, Tomasz Jurdziński, and Andrzej Tarlecki, editors, *Proc. of ICALP'07*, pages 825–837, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

9 J. Richard Büchi and Lawrence H. Landweber. Solving sequential conditions by finite-state strategies. *Transactions of the American Mathematical Society*, 138:295–311, 1969. URL: http://www.jstor.org/stable/1994916.

10 Franck Cassez, Alexandre David, Emmanuel Fleury, Kim G. Larsen, and Didier Lime. Efficient on-the-fly algorithms for the analysis of timed games. In Martín Abadi and Luca de Alfaro, editors, *Proc. of CONCUR'05*, pages 66–80, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

11 Lorenzo Clemente, Wojciech Czerwiński, Sławomir Lasota, and Charles Paperman. Regular separability of parikh automata. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *Proc. of ICALP'17*, volume 80, pages 117:1–117:13, 2017. doi:10.4230/LIPIcs.ICALP.2017.117.

12 Lorenzo Clemente, Wojciech Czerwinski, Slawomir Lasota, and Charles Paperman. Separability of Reachability Sets of Vector Addition Systems. In *Proc. of STACS'17*, volume 66 of *LIPICs*, pages 24:1–24:14, 2017. doi:10.4230/LIPIcs.STACS.2017.24.

13 Lorenzo Clemente, Sławomir Lasota, and Radosław Piórkowski. Timed games and deterministic separability. *arXiv e-prints*, page arXiv:2004.12868, April 2020. arXiv:2004.12868.

14 Lorenzo Clemente, Paweł Parys, Sylvain Salvati, and Igor Walukiewicz. The diagonal problem for higher-order recursion schemes is decidable. In *Proc. of LICS'16*, 2016. doi:10.1145/2933575.2934527.

15 Hubert Comon and Yan Jurski. Timed automata and the theory of real numbers. In *Proc. of CONCUR'99*, CONCUR '99, pages 242–257, London, UK, UK, 1999. Springer-Verlag.

16 Wojciech Czerwiński and Sławomir Lasota. Regular Separability of One Counter Automata. *Logical Methods in Computer Science*, Volume 15, Issue 2, June 2019. URL: https://lmcs.episciences.org/5563.

17 Wojciech Czerwinski, Slawomir Lasota, Roland Meyer, Sebastian Muskalla, K. Narayan Kumar, and Prakash Saivasan. Regular Separability of Well-Structured Transition Systems. In Sven Schewe and Lijun Zhang, editors, *29th International Conference on Concurrency Theory (CONCUR 2018)*, volume 118 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 35:1–35:18, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.CONCUR.2018.35.

18 Wojciech Czerwiński, Wim Martens, and Tomáš Masopust. Efficient separability of regular languages by subsequences and suffixes. In *Proc. of ICALP'14*, ICALP'13, pages 150–161, Berlin, Heidelberg, 2013. Springer-Verlag. doi:10.1007/978-3-642-39212-2_16.

19 Wojciech Czerwiński, Wim Martens, Lorijn van Rooijen, and Marc Zeitoun. A note on decidable separability by piecewise testable languages. In *Proc. of FCT'15*, 2015. doi:10.1007/978-3-319-22177-9_14.

20 Luca de Alfaro, Marco Faella, Thomas A. Henzinger, Rupak Majumdar, and Mariëlle Stoelinga. The element of surprise in timed games. In Roberto Amadio and Denis Lugiez, editors, *Proc. of CONCUR'03*, pages 144–158, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

21 C. Dima. Computing reachability relations in timed automata. In *Proc. of LICS'02*, pages 177–186, 2002.

22 Deepak D'souza and P. Madhusudan. Timed control synthesis for external specifications. In Helmut Alt and Afonso Ferreira, editors, *Proc. of STACS'02*, pages 571–582, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

23 John Fearnley and Marcin Jurdzinski. Reachability in two-clock timed automata is PSPACE-complete. *Inf. Comput.*, 243:26–36, 2015.

24 Olivier Finkel. Undecidable problems about timed automata. In *Proc. of FORMATS'06*, FORMATS'06, pages 187–199, Berlin, Heidelberg, 2006. Springer-Verlag. doi:10.1007/11867340_14.

25 Martin Fränzle, Karin Quaas, Mahsa Shirmohammadi, and James Worrell. Effective definability of the reachability relation in timed automata. *Information Processing Letters*, 153:105871, 2020. doi:10.1016/j.ipl.2019.105871.

26 Paul Gastin, Sayan Mukherjee, and B. Srivathsan. Reachability in Timed Automata with Diagonal Constraints. In Sven Schewe and Lijun Zhang, editors, *Proc. of CONCUR'18*, volume 118 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 28:1–28:17, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.CONCUR.2018.28.

27 Paul Gastin, Sayan Mukherjee, and B. Srivathsan. Fast algorithms for handling diagonal constraints in timed automata. In Isil Dillig and Serdar Tasiran, editors, *Computer Aided Verification*, pages 41–59, Cham, 2019. Springer International Publishing.

28 Jean Goubault-Larrecq and Sylvain Schmitz. Deciding Piecewise Testable Separability for Regular Tree Languages. In *Proc. of ICALP'16*, volume 55 of *LIPIcs*, pages 97:1–97:15, 2016. doi:10.4230/LIPIcs.ICALP.2016.97.

29 R. Govind, Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz. Revisiting Local Time Semantics for Networks of Timed Automata. In Wan Fokkink and Rob van Glabbeek, editors, *Proc. of CONCUR 2019*, volume 140 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:15, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.CONCUR.2019.16.

30 Matthew Hague, Jonathan Kochems, and C.-H. Luke Ong. Unboundedness and downward closures of higher-order pushdown automata. In *Proc. of POPL'16*, POPL 2016, pages 151–163, New York, NY, USA, 2016. ACM. doi:10.1145/2837614.2837627.

31 Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz. Better abstractions for timed automata. *Information and Computation*, 251:67–90, 2016. doi:10.1016/j.ic.2016.07.004.

32 H. B. Hunt, III. On the decidability of grammar problems. *J. ACM*, 29(2):429–447, April 1982. doi:10.1145/322307.322317.

33 Marcin Jurdziński and Ashutosh Trivedi. Reachability-time games on timed automata. In *Proc. of ICALP'07*, pages 838–849, Berlin, Heidelberg, 2007. Springer-Verlag. URL: http://dl.acm.org/citation.cfm?id=2394539.2394637.

34 Eryk Kopczynski. Invisible pushdown languages. In *Proc. of LICS'16*, pages 867–872, 2016. doi:10.1145/2933575.2933579.

35 Pavel Krčál and Radek Pelánek. On sampled semantics of timed systems. In Sundar Sarukkai and Sandeep Sen, editors, *Proc. of FSTTCS'05*, volume 3821 of *LNCS*, pages 310–321. Springer, 2005. doi:10.1007/11590156_25.

36 M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In G. Gopalakrishnan and S. Qadeer, editors, *Proc. of CAV'11*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.

37 Oded Maler and Amir Pnueli. On recognizable timed languages. In Igor Walukiewicz, editor, *Proc. of FOSSACS'04*, volume 2987 of *LNCS*, pages 348–362. Springer Berlin Heidelberg, 2004. doi:10.1007/978-3-540-24727-2_25.

38 Oded Maler, Amir Pnueli, and Joseph Sifakis. On the synthesis of discrete controllers for timed systems. In Ernst W. Mayr and Claude Puech, editors, *Proc. of STACS'95*, pages 229–242, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

39 Brian Nielsen and Arne Skou. Automated test generation from timed automata. *International Journal on Software Tools for Technology Transfer*, 5(1):59–77, November 2003. doi:10.1007/s10009-002-0094-1.

40 Thomas Place, Lorijn Rooijen, and Marc Zeitoun. Separating regular languages by piecewise testable and unambiguous languages. In Krishnendu Chatterjee and Jirí Sgall, editors, *Proc. of MFCS'13*, pages 729–740. Springer, 2013. doi:10.1007/978-3-642-40313-2_64.

41 Thomas Place, Lorijn van Rooijen, and Marc Zeitoun. Separating regular languages by locally testable and locally threshold testable languages. *LMCS*, 10(3), 2014. doi:10.2168/LMCS-10(3:24)2014.

42 Thomas Place and Marc Zeitoun. Going higher in the first-order quantifier alternation hierarchy on words. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Proc. of ICALP'14*, pages 342–353, Berlin, Heidelberg, 2014. Springer. doi:10.1007/978-3-662-43951-7_29.

43 Thomas Place and Marc Zeitoun. Separating regular languages with first-order logic. *Logical Methods in Computer Science*, 12(1), 2016. doi:10.2168/LMCS-12(1:5)2016.

44 Michael O. Rabin. Weakly definable relations and special automata. In Yehoshua Bar-Hillel, editor, *Mathematical Logic and Foundations of Set Theory*, volume 59 of *Studies in Logic and the Foundations of Mathematics*, pages 1–23. Elsevier, 1970. doi:10.1016/S0049-237X(08)71929-3.

45 Thomas G. Szymanski and John H. Williams. Noncanonical extensions of bottom-up parsing techniques. *SIAM Journal on Computing*, 5(2):231–250, 1976. doi:10.1137/0205019.

46 Martin Tappler, Bernhard K. Aichernig, Kim Guldstrand Larsen, and Florian Lorber. Time to learn - learning timed automata from tests. In Étienne André and Mariëlle Stoelinga, editors, *Proc. of FORMATS'19*, pages 216–235, Cham, 2019. Springer International Publishing.

47 Ramanathan S. Thinniyam and Georg Zetzsche. Regular Separability and Intersection Emptiness Are Independent Problems. In Arkadev Chattopadhyay and Paul Gastin, editors, *Proc. of FSTTCS'19*, volume 150 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 51:1–51:15, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.FSTTCS.2019.51.

48 Stavros Tripakis. Folk theorems on the determinization and minimization of timed automata. *Inf. Process. Lett.*, 99(6):222–226, September 2006.

49 Sicco Verwer, Mathijs de Weerdt, and Cees Witteveen. An algorithm for learning real-time automata. In *Proc of. the Annual Belgian-Dutch Machine Learning Conference (Benelearn'078)*, 2007.

50 H. Wong-Toi and G. Hoffmann. The control of dense real-time discrete event systems. In *Proc. of CDC'91*, volume 2 of *Proceedings of the 30th IEEE Conference on Decision and Control*, pages 1527–1528, December 1991. doi:10.1109/CDC.1991.261658.

Dynamic Complexity of Reachability: How Many Changes Can We Handle?

Samir Datta
Chennai Mathematical Institute, India
sdatta@cmi.ac.in

Pankaj Kumar
Chennai Mathematical Institute, India
Department of Applied Mathematics, Charles University, Prague, Czech Republic
pankaj@kam.mff.cuni.cz

Anish Mukherjee
Institute of Informatics, University of Warsaw, Poland
anish@mimuw.edu.pl

Anuj Tawari
Chennai Mathematical Institute, India
atawari@cmi.ac.in

Nils Vortmeier
TU Dortmund, Germany
nils.vortmeier@tu-dortmund.de

Thomas Zeume
Ruhr-Universität Bochum, Germany
thomas.zeume@rub.de

Abstract

In 2015, it was shown that reachability for arbitrary directed graphs can be updated by first-order formulas after inserting or deleting single edges. Later, in 2018, this was extended for changes of size $\frac{\log n}{\log \log n}$, where n is the size of the graph. Changes of polylogarithmic size can be handled when also majority quantifiers may be used.

In this paper we extend these results by showing that, for changes of polylogarithmic size, first-order update formulas suffice for maintaining (1) undirected reachability, and (2) directed reachability under insertions. For classes of directed graphs for which efficient parallel algorithms can compute non-zero circulation weights, reachability can be maintained with update formulas that may use "modulo 2" quantifiers under changes of polylogarithmic size. Examples for these classes include the class of planar graphs and graphs with bounded treewidth. The latter is shown here.

As the logics we consider cannot maintain reachability under changes of larger sizes, our results are optimal with respect to the size of the changes.

2012 ACM Subject Classification Theory of computation → Complexity theory and logic

Keywords and phrases Dynamic complexity, reachability, complex changes

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.122

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

Related Version A full version of this paper is available at `http://arxiv.org/abs/2004.12739`.

Funding *Samir Datta*: The author was partially funded by a grant from Infosys foundation and SERB-MATRICS grant MTR/2017/000480.
Pankaj Kumar: The author was supported by Czech Science Foundation GAČR (grant #19-27871X).
Anish Mukherjee: The author was supported by the ERC CoG grant TUgbOAT no 772346.
Nils Vortmeier: The author acknowledges the financial support by DFG grant SCHW 678/6-2.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 122; pp. 122:1–122:19
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Acknowledgements The first author would like to thank Chetan Gupta for finding a problem in a previous version of the proof of Theorem 14.

1 Introduction

Suppose we are given a graph G whose edge relation is subjected to insertions and deletions of edges. Which resources are required to update the reachability relation of the graph?

Recently it was shown that if one is allowed to store auxiliary relations, then the reachability relation can be updated after single edge insertions and deletions using first-order logic formulas with access to the graph, the stored relations, and the changed edges [4]. In other words, the reachability query is contained in the dynamic complexity class DynFO [19]. From a database perspective, this means that it can be updated with core-SQL queries; from the perspective of circuit complexity, this means that reachability can be updated by circuits of polynomial size in constant-time due to the correspondence of first-order logic and AC^0 established by Barrington, Immerman, and Straubing [1].

Understanding single edge insertions and deletions is an important first step. Yet in applications, changes to a graph G often come as bulk set ΔE of changed edges. It is natural to ask, how large the set ΔE of edges can be such that reachability can be maintained with the same resources as for single edge changes – that is with first-order formulas or, respectively, AC^0 circuits. Using existing lower bounds for circuits [21], it is easy to see that DynFO (or DynAC^0, respectively) cannot handle changes of size larger than polylogarithmic for many queries, including the reachability query (see Section 3 for a more detailed discussion).

The best one can hope for is to maintain reachability with first-order formulas for changes of polylogarithmic size with respect to the size of the graph. In a first step, a subset of the authors showed that reachability can be maintained in $\mathsf{DynFO}(\leq, +, \times)$ under changes of size $O(\frac{\log n}{\log \log n})$ [7]. Here, the class $\mathsf{DynFO}(\leq, +, \times)$ extends DynFO by access to built-in arithmetic, which for technical reasons is more natural for bulk changes. Unfortunately, the techniques used in [7] seem only be able to handle changes of polylogarithmic size in the extension of DynFO by majority quantifiers, that is, in the class $\mathsf{DynFO{+}Maj}(\leq, +, \times)$.

In this paper we make progress on handling changes of polylogarithmic size in DynFO by attacking the challenge from two directions. First, we establish two restrictions for which reachability can be maintained under these changes.

▶ **Main Theorem 1.** *Reachability can be maintained in* $\mathsf{DynFO}(\leq, +, \times)$ *under*

- *insertions of polylogarithmically many edges; and*
- *insertions and deletions of polylogarithmically many edges if the graph remains undirected.*

As second contribution of this paper, we provide a meta-theorem for establishing classes of graphs for which reachability can be maintained under polylogarithmic-size changes with a slight extension of first-order logic. In this extension, $\mathsf{DynFO{+}Mod\,2}(\leq, +, \times)$, formulas used for updating the reachability information and the auxiliary relations may use parity quantifiers in addition to the traditional universal and existential quantifiers.

▶ **Main Theorem 2.** *Reachability can be maintained in* $\mathsf{DynFO{+}Mod\,2}(\leq, +, \times)$ *under insertions and deletions of polylogarithmically many edges on classes of graphs for which polynomially bounded non-zero circulation weights can be computed in* AC.

Here a weighting function for the edges of a graph has *non-zero circulation*, if the weight of every directed cycle is non-zero (see Section 6 for details). The class AC contains queries computable by circuits of polynomial size and polylogarithmic depth. Examples for graph classes for which non-zero circulation weights can be computed in AC include the class of planar graphs and graphs with bounded treewidth. The latter is shown here. We note that isolating weights, a concept closely related to non-zero circulation weights, have been used previously in dynamic complexity for establishing that reachability is in non-uniform DynFO+Mod $2(\leq, +, \times)$ under single edge changes [3], a precursor result to reachability in DynFO.

For our results, we employ two techniques of independent interest. The first technique relies on the power of first-order logic on structures of polylogarithmic size. It is well-known that reachability can be computed by a uniform circuit family of size $N^{\mathcal{O}(N^{1/d})}$ and depth $2d$. An immediate consequence is that all NL-queries can be expressed by first-order formulas for graphs with n nodes but only polylogarithmically many edges. Thus, for maintaining a query under changes ΔE of polylogarithmic size, a dynamic program can (1) do an arbitrary NL-computation on ΔE, and (2) update the auxiliary data by combining the computed information with the previous auxiliary data using a first-order formula.

The second technique we rely on is a slight generalization of the "Muddling Lemma" from [6]. The Muddling Lemma reduces the requirements for proving that a query is in DynFO: a query is in DynFO if, essentially, one can update the query for polylog many steps starting from auxiliary data precomputed in AC. Here we observe that this can be strengthened for changes of polylogarithmic size: a query is in DynFO if, essentially, one can update the query under one polylogarithmic-size change from auxiliary data precomputed in AC.

Parts of the results presented here have been included in the PhD thesis of Nils Vortmeier [24].

Outline. After recalling the dynamic complexity framework in Section 2, we shortly outline barriers for the size of bulk changes in Section 3 and recall useful techniques in Section 4. Afterwards we present our results for DynFO in Section 5 and for DynFO+Mod $2(\leq, +, \times)$ in Section 6.

2 The dynamic setting

We briefly repeat the essentials of dynamic complexity, closely following [7] which in turn builds on [20]. The goal of a *dynamic program* is to answer a given query on a relational *input structure* subjected to changes that insert tuples into the input relations or delete tuples from them. The program may use auxiliary information represented by an *auxiliary structure* over the same domain as the input structure. Initially, both input and auxiliary structure are empty; and the domain is fixed during each run of the program. Whenever a change to the input structure occurs, the auxiliary structure is updated by means of first-order formulas.

Changes. For a (relational) structure $\mathcal{I}$ over domain D and schema σ, a change $\Delta\mathcal{I}$ consists of sets R^+ and R^- of tuples for each relation symbol $R \in \sigma$. The result $\mathcal{I} + \Delta\mathcal{I}$ of an application of the change $\Delta\mathcal{I}$ to $\mathcal{I}$ is the input structure where $R^{\mathcal{I}}$ is changed to $(R^{\mathcal{I}} \cup R^+) \setminus R^-$. The *size* of $\Delta\mathcal{I}$ is the total number of tuples in relations R^+ and R^- and the set of *affected elements* is the (active) domain of tuples in $\Delta\mathcal{I}$.

Dynamic Programs and Maintenance of Queries. A dynamic program consists of a set of update rules that specify how auxiliary relations are updated after changing the input structure. Let $\mathcal{I}$ be the current input structure over schema σ and let $\mathcal{A}$ be the auxiliary structure over some schema σ_{aux}. An *update rule* for updating an ℓ-ary auxiliary relation T after a change is a first-order formula φ over schema $\sigma \cup \sigma_{\text{aux}}$ with ℓ free variables. After a change $\Delta\mathcal{I}$, the new version of T is $T \stackrel{\text{def}}{=} \{\bar{a} \mid (\mathcal{I} + \Delta\mathcal{I}, \mathcal{A}) \models \varphi(\bar{a})\}$, so, the updated auxiliary relation includes all tuples $\bar{a}$ such that $\varphi(\bar{a})$ is satisfied when it is evaluated on the changed input structure and the old auxiliary structure. Note that a dynamic program can choose to have access also to the old input structure by storing it in its auxiliary relations.

For a state $\mathcal{S} = (\mathcal{I}, \mathcal{A})$ of the dynamic program $\mathcal{P}$ with input structure $\mathcal{I}$ and auxiliary structure $\mathcal{A}$ we denote the state of the program after applying a change sequence α and updating the auxiliary relations accordingly by $\mathcal{P}_\alpha(\mathcal{S})$.

The dynamic program *maintains* a q-ary query Q under changes of size k if it has a q-ary auxiliary relation $\textsc{Ans}$ that at any time stores the result of Q applied to the current input structure. More precisely, for each non-empty sequence α of changes of size k, the relation $\textsc{Ans}$ in $\mathcal{P}_\alpha(\mathcal{S}_\emptyset)$ and $Q(\alpha(\mathcal{I}_\emptyset))$ coincide, where the state $\mathcal{S}_\emptyset \stackrel{\text{def}}{=} (\mathcal{I}_\emptyset, \mathcal{A}_\emptyset)$ consists of an input structure $\mathcal{I}_\emptyset$ and an auxiliary structure $\mathcal{A}_\emptyset$ over some common domain that both have empty relations, and $\alpha(\mathcal{I}_\emptyset)$ is the input structure after applying α.

If a dynamic program maintains a query, we say that the query is in DynFO. Similarly to DynFO one can define variants with built-in auxiliary relations and with more powerful update formulas. For instance, the class $\mathsf{DynFO}(\leq, +, \times)$ contains queries that can be maintained by first-order update formulas with access to three particular auxiliary relations $<, +$, and $\times$ which are initialized as a linear order and the corresponding addition and multiplication relations; in the class $\mathsf{DynFO{+}Mod\ p}$, update formulas may use modulo-p-quantifiers in addition to existential and universal quantifiers.

We state our results for dynamic classes with access to the arithmetic relations $\leq, +$ and $\times$. Handling bulk changes without access to arithmetic leads to technical issues which distract from the fundamental dynamic properties. See [7, 24] for further discussions on this topic and how our results can be stated for DynFO in an adapted setting which takes these technical issues into account.

For the construction of dynamic programs in this paper we assume that changes either only insert edges or only delete edges. This is no restriction, as corresponding update formulas can be combined to process a change that inserts and deletes edges at the same time, by first processing the inserted edges and then processing the deleted edges.

3 Barriers for the size of bulk changes

In the following we outline why it is not possible to maintain reachability under changes of larger than polylogarithmic size with first-order formulas, even in the presence of parity quantifiers.

The idea is simple. A classical result by Smolensky states that for computing the number of ones modulo a prime q occurring in a bit string of length n, an $\mathsf{AC}[p]$ circuit of depth d requires $2^{\Omega(n^{1/2d})}$ many gates, for each prime p distinct from q (see [21] or, for a modern exposition, [17, Theorem 12.27]). A simple, well-known reduction yields that deciding reachability for graphs with n edges which are disjoint unions of paths also requires $\mathsf{AC}[p]$ circuits of size $2^{\Omega(n^{1/2d})}$. Indeed, computing the number of ones in $w = a_1 \cdots a_n$ modulo q can be reduced to reachability as follows. Consider the graph with nodes $\{(i,k) \mid 1 \leq i \leq n+1 \text{ and } 0 \leq k < q\}$ and edges $\{((i,k),(i+1,k)) \mid a_i = 0\} \cup \{((i,k),(i+1,k+1 \bmod q)) \mid a_i = 1\}$. It is easy to see that (i) the graph has $\mathcal{O}(n)$ edges and is a disjoint union of q paths, and (ii) there is a path from $(1,0)$ to $(n+1,0)$ if and only if the number of ones in w is 0 modulo q.

These lower bounds for circuit sizes immediately translate into lower bounds for first-order formulas with modulo p quantifiers via the correspondence due to Barrington, Immerman, and Straubing [1].

▶ **Theorem 1.** *Let $f(n) \in \log^{\omega(1)} n$ be a function from $\mathbb{N}$ to $\mathbb{N}$ and let p be a prime. There is no* FO+Mod p *formula with access to built-in relations that defines*

1. *whether the size of a unary relation U with $|U| \leq f(n)$ is divisible by q, for primes q distinct from p;*
2. *reachability in graphs with at most $f(n)$ edges, even for disjoint unions of paths.*

Proof sketch.

(a) Let $f(n)$ be some function from $\log^{\omega(1)} n$. Suppose, towards a contradiction, that there is an FO+Mod p formula with access to built-in relations that defines whether the size of a unary relation U with $|U| \leq f(n)$ is divisible by q, for some primes $p \neq q$. Then, by [1], for every n there is an AC$[p]$ circuit of some fixed depth d that decides that question for inputs if size n, and the size of this circuit is polynomial in n. That is a contradiction, as by Smolensky's lower bound every such circuit needs to have size $2^{\Omega(f(n)^{1/2d})} = 2^{\log(n)^{\omega(1)}} = n^{\omega(1)}$.

(b) This part can be proven analogously to Part (a), using the circuit lower bound for graph reachability. ◀

Those lower bounds have the immediate consequence that DynFO cannot deal with bulk changes of larger than polylogarithmic size. Indeed, from any formula that updates the result of a query after an insertion of $f(n)$ tuples into an initially empty input relation one can construct a formula that defines the query for inputs of size $f(n)$.

▶ **Corollary 2.** *Let $f(n) \in \log^{\omega(1)} n$ be a function from $\mathbb{N}$ to $\mathbb{N}$ and let p be a prime. Then the following queries cannot be maintained in* DynFO+Mod p *for bulk changes of size $\leq f(n)$, even if the auxiliary relations may be initialized arbitrarily:*

1. *divisibility of the size of a unary relation by a prime $q \neq p$, and*
2. *reachability in graphs, even if restricted to disjoint unions of paths.*

4 Techniques and Tools

In the previous section we recalled that FO$(\leq, +, \times)$, even if equipped with modulo p quantifiers, is not very expressive in general. That changes when we are only interested in small substructures of our input: FO$(\leq, +, \times)$ can express every NL-computable query on subgraphs of polylogarithmic size.

▶ **Theorem 3.** *Let k and c be arbitrary natural numbers, and let Q be a k-ary,* NL*-computable graph query. There is an* FO$(\leq, +, \times)$ *formula φ over schema $\{E, D\}$ such that for any graph G with n nodes, any subset D of its nodes of size at most $\log^c n$ and any k-tuple $\bar{a} \in D^k$: $\bar{a} \in Q(G[D])$ if and only if $(G, D) \models \varphi(\bar{a})$. Here, $G[D]$ denotes the subgraph of G induced by D.*

Proof. We prove the result for the reachability query. As reachability is NL-complete under FO$(\leq, +, \times)$-reductions [16], and every FO$(\leq, +, \times)$-reduction maps an instance of size $\log^c n$ to an instance of size $\log^{cd} n$ for a fixed $d \in \mathbb{N}$, the full result follows.

It is well-known (see for example [2, p. 613]), that for every $d \in \mathbb{N}$ there is a uniform circuit family for reachability where the circuit for inputs of size N has depth $2d$ and size $N^{\mathcal{O}(N^{1/d})}$. Suppose the input size N is only $\log^c n$, for some $c \in \mathbb{N}$ and pick $d \overset{\text{def}}{=} 2c$. Then the circuit size

$$\begin{aligned}N^{\mathcal{O}(N^{1/d})} =&(\log^c n)^{\mathcal{O}((\log^c n)^{1/d})}\\ =&(\log n)^{c\mathcal{O}((\log n)^{c/d})} = (\log n)^{\mathcal{O}((\log n)^{c/2c})} = (\log n)^{\mathcal{O}(\sqrt{\log n})}\\ =&2^{\mathcal{O}(\log\log n\sqrt{\log n})} \subseteq 2^{\mathcal{O}(\sqrt{\log n}\sqrt{\log n})} = 2^{\mathcal{O}(\log n)} = n^{\mathcal{O}(1)}\end{aligned}$$

is polynomial in n, so, the circuit is a uniform AC^0 circuit for reachability for graphs of size $\log^c n$. The existence of φ follows by the equivalence of uniform AC^0 and $\mathsf{FO}(\leq, +, \times)$ [1]. ◀

The Muddling Lemma simplifies the maintenance of queries under single edge changes [6]. It states that for many natural queries Q, in order to show that Q can be maintained, it is enough to show that the query can be maintained for a bounded number of steps. In the following we recall the necessary notions and extend the lemma to bulk changes.

A query Q is *almost domain-independent* if there is a $c \in \mathbb{N}$ such that $Q(\mathcal{A})[(\text{adom}(\mathcal{A}) \cup B)] = Q(\mathcal{A}[(\text{adom}(\mathcal{A}) \cup B)])$ for all structures $\mathcal{A}$ and sets $B \subseteq A \setminus \text{adom}(\mathcal{A})$ with $|B| \geq c$. Here, $\text{adom}(\mathcal{A})$ denotes the *active domain*, i.e. the set of domain elements that are used in some tuple of $\mathcal{A}$. A query Q is *$(\mathcal{C}, f)$-maintainable*, for some complexity class $\mathcal{C}$ and some function $f : \mathbb{N} \to \mathbb{R}$, if there is a dynamic program $\mathcal{P}$ and a $\mathcal{C}$-algorithm $\mathbb{A}$ such that for each input structure $\mathcal{I}$ over a domain of size n, each linear order $\leq$ on the domain, and each change sequence α of length $|\alpha| \leq f(n)$, the relation Q in $\mathcal{P}_\alpha(\mathcal{S})$ and $Q(\alpha(\mathcal{I}))$ coincide, where $\mathcal{S} = (\mathcal{I}, \mathbb{A}(\mathcal{I}, \leq))$.

The Muddling Lemma from [6] has been formulated for bulk changes in [7, 24][1].

▶ **Theorem 4** ([7, 24]). *Let Q be an NL-computable, almost domain independent query, and let $c \in \mathbb{N}$ be arbitrary. If the query Q is $(\mathsf{AC}^d, \log^d n)$-maintainable under changes of size $\log^c n$ for some $d \in \mathbb{N}$, then Q is in $\mathsf{DynFO}(\leq, +, \times)$ under changes of size $\log^c n$.*

The previous theorem can be strengthened as follows.

▶ **Theorem 5.** *Let Q be an NL-computable, almost domain independent query, and let $c \in \mathbb{N}$ be arbitrary. If the query Q is $(\mathsf{AC}^d, 1)$-maintainable under changes of size $\log^{c+d} n$ for some $d \in \mathbb{N}$, then Q is in $\mathsf{DynFO}(\leq, +, \times)$ under changes of size $\log^c n$.*

Proof. Let Q and d be as in the theorem statement, and let $\mathbb{A}$ be an AC^d algorithm and $\mathcal{P}$ a dynamic program that witness that Q is $(\mathsf{AC}^d, 1)$-maintainable under changes of size $\log^{c+d} n$. By Theorem 4 it suffices to show that there is an AC^d algorithm $\mathbb{A}'$ and a dynamic program $\mathcal{P}'$ that witness that Q is $(\mathsf{AC}^d, \log^d n)$-maintainable under changes of size $\log^c n$.

We choose $\mathbb{A}'$ as $\mathbb{A}$. The program $\mathcal{P}'$ just stores the at most $\log^{c+d} n$ changes that accumulate during the $\log^d n$ steps, and in each step uses $\mathcal{P}$ to answer Q, using the initial auxiliary relations computed by $\mathbb{A}$. ◀

5 Handling Polylog Changes with DynFO

So far we do not know how to maintain directed reachability under polylogarithmically many changes in $\mathsf{DynFO}(\leq, +, \times)$. In this section we show that reachability can be maintained in $\mathsf{DynFO}(\leq, +, \times)$ under insertions of polylogarithmically many edges for arbitrary graphs (disallowing any deletions), and under insertions *and* deletions of polylogarithmic size for undirected graphs.

[1] The statement and proof in [7] is slightly flawed and has been corrected in [24].

The general idea is similar in both cases. After changing polylogarithmically many edges with an effect on nodes V_{aff}, the dynamic program (1) computes a structure of polylogarithmic size on V_{aff}, (2) uses Theorem 3 to compute helpful information for this structure, and (3) updates the auxiliary relations by combining this information with the previous auxiliary data. Both (1) and (3) are performed by first-order formulas, and (2) uses an NL-computation.

▶ **Theorem 6.** *Reachability is in* $\mathsf{DynFO}(\leq, +, \times)$ *under insertions of size* $\log^c n$*, for every* $c \in \mathbb{N}$.

Proof. Let $c \in \mathbb{N}$ be fixed. We construct a dynamic program with a single auxiliary relation $\textsc{Ans}$ which stores the transitive closure of the current graph.

Whenever a set E^+ of edges is inserted into the current graph $G = (V, E)$, the dynamic program updates $\textsc{Ans}$ with the help of the transitive closure relation of a graph H defined as follows. The nodes of H are the nodes V_{aff} affected by the change, that is, the nodes incident to edges in E^+. The edge set E_H of H contains the newly inserted edges E^+, and additionally edges (u, v) for all pairs (u, v) of nodes from V_{aff} that are connected by a path in G. Observe that H is of size $O(\log^c n)$ and first-order definable from G, E^+ and $\textsc{Ans}$. Hence, by Theorem 3, the transitive closure of H can be defined by a first-order formula.

The transitive closure relation of $G' \overset{\text{def}}{=} (V, E \cup E^+)$ can now be constructed from the transitive closures of G and H. To this end observe that the transitive closure of H equals the transitive closure relation of G' restricted to V_{aff}: it accounts for all paths from a node $u \in V_{\text{aff}}$ to another node $v \in V_{\text{aff}}$ that may use both newly inserted edges and edges that are already present in G. For this reason, every path ρ in G' consists of three consecutive subpaths $\rho_1\rho_2\rho_3 = \rho$, where ρ_1 and ρ_3 are defined as the maximal subpaths of ρ that do not rely on edges from E^+. These subpaths already exist in G and are represented in $\textsc{Ans}$. The subgraph ρ_2 by definition starts and ends at nodes from V_{aff}, so its existence is given by the transitive closure relation of H.

Hence, the transitive closure of G' can be defined by the formula $\varphi(s, t) \overset{\text{def}}{=} \textsc{Ans}(s, t) \vee \exists x_1 \exists x_2 \big(\textsc{Ans}(s, x_1) \wedge \text{TC}_H(x_1, x_2) \wedge \textsc{Ans}(x_2, t)\big)$. ◀

▶ **Theorem 7.** *Reachability on undirected graphs can be maintained in* $\mathsf{DynFO}(\leq, +, \times)$ *under changes of size* $\log^c n$*, for every* $c \in \mathbb{N}$.

Proof. The dynamic program from [9] that maintains undirected reachability in DynFO under single-edge changes uses, in addition to the transitive closure relation of the input graph, two binary auxiliary relations that represent a directed spanning forest of the input graph and its transitive closure, respectively. We show that these relations can still be maintained in $\mathsf{DynFO}(\leq, +, \times)$ under changes of $\log^c n$ many edges, for fixed $c \in \mathbb{N}$.

Recall that it suffices to treat insertions and deletions independently, as they can be handled subsequently by a dynamic program.

For edge insertions, the construction idea is very similar to the proof of Theorem 6. We define a graph H, where nodes correspond to connected components of the input graph that include an affected node, and edges indicate that some inserted edge connects the respective connected components. As this graph is of polylogarithmic size, thanks to Theorem 3 we can express a spanning forest for H and its transitive closure in $\mathsf{FO}(\leq, +, \times)$, which is sufficient to update the respective relations for the whole input graph.

In the case of edge deletions, the update formulas need to replace deleted spanning tree edges, whenever this is possible. Our approach is very similar to the case of edge insertions. The spanning tree decomposes into polylogarithmically many connected components when edges are deleted. These components can be merged again if non-tree edges exist that connect them, and these edges become tree edges of the spanning forest. For a correspondingly defined graph of polylogarithmic size we can again define a spanning forest and its transitive closure, and from this information select the new tree edges.

We explain both cases in more detail. Let $G = (V, E)$ be the undirected input graph of size n with transitive closure $\textsc{Ans}$, and let S and TC_S be a directed spanning forest for G and its transitive closure, respectively. Suppose that a set E^+ of size at most $\log^c n$ is inserted. We define a graph H as follows. It contains a node $v \in V$ if (1) v is affected, that is, if E^+ contains an edge of v, and (2) v is the smallest affected node in its connected component of G with respect to $\leq$. It contains an edge (u, v) if $(u', v') \in E^+$ for some nodes u', v' with $(u, u') \in \textsc{Ans}$ and $(v, v') \in \textsc{Ans}$, so, if the connected components of u and v are connected by an inserted edge. The graph H is easily seen to be FO-definable using $\textsc{Ans}$. Because H is of polylogarithmic size with respect to n and a spanning forest of a graph can be computed[2] in NL, we can define a spanning tree S_H as well as its transitive closure TC_{S_H} in $\mathsf{FO}(\leq, +, \times)$, thanks to Theorem 3.

The update formulas define updated auxiliary relations for the graph $G' = (V, E \cup E^+)$ as follows. Intuitively, an edge $(u, v) \in S_H$ means that the connected components of u and v in G shall be connected in G' directly by a new tree edge. There might be several edges in E^+ that may serve this purpose, and we need to choose one of them. So, an edge $(u', v') \in E^+$ becomes part the updated spanning forest if there is an edge $(u, v) \in S_H$ such that u' and u as well as v' and v are in the same connected component of G, respectively, and (u', v') is the lexicographically minimal edge with these properties. This is clearly $\mathsf{FO}(\leq, +, \times)$-expressible using the old auxiliary relations. The old tree edges from S are taken over to the updated version, although some directions need to be inverted, if for a newly chosen tree edge (u', v') the node v' was not the root of the directed spanning tree of its connected component. First-order formulas that determine which edges need to be reversed and that provide the adjusted transitive closure for the components of the spanning forest are given in [9]. The relation TC_S is updated by combining this information with TC_{S_H}.

We note that $\textsc{Ans}$ is first-order expressible from TC_S. In conclusion, all auxiliary relations can be updated in $\mathsf{FO}(\leq, +, \times)$.

Now suppose that a set E^- of at most $\log^c n$ edges is deleted. Let S' be the spanning forest that results from S after all tree edges from E^- are removed, and let $\text{TC}_{S'}$ be its transitive closure, which is easily FO-expressible from TC_S. Similarly as above we define a graph H, with nodes being the minimal affected nodes in a weakly connected component of S', which are connected by an edge if the respective weakly connected components of S' are connected by some edge from $E \setminus E^-$. The same way as above, $\mathsf{FO}(\leq, +, \times)$ formulas can define a spanning forest and its transitive closure for H and then use this information to define a spanning forest and its transitive closure for the changed graph $G' = (V, E \setminus E^-)$. ◀

6 Handling Polylog Changes with DynFO+Mod 2

While we have seen, in the last section, that reachability for directed graphs can be maintained under edge insertions of polylogarithmic size, a matching result for edge deletions is still missing. Two intermediate results were shown in [7], building on the work of [13]: reachability can be maintained in $\mathsf{DynFO}(\leq, +, \times)$ under insertions and deletions that affect $\frac{\log n}{\log \log n}$ nodes, and in $\mathsf{DynFO{+}Maj}$ under insertions and deletions of polylogarithmically many edges.

[2] For example the breadth-first spanning forest with the minimal nodes of each component, with respect to $\leq$, as roots can be computed with the inductive counting technique due to Immerman and Szelepcsényi [15, 22].

In this section, we adapt the proof of the latter result, also using ideas that appear in [3], and show that reachability can be maintained in DynFO+Mod $2(\leq, +, \times)$ under edge insertions and deletions of polylogarithmic size for classes of directed graphs for which non-vanishing weight assignments can be computed in AC, that is, by polynomial-size circuits of polylogarithmic depth. This is possible for example for planar graphs [23], as well as for graphs with bounded treewidth, as we show towards the end of this section.

We start by giving the necessary definitions regarding isolating and non-vanishing weight assignments.

6.1 Isolating and non-vanishing weights

A *weighted* directed graph (G, w) consists of a graph $G = (V, E)$ and a *weight assignment* $w \colon E \to \mathbb{Z}$ that assigns an integer weight $w(e)$ to each edge $e \in E$. The weight assignment w is *bounded* by a function $f(|V|)$ if w assigns only weights from the interval $[-f(|V|), f(|V|)]$.

The weighted graph (G, w) is *min-unique* if (1) w only gives positive weights to the edges E, and (2) if some path from s to t exists, for some pair s, t of nodes, then there is a unique path from s to t with minimum weight under w. Here, the weight of a path (and in general every sequence of edges) is the sum of the weights of its edges. If (G, w) is min-unique, we say that w *isolates* (minimal paths in) G.

Define $\overleftrightarrow{G} = (V, \overleftrightarrow{E})$ to be the *bidirected extension* of G, where $\overleftrightarrow{E} \overset{\text{def}}{=} \{(u, v), (v, u) \mid (u, v) \in E\}$. A weight assignment w is *skew-symmetric* if $w(u, v) = -w(v, u)$ for all $(u, v) \in \overleftrightarrow{E}$. It has *non-zero circulation* if the weight of every simple directed cycle in $\overleftrightarrow{G}$ is non-zero (here, a cycle is *simple* if no node occurs twice).

From polynomially bounded non-zero circulation weights for $\overleftrightarrow{G}$ we can easily compute isolating weights for G.

▶ **Lemma 8.** *Let $G = (V, E)$ be a graph with n nodes, and let w be skew-symmetric non-zero circulation weight assignment for $\overleftrightarrow{G}$, which is bounded by n^k for some $k \in \mathbb{N}$. Then w' with $w'(e) = w(e) + n^{k+2}$ for every $e \in E$ isolates G.*

Proof. All weights in w' are clearly positive. It remains to show that w' isolates minimal paths in G. Assume, towards a contradiction, that there are two different s-t-paths ρ_1, ρ_2 with the same minimal weight under w' in G, for some nodes s and t. Without loss of generality, they are both simple paths, as otherwise they cannot be minimal. Let u be the last node visited by both paths before they differ for the first time, and let v be the first node after u that is visited by both paths. Let ρ_1^{uv}, ρ_2^{uv} be the subpaths in ρ_1, ρ_2 from u to v, respectively. If these subpaths have different weights, say, $w'(\rho_1^{uv}) < w'(\rho_2^{uv})$, then we can replace ρ_2^{uv} by ρ_1^{uv} in ρ_2 and get a lighter path, contradicting the assumption that both ρ_1 and ρ_2 are paths with minimal weight. So, $w'(\rho_1^{uv}) = w'(\rho_2^{uv})$ needs to hold. Then also $w(\rho_1^{uv}) = w(\rho_2^{uv})$ holds, because $w(\rho)$ and $w'(\rho)$ differ by a multiple of n^{k+2} for any path ρ, and the difference between $w(\rho)$ and $w(\rho')$ is at most n^{k+1}, for simple paths ρ and ρ'. So, w' cannot compensate weight differences under w. But then the concatenation of ρ_1^{uv} and the reverse of ρ_2^{uv} is a simple cycle in $\overleftrightarrow{G}$ with weight $w(\rho_1^{uv}) - w(\rho_2^{uv}) = 0$, contradiction the assumption that w has non-zero circulation. ◀

We explain how (families of) polynomially bounded weight assignments for graphs are represented in relational structures. Let V be the node set of a weighted graph of size n. We identify V with the set $\{0, \ldots, n-1\}$ of numbers according to the given linear order $\leq$. A tuple $(a_1, \ldots, a_k)$ of nodes then represents the number $\sum_{i=1}^{k} a_i n^{i-1}$. A (partial) function $f \colon V^k \to V^\ell$ is represented as a $(k+\ell)$-ary relation F over V, such that for each $\bar{a} \in V^k$ there is at most one $\bar{b} \in V^\ell$ with $(\bar{a}, \bar{b}) \in F$. We say that f is $\mathsf{FO}(\leq, +, \times)$-definable if F is defined by an $\mathsf{FO}(\leq, +, \times)$ formula $\psi(\bar{x}, \bar{y})$, where $\bar{x} = x_1, \ldots, x_k$ and $\bar{y} = y_1, \ldots, y_\ell$. An $\mathsf{FO}(\leq, +, \times)$ formula $\psi(\bar{z}, \bar{x}, \bar{y})$, with $\bar{z} = z_1, \ldots, z_m$, defines a family $\{f(\bar{c})\colon V^k \to V^\ell \mid \bar{c} \in V^m\}$ of functions.

6.2 Maintaining Reachability in weighted graphs

We now state and prove the main result of this section.

▶ **Theorem 9.** *Let $\mathcal{G}$ be a class of graphs for which polynomially bounded skew-symmetric non-zero circulation weights can be computed in* AC. *Then, reachability for graphs in $\mathcal{G}$ is in* $\mathsf{DynFO{+}Mod}\,2(\leq, +, \times)$ *under changes of size $\log^c n$, for every $c \in \mathbb{N}$.*

Although this result leaves open whether reachability can be maintained in $\mathsf{DynFO}(\leq, +, \times)$ under polylogarithmically many edge changes, note that, in light of Corollary 2, it gives a tight upper bound for the size of changes that can be handled in $\mathsf{DynFO{+}Mod}\,2(\leq, +, \times)$.

We outline the proof strategy, which closely follows the strategy from [7]. Suppose, we are given a weighted directed graph (G, w) where $G = (V, E)$ is a graph with n nodes and w is an isolating weight assignment. We represent this weighted graph by an $n \times n$ matrix $A_{(G,w)}(x)$ as follows: if $(u, v) \in E$, then the u-v-entry of $A_{(G,w)}(x)$ is $x^{w(u,v)}$, where x is a formal variable, otherwise the u-v-entry is 0.

The matrix $D \stackrel{\text{def}}{=} \sum_{i=0}^{\infty}(A_{G_w}(x))^i$ is a matrix of formal power series in the formal variable x, and from an s-t-entry $\sum_{i=0}^{\infty} c_i x^i$ of this matrix we can read the number c_i of paths from s to t with weight i. Our goal is to determine the coefficients c_i modulo 2, for all i up to some polynomial bound. From this information we can deduce whether there is a path from s to t in G: as w isolates minimal paths in G, if there is some path from s to t, then there is a unique path with minimal weight, which means that for the weight ℓ of this path we have $c_\ell \equiv 1 \pmod 2$. Otherwise, if no path from s to t exists, $c_\ell \equiv 0 \pmod 2$ for all i.

We use the following insights to actually compute and update the coefficients c_i. Notice that the matrix D is invertible over the ring of formal power series (see [7] and its full version [8]) and can be written as $D = (I - A_{(G,w)}(x))^{-1}$, where I is the identity matrix.

So, we need to compute and update the inverse of a matrix. This cannot be done effectively for matrices of inherently infinite formal power series. For this reason we compute D only approximately. A *b-approximation* C of D, for some $b \in \mathbb{N}$, is a matrix of formal polynomials that agrees with the entries of D on the low-degree coefficients c_i for all $i \leq b$. This precision is preserved by the matrix operations we use, see [7, Proposition 14]. Note that it is sufficient to maintain an approximation of D, as for a weighted graph with polynomially bounded weights the maximal possible weight $w_{\max}$ of a minimal path is bounded by a polynomial, and thus only the coefficients c_i with $i \leq w_{\max}$ are relevant.

To update the matrix inverse, we employ the Sherman-Morrison-Woodbury identity (cf. [12]). This identity states that when updating a matrix A to a matrix $A + \Delta A$, with ΔA writeable as matrix product UBV, the inverse of A can be updated as follows:

$$(A + \Delta A)^{-1} = (A + UBV)^{-1} = A^{-1} - A^{-1}U(I + BVA^{-1}U)^{-1}BVA^{-1}.$$

When ΔA has only k non-zero rows and columns, there is a decomposition UBV where B is a $k \times k$ matrix.

The right-hand side can be computed in $\mathsf{FO{+}Mod}\,2(\leq, +, \times)$ for $k \stackrel{\text{def}}{=} \log^c n$. To see this, we observe that also $I + BVA^{-1}U$ is a $k \times k$ matrix. Computing the right-hand side now requires multiplication and iterated addition of polynomials over $\mathbb{Z}$ as well as the computation of the inverse of a $k \times k$ matrix. As all computations are done modulo 2, this is indeed possible in FO+Mod $2(\leq, +, \times)$ for (matrices of) polynomials with polynomial degree using results of [11]. We provide more details later.

As we work with isolating weight assignments, our update routines also need to assign weights to changed edges such that the resulting weight assignment is again isolating. We show that this can be done if we start with (slightly adjusted) non-zero circulation weights. Using Theorem 5 we can assume that such an assignment is given, and that we only need to update the weights once.

Proof sketch (of Theorem 9). Let c be arbitrary. Thanks to Theorem 5 it suffices to show that there is a $d \in \mathbb{N}$ such that reachability is $(\mathsf{AC}^d, 1)$-maintainable by a dynamic program $\mathcal{P}$ under changes of size $\log^{c+d} n$. Let $d' \in \mathbb{N}$ be such that polynomially bounded skew-symmetric non-zero circulation weights for graphs from $\mathcal{G}$ can be computed in $\mathsf{AC}^{d'}$, and set $d \overset{\text{def}}{=} \max(2, d')$.

Let $G = (V, E)$ be a graph with n nodes. Let u be skew-symmetric non-zero circulation weights for G and let n^k be the polynomial bound on the weights. Further, let w be the weight assignment that gives weight $n^{k+2} + u(e)$ to each edge $e \in E$. Notice that w is polynomially bounded by n^{k+3} and isolates G according to Lemma 8.

The AC^d initialization computes, as auxiliary information, the weightings u and w and an n^b-approximation C of $(I - A_{(G,w)}(x))^{-1} \bmod 2$, that is, a matrix of formal polynomials in x that agree with the formal power series in $(I - A_{(G,w)}(x))^{-1} \bmod 2$ on the coefficients up to degree n^b. Here, $b \in \mathbb{N}$ is a constant to be determined later.

When changing G via a change ΔE with deletions E^- and insertions E^+, the dynamic program $\mathcal{P}$ handles deletions and insertions subsequently:

(1) Handling of deletions:
 - **(a)** Define isolating weights w^- for $G^- \overset{\text{def}}{=} (V, E \setminus E^-)$. The weights w^- will differ from w only for the at most $\log^{c+d} n$ edges in E^-.
 - **(b)** Compute an n^b-approximation of $(I - A_{(G^-,w^-)}(x))^{-1} \bmod 2$ using the existing n^b-approximation of $(I - A_{(G,w)}(x))^{-1} \bmod 2$.

(2) Handling of insertions:
 - **(a)** Define a family $W^{-/+}$ of weightings such that one member of the family is isolating for $G^{-/+} \overset{\text{def}}{=} (V, (E \setminus E^-) \cup E^+)$. All weightings of the family will differ from w^- only for the at most $\log^{c+d} n$ edges in E^+.
 - **(b)** Compute an n^b-approximation of $(I - A_{(G^{-/+},w^{-/+})}(x))^{-1} \bmod 2$ using the existing n^b-approximation of $(I - A_{(G^-,w^-)}(x))^{-1} \bmod 2$ for all members $w^{-/+}$ of $W^{-/+}$.

We first explain Steps (1a) and (2a) in more detail. For computing the isolating weights w^-, the program proceeds as follows. Skew-symmetric non-zero circulation weights u^- for G^- are obtained from the non-zero circulation weights u for G by setting the weight of deleted edges $e \in E^-$ to 0. As u^- gives the same weight to all simple cycles in G^- as u gives to these cycles in G, it has non-zero circulation. Now, the weight assignment w^- defined by $n^{k+2} + u^-(e)$ is isolating for G^- due to Lemma 8, and differs from w only for edges in E^-.

Computing the isolation weights for insertions is more challenging. In Lemma 11 below we show that from G^-, its transitive closure, and a set E^+ of edges of polylogarithmic size one can $\mathsf{FO}(\leq, +, \times)$-define a family $W^{-/+}$ of weight assignments such that one of these assignments is isolating for $G^{-/+}$.

For both Steps (1b) and (2b), the inverse of a matrix of polynomials over $\mathbb{Z}_2$ of polynomial degree needs to be updated after changing polylogarithmically many entries (i.e. entries corresponding to E^- and E^+, respectively). Inverses can be updated under such changes in

$\mathsf{FO{+}Mod}\,2(\leq,+,\times)$ due to Lemma 10 (see below) and the observation that changes ΔA of size $\log^{c+d} n$ to such a matrix can be decomposed into UBV as required by Lemma 10, see Lemma 7 in [7]. For Step (2b) this is done in parallel for all members of $W^{-/+}$.

For checking whether there is a path from s to t after the change ΔE to G, the dynamic program checks whether there is a member of $W^{-/+}$ such that the s-t-entry of $(I - A_{(G^{-/+},w^{-/+})}(x))^{-1} \bmod 2$ is non-zero. Since one member of $W^{-/+}$ is isolating, a path will be discovered this way. ◀

In the remainder of this subsection we show how inverses for matrices of polynomials can be updated under changes of polylogarithmic size, and how weights for inserted edges can be found.

The following lemma is obtained using the same techniques as in [7]. Here, $\mathbb{Z}_2[[x]]$ denotes the ring of formal power series with coefficients from $\mathbb{Z}_2$, and $\mathbb{Z}_2[x]$ denotes its subring that consists of all finite polynomials.

▶ **Lemma 10.** *Suppose $A \in \mathbb{Z}_2[[x]]^{n\times n}$ is invertible over $\mathbb{Z}_2[[x]]$, and $C \in \mathbb{Z}_2[x]^{n\times n}$ is an m-approximation of A^{-1}. If $A + \Delta A$ is invertible over $\mathbb{Z}_2[[x]]$ and ΔA can be written as UBV with $U \in \mathbb{Z}_2[x]^{n\times k}, B \in \mathbb{Z}_2[x]^{k\times k}$, and $V \in \mathbb{Z}_2[x]^{k\times n}$, then*

$$(A + \Delta A)^{-1} \approx_m C - CU(I + BVCU)^{-1}BVC$$

Furthermore, if $k \leq \log^c n$ for some fixed c and all involved polynomials have polynomial degree in n, then the right-hand side can be defined in $\mathsf{FO{+}Mod}\,2(\leq,+,\times)$ from C and ΔA.

Proof sketch. The correctness of the equation can be proved exactly as in Proposition 14 in [7] (there, this is proved for $\mathbb{Z}[[x]]$ instead of $\mathbb{Z}_2[[x]]$).

We argue that the right-hand side can be defined in $\mathsf{FO{+}Mod}\,2(\leq,+,\times)$. The involved matrix additions and multiplications modulo 2 can easily be expressed in $\mathsf{FO{+}Mod}\,2(\leq,+,\times)$, see [11]. It remains to explain how the inverse of the $\log^c n \times \log^c n$ matrix $I + BVCU$ can be found.

To this end, recall that the i-j-entry of the inverse of a matrix D is equal to $(-1)^{i+j}\frac{\det D_{ji}}{\det D}$, where D_{ji} is obtained from D by removing the j-th row and the i-th column.

So, it is sufficient to show that the determinant of a $\log^c n \times \log^c n$ matrix of polynomials with polynomial degree can be expressed modulo 2. In [7, Lemma 15] it was shown that such a determinant can be expressed in $\mathsf{FO{+}Maj}(\leq,+,\times)$, by observing that one only needs to be able to express the sum of polynomially many polynomials and the product of $\log^c n$ many polynomials. This observation is still valid for computing the determinant modulo 2 in $\mathsf{FO{+}Mod}\,2(\leq,+,\times)$. Both kind of computations are possible modulo 2 in $\mathsf{FO{+}Mod}\,2(\leq,+,\times)$ as well [11]. ◀

▶ **Lemma 11.** *Let $G = (V,E)$ be a graph and let $n = |V|$. Further, let w be a polynomially bounded isolating weight assignment for G, and let E^+ be a set of $\mathcal{O}(\log^c n)$ edges that is disjoint from E, for some $c \in \mathbb{N}$. Then there is a family W' of polynomially many polynomially bounded weight assignments such that*

1. *W' is $\mathsf{FO}(\leq,+,\times)$-definable from G, $\textsc{Reach}(G)$, E^+ and w,*
2. *all $w' \in W'$ agree with w on E,*
3. *at least one $w' \in W'$ is isolating for $(V, E \cup E^+)$.*

The proof works along the following lines. We use the approach from [18] to obtain weights for the inserted edges with the following idea: if there is an s-t-path that uses at least one inserted edge from E^+, then there is a unique minimal path under all s-t-paths that use at least one such edge, where we ignore the weight of the paths that is contributed by edges from E. We multiply these constructed weights for the edges from E^+ by a large polynomial to ensure that the combined weight assignment with the existing weights for edges in E is isolating for the graph $(V, E \cup E^+)$.

The approach from [18] does not lead to polynomially bounded weights in the size of the graph it is used for. We construct them for a graph with $N = \mathcal{O}(\log^c n)$ many nodes, and although they are not polynomially bounded in N, they are in n.

We now get to the details of the construction. In the following, we consider graphs with two sets of edges. An *adorned* graph $G = (V, E, F)$ has, besides the set E of *real* edges, a further set F of *fictitious* edges, which is not necessarily disjoint from E. For each pair s, t of nodes, let $\mathcal{P}'_{s,t}$ be the set of s-t-paths in G that use at least one real edge $e \in E$ and arbitrarily many fictitious edges $e' \in F$. Let $\mathcal{P}_{s,t}$ be the set of edge sequences that result from $\mathcal{P}'_{s,t}$ by removing the fictitious edges from the paths.

We say that a weight assignment w *real-isolates* G, if (1) it maps each real edge $e \in E$ to a positive integer, and (2) each non-empty $\mathcal{P}_{s,t}$ has a unique minimal element under w. In the following, we will need a stronger property. We say that w *strongly* real-isolates G, if in addition for each pair $\mathcal{P}_{s,t}$ and $\mathcal{P}_{s',t'}$ of non-empty sets with $(s,t) \neq (s',t')$ the unique minimal elements of $\mathcal{P}_{s,t}$ and $\mathcal{P}_{s',t'}$ have different weights under w.

The following lemma can be proved along the lines of [18], see the full version for details.

▶ **Lemma 12.** *There is a constant $\beta \in \mathbb{N}$ such that for every natural number N and every adorned graph $G = (V, E, F)$ with $V = \{1, \dots, N\}$ there is a sequence $\bar{p} = p_1, p_2, \dots, p_{\log N}$ of primes, each consisting of at most $(\beta - 2)\log N$ bits, such that the weight assignment*

$$w_{\bar{p}}(e) = \begin{cases} \sum_{j=1}^{\log N} N^{\beta(\log N - j)}(w_0(e) \bmod p_j) & e \in E \\ 0 & e \in F \end{cases}$$

strongly real-isolates G. Here, $w_0(u,v) \overset{def}{=} 2^{(N+1)u+v}$.

Using this lemma, we can prove Lemma 11.

Proof of Lemma 11. Let $V_{\text{aff}} \subseteq V$ be the set of nodes with edges in E^+. We construct an adorned graph $H = (V_H, E_H, F_H)$ with node set $V_H \overset{\text{def}}{=} V_{\text{aff}}$ as follows. The set E_H of real edges is $E_H \overset{\text{def}}{=} E^+$, and the set F_H of fictitious edges is $F_H \overset{\text{def}}{=} \{(u,v) \mid u, v \in V_H, (u,v) \in \textsc{Reach}(G)\}$. So, a fictitious edge (u,v) of H represents the existence of a u-v-path in G.

Let $\beta, \bar{p}$ and $w_{\bar{p}}$ be as promised to exist for H by Lemma 12. Further, let n^k be the upper bound on the weights of w. We define the weight assignment w' for $G' \overset{\text{def}}{=} (V, E \cup E^+)$ as follows.

- $w'(e) = w(e)$ for all $e \in E$,
- $w'(e) = n^{k+2} \cdot w_{\bar{p}}(e)$ for all $e \in E^+ = E_H$.

We show that w' isolates G' first, afterwards we show that w' is a member of an $\mathsf{FO}(\leq, +, \times)$-definable family of weightings.

For showing that w' isolates G' suppose, towards a contradiction, that there are two lightest simple s-t-paths π, ρ in G' with respect to w', for some nodes s and t. Let $\pi_1\pi_2\pi_3 = \pi$ and $\rho_1\rho_2\rho_3 = \rho$ be the subpaths of π and ρ such that edges from E^+ are only used in π_2 and ρ_2 and those subpaths are minimal with that property. Notice that both π_2 and ρ_2 are non-empty, as otherwise ρ and π are also lightest paths in G with respect to w, contradicting the assumption that w isolates G. Let π' and ρ' be the paths in H that correspond to π_2 and ρ_2, where subpaths of π_2 and ρ_2 are replaced by fictitious edges. We consider two cases.

If $w_{\bar{p}}(\pi') \neq w_{\bar{p}}(\rho')$, then the total weight of π and ρ contributed by the edges from E^+ differs by at least n^{k+2}. As the total weight contributed by the remaining edges is upper-bounded by n^{k+1}, we have that $w'(\pi) \neq w'(\rho)$, the desired contradiction.

Thus assume, without loss of generality, that $w_{\bar{p}}(\pi') = w_{\bar{p}}(\rho')$. We can assume that both paths π' and ρ' are lightest paths in H: if, say, π' is not a lightest path, then we can replace π_2 in π by a path that uses lighter edges from E^+, leading to an overall lighter path by the argument of the previous case. Then, because $w_{\bar{p}}$ strongly isolates H, the edges from $E_H = E^+$ used in π' and ρ' must be equal, and the same is true for π_2 and ρ_2. These edges must also be used in the same order, as otherwise a path with fewer edges from E^+ exists, which by the argument of previous case is lighter than both π and ρ. Because π and ρ are different paths, there must be subpaths π^* and ρ^* that consist only of edges from E and are both simple u-v-paths, for some nodes u and v. As w is isolating for G, not both subpaths can be lightest u-v-paths in G. Say, ρ^* is not such a lightest path. If we replace ρ^* in ρ by the lightest u-v-path in G, we obtain a path that is lighter than ρ, as w' agrees with w on E. So, ρ is not a lightest s-t-path in G' with respect to w', the desired contradiction. It follows that w' is isolating for G'.

The weight assignment $w_{\bar{p}}$ is clearly $\mathsf{FO}(\leq, +, \times)$-definable from $G, \textsc{Reach}(G), E^+, w$ and $\bar{p}$, as H is $\mathsf{FO}(\leq, +, \times)$-definable, the involved numbers consist of at most polylogarithmically many bits, and $\mathsf{FO}(\leq, +, \times)$ can express the necessary arithmetic on numbers of that magnitude (see [14, Theorem 5.1]). The sequence $\bar{p}$ consists of $\mathcal{O}(\log\log n)$ many primes (as H is of size polylog) which in turn are represented by $\mathcal{O}(\log\log n)$ many bits, because H has only polylogarithmic size in n. So, $\bar{p}$ can be represented by a tuple of nodes from V, and it follows that a family W' of weight assignments with $w' \in W'$ is $\mathsf{FO}(\leq, +, \times)$-definable from $G, \textsc{Reach}(G), E^+$ and w. ◀

6.3 Computing weights for bounded-treewidth graphs

In this section, we show that isolating weights for graphs of bounded treewidth can be computed in LOGSPACE. As an immediate consequence, reachability can be maintained for such graphs under changes of polylogarithmic size.

A *tree decomposition* $\mathcal{T} = (T, \mathcal{B})$ of a graph $G = (V, E)$ consists of a (rooted, directed) tree $T = (I, F, r)$, with (tree) nodes I, (tree) edges F, a distinguished root node $r \in I$, and a function $\mathcal{B}\colon I \to 2^V$ such that

(1) the set $\{i \in I \mid v \in \mathcal{B}(i)\}$ is non-empty for each node $v \in V$,
(2) there is an $i \in I$ with $\{u, v\} \subseteq \mathcal{B}(i)$ for each edge $(u, v) \in E$, and
(3) the subgraph $T[\{i \in I \mid v \in \mathcal{B}(i)\}]$ is connected for each node $v \in V$.

We refer to the number of children of a node i of T as its *degree*, and to the set $\mathcal{B}(i)$ as its *bag*. We denote the parent node of i by parent(i). The *width* of a tree decomposition is defined as the maximal size of a bag minus 1. The *treewidth* of a graph G is the minimal width among all tree decompositions of G. A tree decomposition is binary, if all tree nodes have degree at most 2. Its depth is the length of a longest path from the root r to a leaf of T. We inductively define the *height* $h(i)$ of i to be 1 if i is a leaf, and $h(i') + 1$ if i is an inner tree node and i' is a child of i with maximal height. For a node $v \in V$ we denote by $B(v)$ the highest bag that contains u, and let $h(u) \overset{\text{def}}{=} h(B(u))$. This bag $B(v)$ is well-defined for each node v thanks to condition (3) of the definition of a tree decomposition.

We usually identify tree nodes i and their bag $\mathcal{B}(i)$, and use the above notions and measures directly for bags. We also abuse notation and write $B \in \mathcal{B}$ if $B = \mathcal{B}(i)$ for some tree node i.

As a first step we construct isolation weights for bounded treewidth graphs that additionally have bounded degree. Isolation weights for all graphs of bounded treewidth are provided afterwards.

▶ **Proposition 13.** *Let $c, d, k \in \mathbb{N}$ be fixed. Let $G = (V, E)$ be a graph with maximal degree d, and let n be the number of its nodes. Let $\mathcal{T}$ be a binary tree decomposition of G with width k and depth at most $c \log n$. A polynomially bounded skew-symmetric weight assignment with non-zero circulation for G can be computed in* LOGSPACE.

Proof. The idea for assigning weights is the following. We associate each edge with one bag of the tree decomposition, namely the highest bag that contains one endpoint of the edge. For each bag B, we denote the set of all edges that are associated with B by $S(B)$. As the width of $\mathcal{T}$ and the degree of G are bounded by a constant, so is $|S(B)|$. An edge e is assigned a weight that depends exponentially on the height of the bag B it is associated with, and also exponentially on its position in some linear order on $S(B)$. For each cycle C there is a unique highest bag B_C that some of the cycle's edges is associated with. The idea for establishing non-zero circulation of C is that its weight is dominated by the weight of the unique edge which (1) is associated with B_C and (2) has largest index in the linear order on $S(B)$ among all edges of the cycle. As the height of a bag is logarithmic in n and $|S(B)|$ is bounded by a constant, the weight of every edge is polynomial in n.

We now proceed to the details. For each $e \in \vec{E}$, let B_e be the (unique) highest bag that contains one of the end points of e. For a bag B, define the set $S(B)$ of its associated edges as $S(B) \overset{\text{def}}{=} \{e \in \vec{E} \mid B = B_e\}$. Observe that the sets $S(B)$ partition the set $\vec{E}$ of edges and that the size of $S(B)$ is bounded by a constant $\beta \overset{\text{def}}{=} 2d(k+1)$, as each bag B contains at most $k+1$ nodes and each node has degree at most d in G and therefore degree at most $2d$ in $\vec{G}$. For each $S(B)$ we fix an enumeration of its elements[3]. Now, for each edge e, we set $h(e) = h(B_e)$ and $\ell(e) = i$, if e is the i-th element in the enumeration of $S(B_e)$.

We set the weight $w(e)$ of an edge $e = (u, v)$ with $u \leq v$ to be $w(e) \overset{\text{def}}{=} (4\beta \cdot 3^\beta + 2)^{h(e)} \cdot 3^{\ell(e)}$. The weight of an edge (u, v) with $u > v$ is $w(u, v) = -w(v, u)$. Notice that this weight assignment is polynomially bounded and skew-symmetric and can be computed in LOGSPACE. We now show that it has non-zero circulation.

Let $\mathcal{C}$ be any simple cycle in $\vec{G}$, and let $e_1, \ldots, e_m$ be an enumeration of its edges. Without loss of generality we assume that e_1 is the edge with the maximal weight among all edges in $\mathcal{C}$. This edge is well-defined, as there is a unique highest bag B such that $S(B)$ contains an edge of $\mathcal{C}$, and the term $4\beta \cdot 3^\beta + 2$ is strictly greater than $3^{\ell(e)}$ for any value of $\ell(e)$.

We show $|w(e_1)| > |w(e_2) + \cdots + w(e_m)|$, which implies the claim. Actually, we show that the weight of $w(e_1)$ exceeds the combined weight of all other edges e that are either in $S(B)$ and have $\ell(e) < \ell(e_1)$ or are in $S(B')$ for some bag B' below B in the tree decomposition. Note that there are $\sum_{h=1}^{h(e_1)-1} 2^{h(e_1)-h}$ many of those bags B', each $S(B')$ contains at most β edges, and the weight of each edge is upper bounded by $(4\beta \cdot 3^\beta + 2)^{h(B')} \cdot 3^\beta$.

$$\begin{aligned}
&|w(e_2) + \cdots + w(e_m)| \\
&\quad < \sum_{i=1}^{l(e_1)-1} (4\beta \cdot 3^\beta + 2)^{h(e_1)} \cdot 3^i + \sum_{h=1}^{h(e_1)-1} 2^{h(e_1)-h} \cdot \beta \cdot (4\beta \cdot 3^\beta + 2)^h \cdot 3^\beta \\
&\quad = \sum_{i=1}^{l(e_1)-1} (4\beta \cdot 3^\beta + 2)^{h(e_1)} \cdot 3^i + \sum_{h=1}^{h(e_1)-1} 2^{h(e_1)} \cdot \beta \cdot (2\beta \cdot 3^\beta + 1)^h \cdot 3^\beta
\end{aligned}$$

[3] As we devise an LOGSPACE algorithm, we can assume the existence of a linear order on the input.

$$
\begin{aligned}
&= (4\beta \cdot 3^{\beta} + 2)^{h(e_1)} \cdot \sum_{i=1}^{l(e_1)-1} 3^i + 2^{h(e_1)} \cdot \beta \cdot 3^{\beta} \cdot \sum_{h=1}^{h(e_1)-1} (2\beta \cdot 3^{\beta} + 1)^h \\
&= (4\beta \cdot 3^{\beta} + 2)^{h(e_1)} \cdot \frac{3^{l(e_1)} - 3}{2} + 2^{h(e_1)} \cdot \beta \cdot 3^{\beta} \cdot \frac{(2\beta \cdot 3^{\beta} + 1)^{h(e_1)} - (2\beta \cdot 3^{\beta} + 1)}{2\beta \cdot 3^{\beta}} \\
&< (4\beta \cdot 3^{\beta} + 2)^{h(e_1)} \cdot 3^{\ell(e_1)} = |w(e_1)|
\end{aligned}
$$
◀

Non-zero circulation weights cannot only be computed in LOGSPACE for bounded-treewidth graphs with bounded degree, as given by Proposition 13, but also for all graphs with bounded treewidth. Also, using a result of Elberfeld, Jakoby and Tantau [10], no tree decomposition needs to be given as input.

▶ **Theorem 14.** *Let $k \in \mathbb{N}$ be fixed and let $G = (V, E)$ be a graph with treewidth at most k. A polynomially bounded skew-symmetric weight assignment with non-zero circulation for G can be computed in* LOGSPACE.

The idea for proving Theorem 14 is as follows. From a given graph G with treewidth at most k we construct a graph G' with treewidth and degree $\mathcal{O}(k)$ as well as a tree decomposition. The graph G' basically results from a tree decomposition $\mathcal{T}$ of G by making a copy of a node v for every bag of $\mathcal{T}$ that contains v. These copies are connected by an edge if the corresponding bags in $\mathcal{T}$ are. Using Proposition 13, we obtain non-zero circulation weights for G', and we show that they can be translated to non-zero circulation weights for G.

Proof. Fix $k \in \mathbb{N}$ and let $G = (V, E)$ be a graph with treewidth at most k. Let n be the size of V. There are constants $c_1, c_2 \in \mathbb{N}$ that only depend on k such that a binary tree decomposition $\mathcal{T} = (T, \mathcal{B})$ of G of width at most $c_1 k$ and depth at most $c_2 \log n$ can be computed in LOGSPACE [10]. From G and $\mathcal{T}$ we construct a graph $G' = (V', E')$ as follows. Let V' be the set $V' \overset{\text{def}}{=} \{v_B \mid B \in \mathcal{B}, v \in B\}$ and let $E' \overset{\text{def}}{=} \{(v_B, v_{B'}) \mid B' = \text{parent}(B)\} \cup \{(u_B, v_B) \mid (u, v) \in E, u \notin \text{parent}(B) \text{ or } v \notin \text{parent}(B)\}$. So, we have one copy v_B of a node $v \in V$ for each bag B such that v is contained in B. Two copies of a node are connected by an edge if they originate from adjacent bags in the tree decomposition, and there is an edge between two copies u_B and v_B, originating from the same bag B, if B is the highest bag of $\mathcal{T}$ that contains both endpoints u and v.

The degree of G' is bounded by $c_1 k + 3$. The tree decomposition $\mathcal{T}' = (T, \mathcal{B}')$ that replaces each bag B of $\mathcal{T}$ by $\{v_B \mid v \in B\} \cup \{v_{\text{parent}(B)} \mid v \in \text{parent}(B)\}$ is a tree decomposition of G' and has width at most $2c_1 k + 1$. Furthermore, it is binary and has depth at most $c_2 \log n$. So, by Proposition 13, one can compute in LOGSPACE polynomially bounded, skew-symmetric non-zero circulation weights w' for G'.

We construct a weight function w for $\overleftrightarrow{G}$ as follows. For that, we associate with each edge $(u, v) \in \overleftrightarrow{E}$ a sequence $P(u, v)$ of edges in $\overleftrightarrow{G'}$. Recall that for each edge (u, v) there is a highest bag in which both u and v appear. The bag above that bag contains either (a) none of the two vertices, or (b) v but not u, or (c) u but not v. The definition of $P(u, v)$ distinguishes these three cases:

1. Suppose $B(u) = B(v)$. We set $P(u, v) = (u_{B(u)}, v_{B(v)})$.
2. Suppose $B(u)$ is a proper descendant of $B(v)$. Let $B = B(u)$ and $B' = B(v)$. We set $P(u, v) = (u_B, v_B), (v_B, v_{\text{parent}(B)}), \ldots, (v_{\text{parent}(\cdots(\text{parent}(B)))}, v_{B'})$.
3. Suppose $B(v)$ is a proper descendant of $B(u)$. Let $B = B(u)$ and $B' = B(v)$. We set $P(u, v) = (u_B, u_{\text{parent}(\cdots(\text{parent}(B')))}), \ldots, (u_{\text{parent}(B')}, u_{B'}), (u_{B'}, v_{B'})$.

Now, let $w(u, v)$ be the sum $\sum_{e \in P(u,v)} w'(e)$ of the weights of the edges e in $P(u, v)$. Because w' is a polynomially bounded skew-symmetric weight assignment, so is w.

It remains to show that w has non-zero circulation. Let $\mathcal{C}$ be an arbitrary simple cycle in $\overleftrightarrow{G}$. We need to show that w assigns a non-zero weight to $\mathcal{C}$. Let $e_1, \dots, e_m$ be the sequence of edges that constitutes $\mathcal{C}$, and let $\mathcal{W}'$ be the sequence $P(e_1), \dots, P(e_m)$ of edges. By definition, the weight of $\mathcal{C}$ under w is the same as the weight of $\mathcal{W}'$ under w'.

Note that $\mathcal{W}'$ constitutes a cycle in $\overleftrightarrow{G}'$ which is not necessarily simple: some nodes might be visited more than once. We show that we can construct from $\mathcal{W}'$ a simple cycle $\mathcal{C}'$ by removing parts of $\mathcal{W}'$ with total weight 0. As a result, $\mathcal{C}'$ has the same weight as $\mathcal{W}'$ under w'. Because w' has non-zero circulation, the weight of $\mathcal{C}'$ and $\mathcal{W}'$ is non-zero, and so is the weight of $\mathcal{C}$ under w.

Suppose that some node u_B is visited twice by $\mathcal{W}'$. Then $\mathcal{W}'$ has the subsequence $P(v,u)P(u,v')$ for some nodes v and v', because u is visited only once in $\mathcal{C}$ and no node appears twice in a single sequence $P(u,u')$. Moreover, it most be that $h(u)$ is greater than both $h(v)$ and $h(v')$, or smaller than both $h(v)$ and $h(v')$, and either $B(v)$ is a descendent of $B(v')$ or $B(v')$ is a descendant of $B(v)$. We consider the case that $h(u)$ is greater than both $h(v)$ and $h(v')$, and $B(v')$ is a descendant of $B(v)$. The other cases are analogous. Then $P(v,u)P(u,v')$ visits the nodes $v_{B(v)}, u_{B(v)}, u_{\text{parent}(B(v))}, \dots, u_{B(u)}, \dots u_{\text{parent}(B(v))}, u_{B(v)}, \dots, u_{B(v')}, v'_{B(v')}$ in that order. The closed walk from $u_{B(v)}$ to $u_{B(u)}$ and back to $u_{B(v)}$ has, because of skew-symmetry, a total weight of 0 under w'. So, the corresponding edges can be removed from $\mathcal{W}'$ without changing the weight. Repeating this step results in a simple cycle $\mathcal{C}'$ with the same weight under w' as $\mathcal{C}$ under w. As w' has non-zero circulation, the weight of $\mathcal{C}'$ is non-zero, and so is the weight of $\mathcal{C}$. ◀

7 Conclusion

The complexity of maintaining (variants of) the reachability query is the dominant research question in dynamic complexity theory. With this paper we basically settle this question for reachability in undirected graphs, at least with respect to the size of a change: reachability in undirected graphs is in DynFO$(\leq, +, \times)$ if and only if the changes have at most polylogarithmic size. For reachability in directed graphs, we can only show this for insertions of polylogarithmic size, and the main open problem is whether this can be extended to also allow for deletions of single edges, non-constantly many edges, or even polylogarithmically many edges.

We give preliminary results for classes of graphs for which non-zero circulation weights can be computed in AC: reachability for these graphs is in DynFO+Mod 2$(\leq, +, \times)$ under insertions and deletions of polylogarithmic size. We show that one can compute such weight assignments for graphs with bounded treewidth. Other graph classes for which this is possible include the class of planar graphs [23], and in general all graphs with bounded genus, which one can show using results from [5].

A question for further research is whether reachability for classes of directed graphs can be maintained in DynFO$(\leq, +, \times)$ under insertions and deletions of polylogarithmic size. Candidate classes are graphs with bounded treewidth, and directed acyclic graphs.

References

1 David A. Mix Barrington, Neil Immerman, and Howard Straubing. On uniformity within NC^1. *J. Comput. Syst. Sci.*, 41(3):274–306, 1990. doi:10.1016/0022-0000(90)90022-D.

2 Xi Chen, Igor Carboni Oliveira, Rocco A. Servedio, and Li-Yang Tan. Near-optimal small-depth lower bounds for small distance connectivity. In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 612–625. ACM, 2016. doi:10.1145/2897518.2897534.

3 Samir Datta, William Hesse, and Raghav Kulkarni. Dynamic complexity of directed reachability and other problems. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, volume 8572 of *Lecture Notes in Computer Science*, pages 356–367. Springer, 2014. doi:10.1007/978-3-662-43948-7_30.

4 Samir Datta, Raghav Kulkarni, Anish Mukherjee, Thomas Schwentick, and Thomas Zeume. Reachability is in DynFO. *J. ACM*, 65(5):33:1–33:24, August 2018. doi:10.1145/3212685.

5 Samir Datta, Raghav Kulkarni, Raghunath Tewari, and N. V. Vinodchandran. Space complexity of perfect matching in bounded genus bipartite graphs. *J. Comput. Syst. Sci.*, 78(3):765–779, 2012. doi:10.1016/j.jcss.2011.11.002.

6 Samir Datta, Anish Mukherjee, Thomas Schwentick, Nils Vortmeier, and Thomas Zeume. A Strategy for Dynamic Programs: Start over and Muddle through. *Logical Methods in Computer Science*, Volume 15, Issue 2, May 2019. doi:10.23638/LMCS-15(2:12)2019.

7 Samir Datta, Anish Mukherjee, Nils Vortmeier, and Thomas Zeume. Reachability and distances under multiple changes. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 120:1–120:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.120.

8 Samir Datta, Anish Mukherjee, Nils Vortmeier, and Thomas Zeume. Reachability and distances under multiple changes. *CoRR*, abs/1804.08555, 2018. arXiv:1804.08555.

9 Guozhu Dong and Jianwen Su. Arity bounds in first-order incremental evaluation and definition of polynomial time database queries. *J. Comput. Syst. Sci.*, 57(3):289–308, 1998. doi:10.1006/jcss.1998.1565.

10 Michael Elberfeld, Andreas Jakoby, and Till Tantau. Logspace versions of the theorems of Bodlaender and Courcelle. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 143–152. IEEE Computer Society, 2010. doi:10.1109/FOCS.2010.21.

11 Alexander Healy and Emanuele Viola. Constant-depth circuits for arithmetic in finite fields of characteristic two. In Bruno Durand and Wolfgang Thomas, editors, *STACS 2006, 23rd Annual Symposium on Theoretical Aspects of Computer Science, Marseille, France, February 23-25, 2006, Proceedings*, volume 3884 of *Lecture Notes in Computer Science*, pages 672–683. Springer, 2006. doi:10.1007/11672142_55.

12 Harold V Henderson and Shayle R Searle. On deriving the inverse of a sum of matrices. *Siam Review*, 23(1):53–60, 1981.

13 William Hesse. The dynamic complexity of transitive closure is in DynTC^0. *Theor. Comput. Sci.*, 296(3):473–485, 2003. doi:10.1016/S0304-3975(02)00740-5.

14 William Hesse, Eric Allender, and David A. Mix Barrington. Uniform constant-depth threshold circuits for division and iterated multiplication. *J. Comput. Syst. Sci.*, 65(4):695–716, 2002. doi:10.1016/S0022-0000(02)00025-9.

15 Neil Immerman. Nondeterministic space is closed under complementation. *SIAM J. Comput.*, 17(5):935–938, 1988. doi:10.1137/0217058.

16 Neil Immerman. *Descriptive complexity.* Graduate texts in computer science. Springer, 1999. doi:10.1007/978-1-4612-0539-5.

17 Stasys Jukna. *Boolean function complexity: advances and frontiers*, volume 27. Springer Science & Business Media, 2012.

18 Vivek Anand T. Kallampally and Raghunath Tewari. Trading determinism for time in space bounded computations. In *41st International Symposium on Mathematical Foundations of Computer Science, MFCS 2016, August 22-26, 2016 - Kraków, Poland*, pages 10:1–10:13, 2016. doi:10.4230/LIPIcs.MFCS.2016.10.

19 Sushant Patnaik and Neil Immerman. Dyn-FO: A parallel, dynamic complexity class. *J. Comput. Syst. Sci.*, 55(2):199–209, 1997. doi:10.1006/jcss.1997.1520.

20 Thomas Schwentick and Thomas Zeume. Dynamic complexity: recent updates. *SIGLOG News*, 3(2):30–52, 2016. doi:10.1145/2948896.2948899.

21 Roman Smolensky. Algebraic methods in the theory of lower bounds for boolean circuit complexity. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 77–82, 1987. doi:10.1145/28395.28404.

22 Róbert Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Inf.*, 26(3):279–284, 1988. doi:10.1007/BF00299636.

23 Raghunath Tewari and N. V. Vinodchandran. Green's theorem and isolation in planar graphs. *Inf. Comput.*, 215:1–7, 2012. doi:10.1016/j.ic.2012.03.002.

24 Nils Vortmeier. *Dynamic expressibility under complex changes*. PhD thesis, TU Dortmund University, Germany, 2019. doi:10.17877/DE290R-20434.

The Strahler Number of a Parity Game

Laure Daviaud
CitAI, Department of Computer Science, City, University of London, UK
Laure.Daviaud@city.ac.uk

Marcin Jurdziński
Department of Computer Science, University of Warwick, Coventry, UK
Marcin.Jurdzinski@warwick.ac.uk

K. S. Thejaswini
Department of Computer Science, University of Warwick, Coventry, UK
Thejaswini.Raghavan.1@warwick.ac.uk

Abstract

The Strahler number of a rooted tree is the largest height of a perfect binary tree that is its minor. The Strahler number of a parity game is proposed to be defined as the smallest Strahler number of the tree of any of its attractor decompositions. It is proved that parity games can be solved in quasi-linear space and in time that is polynomial in the number of vertices n and linear in $(d/2k)^k$, where d is the number of priorities and k is the Strahler number. This complexity is quasi-polynomial because the Strahler number is at most logarithmic in the number of vertices. The proof is based on a new construction of small Strahler-universal trees.

It is shown that the Strahler number of a parity game is a robust, and hence arguably natural, parameter: it coincides with its alternative version based on trees of progress measures and – remarkably – with the register number defined by Lehtinen (2018). It follows that parity games can be solved in quasi-linear space and in time that is polynomial in the number of vertices and linear in $(d/2k)^k$, where k is the register number. This significantly improves the running times and space achieved for parity games of bounded register number by Lehtinen (2018) and by Parys (2020).

The running time of the algorithm based on small Strahler-universal trees yields a novel trade-off $k \cdot \lg(d/k) = O(\log n)$ between the two natural parameters that measure the structural complexity of a parity game, which allows solving parity games in polynomial time. This includes as special cases the asymptotic settings of those parameters covered by the results of Calude, Jain Khoussainov, Li, and Stephan (2017), of Jurdziński and Lazić (2017), and of Lehtinen (2018), and it significantly extends the range of such settings, for example to $d = 2^{O\left(\sqrt{\lg n}\right)}$ and $k = O\left(\sqrt{\lg n}\right)$.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory; Theory of computation → Theory and algorithms for application domains; Theory of computation → Design and analysis of algorithms; Theory of computation → Logic and verification

Keywords and phrases parity game, attractor decomposition, progress measure, universal tree, Strahler number

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.123

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

Related Version A full version of the paper is available at [7], `https://arxiv.org/abs/2003.08627`.

Funding This work has been supported by the EPSRC grant EP/P020992/1 (Solving Parity Games in Theory and Practice). It has started when the third author was supported by the Department of Computer Science at the University of Warwick as a visiting student from Chennai Mathematical Institute, India.

Acknowledgements We thank colleagues in the following human chain for contributing to our education on the Strahler number of a tree: the first and the second authors have learnt it from the third author, who has learnt it from K Narayan Kumar, who has learnt it from Dmitry Chistikov,

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 123; pp. 123:1–123:19
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

who has learnt it from Radu Iosif and Stefan Kiefer at a workshop on Infinite-State Systems, which has been organized and hosted by Joël Ouaknine and Prakash Panangaden at Bellairs Research Institute.

1 Context

Parity Games. Parity games are a fundamental model in automata theory and logic [8, 32, 17, 2], and their applications to verification, program analysis, and synthesis. In particular, they are intimately linked to the problems of emptiness and complementation of non-deterministic automata on trees [8, 32], model checking and satisfiability of fixpoint logics [9, 2], and evaluation of nested fixpoint expressions [1, 18]. It is a long-standing open problem whether parity games can be solved in polynomial time [9].

The impact of parity games goes well beyond their home turf of automata theory, logic, and formal methods. For example, an answer [14] of a question posed originally for parity games [31] has strongly inspired major breakthroughs on the computational complexity of fundamental algorithms in stochastic planning [12] and linear optimization [15, 16].

Strahler Number. The Strahler number has been proposed by Horton (1945) and made rigorous by Strahler (1952), in their morphological study of river networks in hydrogeology. It has been also studied in other sciences, such as botany, anatomy, neurophysiology, physics, and molecular biology, where branching patterns appear. The Strahler number has been identified in computer science by Ershov [10] as the smallest number of registers needed to evaluate an arithmetic expression. It has since been rediscovered many times in various areas of computer science; see the surveys of Knuth [23], Viennot [30], and Esparza, Luttenberger, and Schlund [11].

Related Work. A major breakthrough in the quest for a polynomial-time algorithm for parity games was achieved by Calude, Jain, Khoussainov, Li, and Stephan [3], who have given the first quasi-polynomial algorithm. Other quasi-polynomial algorithms have been developed soon after by Jurdziński and Lazić [20], and Lehtinen [24]. Czerwiński, Daviaud, Fijalkow, Jurdziński, Lazić, and Parys [4] have introduced the concepts of *universal trees* and *separating automata*, and argued that all the aforementioned quasi-polynomial algorithms were intimately linked to them.

By establishing a quasi-polynomial lower bound on the size of universal trees, Czerwiński et al. have highlighted the fundamental limitations of the above approaches, motivating further the study of the attractor decomposition algorithm due to McNaughton [27] and Zielonka [32]. Parys [28] has proposed an ingenious quasi-polynomial version of McNaughton-Zielonka algorithm, but Lehtinen, Schewe, and Wojtczak [26], and Jurdziński and Morvan [21] have again strongly linked all quasi-polynomial variants of the attractor decomposition algorithm to universal trees.

Among several prominent quasi-polynomial algorithms for parity games, Lehtinen's approach [24] has relatively least attractive worst-case running time bounds. Parys [29] has offered some running-time improvements to Lehtinen's algorithm, but it remains significantly worse than state-of-the-art bounds of Jurdziński and Lazić [20], and Fearnley, Jain, de Keijzer, Schewe, Stephan, and Wojtczak [13], in particular because it always requires at least quasi-polynomial working space.

Our Contributions. We propose the Strahler number as a parameter that measures the structural complexity of dominia in a parity game and that governs the computational complexity of the most efficient algorithms currently known for solving parity games. We establish that the Strahler number is a robust, and hence natural, parameter by proving that it coincides with its version based on trees of progress measures and with the register number defined by Lehtinen [24].

We give a construction of small Strahler-universal trees that, when used with the progress measure lifting algorithm [19, 20] or with the universal attractor decomposition algorithm [21], yield algorithms that work in quasi-linear space and quasi-polynomial time. Moreover, usage of our small Strahler-universal trees allows to solve parity games in polynomial time for a wider range of asymptotic settings of the two natural structural complexity parameters (number of priorities d and the Strahler/register number k) than previously known, and that covers as special cases the $k = O(1)$ criterion of Lehtinen [24] and the $d < \lg n$ and $d = O(\log n)$ criteria of of Calude et al. [3], and of Jurdziński and Lazić [20], respectively.

Proofs. Proofs of some of our technical results can be found in the long version of this extended abstract available on arXiv [7].

2 Dominions, Attractor Decompositions, and Their Trees

Strategies, Traps, and Dominions. A *parity game* [8] $\mathcal{G}$ consists of a finite directed graph (V, E), a partition $(V_{\text{Even}}, V_{\text{Odd}})$ of the set of vertices V, and a function $\pi : V \to \{0, 1, \ldots, d\}$ that labels every vertex $v \in V$ with a non-negative integer $\pi(v)$ called its *priority*. We say that a cycle is *even* if the highest vertex priority on the cycle is even; otherwise the cycle is *odd*. We say that a parity game is (n, d)*-small* if it has at most n vertices and all vertex priorities are at most d.

For a set S of vertices, we write $\mathcal{G} \cap S$ for the substructure of $\mathcal{G}$ whose graph is the subgraph of (V, E) induced by the sets of vertices S. Sometimes, we also write $\mathcal{G} \setminus S$ to denote $\mathcal{G} \cap (V \setminus S)$. We assume throughout that every vertex has at least one outgoing edge, and we reserve the term *subgame* to substructures $\mathcal{G} \cap S$, such that every vertex in the subgraph of (V, E) induced by S has at least one outgoing edge.

A (positional) *Steven strategy* is a set $\sigma \subseteq E$ of edges such that:

- for every $v \in V_{\text{Even}}$, there is an edge $(v, u) \in \sigma$,
- for every $v \in V_{\text{Odd}}$, if $(v, u) \in E$ then $(v, u) \in \sigma$.

For a non-empty set of vertices R, we say that a Steven strategy σ *traps Audrey in* R if $w \in R$ and $(w, u) \in \sigma$ imply $u \in R$. We say that a set of vertices R is a *trap for Audrey* [32] if there is a Steven strategy that traps Audrey in R. Observe that if R is a trap in a game $\mathcal{G}$ then $\mathcal{G} \cap R$ is a subgame of $\mathcal{G}$. For a set of vertices $D \subseteq V$, we say that a Steven strategy σ is a *Steven dominion strategy on* D if σ traps Audrey in D and every cycle in the subgraph (D, σ) is even. Finally, we say that a set D of vertices is a *Steven dominion* [22] if there is a Steven dominion strategy on it.

Audrey strategies, trapping Steven, and Audrey dominions are defined in an analogous way by swapping the roles of the two players. We note that the sets of Steven dominions and of Audrey dominions are each closed under union, and hence the largest Steven and Audrey dominions exist, and they are the unions of all Steven and Audrey dominions, respectively. Moreover, every Steven dominion is disjoint from every Audrey dominion.

Attractor Decompositions. In a parity game $\mathcal{G}$, for a target set of vertices B ("bullseye") and a set of vertices A such that $B \subseteq A$, we say that a Steven strategy σ is a *Steven reachability strategy to B from A* if every infinite path in the subgraph (V, σ) that starts from a vertex in A contains at least one vertex in B.

For every target set B, there is the largest (with respect to set inclusion) set from which there is a Steven reachability strategy to B in $\mathcal{G}$; we call this set the *Steven attractor to B in $\mathcal{G}$* [32]. *Audrey reachability strategies* and *Audrey attractors* are defined analogously. We highlight the simple fact that if A is an attractor for a player in $\mathcal{G}$ then its complement $V \setminus A$ is a trap for them.

If $\mathcal{G}$ is a parity game in which all priorities do not exceed a non-negative even number d then we say that $\mathcal{H} = \langle A, (S_1, \mathcal{H}_1, A_1), \ldots, (S_\ell, \mathcal{H}_\ell, A_\ell)\rangle$ is a *Steven d-attractor decomposition* [5, 6, 21] of $\mathcal{G}$ if:

- A is the Steven attractor to the (possibly empty) set of vertices of priority d in $\mathcal{G}$;

and setting $\mathcal{G}_1 = \mathcal{G} \setminus A$, for all $i = 1, 2, \ldots, \ell$, we have:

- S_i is a non-empty trap for Audrey in $\mathcal{G}_i$ in which every vertex priority is at most $d - 2$;
- $\mathcal{H}_i$ is a Steven $(d-2)$-attractor decomposition of subgame $\mathcal{G} \cap S_i$;
- A_i is the Steven attractor to S_i in $\mathcal{G}_i$;
- $\mathcal{G}_{i+1} = \mathcal{G}_i \setminus A_i$;

and the game $\mathcal{G}_{\ell+1}$ is empty. If $d = 0$ then we require that $\ell = 0$.

The following proposition states that if a subgame induced by a trap for Audrey has a Steven attractor decomposition then the trap is a Steven dominion. Indeed, a routine proof argues that the union of all the Steven reachability strategies, implicit in the attractors listed in the decomposition, is a Steven dominion strategy.

▶ **Proposition 1** ([32, 5, 21]). *If d is even, R is a trap for Audrey in $\mathcal{G}$, and there is a Steven d-attractor decomposition of $\mathcal{G} \cap R$, then R is a Steven dominion in $\mathcal{G}$.*

Attractor decompositions for Audrey can be defined in the analogous way by swapping the roles of players as expected, and then a dual version of the proposition holds routinely.

The following theorem implies that every vertex in a parity game is either in the largest Steven dominion or in the largest Audrey dominion – it is often referred to as the *positional determinacy theorem* for parity games.

▶ **Theorem 2** ([8, 27, 32, 21]). *For every parity game $\mathcal{G}$, there is a partition of the set of vertices into a trap for Audrey W_{Even} and a trap for Steven W_{Odd}, such that there is a Steven attractor decomposition of $\mathcal{G} \cap W_{\text{Even}}$ and an Audrey attractor decomposition of $\mathcal{G} \cap W_{\text{Odd}}$.*

Ordered Trees and Their Strahler Numbers. Ordered trees are defined inductively; the trivial tree $\langle\rangle$ is an ordered tree and so is a sequence $\langle T_1, T_2, \ldots, T_\ell\rangle$, where T_i is an ordered tree for every $i = 1, 2, \ldots, \ell$. The trivial tree has only one node called the root, which is a leaf; and a tree of the form $\langle T_1, T_2, \ldots, T_\ell\rangle$ has the root with k children, the root is not a leaf, and the i-th child of the root is the root of ordered tree T_i.

Because the trivial tree $\langle\rangle$ has just one node, we sometimes write $\circ$ to denote it. If T is an ordered tree and i is a positive integer, then we use the notation T^i to denote the sequence $T, T, \ldots, T$ consisting of i copies of tree T. Then the expression $\left\langle T^i \right\rangle = \langle T, \ldots, T\rangle$ denotes the tree whose root has i children, each of which is the root of a copy of T. We also use the $\cdot$ symbol to denote concatenation of sequences, which in the context of ordered trees can be interpreted as sequential composition of trees by merging their roots; for example, $\langle\langle\circ^3\rangle\rangle \cdot \left\langle \circ^4, \langle\langle\circ\rangle\rangle^2 \right\rangle = \left\langle \langle\circ^3\rangle, \circ^4, \langle\langle\circ\rangle\rangle^2 \right\rangle = \langle\langle\circ, \circ, \circ\rangle, \circ, \circ, \circ, \circ, \langle\langle\circ\rangle\rangle, \langle\langle\circ\rangle\rangle\rangle$.

For an ordered tree T, we write $\operatorname{height}(T)$ for its *height* and $\operatorname{leaves}(T)$ for its *number of leaves*, which are defined by the following routine induction: the trivial tree $\langle\rangle = \circ$ has 1 leaf and its height is 1; the number of leaves of tree $\langle T_1, T_2, \ldots, T_\ell\rangle$ is the sum of the numbers of leaves of trees T_1, T_2, ..., T_ℓ; and its height is 1 plus the maximum height of trees T_1, T_2, ..., T_ℓ. For example, the tree $\left\langle\left\langle\circ^3\right\rangle, \circ^4, \langle\langle\circ\rangle\rangle^2\right\rangle$ has 9 leaves and height 4 We say that an ordered tree is (n, h)*-small* if it has at most n leaves and its height is at most h.

The *Strahler number* $\operatorname{Str}(T)$ of a tree T is defined to be the largest height of a perfect binary tree that is a minor of T. Alternatively, it can be defined by the following structural induction: the Strahler number of the trivial tree $\langle\rangle = \circ$ is 1; and if $T = \langle T_1, \ldots, T_\ell\rangle$ and m is the largest Strahler number of trees $T_1, \ldots, T_\ell$, then $\operatorname{Str}(T) = m$ if there is a unique i such that $\operatorname{Str}(T_i) = m$, and $\operatorname{Str}(T) = m+1$ otherwise. For example, we have $\operatorname{Str}\left(\left\langle\left\langle\circ^3\right\rangle, \circ^4, \langle\langle\circ\rangle\rangle^2\right\rangle\right) = 2$ because $\operatorname{Str}(\circ) = \operatorname{Str}(\langle\langle\circ\rangle\rangle) = 1$ and $\operatorname{Str}\left(\left\langle\circ^3\right\rangle\right) = 2$.

▶ **Proposition 3.** *For every (n, h)-small tree T, we have* $\operatorname{Str}(T) \leq h$ *and* $\operatorname{Str}(T) \leq \lfloor\lg n\rfloor + 1$.

Trees of Attractor Decompositions. The definition of an attractor decomposition is inductive and we define an ordered tree that reflects the hierarchical structure of an attractor decomposition. If d is even and $\mathcal{H} = \langle A, (S_1, \mathcal{H}_1, A_1), \ldots, (S_\ell, \mathcal{H}_\ell, A_\ell)\rangle$ is a Steven d-attractor decomposition then we define the *tree of attractor decomposition* $\mathcal{H}$ [6, 21], denoted by $T_{\mathcal{H}}$, to be the trivial ordered tree $\langle\rangle$ if $\ell = 0$, and otherwise, to be the ordered tree $\langle T_{\mathcal{H}_1}, T_{\mathcal{H}_2}, \ldots, T_{\mathcal{H}_\ell}\rangle$, where for every $i = 1, 2, \ldots, \ell$, tree $T_{\mathcal{H}_i}$ is the tree of attractor decomposition $\mathcal{H}_i$. Trees of Audrey attractor decompositions are defined analogously.

Observe that the sets $S_1, S_2, \ldots, S_\ell$ in an attractor decomposition as above are non-empty and pairwise disjoint, which implies that trees of attractor decompositions are small relative to the number of vertices and the number of distinct priorities in a parity game. The following proposition can be proved by routine structural induction.

▶ **Proposition 4** ([6, 21])**.** *If $\mathcal{H}$ is an attractor decomposition of an (n, d)-small parity game then its tree $T_{\mathcal{H}}$ is $(n, \lceil d/2\rceil + 1)$-small.*

We define the *Strahler number of an attractor decomposition* $\mathcal{H}$, denoted by $\operatorname{Str}(\mathcal{H})$, to be the Strahler number $\operatorname{Str}(T_{\mathcal{H}})$ of its tree $T_{\mathcal{H}}$. We define the *Strahler number of a parity game* to be the maximum of the smallest Strahler numbers of attractor decompositions of the largest Steven and Audrey dominions, respectively.

3 Strahler Strategies in Register Games

This section establishes a connection between the register number of a parity game defined by Lehtinen [24] and the Strahler number. More specifically, we argue that from every Steven attractor decomposition of Strahler number k, we can derive a dominion strategy for Steven in the k-register game. Once we establish the Strahler number upper bound on the register number, we are faced with the following two natural questions:

▶ Question 5. Do the Strahler and the register numbers coincide?

▶ Question 6. Can the relationship between Strahler and register numbers be exploited algorithmically, in particular, to improve the running time and space complexity of solving register games studied by Lehtinen [24] and Parys [29]?

This work has been motivated by those two questions and it answers them both positively (Lemma 7 and Theorem 8, and Theorem 26, respectively).

For every positive number k, a Steven *k-register game* on a parity game $\mathcal{G}$ is another parity game $\mathcal{R}^k(\mathcal{G})$ whose vertices, edges, and priorities will be referred to as *states*, *moves*, and *ranks*, respectively, for disambiguation. The states of the Steven k-register game on $\mathcal{G}$ are either pairs $(v, \langle r_k, r_{k-1}, \ldots, r_1\rangle)$ or triples $(v, \langle r_k, r_{k-1}, \ldots, r_1\rangle, p)$, where v is a vertex in $\mathcal{G}$, $d \geq r_k \geq r_{k-1} \geq \cdots \geq r_1 \geq 0$, and $1 \leq p \leq 2k+1$. The former states have rank 1 and the latter have rank p. Each number r_i, for $i = k, k-1, \ldots, 1$, is referred to as the value of the i-th register in the state. Steven owns all states $(v, \langle r_k, r_{k-1}, \ldots, r_1\rangle)$ and the owner of vertex v in $\mathcal{G}$ is the owner of states $(v, \langle r_k, r_{k-1}, \ldots, r_1\rangle, p)$ for every p. How the game is played by Steven and Audrey is determined by the available moves:

- at every state $(v, \langle r_k, r_{k-1}, \ldots, r_1\rangle)$, Steven picks i, such that $0 \leq i \leq k$, and *resets* registers $i, i-1, i-2, \ldots, 1$, leading to state $\left(v, \left\langle r'_k, \ldots, r'_{i+1}, r'_i, 0, \ldots, 0\right\rangle, p\right)$ of rank p and with updated register values, where:

$$p = \begin{cases} 2i & \text{if } i \geq 1 \text{ and } \max\left(r_i, \pi(v)\right) \text{ is even,} \\ 2i+1 & \text{if } i = 0, \text{ or if } i \geq 1 \text{ and } \max\left(r_i, \pi(v)\right) \text{ is odd;} \end{cases}$$

  $r'_j = \max(r_j, \pi(v))$ for $j \geq i+1$, and $r'_i = \pi(v)$;
- at every state $(v, \langle r_k, r_{k-1}, \ldots, r_1\rangle, p)$, the owner of vertex v in $\mathcal{G}$ picks an edge (v, u) in $\mathcal{G}$, leading to state $(u, \langle r_k, r_{k-1}, \ldots, r_1\rangle)$ of rank 1 and with unchanged register values.

For example, at state $(v, \langle 9, 6, 4, 4, 3\rangle)$ of rank 1, if the priority $\pi(v)$ of vertex v is 5 and Steven picks $i = 3$, this leads to state $(v, \langle 9, 6, 5, 0, 0\rangle, 7)$ of rank $2i+1 = 7$ because $\max(r_3, \pi(v)) = \max(4, 5) = 5$ is odd, $r'_4 = \max(r_4, \pi(v)) = \max(6, 5) = 6$, and $r'_3 = \pi(v) = 5$.

Observe that the first components of states on every cycle in game $\mathcal{R}^k(\mathcal{G})$ form a (not necessarily simple) cycle in parity game $\mathcal{G}$; we call it the cycle in $\mathcal{G}$ *induced* by the cycle in $\mathcal{R}^k(\mathcal{G})$. If a cycle in $\mathcal{R}^k(\mathcal{G})$ is even (that is, the highest state rank on it is even) then the induced cycle in $\mathcal{G}$ is also even. Lehtinen [24, Lemmas 3.3 and 3.4] has shown that a vertex v is in the largest Steven dominion in $\mathcal{G}$ if and only if there is a positive integer k such that a state $(v, \overline{r})$, for some register values $\overline{r}$ is in the largest Steven dominion in $\mathcal{R}^k(\mathcal{G})$. Lehtinen and Boker [25, a comment after Definition 3.1] have further clarified that for every k, if a player has a dominion strategy in $\mathcal{R}^k(\mathcal{G})$ from a state whose first component is a vertex v in $\mathcal{G}$, then they also have a dominion strategy in $\mathcal{R}^k(\mathcal{G})$ from every state whose first component is v. This allows us to say without loss of rigour that a vertex v in $\mathcal{G}$ is in a dominion in $\mathcal{R}^k(\mathcal{G})$.

By defining the *(Steven) register number* [24, Definition 3.5] of a parity game $\mathcal{G}$ to be the smallest number k such that all vertices v in the largest Steven dominion in $\mathcal{G}$ are in a Steven dominion in $\mathcal{R}^k(\mathcal{G})$, and by proving the $1 + \lg n$ upper bound on the register number of every (n, d)-small parity game [24, Theorem 4.7], Lehtinen has contributed a novel quasi-polynomial algorithm for solving parity games, adding to those by Calude et al. [3] and Jurdziński and Lazić [20].

Lehtinen [24, Definition 4.8] has also considered the concept of a Steven *defensive dominion strategy* in a k-register game (for brevity, we call it a k-defensive strategy): it is a Steven dominion strategy on a set of states in $\mathcal{R}^k(\mathcal{G})$ in which there is no state of rank $2k+1$. Alternatively, the same concept can be formalized by defining the *defensive k-register game* $\mathcal{D}^k(\mathcal{G})$, which is played exactly like the k-register game $\mathcal{R}^k(\mathcal{G})$, but in which Audrey can also win just by reaching a state of rank $2k+1$. Note that the game $\mathcal{D}^k(\mathcal{G})$ can be thought of as having the winning criterion for Steven as being a conjunction of a parity and a safety criteria, and the winning criterion for Audrey as a disjunction of a parity and a reachability criteria. Routine arguements allow to extend positional determinacy from parity games to such games with combinations of parity, and safety or reachability winning criteria.

We follow Lehtinen [24, Definition 4.9] by defining the *(Steven) defensive register number* of a Steven dominion D in $\mathcal{G}$ as the smallest number k such that Steven has a defensive dominion strategy in $\mathcal{R}^k(\mathcal{G})$ on a set of states that includes all $(v, \langle r_k, \ldots, r_1 \rangle)$ for $v \in D$, and such that r_k is an even number at least as large as every vertex priority in D. We propose to call it the *Lehtinen number* of a Steven dominion in $\mathcal{G}$ to honour Lehtinen's insight that led to this – as we argue in this work – fundamental concept. We also define the Lehtinen number of a vertex in $\mathcal{G}$ to be the smallest Lehtinen number of a Steven dominion in $\mathcal{G}$ that includes the vertex, and the Lehtinen number of a parity game to be the Lehtinen number of its largest Steven dominion. We also note that the register and the Lehtinen numbers of a parity game nearly coincide (they differ by at most one), and hence the conclusions of our analysis of the latter also apply to the former.

▶ **Lemma 7.** *The Lehtinen number of a parity game is no larger than its Strahler number.*

The arguments used in our proof of this lemma are similar to those used in the proof of the main result of Lehtinen [24, Theorem 4.7]. Our contribution here is to pinpoint the Strahler number of an attractor decomposition as the structural parameter of a dominion that naturally bounds the number of registers used in Lehtinen's construction of a defensive dominion strategy.

Proof of Lemma 7. Consider a parity game $\mathcal{G}$ and let d be the least even integer no smaller than any of the priority in $\mathcal{G}$. Consider a Steven d-attractor decomposition $\mathcal{H}$ of $\mathcal{G}$ of Strahler number k. We construct a defensive k-register strategy for Steven on $\mathcal{R}^k(\mathcal{G})$. The strategy is defined inductively on the height of $\mathcal{T}_{\mathcal{H}}$, and has the additional property of being *$\mathcal{G}$-positional* in the following sense: if $((v, \langle r_k, \ldots, r_1 \rangle), (v, \langle r'_k, \ldots, r'_1 \rangle, p))$ is a move then the register reset by Steven only depends on v, not on the values in the registers. Similarly, if $((v, \langle r_k, \ldots, r_1 \rangle, p), (u, \langle r_k, \ldots, r_1 \rangle))$ is a move and v is owned by Steven, u only depends on v and not on the values of the registers or p.

Strategy for Steven. If $\mathcal{H} = \langle A, \emptyset \rangle$, then $\mathcal{G}$ consists of the set of vertices of priority d and of its Steven attractor. In this case, Steven follows the strategy induced by the reachability strategy in A to the set of vertices of priority d, only resetting register r_1 immediately after visiting a state with first component a vertex of priority d in $\mathcal{G}$. More precisely, the Steven defensive strategy is defined with the following moves:

- $((v, \langle r_1 \rangle), (v, \langle r_1 \rangle, 1))$ if v is not a vertex of priority d in $\mathcal{G}$;
- $((v, \langle r_1 \rangle), (v, \langle r'_1 \rangle, 2))$ if v is a vertex of priority d in $\mathcal{G}$ and $r'_1 = \max(r_1, d)$ is even;
- $((v, \langle r_1 \rangle), (v, \langle r'_1 \rangle, 3))$ if v is a vertex of priority d in $\mathcal{G}$ and $r'_1 = \max(r_1, d)$ is odd (we state this case for completeness but this will never occur);
- $((v, \langle r_1 \rangle, p), (u, \langle r_1 \rangle))$ where (v, u) belongs to the Steven reachability strategy from A to the set of vertices of priority d in $\mathcal{G}$.

Note that this strategy is $\mathcal{G}$-positional.

Suppose now that $\mathcal{H} = \langle A, (S_1, \mathcal{H}_1, A_1), \ldots, (S_\ell, \mathcal{H}_\ell, A_\ell) \rangle$ and that it has Strahler number k. For all $i = 1, 2, \ldots, \ell$, let k_i be the Strahler number of $\mathcal{H}_i$. By induction, for all i, we have a Steven defensive k_i-register strategy σ_i, which is $(\mathcal{G} \cap S_i)$-positional, on a set of states Ω_i in $\mathcal{R}^{k_i}(\mathcal{G} \cap S_i)$ including all the states $(v, \langle r_{k_i}, \ldots, r_1 \rangle)$ for $v \in S_i$ and r_{k_i} an even number at least as large as every vertex priority in S_i. Let Γ_i be the set of states in $\mathcal{R}^k(\mathcal{G} \cap S_i)$ defined as all the states $(v, \langle d, r_{k-1}, \ldots, r_1 \rangle)$ for $v \in S_i$ if $k_i \neq k$ and as the union of the states $(v, \langle d, r_{k-1}, \ldots, r_1 \rangle)$ for $v \in S_i$ and Ω_i, otherwise.

The strategy σ_i induces a strategy on Γ_i in $\mathcal{R}^k(\mathcal{G} \cap S_i)$ by simply ignoring registers $r_{k_i+1}, \ldots, r_k$, and using $(\mathcal{G} \cap S_i)$-positionality to define moves from the states not in Ω_i. More precisely, in a state $(v, \langle r_k, \ldots, r_1 \rangle)$, Steven resets register j if and only if register j is reset in a state $\left(v, \left\langle r'_{k_i}, \ldots, r'_1 \right\rangle\right)$ of Ω_i according to σ_i. This is well defined by $(\mathcal{G} \cap S_i)$-positionality. Similarly, we add moves $((v, \langle r_k, \ldots, r_1 \rangle, p), (u, \langle r_k, \ldots, r_1 \rangle))$ to the strategy if and only if there is a move $\left(\left(v, \left\langle r'_{k_i}, \ldots, r'_1 \right\rangle, p'\right), \left(u, \left\langle r'_{k_i}, \ldots, r'_1 \right\rangle\right)\right)$ in σ_i. This is again well-defined by $(\mathcal{G} \cap S_i)$-positionality.

This strategy is denoted by τ_i. Note that τ_i is a defensive k-register strategy on Γ_i, which is $\mathcal{G}$-positional.

The Steven defensive strategy in $\mathcal{R}^k(\mathcal{G})$ is defined by the following moves, where S denotes the set of vertices of priority d in $\mathcal{G}$:

- On the set of states with first component a vertex of $A_i \setminus S_i$, the moves are given by τ_i.
- On the set of states with first component a vertex of $A \setminus S$, Steven uses the strategy induced by the reachability strategy from A_i to S_i, without resetting any registers.
- On $\mathcal{R}^k(\mathcal{G} \cap (A \setminus S))$, Steven uses the strategy induced by the reachability strategy from A to S, without resetting any registers.
- On the set of states with first component a vertex of S,
  - $((v, \langle r_k, \ldots, r_1 \rangle), (v, \langle d, 0, \ldots, 0 \rangle, p))$ where v is a vertex in S and $p = 2k$ if $\max(r_k, d)$ is even and $p = 2k + 1$ otherwise.
  - $((v, \langle r_k, \ldots, r_1 \rangle, p), (u, \langle r_k, \ldots, r_1 \rangle))$ for some uniquely chosen u such that (v, u) in E if v is owned by Steven and for all u such that (v, u) in E if v is owned by Audrey.

Observe that this strategy is $\mathcal{G}$-positional.

Correctness of the Strategy. We prove now that the strategy defined above is indeed a defensive k-register strategy. We proceed by induction on the height of $\mathcal{T}_{\mathcal{H}}$ and define a set of states Γ, including all the states $(v, \langle d, r_{k-1}, \ldots, r_1 \rangle)$ such that v is a vertex of $\mathcal{G}$.

Base Case: If the height of $\mathcal{T}_{\mathcal{H}}$ is 0 and $\mathcal{H} = \langle A, \emptyset \rangle$, let Γ be the set of states $(v, \langle r_1 \rangle)$ and $(v, \langle r_1 \rangle, p)$ with v a vertex of $\mathcal{G}$, $1 \leq r_1 \leq d$ and p being either 1 or 2. It is easy to see that the strategy defined above is a defensive dominion strategy on this set.

Inductive step: If $\mathcal{H} = \langle A, (S_1, \mathcal{H}_1, A_1), \ldots, (S_\ell, \mathcal{H}_\ell, A_\ell) \rangle$ with Strahler number k and k_i being the Strahler number of $\mathcal{H}_i$ for all i (note that $k_i \leq k$ for all i, and by definition of Strahler number, there is at most one m such that $k_m = k$), we define Γ to be the set comprising the union of the Γ_i and all the states of the form $(v, \langle r_k, \ldots, r_1 \rangle)$ and $(v, \langle r_k, \ldots, r_1 \rangle, p)$ with v a vertex of $(A_i \setminus S_i) \cup A$ and $1 \leq p \leq 2k$.

Case 1: For each i, $k_i < k$. We first show that Γ is a trap for Audrey for the strategy defined above, showing that rank $2k+1$ can never be reached (implying that the strategy is defensive). This comes from the fact that the register of rank k is only reset in a state $(v, \langle r_k, \ldots, r_1 \rangle)$ with v in S. Since $\max(r_k, d) = d$ is even then this leads to a state $(v, \langle d, 0, \ldots, 0 \rangle, 2k)$. Otherwise, register k is never reset, so a state with rank $2k + 1$ cannot be reached.

Consider now any cycle in $\mathcal{R}^k(\mathcal{G})$ with moves restricted to the strategy constructed above. If this cycle contains a state whose first component is a vertex of S, then as explained above, the highest rank in the cycle is $2k$. Otherwise, the cycle is necessarily in $\mathcal{R}^k(\mathcal{G} \cap S_i)$ for some i. By induction, τ_i is winning and so the cycle is even.

Case 2: There is a unique m such that $k_m = k$. We first show that a state of rank $2k+1$ is never reached. Observe that register k is reset in two places: (1) immediately after a state with first component a vertex of S is visited, (2) if register k is reset by τ_m. In the first case, similarly as shown above, a state of rank $2k$ is reached. In the second case, register k is either reset in a state $(v, \langle d, r_{k-1}, \ldots, r_1\rangle)$, and similarly as above, a state of rank $2k$ is reached, or in a state of Ω_i. In this case, as τ_i is defensive on Ω_i by induction, a state of rank $2k+1$ cannot be reached, and the highest rank that can be reached is $2k$. Proving that every cycle is even is similar to the previous case. ◀

4 Strahler-Optimal Attractor Decompositions

In this section we prove that every parity game whose Lehtinen number is k has an attractor decomposition of Strahler number at most k. In other words, we establish the Lehtinen number upper bound on the Strahler number, which together with Lemma 7 provides a positive answer to Question 5.

▶ **Theorem 8.** *The Strahler number of a parity game is no larger than its Lehtinen number.*

When talking about strategies in parity games in Section 2, we only considered positional strategies, for which it was sufficient to verify the parity criterion on (simple) cycles. Instead, we explicitly consider the parity criterion on infinite paths here, which we find more convenient to establish properties of Audrey strategies in the proof of Theorem 8.

First, we introduce the concepts of *tight* and *offensively optimal* attractor decompositions.

▶ **Definition 9.** *A Steven d-attractor decomposition $\mathcal{H}$ of $\mathcal{G}$ is* tight *if Audrey has a winning strategy from at least one state in $\mathcal{D}^{\mathrm{Str}(\mathcal{H})-1}(\mathcal{G})$ in which the value of register* $\mathrm{Str}(\mathcal{H})-1$ *is d.*

By definition, the existence of a tight Steven d-attractor decomposition on a parity game implies that the Lehtinen number of the game is at least its Strahler number, from which Theorem 8 follows. Offensive optimality of an attractor decomposition, the concept we define next, may seem less natural and more technical than tightness, but it facilitates our proof that every game has a tight attractor decomposition.

▶ **Definition 10.** *Let $\mathcal{H} = \langle A, (S_1, \mathcal{H}_1, A_1), \ldots, (S_\ell, \mathcal{H}_\ell, A_\ell)\rangle$ be a Steven d-attractor decomposition, let games $\mathcal{G}_i$ for $i = 1, 2, \ldots, \ell$ be as in the definition of an attractor decomposition, let A'_i be the Audrey attractor of the set of vertices of priority $d-1$ in $\mathcal{G}_i$, and let $\mathcal{G}'_i = \mathcal{G}_i \setminus A'_i$. We say that $\mathcal{H}$ is* offensively optimal *if for every $i = 1, 2, \ldots, \ell$, we have:*

- *Audrey has a dominion strategy on $\mathcal{R}^{\mathrm{Str}(\mathcal{H}_i)-1}(\mathcal{G}'_i)$;*
- *Audrey has a dominion strategy on $\mathcal{D}^{\mathrm{Str}(\mathcal{H}_i)}(\mathcal{G}'_i \setminus S_i)$.*

Proving that every offensively optimal Steven attractor decomposition is tight (Lemma 11), and that every Steven dominion in a parity game has an offensively optimal Steven attractor decomposition (Lemma 12), will complete the proof of Theorem 8.

▶ **Lemma 11.** *Every offensively optimal Steven attractor decomposition is tight.*

Proof. Let $\mathcal{H} = \langle A, (S_1, \mathcal{H}_1, A_1), \ldots, (S_\ell, \mathcal{H}_\ell, A_\ell)\rangle$ be an offensively optimal d-attractor decomposition of a parity game and let $k = \mathrm{Str}(\mathcal{H})$. We construct a strategy for Audrey in $\mathcal{D}^{k-1}(\mathcal{G})$ that is winning for her from at least one state in which the value of register $k-1$ is d. We define $\mathcal{G}'_i$ and A'_i as in Definition 10.

Case 1: $\mathrm{Str}(\mathcal{H}_i) = k$ for some unique i in $\{1, \ldots, \ell\}$. In this case, we show that Audrey has a dominion strategy on $\mathcal{D}^{k-1}(\mathcal{G}_i)$. Since $\mathcal{G}_i$ is a trap for Steven in $\mathcal{G}$, this gives the desired result. Consider the following strategy in $\mathcal{D}^{k-1}(\mathcal{G}_i)$:

- On the set of states whose vertex components are in A'_i, Audrey follows a strategy induced by the reachability strategy in A'_i to a vertex of priority $d-1$ (picking any move if v is of priority $d-1$);

- In states whose vertex component is in $\mathcal{G}'_i$, Audrey plays a $(k-1)$-register dominion strategy on $\mathcal{R}^{k-1}(\mathcal{G}'_i)$. Such a strategy exists by the definition of offensive optimality and by the assumption that $\mathrm{Str}\,(\mathcal{H}_i) = k$.

This strategy is indeed an Audrey dominion strategy on $\mathcal{D}^{k-1}(\mathcal{G}_i)$, because any play either visits a state whose first component is a vertex in A'_i infinitely often, or it eventually remains in $\mathcal{R}^{k-1}(\mathcal{G}'_i)$. In the former case, the play visits a state whose first component is a vertex of priority $d-1$ infinitely often. In the latter case, the state parity criterion holds. Note that this even defines an Audrey dominion strategy on $\mathcal{R}^{k-1}(\mathcal{G}_i)$.

Case 2: There are $1 \leq i < j \leq \ell$ such that $\mathrm{Str}\,(\mathcal{H}_i) = \mathrm{Str}\,(\mathcal{H}_j) = k-1$. We construct a strategy for Audrey in $\mathcal{D}^{k-1}(\mathcal{G})$ that is winning for her from all states in $\mathcal{G}_j$ whose register $k-1$ has value d. Firstly, since $\mathcal{H}$ is offensively optimal, Audrey has a dominion strategy on $\mathcal{D}^{k-1}(\mathcal{G}'_i \setminus S_i)$, denoted by τ_i, and a dominion strategy on $\mathcal{R}^{k-2}(\mathcal{G}'_i)$, denoted by τ'_i. Moreover, since $\langle \emptyset, (S_j, \mathcal{H}_j, A_j), \ldots, (S_\ell, \mathcal{H}_\ell, A_\ell) \rangle$ is an offensively optimal attractor decomposition of $\mathcal{G}_j$, an argument similar to the one in Case 1. yields that Audrey has a dominion strategy, denoted by τ_j, on $\mathcal{R}^{k-2}(\mathcal{G}_j)$ (note that $\mathcal{G}_j$ is a trap for Steven in $\mathcal{G}$). Consider the following strategy for Audrey in $\mathcal{D}^{k-1}(\mathcal{G})$, starting from a state whose vertex component is in $\mathcal{G}_j$ and register $k-1$ has value d:

- As long as the value of register $k-1$ is larger than $d-1$, Audrey follows the strategy induced by τ_j, while ignoring the value of register $k-1$, as long as this value is larger than $d-1$.
- If the value in register $k-1$ is at most $d-1$:
  - In states whose vertex component is in A'_i, Audrey follows a strategy induced by the reachability strategy from A'_i to a vertex of priority $d-1$ (picking any move if the vertex has priority $d-1$);
  - In states whose vertex component is in $\mathcal{G}'_i \setminus S_i$ and whose register $k-2$ has value at most $d-2$, Audrey follows τ_i;
  - In states whose vertex component is in $\mathcal{G}'_i$ and whose register $k-1$ has value $d-1$, Audrey follows the strategy induced by τ'_i, while ignoring the value of regiser $k-1$.

Audrey plays any move if none of the above applies.

We argue that this strategy is winning for Audrey in $\mathcal{D}^{k-1}(\mathcal{G})$ from states whose vertex component is in $\mathcal{G}_j$ and register $k-1$ has value d. Consider an infinite path that starts in such a state. As long as register $k-1$ has value d, Audrey follows τ_j. If Steven never resets register $k-1$ then Audrey wins. Otherwise, once register $k-1$ has been reset, its value is at most $d-1$. Note that $\mathcal{G}_j$ is included in $A'_i \cup (\mathcal{G}'_i \setminus S_i)$. If register $k-1$ has a value smaller than $d-1$, and the play never visits a state whose vertex component is in A'_i, then Audrey has followed τ_i along the play (she has never left $\mathcal{G}'_i \setminus S_i$ as the only way for Steven to go out $\mathcal{G}'_i \setminus S_i$ is to go to A'_i) and wins. Otherwise, the play visits a state whose vertex component is in A'_i, and so it visits a state whose vertex component has priority $d-1$, leading to a state in which register $k-1$ has value $d-1$. Finally, if a state whose vertex component is in A'_i is visited infinitely many times then Audrey wins. Otherwise, Audrey eventually plays according to τ'_i. If Steven never resets register $k-1$ then Audrey wins. Otherwise, if Steven resets register $k-1$, which at this point has value $d-1$, a state of rank $2k-1$ is visited and Audrey wins. ◀

▶ **Lemma 12.** *Every Steven dominion in a parity game has an offensively optimal Steven attractor decomposition.*

Proof. Consider a parity game $\mathcal{G}$ which is a Steven dominion. Let k be the Lehtinen number of $\mathcal{G}$ and let d be the largest even value such that $\pi^{-1}(\{d, d-1\}) \neq \emptyset$. We construct an offensively optimal Steven attractor decomposition by induction.

If $d = 0$, it is enough to consider $\langle A, \emptyset \rangle$, where A is the set of all vertices in $\mathcal{G}$.

If $d > 1$, let A be the Steven attractor of the set of vertices of priority d in $\mathcal{G}$. Let $\mathcal{G}_0 = \mathcal{G} \setminus A$. If $\mathcal{G}_0 = \emptyset$ then $\langle A, \emptyset \rangle$ is an offensively optimal Steven attractor decomposition for $\mathcal{G}$. Otherwise, $\mathcal{G}_0$ is a non-empty trap for Steven in $\mathcal{G}$ and therefore $\mathcal{G}_0$ has a Lehtinen number at most k. Let A' be the Audrey attractor of all the vertices of priority $d-1$ in the sub-game $\mathcal{G}_0$ and let $\mathcal{G}'_0 = \mathcal{G}_0 \setminus A'$.

Given a positive integer b, let L^b be the largest dominion in $\mathcal{G}'_0$ such that Steven has a dominion strategy on $\mathcal{D}^b(\mathcal{G}'_0)$. We define m to be the smallest number such that $L^m \neq \emptyset$ and let $S_0 = L^m$. We show that $m \leq k$. To prove this, we construct an Audrey dominion strategy on $\mathcal{D}^b(\mathcal{G}_0)$ for all b such that $L^b = \emptyset$. Since the Lehtinen number of $\mathcal{G}_0$ is at most k, this implies that $m \leq k$. The Audrey dominion strategy on $\mathcal{D}^b(\mathcal{G}_0)$, assuming $L^b = \emptyset$, is as follows:

- If the vertex component of a state is in A' then Audrey uses the strategy in A' induced by the reachability strategy to vertices of priority $d-1$;
- If the vertex component of a state is in $\mathcal{G}'_0$ then Audrey uses her dominion strategy on $\mathcal{D}^b(\mathcal{G}'_0)$, which exists because the Steven dominion L^b in $\mathcal{D}^b(\mathcal{G}'_0)$ is empty.

Any play following the above strategy and visiting infinitely often a state of $\mathcal{D}^b(\mathcal{G}_0 \cap A')$ is winning for Audrey. A play following the above strategy and remaining eventually in $\mathcal{D}^b(\mathcal{G}'_0)$ is also winning for Audrey.

Let $\mathcal{H}_0$ be the $(d-2)$-attractor decomposition of S_0 obtained by induction. In particular, $\mathcal{H}_0$ is offensively optimal.

Let A_0 be the Steven attractor to S_0 in $\mathcal{G}_0$ and let $\mathcal{G}_1 = \mathcal{G}_0 \setminus A_0$. Subgame $\mathcal{G}_1$ is a trap for Steven and therefore it is a Steven dominion. Let $\mathcal{H}' = \langle \emptyset, (S_1, \mathcal{H}_1, A_1), \ldots, (S_\ell, \mathcal{H}_\ell, A_\ell) \rangle$ be an offensively optimal Steven d-attractor decomposition of $\mathcal{G}_1$ obtained by induction.

We claim that $\mathcal{H} = \langle A, (S_0, \mathcal{H}_0, A_0), (S_1, \mathcal{H}_1, A_1), \ldots, (S_\ell, \mathcal{H}_\ell, A_\ell) \rangle$ is an offensively optimal Steven d-attractor decomposition of $\mathcal{G}$. Since $\mathcal{H}'$ is offensively optimal, it is enough to show that:

- Audrey has a dominion strategy on $\mathcal{R}^{\mathrm{Str}(\mathcal{H}_0)-1}(\mathcal{G}'_0)$,
- Audrey has a dominion strategy on $\mathcal{D}^{\mathrm{Str}(\mathcal{H}_0)}(\mathcal{G}'_0 \setminus S_0)$.

Since $\mathcal{H}_0$ is offensively optimal, Audrey has a dominion strategy in $\mathcal{R}^{\mathrm{Str}(\mathcal{H}_0)-1}(S_0)$, by Lemma 11, and hence $m \geq \mathrm{Str}\,(\mathcal{H}_0)$. Moreover, by construction of S_0, Audrey has a dominion strategy on $\mathcal{D}^m(\mathcal{G}'_0 \setminus S_0)$. This implies that Audrey has a dominion strategy on $\mathcal{D}^{\mathrm{Str}(\mathcal{H}_0)}(\mathcal{G}'_0 \setminus S_0)$.

By choice of m, Steven does not have a defensive dominion strategy on $\mathcal{D}^{\mathrm{Str}(\mathcal{H}_0)-1}(\mathcal{G}'_0)$ from any state. This means that for all states s, Audrey has a winning strategy τ_s on $\mathcal{D}^{\mathrm{Str}(\mathcal{H}_0)-1}(\mathcal{G}'_0)$ starting in s. We construct a dominion strategy for her on $\mathcal{R}^{\mathrm{Str}(\mathcal{H}_0)-1}(\mathcal{G}'_0)$: after every visit to a state of rank $2\mathrm{Str}\,(\mathcal{H}_0) - 1$, Audrey follows τ_s, where s is the first state that follows on the path and whose rank is smaller than $2\mathrm{Str}\,(\mathcal{H}_0) - 1$. This defines a dominion strategy on $\mathcal{R}^{\mathrm{Str}(\mathcal{H}_0)-1}(\mathcal{G}'_0)$. ◀

5 Strahler-Universal Trees

Our attention now shifts to tackling Question 6. The approach is to develop constructions of small ordered trees into which trees of attractor decompositions or of progress measures can be embedded. Such trees can be seen as natural search spaces for dominion strategies, and existing meta-algorithms such as the universal attractor decomposition algorithm [21] and progress measure lifting algorithm [19, 20] can use them to guide their search, performed in time proportional to the size of the trees in the worst case.

An ordered tree is *universal* for a class of trees if all trees from the class can be embedded into it. The innovation offered in this work is to develop optimized constructions of trees that are universal for classes of trees whose complex structural parameter, such as the Strahler number, is bounded. This is in contrast to less restrictive universal trees introduced by Czerwiński et al. [4] and implicitly constructed by Jurdziński and Lazić [20], whose sizes therefore grow faster with size parameters, leading to slower algorithms.

Firstly, we give an inductive construction of Strahler-universal trees and an upper bound on their numbers of leaves. Then we introduce labelled ordered trees, provide a succinct bit-string labelling of the Strahler-universal trees, and give an alternative and more explicit characterization of the succinctly-labelled Strahler-universal trees. Finally, we argue how the succinct bit-string labelling of Strahler-universal trees facilitates efficient computation of the so-called "level-p successors" in them, which is the key computational primitive that allows using ordered trees to solve parity games. The constructions and techniques we develop here are inspired by and significantly refine those introduced by Jurdziński and Lazić [20].

Strahler-Universal Trees and Their Sizes. Intuitively, an ordered tree *can be embedded in* another if the former can be obtained from the latter by pruning some subtrees. More formally, the trivial tree $\langle\rangle$ can be embedded in every ordered tree, and $\langle T_1, T_2, \ldots, T_k\rangle$ can be embedded in $\langle T'_1, T'_2, \ldots, T'_\ell\rangle$ if there are indices $i_1, i_2, \ldots, i_k$ such that $1 \le i_1 < i_2 < \cdots < i_k \le \ell$ and for every $j = 1, 2, \ldots, k$, we have that T_j can be embedded in T'_{i_j}.

An ordered tree is (n, h)*-universal* [4] if every (n, h)-small ordered tree can be embedded in it. We define an ordered tree to be k*-Strahler* (n, h)*-universal* if every (n, h)-small ordered tree whose Strahler number is at most k can be embedded in it, and we give a construction of small Strahler-universal trees.

▶ **Definition 13** (Trees $U^k_{t,h}$ and $V^k_{t,h}$). *For all $t \ge 0$, we define trees $U^k_{t,h}$ (for all h and k such that $h \ge k \ge 1$) and $V^k_{t,h}$ (for all h and k such that $h \ge k \ge 2$) by mutual induction:*

1. *if $h = k = 1$ then $U^k_{t,h} = \langle\rangle$;*
2. *if $h > 1$ and $k = 1$ then $U^k_{t,h} = \left\langle U^k_{t,h-1}\right\rangle$;*
3. *if $h \ge k \ge 2$ and $t = 0$ then $U^k_{t,h} = V^k_{t,h} = \left\langle U^{k-1}_{t,h-1}\right\rangle$;*
4. *if $h \ge k \ge 2$ and $t \ge 1$ then $V^k_{t,h} = V^k_{t-1,h} \cdot \left\langle U^{k-1}_{t,h-1}\right\rangle \cdot V^k_{t-1,h}$;*
5. *if $h = k \ge 2$ and $t \ge 2$ then $U^k_{t,h} = V^k_{t,h}$;*
6. *if $h > k \ge 2$ and $t \ge 2$ then $U^k_{t,h} = V^k_{t,h} \cdot \left\langle U^k_{t,h-1}\right\rangle \cdot V^k_{t,h}$.*

For $g \ge 0$, let I_g be the trivial tree, that is the tree with exactly one leaf, of height g. For example, $I_1 = \langle\rangle$ and $I_3 = \langle\langle\langle\rangle\rangle\rangle = \langle\langle\circ\rangle\rangle$. It is routine to verify that if $h \ge k = 1$ or $t = 0$ then $U^k_{t,h} = I_h$, and if $h \ge k \ge 2$ and $t = 0$ then $V^k_{t,h} = I_h$.

▶ **Lemma 14.** *For all $n \ge 1$ and $h \ge k \ge 1$, the ordered tree $U^k_{\lfloor \lg n\rfloor,h}$ is k-Strahler (n, h)-universal.*

Proof. We say that a tree has *weak Strahler number* at most k if every subtree rooted in a child of the root has Strahler number at most $k - 1$. A tree is then *weakly k-Strahler (n, h)-universal* if every (n, h)-small ordered tree whose weak Strahler number is at most k can be embedded in it. We proceed by induction on the number of leaves in an ordered tree and its height, using the following strengthened inductive hypothesis:

- for all $n \ge 1$ and $h \ge k \ge 1$, ordered tree $U^k_{\lfloor \lg n\rfloor,h}$ is k-Strahler (n, h)-universal;
- for all $n \ge 1$ and $h \ge k \ge 2$, ordered tree $V^k_{\lfloor \lg n\rfloor,h}$ is weakly k-Strahler (n, h)-universal.

Let T be an (n,h)-small ordered tree of Strahler number at most k. If $n=1$, $h=1$, or $k=1$, then T is the trivial tree (with just one leaf) of height at most h, and hence it can be embedded in $U^k_{\lfloor \lg n \rfloor, h} = I_h$, the trivial tree of height h. Likewise, if $h \geq k \geq 2$ and $n=1$, then T is the trivial tree of height at most h, and hence it can be embedded in $V^k_{\lfloor \lg n \rfloor, h} = I_h$, the trivial tree of height h.

Otherwise, we have that $T = \langle T_1, \dots, T_j \rangle$ for some $j \geq 1$. We consider two cases: either $\operatorname{Str}(T_i) \leq k-1$ for all $i = 1, \dots, j$, or there is q such that $\operatorname{Str}(T_q) = k$. Note that by Proposition 3, the latter case can only occur if $h > k$.

If $\operatorname{Str}(T_i) \leq k-1$ for all $i = 1, \dots, j$, then we argue that T can be embedded in $V^k_{\lfloor \lg n \rfloor, h}$, and hence also in $U^k_{\lfloor \lg n \rfloor, h}$, because $V^k_{\lfloor \lg n \rfloor, h}$ can be embedded in $U^k_{\lfloor \lg n \rfloor, h}$ by definition (see items 3., 5., and 6. of Definition 13). Let p (a pivot) be an integer such that both trees $T' = \langle T_1, \dots, T_{p-1} \rangle$ and $T'' = \langle T_{p+1}, \dots, T_j \rangle$ are $(\lfloor n/2 \rfloor, h)$-small. Then by the strengthened inductive hypothesis, each of the two trees T' and T'' can be embedded in tree $V^k_{\lfloor \lg \lfloor n/2 \rfloor \rfloor, h} = V^k_{\lfloor \lg n \rfloor - 1, h}$ and tree T_p can be embedded in $U^{k-1}_{\lfloor \lg n \rfloor, h-1}$. It then follows that tree $T = T' \cdot \langle T_p \rangle \cdot T''$ can be embedded in $V^k_{\lfloor \lg n \rfloor, h} = V^k_{\lfloor \lg n \rfloor - 1, h} \cdot \left\langle U^{k-1}_{\lfloor \lg n \rfloor, h-1} \right\rangle \cdot V^k_{\lfloor \lg n \rfloor - 1, h}$.

If $\operatorname{Str}(T_q) = k$ for some q (the pivot), then we argue that T can be embedded in $U^k_{\lfloor \lg n \rfloor, h}$. Note that each of the two trees $T' = \langle T_1, \dots, T_{q-1} \rangle$ and $T'' = \langle T_{q+1}, \dots, T_j \rangle$ is (n,h)-small and all trees $T_1, \dots, T_{q-1}$ and $T_{q+1}, \dots, T_j$ have Strahler numbers at most $k-1$. By the previous paragraph, it follows that each of the two trees T' and T'' can be embedded in $V^k_{\lfloor \lg n \rfloor, h}$. Moreover, tree T_q is $(n, h-1)$-small and hence, by the inductive hypothesis, it can be embedded in $U^k_{\lfloor \lg n \rfloor, h-1}$. It follows that tree $T = T' \cdot \langle T_q \rangle \cdot T''$ can be embedded in $U^k_{\lfloor \lg n \rfloor, h} = V^k_{\lfloor \lg n \rfloor, h} \cdot \left\langle U^k_{\lfloor \lg n \rfloor, h-1} \right\rangle \cdot V^k_{\lfloor \lg n \rfloor, h}$. ◀

▶ **Lemma 15.** *For all $t \geq 0$, we have:*

- *if $h \geq k = 1$ then* $\operatorname{leaves}\left(U^k_{t,h}\right) = 1$;
- *if $h \geq k \geq 2$ then* $\operatorname{leaves}\left(U^k_{t,h}\right) \leq 2^{t+k} \binom{t+k-2}{k-2} \binom{h-1}{k-1}$.

▶ **Theorem 16.** *For $k \leq \lg n$, the number of leaves of the k-Strahler (n,h)-universal ordered trees $U^k_{\lfloor \lg n \rfloor, h}$ is $n^{O(1)} \cdot (h/k)^k = n^{k \lg(h/k)/\lg n + O(1)}$, which is polynomial in n if $k \cdot \lg(h/k) = O(\log n)$. In more detail, the number is at most $n^{c(n)} \cdot (h/k)^k$, where $c(n) = 5.45$ if $k \leq \lg n$, $c(n) = 3 + o(1)$ if $k = o(\log n)$, and $c(n) = 1 + o(1)$ if $k = O(1)$.*

▶ Remark 17. By Proposition 3 and Lemma 14, for all positive integers n and h, the tree $U^{\lfloor \lg n \rfloor + 1}_{\lfloor \lg n \rfloor, h}$ is (n,h)-universal. Theorem 16 implies that the number of leaves of $U^{\lfloor \lg n \rfloor + 1}_{\lfloor \lg n \rfloor, h}$ is $n^{\lg(h/\lg n) + O(1)}$, which matches the asymptotic number of leaves of (n,h)-universal trees of Jurdziński and Lazić [20, Lemma 6]. In particular, if $h = O(\log n)$ then $\lg(h/\lg n) = O(1)$, and hence the number of leaves of $U^{\lfloor \lg n \rfloor + 1}_{\lfloor \lg n \rfloor, h}$ is polynomial in n.

Labelled Strahler-Universal Trees. *Labelled ordered tree* are similar to ordered trees: the trivial tree $\langle \rangle$ is an *A-labelled ordered tree* and so is a sequence $\langle (a_1, \mathcal{L}_1), (a_2, \mathcal{L}_2), \dots, (a_k, \mathcal{L}_k) \rangle$, where $\mathcal{L}_1$, $\mathcal{L}_2$, …, $\mathcal{L}_k$ are A-labelled ordered trees, and a_1, a_2, …, a_k are distinct elements of a linearly ordered set $(A, \leq)$ and $a_1 < a_2 < \cdots < a_k$ in that linear order. We define the *unlabelling* of a labelled ordered tree $\langle (a_1, \mathcal{L}_1), (a_2, \mathcal{L}_2), \dots, (a_k, \mathcal{L}_k) \rangle$, by straightforward induction, to be the ordered tree $\langle T_1, T_2, \dots, T_k \rangle$, where T_i is the unlabelling of $\mathcal{L}_i$ for every $i = 1, 2, \dots, k$. An *A-labelling* of an ordered tree T is an A-labelled tree $\mathcal{L}$ whose unlabelling is T. We define the *natural labelling* of an ordered tree $T = \langle T_1, \dots, T_k \rangle$, again by a straightfoward induction, to be the $\mathbb{N}$-labelled tree $\langle (1, \mathcal{L}_1), \dots, (k, \mathcal{L}_k) \rangle$, where $\mathcal{L}_1$, …, $\mathcal{L}_k$ are the natural labellings of trees T_1, …, T_k.

For an A-labelled tree $\langle (a_1, \mathcal{L}_1), \ldots, (a_k, \mathcal{L}_k) \rangle$, its set of *nodes* is defined inductively to consist of the root $\langle\rangle$ and all the sequences in A^* of the form $\langle a_i \rangle \cdot v$, where $v \in A^*$ is a node in $\mathcal{L}_i$ for some $i = 1, \ldots, k$, and where the symbol $\cdot$ denotes concatenation of sequences. For example, the natural labelling of tree $\left\langle \langle \circ^3 \rangle, \circ^4, \langle\langle \circ \rangle\rangle^2 \right\rangle$ has the set of nodes that consists of the following set of leaves $\langle 1,1 \rangle$, $\langle 1,2 \rangle$, $\langle 1,3 \rangle$, $\langle 2 \rangle$, $\langle 3 \rangle$, $\langle 4 \rangle$, $\langle 5 \rangle$, $\langle 6,1,1 \rangle$, $\langle 7,1,1 \rangle$, and all of their prefixes. Indeed, the set of nodes of a labelled ordered tree is always prefix-closed. Moreover, if $L \subseteq A^*$ then its closure under prefixes uniquely identifies a labelled ordered tree that we call the labelled ordered tree *generated* by L, and its unlabelling is the ordered tree generated by L. For example, the set $\{ \langle 1 \rangle, \langle 3,1 \rangle, \langle 3,4,1 \rangle, \langle 6,1 \rangle \}$ generates ordered tree $\langle \circ, \langle \circ, \langle \circ \rangle \rangle, \langle \circ \rangle \rangle$.

Consider the following linear order on the set $\{0,1\}^*$ of bit strings: for each bit $b \in \{0,1\}$, and for all bit strings $\beta, \beta' \in \{0,1\}^*$, if ε is the empty string, then we have $0\beta < \varepsilon$, $\varepsilon < 1\beta$, and $b\beta < b\beta'$ iff $\beta < \beta'$.

For a bit string $\beta \in \{0,1\}^*$, we write $|\beta|$ for the number of bits used in the string. For example, we have $|\varepsilon| = 0$ and $|010| = 3$, and $|11| = 2$. Suppose that $\langle \beta_i, \beta_{i-1}, \ldots, \beta_1 \rangle$ is a node in a $\{0,1\}^*$-labelled ordered tree. Then if $\beta_j = b\beta$ for some $j = 1, 2, \ldots, i$, $b \in \{0,1\}$, and $\beta \in \{0,1\}^*$, then we refer to the first bit b as the *leading bit* in β_j, and we refer to all the following bits in β as *non-leading bits* in β_j. For example, node $\langle \varepsilon, 010, \varepsilon, \varepsilon, 11 \rangle$ has two non-empty strings and hence two leading bits, and it uses three non-leading bits overall, because $|010| + |11| - 2 = 3$.

For a bit $b \in \{0,1\}$ and a $\{0,1\}^*$-labelled ordered tree $\mathcal{L} = \langle (\beta_1, \mathcal{L}_1), \ldots, (\beta_\ell, \mathcal{L}_\ell) \rangle$, we define the $\{0,1\}^*$-labelled ordered tree $[\mathcal{L}]^b$ to be equal to $\mathcal{L} = \langle (b\beta_1, \mathcal{L}_1), \ldots, (b\beta_\ell, \mathcal{L}_\ell) \rangle$. In other words, $[\mathcal{L}]^b$ is the labelled ordered tree that is obtained from $\mathcal{L}$ by adding an extra copy of bit b as the leading bit in the labels of all children of the root of $\mathcal{L}$.

The inductive structure of the next definition is identical to that of Definition 13, and hence labelled ordered trees $\mathcal{U}^k_{t,h}$ and $\mathcal{V}^k_{t,h}$ defined here are labellings of the ordered trees $U^k_{t,h}$ and $V^k_{t,h}$, respectively.

▶ **Definition 18** (Trees $\mathcal{U}^k_{t,h}$ and $\mathcal{V}^k_{t,h}$). *For all $t \geq 0$, we define $\{0,1\}^*$-labelled ordered trees $\mathcal{U}^k_{t,h}$ (for all h and k such that $h \geq k \geq 1$) and $\mathcal{V}^k_{t,h}$ (for all h and k such that $h \geq k \geq 2$) by mutual induction:*

1. *if $h = k = 1$ then $\mathcal{U}^k_{t,h} = \langle\rangle$;*
2. *if $h > 1$ and $k = 1$ then $\mathcal{U}^k_{t,h} = \left\langle \left(\varepsilon, \mathcal{U}^k_{t,h-1} \right) \right\rangle$;*
3. *if $h \geq k \geq 2$ and $t = 0$ then $\mathcal{V}^k_{t,h} = \left\langle \left(\varepsilon, \mathcal{U}^{k-1}_{t,h-1} \right) \right\rangle$ and $\mathcal{U}^k_{t,h} = \left[\mathcal{V}^k_{t,h} \right]^0 = \left\langle \left(0, \mathcal{U}^{k-1}_{t,h-1} \right) \right\rangle$;*
4. *if $h \geq k \geq 2$ and $t \geq 1$ then $\mathcal{V}^k_{t,h} = \left[\mathcal{V}^k_{t-1,h} \right]^0 \cdot \left\langle \left(\varepsilon, \mathcal{U}^{k-1}_{t,h-1} \right) \right\rangle \cdot \left[\mathcal{V}^k_{t-1,h} \right]^1$;*
5. *if $h = k \geq 2$ and $t \geq 1$ then $\mathcal{U}^k_{t,h} = \left[\mathcal{V}^k_{t,h} \right]^0$;*
6. *if $h > k \geq 2$ and $t \geq 1$ then $\mathcal{U}^k_{t,h} = \left[\mathcal{V}^k_{t,h} \right]^0 \cdot \left\langle \left(\varepsilon, \mathcal{U}^k_{t,h-1} \right) \right\rangle \cdot \left[\mathcal{V}^k_{t,h} \right]^1$.*

The inductive definition of labelled ordered trees $\mathcal{U}^k_{t,h}$ and $\mathcal{V}^k_{t,h}$ makes it straightforward to argue that their unlabellings are equal to trees $U^k_{t,h}$ and $V^k_{t,h}$, respectively, and hence to transfer to them Strahler-universality established in Lemma 14 and upper bounds on the numbers of leaves established in Lemma 15 and Theorem 16. We now give an alternative and more explicit characterization of those trees, which will be more suitable for algorithmic purposes. To that end, we define $\{0,1\}^*$-labelled trees $\mathcal{B}^k_{t,h}$ and $\mathcal{C}^k_{t,h}$ and then we argue that they are equal to trees $\mathcal{U}^k_{t,h}$ and $\mathcal{V}^k_{t,h}$, respectively, by showing that they satisfy all the recurrences in Definition 18.

▶ **Definition 19** (Trees $\mathcal{B}^k_{t,h}$ and $\mathcal{C}^k_{t,h}$). *For all $t \geq 0$ and $h \geq k \geq 1$, we define $\{0, 1\}^*$-labelled ordered trees $\mathcal{B}^k_{t,h}$ as the tree generated by sequences $\langle \beta_{h-1}, \ldots, \beta_1 \rangle$ such that:*

1. *the number of non-empty bit strings among β_{h-1}, ..., β_1 is $k-1$;*
2. *the number of bits used in bit strings β_{h-1}, ..., β_1 overall is at most $(k-1)+t$;*

and for every $i = 1, \ldots, h-1$, we have the following:

3. *if there are less than $k-1$ non-empty bit strings among β_{h-1}, ..., β_{i+1}, but there are t non-leading bits used in them, then $\beta_i = 0$;*
4. *if all bit strings β_i, ..., β_1 are non-empty, then each of them has 0 as its leading bit.*

For all $t \geq 0$ and $h \geq k \geq 2$, we define $\{0, 1\}^*$-labelled ordered trees $\mathcal{C}^k_{t,h}$ as the tree generated by sequences $\langle \beta_{h-1}, \ldots, \beta_1 \rangle$ such that:

1. *the number of non-empty bit strings among β_{h-2}, ..., β_1 is $k-2$;*
2. *the number of bits used in bit strings β_{h-1}, ..., β_1 overall is at most $(k-2)+t$;*

and for every $i = 1, \ldots, h-1$, we have the following:

3. *if there are less than $k-2$ non-empty bit strings among β_{h-2}, ..., β_{i+1}, but there are $t - |\beta_{h-1}|$ non-leading bits used in them, then $\beta_i = 0$;*
4. *if all bit strings β_i, ..., β_1 are non-empty, then each of them has 0 as its leading bit.*

▶ **Lemma 20.** *For all $t \geq 0$ and $h \geq k \geq 1$, we have $\mathcal{U}^k_{t,h} = \mathcal{B}^k_{t,h}$.*

The following corollary follows from Lemma 20, and from the identical inductive structures of Definitions 13 and 18.

▶ **Corollary 21.** *For all $t \geq 0$ and $h \geq k \geq 1$, the unlabelling of $\mathcal{B}^k_{t,h}$ is equal to $U^k_{t,h}$.*

Efficiently Navigating Labelled Strahler-Universal Trees. The computation of the *level-p successor* of a leaf in a labelled ordered tree of height h is the following problem: given a leaf $\langle \beta_h, \beta_{h-1}, \ldots, \beta_1 \rangle$ in the tree and given a number p, such that $1 \leq p \leq h$, compute the $<_{\text{lex}}$-smallest leaf $\langle \beta'_h, \beta'_{h-1}, \ldots, \beta'_1 \rangle$ in the tree, such that $\langle \beta_h, \ldots, \beta_p \rangle <_{\text{lex}} \langle \beta'_h, \ldots, \beta'_p \rangle$. As (implicitly) explained by Jurdziński and Lazić [20, Proof of Theorem 7], the level-p successor computation is the key primitive used extensively in an implementation of a progress measure lifting algorithm.

▶ **Lemma 22.** *Every leaf in tree $\mathcal{B}^k_{t,h}$ can be represented using $O\left((k+t)\log h\right)$ bits and for every $p = 1, 2, \ldots, h$, the level-p successor of a leaf in tree $\mathcal{B}^k_{t,h}$ can be computed in time $O\left((k+t)\log h\right)$.*

6 Progress-Measure Strahler Numbers

Consider a parity game $\mathcal{G}$ in which all vertex priorities are at most an even number d. If $(A, \leq)$ is a well-founded linear order then we write sequences in $A^{d/2}$ in the following form $\langle m_{d-1}, m_{d-3}, \ldots, m_1 \rangle$, and for every priority $p \in \{0, 1, \ldots, d\}$, we define the *$p$-truncation* of $\langle m_{d-1}, m_{d-3}, \ldots, m_1 \rangle$, denoted by $\langle m_{d-1}, m_{d-3}, \ldots, m_1 \rangle|_p$, to be the sequence $\langle m_{d-1}, \ldots, m_{p+2}, m_p \rangle$ if p is odd and $\langle m_{d-1}, \ldots, m_{p+3}, m_{p+1} \rangle$ if p is even. We use the lexicographic order $\leq_{\text{lex}}$ to linearly order the set $A^* = \bigcup_{i=0}^{\infty} A^i$.

A *Steven progress measure* [8, 19, 20] on a parity game $\mathcal{G}$ is a map $\mu : V \to A^{d/2}$ such that for every vertex $v \in V$:

- if $v \in V_{\text{Even}}$ then there is a μ-progressive edge $(v, u) \in E$;
- if $v \in V_{\text{Odd}}$ then every edge $(v, u) \in E$ is μ-progressive;

where we say that an edge $(v, u) \in E$ is *μ-progressive* if:

- if $\pi(v)$ is even then $\mu(v)|_{\pi(v)} \geq_{\text{lex}} \mu(u)|_{\pi(v)}$;
- if $\pi(v)$ is odd then $\mu(v)|_{\pi(v)} >_{\text{lex}} \mu(u)|_{\pi(v)}$.

We define *the tree of a progress measure* μ to be the ordered tree generated by the image of V under μ.

▶ **Theorem 23** ([8, 19, 20])**.** *There is a Steven progress measure on a parity game $\mathcal{G}$ if and only if every vertex in $\mathcal{G}$ is in its largest Steven dominion. If game $\mathcal{G}$ is (n, d)-small then the tree of a progress measure on $\mathcal{G}$ is $(n, d/2 + 1)$-small.*

We define the *Steven progress-measure Strahler number* of a parity game $\mathcal{G}$ to be the smallest Strahler number of a tree of a progress measure on $\mathcal{G}$. The following theorem refines and strengthens Theorems 2 and 23 by establishing that the Steven Strahler number and the Steven progress-measure Strahler number of a parity game nearly coincide.

▶ **Theorem 24.** *The Steven Strahler number and the Steven progress-measure Strahler number of a parity game differ by at most* 1*.*

The translations between progress measures and attractor decompositions are as given by Daviaud, Jurdziński, and Lazić [5]; here we point out that they do not increase the Strahler number of the underlying trees by more than 1. This coincidence of the two complexity measures, one based on attractor decompositions and the other based on progress measures, allows us in Section 7 to use a progress measure lifting algorithm to solve games with bounded Strahler number.

7 Strahler-Universal Progress Measure Lifting Algorithm

Jurdziński and Lazić [20, Section IV] have implicitly suggested that the progress-measure lifting algorithm [19] can be run on any ordered tree and they have established the correctness of such an algorithm if their *succinct multi-counters trees* were used. This has been further clarified by Czerwiński et al. [4, Section 2.3], who have explicitly argued that any $(n, d/2)$-universal ordered tree is sufficient to solve an (n, d)-small parity game in this way. We make explicit a more detailed observation that follows using the same standard arguments (see, for example, Jurdziński and Lazić [20, Theorem 5]).

▶ **Proposition 25.** *Suppose the progress measure-lifting algorithm is run on a parity game $\mathcal{G}$ and on an ordered tree T. Let D be the largest Steven dominion in $\mathcal{G}$ on which there is a Steven progress measure whose tree can be embedded in T. Then the algorithm returns a Steven dominion strategy on D.*

An elementary corollary of this observation is that if the progress-measure lifting algorithm is run on the tree of a progress measure on some Steven dominion in a parity game, then the algorithm produces a Steven dominion strategy on a superset of that dominion. Note that this is achieved in polynomial time because the tree of a progress measure on an (n, d)-small parity game is $(n, d/2)$-small and the running time of the algorithm is dominated by the size of the tree [20, Section IV.B].

▶ **Theorem 26.** *There is an algorithm for solving (n, d)-small parity games of Strahler number k in quasi-linear space and time $n^{O(1)} \cdot (d/2k)^k = n^{k \lg(d/k)/\lg n + O(1)}$, which is polynomial in n if $k \cdot \lg(d/k) = O(\log n)$.*

Proof. By Proposition 3, we may assume that $k \leq 1 + \lg n$. In order to solve an (n, d)-small parity game of Steven Strahler number k, run the progress-measure lifting algorithm for Steven on tree $\mathcal{B}^{k+1}_{\lfloor \lg n \rfloor, d/2+1}$, which is $(k + 1)$-Strahler $(n, d/2 + 1)$-universal by Lemma 14 and Corollary 21. By Theorem 24 and by Proposition 25, the algorithm will then return a Steven dominion strategy on the largest Steven dominion. The running time and space upper bounds follow from Theorem 16, by the standard analysis of progress-measure lifting as in [20, Theorem 7], and by Lemma 22. ◀

▶ Remark 27. We highlight the $k \cdot \lg(d/k) = O(\log n)$ criterion from Theorem 26 as offering a novel trade-off between two natural structural complexity parameters of parity games (number of of priorities d and the Strahler/Lehtinen number k) that enables solving them in time that is polynomial in the number of vertices n. It includes as special cases both the $d < \lg n$ criterion of Calude et al. [3, Theorem 2.8] and the $d = O(\log n)$ criterion of Jurdziński and Lazić [20, Theorem 7] (set $k = \lfloor \lg n \rfloor + 1$ and use Propositions 4 and 3 to justify it), and the $k = O(1)$ criterion of Lehtinen [24, Theorem 3.6] (by Theorem 8).

We argue that the new $k \cdot \lg(d/k) = O(\log n)$ criterion (Theorem 26) enabled by our results (coincidence of the Strahler and the Lehtinen numbers: Theorem 8) and techniques (small and efficiently navigable Strahler-universal trees: Theorem 16, Corollary 21, and Lemma 22) considerably expands the asymptotic ranges of the natural structural complexity parameters in which parity games can be solved in polynomial time. We illustrate it by considering the scenario in which the rates of growth of both k and $\lg d$ as functions of n are $O\big(\sqrt{\log n}\big)$, i.e., d is $2^{O\left(\sqrt{\log n}\right)}$. Note that the number of priorities d in this scenario is allowed to grow as fast as $2^{b \cdot \sqrt{\lg n}}$ for an arbitrary positive constant b, which is significantly larger than what is allowed by the $d = O(\log n)$ criterion of Jurdziński and Lazić [20, Theorem 7]. Indeed, its rate of growth is much larger than any poly-logarithmic function of n, because for every positive constant c, we have $(\lg n)^c = 2^{c \cdot \lg \lg n}$, and $c \cdot \lg \lg n$ is exponentially smaller than $b \cdot \sqrt{\lg n}$. At the same time, the $O\big(\sqrt{\log n}\big)$ rate of growth allowed in this scenario for the Strahler number k substantially exceeds $k = O(1)$ required by Lehtinen [24, Theorem 3.6].

References

1 P. Baldan, B. König, C. Mika-Michalski, and T. Padoan. Fixpoint games on continuous lattices. *Proceedings of the ACM on Programming Languages*, 3(POPL, January 2019):26:1–26:29, 2019.

2 J. C. Bradfield and I. Walukiewicz. *Handbook of Model Checking*, chapter The mu-calculus and model checking, pages 871–919. Springer, 2018.

3 C. S. Calude, S. Jain, B. Khoussainov, W. Li, and F. Stephan. Deciding parity games in quasipolynomial time. In *STOC 2017*, pages 252–263, Montreal, QC, Canada, 2017. ACM.

4 W. Czerwiński, L. Daviaud, N. Fijalkow, M. Jurdziński, R. Lazić, and P. Parys. Universal trees grow inside separating automata: Quasi-polynomial lower bounds for parity games. In *Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019*, pages 2333–2349, San Diego, CA, 2019. SIAM.

5 L. Daviaud, M. Jurdziński, and R. Lazić. A pseudo-quasi-polynomial algorithm for mean-payoff parity games. In *33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018*, pages 325–334, Oxford, UK, 2018. ACM.

6 L. Daviaud, M. Jurdziński, and K. Lehtinen. Alternating weak automata from universal trees. In *30th International Conference on Concurrency Theory, CONCUR 2019*, volume 140 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 18:1–18:14, Amsterdam, the Netherlands, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

7 L. Daviaud, M. Jurdziński, and K. S. Thejaswini. The Strahler number of a parity game, 2020. `arXiv:2003.08627`.

8 E. A. Emerson and C. S. Jutla. Tree automata, mu-calculus and determinacy. In *32nd Annual Symposium on Foundations of Computer Science*, pages 368–377, San Juan, Puerto Rico, 1991. IEEE Computer Society.

9 E. A. Emerson, C. S. Jutla, and P. Sistla. On model-checking for fragments of μ-calculus. In *CAV 1993*, volume 697 of *LNCS*, pages 385–396, Elounda, Greece, 1993. Springer.

10 A. P. Ershov. On programming of arithmetic operations. *Communications of the ACM*, 1(8):3–6, 1958.

11 J. Esparza, M. Luttenberger, and M. Schlund. A brief history of Strahler numbers—with a preface. Technical report, Technical University of Munich, 2016.

12 J. Fearnley. Exponential lower bounds for policy iteration. In *ICALP 2010*, volume 6199 of *LNCS*, pages 551–562, Bordeaux, France, 2010. Springer.

13 J. Fearnley, S. Jain, B. de Keijzer, S. Schewe, F. Stephan, and D. Wojtczak. An ordered approach to solving parity games in quasi-polynomial time and quasi-linear space. *International Journal on Software Tools for Technology Transfer*, 21(3):325–349, 2019.

14 O. Friedmann. An exponential lower bound for the parity game strategy improvement algorithm as we know it. In *LICS 2009*, pages 145–156, Los Angeles, CA, USA, 2009. IEEE Computer Society.

15 O. Friedmann. A subexponential lower bound for Zadeh's pivoting rule for solving linear programs and games. In *IPCO 2011*, volume 6655 of *LNCS*, pages 192–206, New York, NY, USA, 2011. Springer.

16 O. Friedmann, T. D. Hansen, and U. Zwick. Subexponential lower bounds for randomized pivoting rules for the simplex algorithm. In *STOC 2011*, pages 283–292, San Jose, CA, USA, 2011. ACM.

17 E. Grädel, W. Thomas, and T. Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research*, volume 2500 of *LNCS*. Springer, 2002.

18 D. Hausmann and L. Schröder. Computing nested fixpoints in quasipolynomial time, 2019. `arXiv:1907.07020`.

19 M. Jurdziński. Small progress measures for solving parity games. In *17th Annual Symposium on Theoretical Aspects of Computer Science*, volume 1770 of *LNCS*, pages 290–301, Lille, France, 2000. Springer.

20 M. Jurdziński and R. Lazić. Succinct progress measures for solving parity games. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017*, pages 1–9, Reykjavik, Iceland, 2017. IEEE Computer Society.

21 M. Jurdziński and R. Morvan. A universal attractor decomposition algorithm for parity games, 2020. `arXiv:2001.04333`.

22 M. Jurdziński, M. Paterson, and U. Zwick. A deterministic subexponential algorithm for solving parity games. *SIAM Journal on Computing*, 38(4):1519–1532, 2008.

23 D. E. Knuth. *The Art of Computer Programming*. Addison-Wesley, 1973.

24 K. Lehtinen. A modal μ perspective on solving parity games in quasi-polynomial time. In *33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018*, pages 639–648, Oxford, UK, 2018. IEEE.

25 K. Lehtinen and U. Boker. Register games, April 2020. `arXiv:1902.10654`.

26 K. Lehtinen, S. Schewe, and D. Wojtczak. Improving the complexity of Parys' recursive algorithm, 2019. `arXiv:1904.11810`.

27 R. McNaughton. Infinite games played on finite graphs. *Annals of Pure and Applied Logic*, 65(2):149–184, 1993.

28 P. Parys. Parity games: Zielonka's algorithm in quasi-polynomial time. In *MFCS 2019*, volume 138 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 10:1–10:13, Aachen, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

29 P. Parys. Parity games: Another view on Lehtinen's algorithm. In *28th EACSL Annual Conference on Computer Science Logic, CSL 2020*, volume 152 of *LIPIcs*, pages 32:1–32:15, Barcelona, Spain, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

30 X. G. Viennot. Trees everywhere. In *15th Colloquium on Trees in Algebra and Programming*, volume 431 of *LNCS*, pages 18–41, Copenhagen, Denmark, 1990. Springer.

31 J. Vöge and M. Jurdziński. A discrete strategy improvement algorithm for solving parity games. In *CAV 2000*, volume 1855 of *LNCS*, pages 202–215, Chicago, IL, USA, 2000. Springer.

32 W. Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200(1–2):135–183, 1998.

On the Structure of Solution Sets to Regular Word Equations

Joel D. Day[1]
Loughborough University, UK
J.Day@lboro.ac.uk

Florin Manea
Georg-August Universität, Göttingen, Germany
florin.manea@informatik.uni-goettingen.de

Abstract

For quadratic word equations, there exists an algorithm based on rewriting rules which generates a directed graph describing all solutions to the equation. For regular word equations – those for which each variable occurs at most once on each side of the equation – we investigate the properties of this graph, such as bounds on its diameter, size, and DAG-width, as well as providing some insights into symmetries in its structure. As a consequence, we obtain a combinatorial proof that the problem of deciding whether a regular word equation has a solution is in NP.

2012 ACM Subject Classification Mathematics of computing → Combinatorics on words; Theory of computation → Problems, reductions and completeness

Keywords and phrases Quadratic Word Equations, Regular Word Equations, String Solving, NP

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.124

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

Funding Florin Manea's work was supported by the DFG grant MA 5725/2-1.

1 Introduction

A *word equation* is a tuple (α, β), usually written $\alpha \doteq \beta$, such that α and β are words comprised of letters from a *terminal alphabet* $\Sigma = \{\mathtt{a}, \mathtt{b}, \ldots\}$ and *variables* from a set $X = \{x, y, z, \ldots\}$. Solutions are substitutions of the variables for words in Σ^* making both sides identical. For example, one solution to the word equation $x\mathtt{aby} \doteq y\mathtt{ba}x$ is given by $x \to \mathtt{b}$ and $y \to \mathtt{bab}$. A system of equations is a set of equations, and a solution to the system is a substitution for the variables which is a solution to all the equations in the system.

One of the most fundamental questions concerning word equations is the satisfiability problem: determining whether or not a word equation has a solution. Makanin [22] famously showed in 1977 that the satisfiability problem for word equations is decidable by giving a general algorithm. Since then, several further algorithms have been presented. Most notable among these are the algorithm given by Plandowski [25] which demonstrated that the satisfiability problem is in PSPACE, the algorithm based on Lempel-Ziv encodings by Plandowksi and Rytter [26], and the method of recompression by Jeż, which has since been shown to require only non-deterministic linear space [15, 16]. On the other hand, it is easily seen that solving word equations is NP-hard due to fact that the subcase when one side of the equation consists only of terminals is exactly the pattern matching problem which is NP-complete [3, 12]. It remains a long-standing open problem whether or not the satisfiability problem for word equations is contained in NP.

[1] Corresponding author

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 124; pp. 124:1–124:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Recently, there has been elevated interest in solving more general versions of the satisfiability problem, originating from practical applications in e.g. software verification where several *string solving* tools capable of solving word equations are being developed [1, 4, 6, 18, 2] and database theory [14, 13], where one asks whether a given (system of) word equation(s) has a solution which satisfies some additional constraints. Prominent examples include requiring that the substitution for a variable x belongs to some regular language $\mathcal{L}_x$ (regular constraints), or that the lengths of the substitutions of the variables satisfy a set of given linear diophantine equations. Adding regular constraints makes the problem PSPACE complete (see [10, 25, 27], while it is another long standing open problem whether the satisfiability problem with length constraints is decidable. There are also many other kinds of constraints, however many lead to undecidable variants of the satisfiability problem [7, 19]. The main difficulty in dealing with additional constraints is that the solution-sets to word equations are often infinite sets with complex structures. For example, they are not parametrizable [24], and the set of lengths of solutions is generally not definable in Presburger arithmetic [20]. Thus, a better understanding of the solution-sets and their structures is a key aspect of improving our ability to solve problems relating to word equations both in theory and practice.

Quadratic word equations (QWEs) are equations in which each variable occurs at most twice. For QWEs, a conceptually simple and easily implemented algorithm exists which produces a representation of the set of all solutions as a graph. Despite this, however, the satisfiability problem for quadratic equations remains NP-hard, even for severely restricted subclasses [8, 11], while inclusion in NP, and whether the satisfiability problem with length constraints is decidable, have remained open for a long time, just as for the general case.

The algorithm solving QWEs is based on iteratively rewriting the equation(s) according to some simple rules called *Nielsen transformations.* If there exists a sequence of transformations from the original equation to the trivial equation $\varepsilon \doteq \varepsilon$, then the equation has a solution. Otherwise, there is no solution. Hence the satisfiability problem becomes a reachability problem for the underlying rewriting transformation relation, which we denote $\Rightarrow_{NT}$. It is natural to represent this relation as a directed graph $\mathcal{G}^{\Rightarrow_{NT}}$ in which the vertices are word equations and the edges are the rewriting transformations. This has the advantage that the set of all solutions to an equation E corresponds exactly to the set of walks in the graph starting at E and finishing at the trivial equation $\varepsilon \doteq \varepsilon$.[2] Consequently, the properties of the subgraph of $\mathcal{G}^{\Rightarrow_{NT}}$ containing all vertices reachable from E (denoted $\mathcal{G}^{\Rightarrow_{NT}}_{[E]}$) are also informative about the set of solutions to the equation. For example, in [24] a connection is made between the non-parameterisability of the solution set of E and the occurrence of combinations of cycles in the graph. Since equations with a parametrisable solution set are much easier to work with when dealing with additional constraints, this also establishes a connection between the structure of $\mathcal{G}^{\Rightarrow_{NT}}_{[E]}$ and the potential (un)decidability of variants of the satisfiability problem. Moreover, new insights into the structure and symmetries of these graphs are necessary for better understanding and optimising the practical performance of the algorithm.

[2] Each choice of edge in a walk can be seen as a decision about the corresponding solution. It is not necessarily true that different walks will result in different solutions. However, all possible decisions are accounted for, so it is guaranteed that for every solution there is a walk from E to $\varepsilon \doteq \varepsilon$ which corresponds to that solution.

Our Contribution

We consider a subclass of QWEs called regular equations (RWEs) introduced in [23]. A word equation is *regular* if each variable occurs at most once on each side of the equation. Thus, for example, $x\mathtt{ab}y \doteq y\mathtt{ba}x$ is regular while $x\mathtt{ab}x \doteq y\mathtt{ba}y$ is not. Understanding RWEs is a vital step towards understanding the quadratic case, not only because they constitute a significant and general subclass, but also because many non-regular quadratic equations can exhibit the same behaviour as regular ones (consider, e.g. $zz \doteq x\mathtt{ab}yy\mathtt{ba}x$ for which all solutions must satisfy $z = x\mathtt{ab}y = y\mathtt{ba}x$). The satisfiability problem was shown in [8] to be NP-hard for RWEs, and shown to be NP-complete in [9] for some restricted subclasses of RWEs including the classes of regular-reversed and regular-ordered equations.

For RWEs E, we investigate the structure of the graphs $\mathcal{G}^{\Rightarrow_{NT}}_{[E]}$, and as a consequence, are able to describe some of their most important properties. We achieve this by first noting that $\mathcal{G}^{\Rightarrow_{NT}}_{[E]}$ can be divided into strongly connected components $\mathcal{G}^{\Rightarrow}_{[E']}$ for which all the vertices are equations of the same length ($\Rightarrow$ shall be used to denote the restriction of $\Rightarrow_{NT}$ to length preserving transformations only). The “full” graph $\mathcal{G}^{\Rightarrow_{NT}}_{[E]}$ is comprised of these individual components $\mathcal{G}^{\Rightarrow}_{[E']}$ arranged in a DAG-like structure of linear depth (see Section 3) and therefore many properties and parameters of the “full” graph $\mathcal{G}^{\Rightarrow_{NT}}_{[E]}$ are determined by the equivalent properties and parameters of the individual components $\mathcal{G}^{\Rightarrow}_{[E']}$. We then focus on the structure of the subgraphs $\mathcal{G}^{\Rightarrow}_{[E']}$, and as a result are able to give bounds on certain parameters such as diameter, size, and DAG-width.

Our structural results come in two stages, based on whether the equation belongs to a the class of “jumbled” equations introduced in Section 4.3. In the first stage, we consider equations which are not jumbled, and we show that for all such equations E, there exists a jumbled equation $\hat{E}$ such that $\mathcal{G}^{\Rightarrow}_{[E]}$ is comprised mainly of several well-connected near-copies of $\mathcal{G}^{\Rightarrow}_{[\hat{E}]}$. For jumbled equations $\hat{E}$, we show in Section 4.4 that every vertex in $\mathcal{G}^{\Rightarrow}_{[\hat{E}]}$ is close to a vertex in a certain normal form. We show that the vertices in this normal form are determined to a large extent by a property invariant under $\Rightarrow$ introduced in Section 4.2.

With regards to the diameter of $\mathcal{G}^{\Rightarrow}_{[E']}$, we give upper bounds which are polynomial in the length of the equation. It follows that the diameter of the full graph $\mathcal{G}^{\Rightarrow_{NT}}_{[E]}$ is also polynomial, and consequently, that the satisfiability problem for RWEs is NP-complete. This can be generalised to systems of equations satisfying a natural extension of the regularity property (see Section 4.7). We also give exact upper and lower bounds on the number of vertices[3] in $\mathcal{G}^{\Rightarrow}_{[E']}$ for a subclass of RWEs called *basic* RWEs (see Section 4.1), as well as describing exactly for which equations these bounds are achieved. For RWEs which are not basic, we can infer similar bounds, at the cost of a small (linear in the length of the equation) degree of imprecision. Since in the worst case (e.g. for equations without a solution), running the algorithm will perform a full “search” of the graph, the number of vertices is integral to the running time of the algorithm, and is potentially a better indicator of difficult instances than the complexity class alone. An example of this, comes from comparing two subclasses of RWEs called regular-ordered and regular rotated equations. It follows from our results that while both classes have an NP-complete satisfiability problem, if E' is regular-ordered, then $\mathcal{G}^{\Rightarrow}_{[E']}$ will contain at most n vertices, where n is the length of the equation, while if E' is regular rotated, but not regular-ordered, then $\mathcal{G}^{\Rightarrow}_{[E']}$ will contain $\frac{n!}{2}$ vertices, indicating a vast difference in the number of vertices the algorithm would have to visit.

[3] We consider the number of vertices, rather than edges, because it is the number of vertices which is relevant to the performance of the algorithm, and by definition of $\Rightarrow_{NT}$, the out-degree of the graph is bounded by a constant so the the number of edges is linear in the number of vertices.

Motivated by generalisations of the satisfiability problem permitting additional constraints, we also consider the connectivity of the graphs $\mathcal{G}^{\Rightarrow_{NT}}_{[E]}$. To do this, we use DAG-width, a measure for directed graphs which is in several ways analogous to treewidth for undirected graphs. Intuitively, equations for which $\mathcal{G}^{\Rightarrow_{NT}}_{[E]}$ has low DAG-width are likely to be more amenable when considering additional constraints such as length constraints (see Section 3.3). We give an example class of equations for which the DAG-width is unbounded, as well as a class for which the DAG-width is at most two. The latter includes the class of regular-ordered equations which is the most general subclass of QWEs for which it is known that the satisfiability problem with length constraints is decidable [20], and we expect that both cases will be interesting classes to consider in the context of this problem.

2 Preliminaries

For a set S, we denote the cardinality of S by $\text{Card}(S)$. Let Σ be an alphabet. By Σ^*, we denote the set of all words over Σ, and by ε the empty word. By Σ^+, we denote the free semigroup $\Sigma^*\backslash\{\varepsilon\}$. A word u is a prefix (resp. suffix) of a word w if there exists v such that $w = uv$ (resp. $w = vu$). Similarly, u is a factor of w if there exist v, v' such that $w = vuv'$. A prefix/suffix/factor is *proper* if is neither the whole word w, nor ε. The length of a word w is denoted $|w|$, while for $\mathtt{a} \in \Sigma$, $|w|_\mathtt{a}$ denotes the number of occurrences of $\mathtt{a}$ in w. For a word $w = \mathtt{a}_1\mathtt{a}_2 \dots \mathtt{a}_n$ with $\mathtt{a}_i \in \Sigma$ for $1 \leq i \leq n$, the notation $w[i]$ refers to the letter $\mathtt{a}_i$ in the i^{th} position. By w^R, we denote the reversal $\mathtt{a}_n\mathtt{a}_{n-1} \dots \mathtt{a}_1$ of the word w. Two words w_1, w_2 are conjugate (written $w_1 \sim w_2$) if there exist u, v such that $w_1 = uv$ and $w_2 = vu$.

We shall generally distinguish between two types of alphabet: an infinite set $X = \{x_1, x_2, \dots\}$ of variables, and a set $\Sigma = \{\mathtt{a}, \mathtt{b}, \dots\}$ of terminal symbols. We shall assume that $\text{Card}(\Sigma) \geq 2$, and that there exists an order on X leading to a lexicographic order on X^*. For a word $\alpha \in (X \cup \Sigma)^*$, we shall denote by $\text{var}(\alpha)$ the set $\{x \in X \mid x \text{ is a factor of } \alpha\}$. We shall denote by $\text{qv}(\alpha)$ the set $\{x \in \text{var}(\alpha) \mid |\alpha|_x = 2\}$. A word equation is a tuple $(\alpha, \beta) \in (X \cup \Sigma)^* \times (X \cup \Sigma)^*$, usually written $\alpha \doteq \beta$. Solutions are morphisms $h : (X \cup \Sigma)^* \to \Sigma^*$ with $h(\mathtt{a}) = \mathtt{a}$ for all $\mathtt{a} \in \Sigma$ such that $h(\alpha) = h(\beta)$. The satisfiability problem is the problem of deciding algorithmically whether a given word equation has a solution. For equations E given by $\alpha \doteq \beta$, we shall often extend notations regarding words in $(X \cup \Sigma)^*$ to E for convenience, so that, e.g. $|E| = |\alpha\beta|$, $\text{var}(E) = \text{var}(\alpha\beta)$ and $\text{qv}(E) = \text{qv}(\alpha\beta)$. An equation $\alpha \doteq \beta$ is quadratic if $|\alpha\beta|_x \leq 2$ for all $x \in X$. It is regular if $|\alpha|_x \leq 1$ and $|\beta|_x \leq 1$ hold for all $x \in X$. Thus all regular equations are quadratic, but not all quadratic equations are regular. We shall usually abbreviate regular (resp. quadratic) word equation to RWE (resp. QWE). For $Y \subseteq X$, let $\pi_Y : (X \cup \Sigma^*) \to Y^*$ be the morphism such that $\pi_Y(x) = x$ if $x \in Y$ and $\pi_Y(x) = \varepsilon$ otherwise; i.e. π_Y is a projection from $(X \cup \Sigma)^*$ onto Y^*. A regular equation E given by $\alpha \doteq \beta$ is regular-ordered if $\pi_{\text{qv}(E)}(\alpha) = \pi_{\text{qv}(E)}(\beta)$, it is regular rotated if $\pi_{\text{qv}(E)}(\alpha) \sim \pi_{\text{qv}(E)}(\beta)$ and it is regular reversed if $\pi_{\text{qv}(E)}(\alpha) = \pi_{\text{qv}(E)}(\beta)^R$.

Given a set S and binary relation $\mathcal{R} \subseteq S \times S$, we denote the reflexive-transitive closure of $\mathcal{R}$ as $\mathcal{R}^*$. For each $s \in S$, we denote by $[s]_\mathcal{R}$ the set $\{s' \mid s\mathcal{R}^*s'\}$. The relation $\mathcal{R}$ may be represented as a directed graph, which we denote $\mathcal{G}^\mathcal{R}$, with vertices from S and edges from $\mathcal{R}$. Usually, we will be interested in the subgraph of $\mathcal{G}^\mathcal{R}$ containing vertices belonging to $[s]$ for some $s \in S$. Thus, for a subset T of S we shall denote by $\mathcal{G}^\mathcal{R}_T$ the subgraph of $\mathcal{G}^\mathcal{R}$ containing vertices from T. Given a (directed) graph $\mathcal{G}$, with vertices $V(\mathcal{G})$ and edges $E(\mathcal{G})$, a root vertex is some $v \in V(\mathcal{G})$ such that there does not exist $(u, v) \in E(\mathcal{G})$. We denote by $\text{diam}(\mathcal{G})$ the diameter: the maximum length of a shortest (directed) path between two vertices. For $W, V' \subseteq V(\mathcal{G})$, we say that W *guards* V' if for all $(u, v) \in E(\mathcal{G})$ with $u \in V'$, we have $v \in V' \cup W$. If $\mathcal{G}$ is acyclic, we write $v_1 \leq_{\mathcal{G}} v_2$ if there is a directed path from v_1 to v_2 in $\mathcal{G}$ or $v_1 = v_2$. Following [5], A DAG-decomposition of $\mathcal{G}$ is a pair (D, χ) such that D is a directed acyclic graph (DAG) with vertices $V(D)$, and $\chi = \{X_d \mid d \in V(D)\}$ is a family of subsets of $V(\mathcal{G})$ satisfying:

(D1) $V(\mathcal{G}) = \bigcup\limits_{d \in V(D)} X_d$,

(D2) if $d, d', d'' \in V(D)$ such that $d \leq_D d' \leq_D d''$, then $X_d \cap X_{d''} \subseteq X_{d'}$,

(D3) For all edges (d, d') of D, $X_d \cap X_{d'}$ guards $X_{\geq d'} \backslash X_d$, where $X_{\geq d'} = \bigcup\limits_{d'' \geq_D d'} X_{d''}$, and for all root vertices d, $X_{\geq d}$ is guarded by $\emptyset$.

The width of the DAG-decomposition is $\max\{\text{Card}(X_d) \mid d \in V(D)\}$. The DAG-width of $\mathcal{G}$ is the minimum width of any possible DAG-decomposition of $\mathcal{G}$ and is denoted $\text{dgw}(\mathcal{G})$.

3 Solving regular word equations

In this section we present the algorithm for solving QWEs discussed in the introduction as a rewriting system given by a relation $\Rightarrow_{NT}$. The rewriting transformations are derived from morphisms called Nielsen transformations, and we shall abuse this terminology slightly and generally also refer to the rewriting transformations themselves as Nielsen transformations. The Nielsen transformations never introduce new variables or terminal symbols, and never increase the length of the equation. They also preserve the properties of being quadratic (resp. regular). Thus, given a quadratic (resp. regular) word equation, the possible space of all equations reachable via Nielsen transformations is finite. Moreover, given an equation which has a solution h, there is always at least one Nielsen transformation which produces an equation which has a solution, such that the new equation or the new solution is shorter than the previous one. It follows that, given an equation which possesses a solution, it is possible to reach the equation $\varepsilon \doteq \varepsilon$ after finitely many rewriting steps. For a more detailed description of the algorithm, we refer the reader to e.g. Chapter 12 of [21].

3.1 Nielsen transformations

The Nielsen transformations are defined as follows: for $x \in X \cup \Sigma$ and $y \in X$, let $\psi_{x<y} : (X \cup \Sigma)^* \to (X \cup \Sigma)^*$ be the morphism given by $\psi_{x<y}(y) = xy$ and $\psi_{x<y}(z) = z$ if $z \neq y$. We define the rewriting transformations via the relations $\Rightarrow_L, \Rightarrow_R, \Rightarrow_>$ as follows. Suppose we have a QWE E of the form $x\alpha \doteq y\beta$ where $x, y \in X \cup \Sigma$ and $\alpha, \beta \in (X \cup \Sigma)^*$. Then:

1. if $x \in \text{qv}(E)$ and $x \neq y$, then $x\alpha \doteq y\beta \Rightarrow_L x\psi_{y<x}(\alpha) \doteq \psi_{y<x}(\beta)$, and
2. if $y \in \text{qv}(E)$ and $x \neq y$, then $x\alpha \doteq y\beta \Rightarrow_R \psi_{x<y}(\alpha) \doteq y\psi_{x<y}(\beta)$, and
3. if $x \in X \backslash \text{qv}(E)$, then $x\alpha \doteq y\beta \Rightarrow_> x\alpha \doteq \beta$, and
4. if $y \in X \backslash \text{qv}(E)$, then $x\alpha \doteq y\beta \Rightarrow_> \alpha \doteq y\beta$, and
5. if $x = y$, then $x\alpha \doteq y\beta \Rightarrow_> \alpha \doteq \beta$.

Moreover, for a QWE E of the form $\alpha \doteq \beta$ with $\alpha, \beta \in (X \cup \Sigma)^*$, and for each $Y \subseteq \text{var}(E)$, we have the additional transformations $\alpha \doteq \beta \Rightarrow_> \pi_{X \backslash \{Y\}}(\alpha) \doteq \pi_{X \backslash \{Y\}}(\beta)$. Now, our full rewriting relation, $\Rightarrow_{NT}$, is given by $\Rightarrow_L \cup \Rightarrow_R \cup \Rightarrow_>$. For convenience, we shall define $\Rightarrow$ to be $\Rightarrow_L \cup \Rightarrow_R$. We shall call the rewriting transformations in $\Rightarrow$ *length-preserving*, since they are exactly those for which the resulting equation has the same length as the original.

▶ Remark 1. Let E, E' be QWEs such that $E \Rightarrow_{NT} E'$. If E is regular, then E' is regular. Moreover, if $E \Rightarrow E'$, then $\text{var}(E) = \text{var}(E')$, $\text{qv}(E) = \text{qv}(E')$, and $|E| = |E'|$.

If E_1, E_2 are RWEs such that $E_1 \Rightarrow_L E_2$, then it follows from the definitions that there exist $x, y \in X$ and $\alpha_1, \alpha_2, \beta_1, \beta_2, \in (X \backslash \{x, y\})^*$ such that E_1 is given by $x\alpha_1 y\alpha_2 \doteq y\beta_1 x\beta_2$ and E_2 is given by $x\alpha_1 y\alpha_2 \doteq \beta_1 yx\beta_2$. Extending this observation to multiple applications of

$\Rightarrow_L$, we may conclude that the set $\{E_2 \mid E_1 \Rightarrow_L^* E_2'\}$ is exactly the set $\{x\alpha_1 y\alpha_2 \doteq \beta_3 x\beta_2 \mid \beta_3 \sim y\beta_1\}$. A similar statement can be made for $\Rightarrow_R^*$. Consequently, $\Rightarrow_L^*$ and $\Rightarrow_R^*$ are symmetric. Since they are reflexive and transitive by definition, we get the following.

▶ Remark 2. Let E be a RWE and $Z \in \{L, R\}$. Then $\mathrm{Card}(\{E' \mid E \Rightarrow_Z^* E'\}) < |E|$ and $\Rightarrow_Z^*$ is an equivalence relation. It follows that $\Rightarrow^*$ is also an equivalence relation.

The following well-known result forms the basis for the algorithm for solving QWEs.

▶ **Theorem 3** ([21]). *Let E be a QWE. Then E has a solution if and only if $E \Rightarrow_{NT}^* \varepsilon \doteq \varepsilon$.*

3.2 The graph of all solutions

Theorem 3 provides the basis for treating the satisfiability of QWEs as a reachability problem for the rewriting relation $\Rightarrow_{NT}$. Since any relation R is naturally represented as a (directed) graph $\mathcal{G}^R$, it is also natural to interpret the resulting algorithm as a search in the graph $\mathcal{G}_{[E]}^{\Rightarrow_{NT}}$, in order to determine whether a path exists in the graph from the original equation E to the trivial equation $\varepsilon \doteq \varepsilon$. In fact, the graph $\mathcal{G}_{[E]}^{\Rightarrow_{NT}}$ can tell us significantly more than simply whether a solution to E exists: every walk from E to $\varepsilon \doteq \varepsilon$ in $\mathcal{G}_{[E]}^{\Rightarrow_{NT}}$ corresponds to a solution to E and likewise, every solution to E is represented by a walk in $\mathcal{G}_{[E]}^{\Rightarrow_{NT}}$ from E to $\varepsilon \doteq \varepsilon$. Thus the graphs $\mathcal{G}_{[E]}^{\Rightarrow}$ contain a full description of all solutions to E, and as such, their properties and structure are of inherent interest to the study of QWEs and their solutions. An immediate example of this is the diameter, which is strongly related to the complexity of the satisfiability problem, as demonstrated in the following proposition.

▶ **Proposition 4.** *Let $\mathcal{C}$ be a class of QWEs. Suppose there exists $k \in \mathbb{N}$ such that for each $E \in \mathcal{C}$, we have $\mathrm{diam}(\mathcal{G}_{[E]}^{\Rightarrow_{NT}}) \in O(|E|^k)$. Then the satisfiability problem for $\mathcal{C}$ is in NP.*

Many properties will be determined mostly (i.e. up to some small imprecision) on the subgraphs obtained by restricting our rewriting relation to length-preserving transformations only (i.e. to $\Rightarrow$). Since the rewriting relation $\Rightarrow_{NT}$ allows us to preserve or decrease the length, but never increase it again, any walk in the graph will visit a subgraph containing equations of each length only once, and in order of decreasing length.

The following proposition is an example of how we may infer a global property of $\mathcal{G}_{[E]}^{\Rightarrow_{NT}}$ from its "local" values in the individual subgraphs $\mathcal{G}_{[E']}^{\Rightarrow}$.

▶ **Proposition 5.** *Let E be a QWE. Then*

1. $\mathrm{diam}(\mathcal{G}_{[E]}^{\Rightarrow_{NT}}) \leq 1 + (|E| + 1) \max\{\mathrm{diam}(\mathcal{G}_{[E']}^{\Rightarrow}) \mid E \Rightarrow_{NT}^* E'\}$, *and*
2. $\mathrm{dgw}(\mathcal{G}_{[E]}^{\Rightarrow_{NT}}) = \max\{\mathrm{dgw}(\mathcal{G}_{[E']}^{\Rightarrow}) \mid E \Rightarrow_{NT}^* E'\}$.

In what follows, we shall focus predominantly on the structure of the (sub)graphs $\mathcal{G}_{[E']}^{\Rightarrow}$ corresponding to the length-preserving transformations given by $\Rightarrow$. This has the advantage of allowing us to apply further restrictions, including a reduction to the case of basic equations introduced in Section 4.1, without significantly altering the structure of the graph.

3.3 Solving equations modulo constraints

For many kinds of additional constraint, it is possible to adapt the algorithm by finding, for each Nielsen transformation, an appropriate corresponding transformation of the constraints. For example, if $x, y, z \in X$ and we have the length constraint $|x| = |z|$, when we apply the Nielsen transformation associated with $\psi_{y<x}$ to our equation, we replace each occurrence of x with yx. Thus, the updated constraint would be $|x| + |y| = |z|$. However, in some cases, including length constraints, the resulting space of possible combinations of equations and

constraints becomes infinite, meaning the algorithm is no longer guaranteed to terminate. A possible solution to this is to find finite descriptions of the potentially infinite sets of constraints which may occur alongside each equation. The task of finding such descriptions, and consequently the decidability of the corresponding extended satisfiability problems, is dependent on the structural properties of the graph, as can be seen e.g. in [20, 24].

4 Properties of the graphs $\mathcal{G}^{\Rightarrow NT}_{[E]}$ for regular equations E

The remainder of the paper concentrates on describing the structure of the graphs $\mathcal{G}_{[E]}$ for RWEs E. Our general description of $\mathcal{G}_{[E]}$ is comprised of several steps, with each one accounting for a particular aspect. The first step (Section 4.1) describes the effect of terminal symbols, single-occurrence variables, and 'decomposability' on the structure of $\mathcal{G}_{[E]}$, essentially reducing the structure of $\mathcal{G}_{[E]}$ to $\mathcal{G}_{[E']}$ for a "basic" equation E' which does not contain any of these features. The second step (Section 4.3) describes a particular symmetric structure which arises from the same factor(s) occurring on both sides of the equation once we have simplified the equations by eliminating the aforementioned features. This allows for a description of $\mathcal{G}_{[E']}$ as a combination of (near) copies of some smaller graph $\mathcal{G}_{[E'']}$ where E'' is a "jumbled equation" obtained by deleting the appropriate variables from E'. Finally, we are able to show (Section 4.4) that for jumbled equations E'', all vertices in $\mathcal{G}_{[E'']}$ are "close" to a vertex from a small subset conforming to a very particular structure called Lex Normal Form, allowing us to draw conclusions in Sections 4.5 and 4.6 about the diameter, number of vertices and connectivity (DAG-width) of $\mathcal{G}_{[E]}$. Finally, in Section 4.7 we note a generalisation of our results to systems of equations.

4.1 Basic equations: a convenient abstraction

The current section is devoted to reducing the study of the graphs $\mathcal{G}^{\Rightarrow}_{[E]}$ to the case of basic equations. This has several advantages, including a significant reduction in the size of the graphs which is useful for working with examples, as well as allowing for the simpler formulation of precise results, e.g. regarding the size of the graphs in Section 4.6, as well as avoiding unnecessary repetition in the formal statements and their proofs..

▶ **Definition 6** (Basic Equations). *Let E be a QWE given by $\alpha \doteq \beta$. Then E is* decomposable *if there exist proper prefixes α', β' of α and β such that* $\mathrm{var}(\alpha') \cap \mathrm{qv}(E) = \mathrm{var}(\beta') \cap \mathrm{qv}(E)$. *Otherwise, E is* indecomposable. *E is* basic *if it is indecomposable and $\alpha, \beta \in \mathrm{qv}(E)^*$.*

A RWE is basic only if both sides of the equation are permutations of the same set of variables, for example $x_1x_2x_3 \doteq x_3x_1x_2$ and $xywz \doteq wzxy$ are both basic and regular while $xyzw \doteq yxzw$ and $xy \doteq yz$ are not. It is easily verified that the property of being basic is preserved under $\Rightarrow^*$. In order to formally present our reduction from arbitrary RWEs to basic RWEs, we need the following notion for graphs which are structurally similar.

▶ **Definition 7** (Isolated path compression). *Let G_1, G_2 be (directed) graphs. We say that G_1 is an* isolated path compression *of order n of G_2 if G_2 may be obtained from G_1 by replacing each edge (e, e') in G_1 by a path $(e, e_1), (e_1, e_2), \ldots (e_{k-1}, e_k), (e_k, e')$ such that $k \leq n$ and $e_1, e_2, e_3, \ldots, e_k$ are new vertices unique to the edge (e, e').*

Informally, an isolated path compression of a graph is obtained simply by replacing "isolated paths" (paths whose internal vertices are not adjacent to to any vertices outside the path) of a bounded length with single edges. It is easy to see that many structural properties are thus preserved.

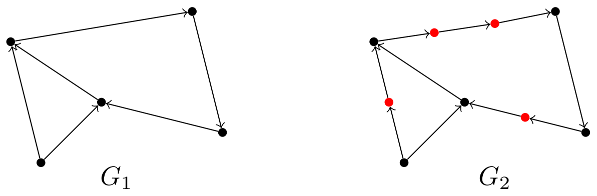


Figure 1 The graph G_1 is an isolated path compression of order two of the graph G_2.

▶ Remark 8. Consider graphs G_1, G_2 such that G_1 is an isolated path compression of order n of G_2. If $\mathrm{dgw}(G_1) = 1$, then $\mathrm{dgw}(G_2) \in \{1, 2\}$. If $\mathrm{dgw}(G_1) \geq 2$, then the $\mathrm{dgw}(G_1) = \mathrm{dgw}(G_2)$. Moreover, $\mathrm{diam}(G_2) \leq (n + 1)\,\mathrm{diam}(G_1)$, and the number of vertices (resp. edges) in G_2 is at most the number of vertices in G_1 plus n times the number of edges of G_1.

Using isolated path compressions, it is possible to describe the structure of the graph $\mathcal{G}^{\Rightarrow}_{[E]}$ for any RWE E in terms of the graph $\mathcal{G}^{\Rightarrow}_{[E']}$ for a basic RWE E'.

▶ **Theorem 9.** *Let E be a RWE given by $\alpha \doteq \beta$. Let α', β' be the shortest non-empty prefixes of α, β respectively such that $\mathrm{var}(\alpha') \cap \mathrm{qv}(E) = \mathrm{var}(\beta') \cap \mathrm{qv}(E)$. Let E' be the equation given by $\pi_{\mathrm{qv}(E)}(\alpha') \doteq \pi_{\mathrm{qv}(E)}(\beta')$. Then E' is basic, and $\mathcal{G}^{\Rightarrow}_{[E']}$ is isomorphic to an isolated path compression of order $|E|$ of $\mathcal{G}^{\Rightarrow}_{[E]}$.*

4.2 A useful invariant

When reasoning about the graphs $\mathcal{G}^{\Rightarrow}_{[E]}$, we need a way to help determine whether, for two equations E_1, E_2, we have $E_1 \Rightarrow^* E_2$. Usually, showing that $E_1 \Rightarrow^* E_2$ is not a problem, since it is sufficient to simply find a sequence of length-preserving Nielsen transformations from E_1 to E_2. However, showing that $E_1 \not\Rightarrow^* E_2$ presents more of a challenge. The naive way would be to enumerate all vertices in $\mathcal{G}^{\Rightarrow}_{[E_1]}$ and show that E_2 is not among them. However, this is not suitable for generic reasoning, and, even in concrete cases, is inelegant and time-consuming. The following is a property of basic RWEs which is preserved under $\Rightarrow$ and thus provides a concise and more general means for showing that $E_1 \not\Rightarrow^* E_2$. It is an indispensable component of the proofs of our main results.

▶ **Definition 10** (The invariant Υ_E)**.** *Let $\#$ be a new symbol not in X. Let E be a basic RWE such that $\mathrm{Card}(\mathrm{var}(E)) > 1$. Then we may write E as $x\alpha_1 y\alpha_2 \doteq y\beta_1 x\beta_2$ with $x, y \in X$ and $\alpha_1, \alpha_2, \beta_1, \beta_2 \in (X \backslash \{x, y\})^*$. Let $\mathcal{Z}_E = \mathrm{var}(\alpha_1\alpha_2\beta_1\beta_2) \cup \{\#\}$. Let the function $Q_E : \mathcal{Z}_E \to X^2$ be defined as follows: for each $z \in \mathcal{Z}_E \backslash \{\#\}$, let $Q_E(z) = (u, v)$ where uz is a factor of $x\alpha_1 y\alpha_2$ and vz is a factor of $y\beta_1 x\beta_2$. Let $Q_E(\#) = (u, v)$ where uy is a factor of $x\alpha_1 y\alpha_2$ and vx is a factor of $y\beta_1 x\beta_2$. Let $\Upsilon_E = \{Q_E(z) \mid z \in \mathcal{Z}_E\}$.*

▶ **Theorem 11.** *Let E_1, E_2 be basic RWEs such that $E_1 \Rightarrow^* E_2$. Then $\Upsilon_{E_1} = \Upsilon_{E_2}$.*

As an example, let E_1 be the basic RWE given by $xuzwy \doteq ywuxz$. Then $\mathcal{Z}_{E_1} = \{u, z, w, \#\}$ and Q_{E_1} is the function such that $Q_{E_1}(u) = (x, w)$, $Q_{E_1}(z) = (u, x)$, $Q_{E_1}(w) = (z, y)$ and $Q_{E_1}(\#) = (w, u)$. Thus, $\Upsilon_{E_1} = \{(w, u), (x, w), (u, x), (z, y)\}$. Similarly, if E_2 is the basic RWE given by $xuwzy \doteq yuxwz$, then $\Upsilon_{E_2} = \{(x, y), (u, x), (w, w), (z, u)\}$. Consequently, we may conclude that $E_1 \not\Rightarrow^* E_2$ (and symmetrically $E_2 \not\Rightarrow^* E_1$).

Since the invariant Υ_E provides a necessary condition on when two basic RWEs belong to the same equivalence class under $\Rightarrow^*$, we might also ask whether it is also sufficient, and hence characteristic. However, this is not the case. For instance, if E_1 is given by $xuvwy \doteq ywvux$ and E_2 is given by $xwvuy \doteq yuvwx$, then $\Upsilon_{E_1} = \Upsilon_{E_2} = \{(x, v), (u, w), (v, y), (w, u)\}$ but it can be verified by enumerating $[E_1]_{\Rightarrow}$ and $[E_2]_{\Rightarrow}$ that $E_1 \not\Rightarrow^* E_2$.

4.3 A special case of symmetry

The invariant property Υ_E introduced in the previous section is a set of pairs of variables. The case that $(x, x) \in \Upsilon_E$ for some $x \in \text{var}(E)$ is special in the sense that it leads to a particular symmetrical structure in $\mathcal{G}^{\Rightarrow}_{[E]}$. Intuitively, $(x, x) \in \Upsilon_E$ when there exists $y \in X$ and $\alpha \doteq \beta \in [E]_{\Rightarrow}$ such that xy is a factor of both α and β. Hence the number of variables x such that $(x, x) \in \Upsilon_E$ is, in a sense, a measure of the "jumbledness" of E.

▶ **Definition 12** (Jumbled Equations and $\Delta(E)$)**.** *Let E be a basic RWE. Let $\Delta(E) = \{x \in \text{var}(E) \mid (x, x) \in \Upsilon_E\}$. If $\text{Card}(\Delta(E)) = 0$, then E is* jumbled.

Note that since Υ_E is invariant under $\Rightarrow^*$, so is the property of being (not) jumbled. Any basic RWE E can be turned into a jumbled equation by simply erasing each $x \in \Delta(E)$.

▶ **Lemma 13.** *Let E be a basic RWE given by $\alpha \doteq \beta$ and let $Y = \text{var}(E)\backslash\Delta(E)$. Then the equation E_Y given by $\pi_Y(\alpha) \doteq \pi_Y(\beta)$ is jumbled.*

The following theorem describes the structure of $\mathcal{G}^{\Rightarrow}_{[E]}$ for a RWE E which is not jumbled in terms of $\mathcal{G}^{\Rightarrow}_{[E_Y]}$ where E_Y is obtained from E by deleting the variables in $\Delta(E)$.

▶ **Theorem 14.** *Let E be a basic RWE given by $\alpha \doteq \beta$. Let $Y = \text{var}(E)\backslash\Delta(E)$. Let E_Y be the equation $\pi_Y(\alpha) \doteq \pi_Y(\beta)$. Let $V = [E_Y]_{\Rightarrow}$. Let Φ be the set of morphisms $\varphi : Y^* \to \text{var}(E)^*$ satisfying $\varphi(y) \in \Delta(E)^*y$ for all $y \in Y$, and $\sum_{y \in Y} |\varphi(y)|_x = 1$ for all $x \in \Delta(E)$. For each $\varphi \in \Phi$, let $\varphi(V)$ denote the set $\{\varphi(\alpha') \doteq \varphi(\beta') \mid \alpha' \doteq \beta' \in V\}$. Then:*

1. *$\bigcup_{\varphi \in \Phi} \varphi(V) \subseteq [E]_{\Rightarrow}$,*
2. *for each $E' \in [E]_{\Rightarrow}$ and $Z \in \{L, R\}$, there exists $E'' \in \bigcup_{\varphi \in \Phi} \varphi(V)$ such that $E' \Rightarrow^*_Z E''$,*
3. *for each $\varphi \in \Phi$, there exists a subgraph $\mathcal{H}_\varphi$ of $\mathcal{G}^{\Rightarrow}_{[E]}$ containing $\varphi(V)$ such that $\mathcal{G}^{\Rightarrow}_{[E_Y]}$ is isomorphic to a structure-preserving contraction of order $\text{Card}(\Delta(E))$ of $\mathcal{H}_\varphi$.*
4. *if $d = \text{diam}(\mathcal{G}^{\Rightarrow}_{[E_Y]})$, then $\text{diam}(\mathcal{G}^{\Rightarrow}_{[E]}) \in O(d|E|^2)$.*

Theorem 14 deserves a few remarks. Firstly, we note that, recalling Remark 2, it follows from statements 1. and 2. of the theorem that $\bigcup_{\varphi \in \Phi} \varphi(V)$ is a dense subset of the vertices of $\mathcal{G}^{\Rightarrow}_{[E]}$ in the sense that every vertex is at most distance $|E|$ away from one contained in $\bigcup_{\varphi \in \Phi} \varphi(V)$. Moreover, since each morphism $\varphi \in \Phi$ is injective, the sets $\varphi(V)$ are pairwise disjoint. Consequently, $\mathcal{G}^{\Rightarrow}_{[E]}$ is made up of many (one for each $\varphi \in \Phi$) slightly modified copies of the graph $\mathcal{G}^{\Rightarrow}_{[\alpha \doteq \beta]}$, with the remaining vertices creating short paths between the different copies. Due to the bound on the diameter, we see that these copies are well connected. Finally, it is worth noting that $\text{Card}(\Phi)$ grows exponentially w.r.t. $\text{Card}(\Delta(E))$.

4.4 Normal forms and block decompositions

Having described the structure $\mathcal{G}^{\Rightarrow}_{[E]}$ for equations E which are not jumbled in the previous section, it remains to consider equations which are jumbled. In this case, the structure of $\mathcal{G}^{\Rightarrow}_{[E]}$ is more intricate and a different approach is required. Our main insight for jumbled equations is the existence of certain normal forms, from which every vertex is polynomial distance away. By constructing these normal forms in a specific way based on reversals, we are able to take full advantage of the invariant Υ_E from Section 4.2 when reasoning about which of these normal forms may occur. The first normal form is defined as follows.

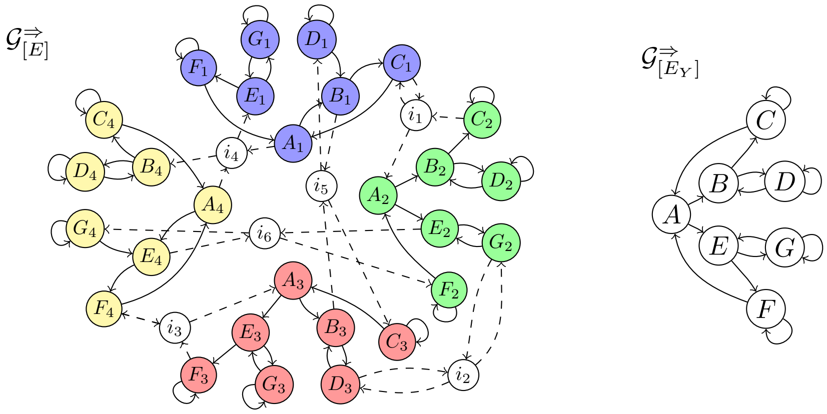


Figure 2 Example illustrating Theorem 14. On the left is $\mathcal{G}^{\Rightarrow}_{[E]}$ for the equation E given by $x_1yx_2x_3x_4 \doteq x_4x_3yx_2x_1$. Note that $\Delta(E) = \{y\}$, so $Y = \{x_1, x_2, x_3, x_4\}$ and E_Y is $x_1x_2x_3x_4 \doteq x_4x_3x_2x_1$. The graph $\mathcal{G}^{\Rightarrow}_{[E_Y]}$ is shown on the right, where the equations in $[E_Y]_{\Rightarrow}$ have been labelled A, B, C, D, E, F, G. The set Φ contains four morphisms φ_i, $1 \leq i \leq 4$, such that $\varphi_i(x_i) = yx_i$ and $\varphi_i(x_j) = x_j$ for $j \neq i$. For each $Z \in \{A, B, C, D, E, F, G\}$ given by $\alpha_Z \doteq \beta_Z$, Z_i denotes the equation $\varphi_i(\alpha_Z) \doteq \varphi_i(\beta_Z)$. The graph $\mathcal{G}^{\Rightarrow}_{[E]}$ contains a "near-copy" of $\mathcal{G}^{\Rightarrow}_{[E_Y]}$ corresponding to each of the morphisms φ_i. Each copy can be made exact by contracting length-two paths (dashed) passing through the intermediate vertices $i_1, i_2, \ldots, i_6$. For example, the subgraph containing the vertices $A_1, B_1, C_1, D_1, E_1, F_1, G_1$ can be made isomorphic to $\mathcal{G}^{\Rightarrow}_{[E_Y]}$ by contracting the paths $(A_1, i_4, E_1), (B_1, i_5, D_1)$, and (C_1, i_1, C_1) into single edges $(A_1, E_1), (B_1, D_1)$ and (C_1, C_1).

▶ **Definition 15** (Normal Form). *Let E be a basic RWE. Then E is in* normal form *if it can be written as $x\alpha_1\alpha_2, \ldots \alpha_k y \doteq y\alpha_1^R\alpha_2^R \ldots \alpha_k^R x$ where $x, y \in X$, $\alpha_i \in X^+$ for $1 \leq i \leq k$, and $|\alpha_i| \leq 3$ for $1 \leq i < k$.*

We can obtain an equation in normal form from any basic RWE by applying a polynomial number of rewriting operations.

▶ **Theorem 16.** *Let E be a jumbled basic RWE. Then there exists $\overline{E}$ which is in normal form and such that $E \Rightarrow^{n_1} \overline{E}$ and $\overline{E} \Rightarrow^{n_2} E$ for some $n_1, n_2 \in O(|E|^3)$.*

The idea behind the first normal form is to divide the RWE into pairs (α_i, α_i^R) which are regular-reversed word equations (although solutions to the full equation E are not necessarily solutions to these smaller equations), and for which all but one belong to a finite number of cases (i.e. three cases depending on the length of α_i). Forcing the sub-equations to be regular-reversed gives us the most control when working with the invariant Υ_E. Some intuition behind this fact can be derived from the observation that if we know that a (complete) basic RWE E is regular-reversed, we can uniquely reconstruct it from the leftmost two variables on the LHS and Υ_E. Indeed, any regular-reversed basic RWE E can be written in the form $x_1x_2 \ldots x_n \doteq x_nx_{n-1} \ldots x_1$, meaning that $\Upsilon_E = \{(x_{i-1}, x_{i+1} \mid 2 \leq i \leq n\} \cup \{(x_{n-1}, x_2)\}$, and if we know x_1, then we may infer from Υ_E all the odd-index variables $(x_3, x_5, \ldots)$ and if we know x_2 then we may infer all the even-index variables $(x_4, x_6, \ldots)$.

Rather than looking at the pairs (α_i, α_i^R) in isolation, in order to take full advantage of the invariant Υ_E, we actually need to consider pairs of the form $(\alpha_i\alpha_{i+1} \ldots \alpha_j, \alpha_i^R\alpha_{i+1}^R \ldots \alpha_j^R)$. We shall call such pairs *blocks*, which we define formally below. Our second normal form will be a restriction of the first, and is based on the notion of blocks.

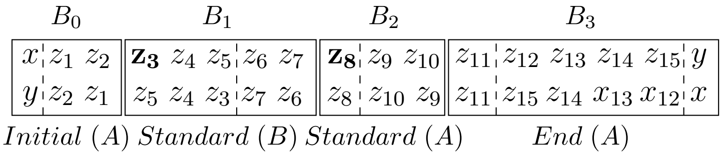

B_0
B_1
B_2
B_3
x z_1 z_2 $\mathbf{z_3}$ z_4 z_5 z_6 z_7 $\mathbf{z_8}$ z_9 z_{10} z_{11} z_{12} z_{13} z_{14} z_{15} y
y z_2 z_1 z_5 z_4 z_3 z_7 z_6 z_8 z_{10} z_9 z_{11} z_{15} z_{14} x_{13} x_{12} x
$Initial\ (A)$
$Standard\ (B)$
$Standard\ (A)$
$End\ (A)$

Figure 3 A depiction of the equation E given by $xz_1z_2z_3z_4z_5z_6z_7z_8z_9z_{10}z_{11}z_{12}z_{13}z_{14}z_{15}y \doteq yz_2z_1z_5z_4z_3z_7z_6z_8z_{10}z_9z_{11}z_{15}z_{14}z_{13}z_{12}x$ where x, y and z_i for $1 \leq i \leq 15$ are variables. The LHS and RHS of the equation are aligned vertically. The block decomposition $\mathfrak{B} = (B_0, B_1, B_2, B_3)$ of E is shown with solid rectangles and with the variety and type of the block written beneath. The additional divisions into the factors α_i, α_i^R required by the definition of normal form are indicated by dashed lines (so that, i.e. $\alpha_1 = z_1z_2$, $\alpha_2 = z_3z_4z_5$, $\alpha_3 = z_6z_7$, $\alpha_4 = z_8, z_5$, $\alpha_5 = z_9z_{10}$, $\alpha_6 = z_{11}$ and $\alpha_7 = z_{12}z_{13}z_{14}z_{15}$). In order for the equation to satisfy the definition of Lex Normal Form, the variables highlighted in bold must be lexicographically minimal with respect to the appropriate sets Γ_i. Note that $\Gamma_1 = \{z_i \mid 3 \leq i \leq 15\} \backslash \{z_4\}$. In particular, Γ_1 consists of the first variable in the block B_1 (x_3) along with (nearly) all variables on the LHS of the equation occurring to the right of z_3, excluding the rightmost variable (y), and since B_1 is Type B, also excluding the second variable in the block B_1 (namely z_4). On the other hand, since B_2 is Type A, in this case we do not need to exclude the second variable in the block B_2, so $\Gamma_2 = \{z_i \mid 8 \leq i \leq 15\}$. Assuming an underlying lexicographic order for which z_{i+1} is greater than z_i, we can conclude that E is in Lex Normal Form.

▶ **Definition 17** (Blocks). *We define 3 variations of blocks which may each have up to two types.*

1. *A* standard block *is a pair* $(\alpha_1\alpha_2 \ldots \alpha_j, \alpha_1^R\alpha_2^R \ldots \alpha_j^R)$ *such that* $j \geq 1$, $\alpha_i \in X^*$ *for* $1 \leq i \leq j$, $|\alpha_1| \in \{1, 3\}$, *and for each* $i, 1 < i \leq j$, $|\alpha_i| = 2$. *It is* Type A *if* $|\alpha_1| = 1$ *and* Type B *if* $|\alpha_1| = 3$.
2. *An* initial block *is a pair* $(x\alpha_1 \ldots \alpha_j, y\alpha_1^R \ldots \alpha_j^R)$ *with* $j \geq 0$, $x, y \in X$ *with* $x \neq y$, *and* $\alpha_i \in (X \backslash \{x, y\})^*$ *for* $1 \leq i \leq j$ *such that* $|\alpha_i| = 2$ *for* $1 \leq i \leq j$. *All initial blocks are* Type A.
3. *An* end block *is a pair* $(\gamma_1\delta y, \gamma_2\delta^R x)$ *where* $x, y \in X$ *with* $x \neq y$, *and* $\gamma_1, \gamma_2, \delta \in (X \backslash \{x, y\})^*$ *with* $|\delta| \geq 1$ *such that* (γ_1, γ_2) *is a block (initial or standard). It is Type A if* (γ_1, γ_2) *is Type A, and Type B otherwise.*

Given an equation which is in normal form, we may decompose it uniquely into blocks in the following manner. The intuition behind this decomposition is that if we fix the invariant property Υ_E, then each block (with the exception of the final block) is determined entirely by the block preceding it and its first (leftmost in the first element) variable. This gives us a crucial degree of control when considering which equations in normal form may appear in $\mathcal{G}^{\Rightarrow}_{[E]}$.

▶ **Definition 18** (Block Decomposition). *Let E be a basic RWE in normal form. Then E may be written as* $x\alpha_1\alpha_2 \ldots \alpha_n y \doteq y\alpha_1^R\alpha_2^R \ldots \alpha_n^R x$ *where* $x, y \in X$, $\alpha_i \in X^+$ *for* $1 \leq i \leq n$, *and* $|\alpha_i| \leq 3$ *for* $1 \leq i < n$. *Let* $I = \{i_1, i_2, \ldots, i_k\} = \{i \mid 1 \leq i < n \text{ and } |\alpha_i| \neq 2\}$ *with* $1 \leq i_1 < i_2 < \ldots < i_k < n$. *If* $I = \emptyset$, *let* $\mathfrak{B} = (E)$. *Otherwise, let* $\mathfrak{B} = (B_0, B_1, \ldots, B_k)$ *where for* $0 \leq j \leq k$, *the* B_j *are blocks such that:*

1. $B_0 = (x\alpha_1 \ldots \alpha_{i_1-1}, y\alpha_1^R \ldots \alpha_{i_1-1}^R)$,
2. $B_k = (\alpha_{i_k} \ldots \alpha_n y, \alpha_{i_k}^R \ldots \alpha_n^R x)$, *and*
3. *for* $1 \leq j < k$, $B_j = (\alpha_{i_j} \ldots \alpha_{i_{j+1}-1}, \alpha_{i_j}^R \ldots \alpha_{i_{j+1}-1}^R)$.

Then $\mathfrak{B}$ is the block decomposition *of E.*

An example illustrating a block decomposition of an equation in normal form is given in Figure 3. Since the blocks are fixed by their first variable, it is natural to ask for which variables we can find an equation in our graph $\mathcal{G}^{\Rightarrow}_{[E]}$ such that the block begins with that variable. In particular, can we find an equation in normal form in $\mathcal{G}^{\Rightarrow}_{[E]}$ for which the first variable of each block is lexicographically minimal when going from left to right? The answer to the question is "nearly". In other words, if we relax the notion slightly to account for some specific cases in which we cannot guarantee minimality, then we can always guarantee the existence of such an equation. This leads to the notion of Lex Normal Form defined below.

▶ **Definition 19** (Lex Normal Form). *Let E be a basic RWE in normal form. Then there exist $x, y \in X$ and $\alpha, \beta \in (X \backslash \{x, y\})^*$ such that E has the form $x\alpha y \doteq y\beta x$. Let $(B_0, B_1, \ldots, B_n)$ be the block decomposition of E. For each i, $0 \leq i \leq n$, let $\gamma_i, \gamma_i' \in X^*$ such that $B_i = (\gamma_i, \gamma_i')$, let $S_i = \{\gamma_i[2], y\}$ whenever B_i is Type B and $S_i = \{y\}$ otherwise, and let $\Gamma_i = \left(\bigcup\limits_{i \leq j \leq n} \text{var}(\gamma_j) \right) \backslash S_i$. A block B_i is* lex-minimal *if $\gamma_i[1]$ is lexicographically minimal in Γ_i. The equation E is in Lex Normal Form (LNF) if, for each i, $0 < i < n$, B_i is lex-minimal.*

Lex Normal Form (see also Fig. 3 for an example) describes the class of equations for which the first variable of each blocks is lexicographically minimal *whenever possible*. We can, in general, guarantee the existence of an equation E' in $\mathcal{G}^{\Rightarrow}_{[E]}$ such that the first variable of each block is lexicographically minimal with the following exceptions. Firstly, we must exclude the first and last blocks (the first block is fixed completely by Υ_E). Secondly, we must only compare the first variable to other variables occurring further right in the LHS of the equation, and excluding the rightmost variable on the LHS of the equation (y in the definition above) and, for blocks of Type B, the second variable in the block. The sets Γ_i in the definition account for these exclusions. It turns out that every vertex in $\mathcal{G}^{\Rightarrow}_{[E]}$ is never more than a polynomial distance away from a vertex corresponding to an equation in LNF.

▶ **Theorem 20.** *Let E be a jumbled basic RWE. Then there exists E' such that E' is in Lex Normal Form, and such that $E \Rightarrow^{n_1} E'$ and $E \ \Rightarrow^{n_2} E$ for some $n_1, n_2 \in O(|E|^4)$.*

4.5 Diameter

It was mentioned in the previous section that the choices for the blocks in a block decomposition of an equation in normal form are restricted by the invariant Υ_E. We shall now make full use of that fact to show that the number of equations in LNF in a single graph $\mathcal{G}^{\Rightarrow}_{[E]}$ is polynomial in $|E|$, and as a consequence that the diameter of $\mathcal{G}^{\Rightarrow}_{[E]}$ is also polynomial. Since each equation in LNF has a unique block decomposition, it is sufficient to count the possible block decompositions for a given value of Υ_E for which the conditions for LNF hold. The restrictions imposed on the blocks by Υ_E are given formally in the following lemmata.

▶ **Lemma 21.** *Let E_1, E_2 be basic RWEs in normal form such that $\Upsilon_{E_1} = \Upsilon_{E_2}$. Let $(B_0, B_1, \ldots, B_k)$ and $(C_0, C_1, \ldots, C_\ell)$ be their respective block decompositions and let $k, \ell > 0$. Then $B_0 = C_0$. Moreover, suppose that $B_i = C_j$, for some $i < k - 1$, $j < \ell - 1$. Let $B_{i+1} = (\gamma_1, \gamma_2)$ and $C_{j+1} = (\delta_1, \delta_2)$ with $\gamma_1, \gamma_2, \delta_1, \delta_2 \in X^*$. If $\gamma_1[1] = \delta_1[1]$, then $B_{i+1} = C_{j+1}$.*

Lemma 21 tells us that the equations in LNF belonging to a single graph $\mathcal{G}^{\Rightarrow}_{[E]}$ are remarkably similar in that they are identical up to the last block of the shorter decomposition.

▶ **Corollary 22.** *Let E_1, E_2 be basic RWEs in LNF such that $\Upsilon_{E_1} = \Upsilon_{E_2}$. Let $(B_0, B_1, \ldots, B_k)$ and $(C_0, C_1, \ldots, C_\ell)$ be their respective block decompositions and suppose that $k, \ell > 0$. Then $B_i = C_i$ for $0 \leq i < \min(k, \ell)$.*

Consequently, two equations in LNF in the graph $\mathcal{G}^{\Rightarrow}_{[E]}$ with block decompositions containing the same number of blocks may differ only in the final block. Clearly, the number of blocks is at most Card(var(E)). Thus, in order to show that there are only polynomially many equations in LNF in $\mathcal{G}^{\Rightarrow}_{[E]}$, it remains to consider the possibilities for the final block.

▶ **Lemma 23.** *Let E_1, E_2 be basic RWEs in normal form such that $\Upsilon_{E_1} = \Upsilon_{E_2}$. Let $(B_0, B_1, \ldots, B_k)$ and $(C_0, C_1, \ldots, C_\ell)$ be their respective block decompositions and suppose that $k, \ell > 0$. Suppose moreover that $B_{k-1} = C_{\ell-1}$. Let $B_k = (\alpha_1\alpha_2\ldots\alpha_n y, \alpha_1^R\alpha_2^R\ldots\alpha_n^R x)$ and $C_\ell = (\beta_1\beta_2\ldots\beta_m y, \beta_1^R\beta_2^R\ldots, \beta_m^R x)$, where $x, y \in X$, $\alpha_1, \alpha_2, \ldots, \alpha_n$, $\beta_1, \beta_2, \ldots, \beta_m \in X^+$, $|\alpha_1| = |\beta_1| \in \{1, 3\}$ and $|\alpha_i|, |\beta_j| = 2$ for $2 \leq i < n$ and $2 \leq j < m$. Then if $\alpha_1[1] = \beta_1[1]$, $n = m$, and $\alpha_n[1] = \beta_m[1]$, we have $B_k = C_\ell$.*

Lemma 23 reveals that the options for last block are dependent only on the choices of three parameters: $\alpha_1[1], \alpha_n[1]$, and n. Since each of these can take at most $|E|$ possible values, there are $|E|^3$ possibilities altogether. Thus for each possible number of blocks, there are at most $|E|^3$ possible block decompositions, and therefore only $|E|^4$ possible block decompositions respecting the invariant Υ_E in total. Since every equation in LNF permits a unique block decomposition, this gives us our desired polynomial bound.

▶ **Theorem 24.** *Let E be a basic RWE. Let S be the set of basic regular equations E' in Lex Normal Form for which $\Upsilon_E = \Upsilon_{E'}$. Then* $\mathrm{Card}(S) \leq |E|^4$.

Since every vertex in $\mathcal{G}^{\Rightarrow}_{[E]}$ is at polynomial distance from a vertex in LNF, and since there are only polynomially many such vertices, it is straightforward to show that the diameter of $\mathcal{G}^{\Rightarrow}_{[E]}$ must also be polynomial: indeed if we have a sufficiently long path between two vertices, then we must have a long path between two vertices which are close to the same vertex in LNF. Since they are close to the same vertex, we can find a shortcut between them, and the initial long path is not minimal. Since the diameter of $\mathcal{G}^{\Rightarrow}_{[E]}$ is polynomial, it follows from Theorem 9 (see also Remark 8) and Proposition 5 that the diameter of $\mathcal{G}^{\Rightarrow NT}_{[E]}$ is polynomial whenever E is regular, even in the case that E is not basic.

▶ **Theorem 25.** *Let E be a basic RWE. Then* $\mathrm{diam}(\mathcal{G}^{\Rightarrow}_{[E]}) \in O(|E|^{10})$. *Consequently, for any RWE E,* $\mathrm{diam}(\mathcal{G}^{\Rightarrow NT}_{[E]}) \in O(|E|^{12})$.

Thus, by Proposition 4, we may infer that the satisfiability problem for RWEs is in NP. It was already shown in [8] that the satisfiability problem for RWEs is NP-hard, and thus we obtain matching upper and lower bounds for its complexity.

▶ **Theorem 26.** *The satisfiability problem for RWEs is NP-complete.*

4.6 Size and DAG-width

While the diameter of $\mathcal{G}^{\Rightarrow}_{[E]}$ is one important parameter, being directly related to the complexity of the satisfiability problem, it is by no means the only interesting one. The overall size of the graphs will also play a central role in the practical performance of the algorithm described in Section 3. For basic RWEs, we have the following tight upper and lower bounds on the number of vertices in the graphs $\mathcal{G}^{\Rightarrow}_{[E]}$.

▶ **Theorem 27.** *Let E be a basic RWE and let $n = \mathrm{Card}(\mathrm{var}(E))$. Suppose that $n > 1$. Let V be the number of vertices in $\mathcal{G}^{\Rightarrow}_{[E]}$. $2^{n-1} - 1 \leq V \leq \frac{n!}{2}$.*

It is worth noting that the lower bound given by Theorem 27 is already exponential in the number of variables. The interpretation of the theorem in the more general (i.e. not basic) setting therefore tells us that the number of vertices in $\mathcal{G}^{\Rightarrow}_{[E]}$ is exponential in the number of variables occurring twice in the appropriate (indecomposable) parts of the LHS and RHS. In other words, we see the rather intuitive fact here that decomposable equations are somehow easier to deal with than indecomposable equations of the same length. The following demonstrates that the bounds given by Theorem 27 are tight.

▶ **Theorem 28.** *Let E be a basic RWE and let $n = \mathrm{Card}(\mathrm{var}(\alpha))$. Suppose that $n > 1$. Let V be the number of vertices in $\mathcal{G}^{\Rightarrow}_{[E]}$. Then:*

1. *$V = 2^{n-1} - 1$ if and only if there exists $E' \in [E]_{\Rightarrow}$ such that E' is regular reversed,*
2. *$V = \frac{n!}{2}$ if and only if there exists $E' \in [E]_{\Rightarrow}$ such that E' is regular rotated.*

In addition to the size we are also able to give some insights about the connectedness of the graphs, which, as discussed in Section 3.3, are of interest when solving RWEs modulo additional constraints. We show firstly that there exist classes of equations E for which $\mathrm{dgw}(\mathcal{G}^{\Rightarrow_{NT}}_{[E]})$ may be arbitrarily large.

▶ **Theorem 29.** *Let $x, y, z_0, z_1, z_2, \ldots, z_n \in X$. Let E be the RWE given by $xz_0z_1z_2 \ldots z_ny \doteq yz_0z_nz_{n-1} \ldots z_1x$. Then $\mathrm{dgw}(\mathcal{G}^{\Rightarrow_{NT}}_{[E]}) > n$.*

Since high connectivity can be seen as an obstacle to deciding the satisfiability problem with additional constraints, it is also worth noting classes for which the DAG-width is bounded by a small constant, such as with those described in the next theorem.

▶ **Theorem 30.** *Let $\alpha_1, \alpha_2, \ldots, \alpha_n, \beta_1, \beta_2, \ldots, \beta_n \in X^*$ such that $\mathrm{var}(\alpha_i) = \mathrm{var}(\beta_i)$ for $1 \leq i \leq n$ and $\mathrm{var}(\alpha_i) \cap \mathrm{var}(\alpha_j) = \emptyset$ for $1 \leq i, j \leq n$ with $i \neq j$. Let E be the RWE $\alpha_1\alpha_2 \ldots \alpha_n \doteq \beta_1\beta_2 \ldots \beta_n$. Then $\mathrm{dgw}(\mathcal{G}^{\Rightarrow_{NT}}_{[E]}) = 2$.*

4.7 Extension to systems of equations

So far, we have considered individual equations. However, it is often the case in practice that there is not just one equation to be solved, but a system of several concurrent equations. However, while constructions exist which transform a system of equations into a single equation (see e.g. [17]), the resulting equation will generally not be quadratic/regular. We extend the definition of regular equations to regular systems as follows.

▶ **Definition 31** (Regular systems). *Let $\Theta = \{\alpha_1 \doteq \beta_1, \alpha_2 \doteq \beta_2, \ldots, \alpha_n \doteq \beta_n\}$ be a system of word equations. An* orientation *of Θ is any element of $\{\alpha_1 \doteq \beta_1, \beta_1 \doteq \alpha_1\} \times \{\alpha_2 \doteq \beta_2, \beta_2 \doteq \alpha_2\} \times \ldots \times \{\alpha_n \doteq \beta_n, \beta_n \doteq \alpha_n\}$. We say that Θ is regular if it has an orientation for which each variable occurs at most once across all LHSs and at most once across all RHSs.*

▶ **Theorem 32.** *The satisfiability problem for regular systems of equations is NP-complete. Moreover, whether a system of word equations is regular can be decided in polynomial time.*

5 Conclusions

A famous algorithm for solving quadratic word equations can be used to produce a (directed) graph containing all solutions to the equation. In the case of regular equations, we have described some underlying structures of these graphs with the intention of better understanding their solution sets. We give bounds on their diameter and number of vertices, as well as provide classes with bounded (resp. unbounded) DAG-width. Probably the most significant result arising from our analysis is that the satisfiability problem for regular word equations is in NP (and thus NP-complete), which we also extend to regular systems of equations.

We leave open many interesting problems, the most obvious of which is to generalise our results to the (full) quadratic case. We also believe that our analysis and techniques open up the possibility to investigate in far more detail the graphs $\mathcal{G}^{\Rightarrow}_{[E]}$, even in the case of regular equations. For example, in light of our results, it seems reasonable to suggest that determining whether $E_1 \Rightarrow^* E_2$ for two regular equations E_1 and E_2 may be done in polynomial time. A particularly nice characterisation of E_1 and E_2 such that $E_1 \Rightarrow^* E_2$ might yield a much quicker algorithm than the one resulting from our bound on the diameter of $\mathcal{G}_{[E]}^{\Rightarrow NT}$ by significantly reducing the degree of the polynomial. We also expect that a detailed analysis of the length-reducing transformations and symmetries which may be found there would be particularly helpful in understanding further the structure of solution sets and the performance of algorithms solving regular equations in practice.

Finally, we mention the task of investigating the decidability of the satisfiability problem for regular equations with additional constraints, in particular length constraints, with the hope that having identified cases where the DAG-width is particularly high/low, along with improved means to describe precisely the structure of the solution-graphs, might provide some useful hints with how to proceed in this direction.

References

1 P. A. Abdulla, M. F. Atig, Y. Chen, L. Holík, A. Rezine, P. Rümmer, and J. Stenman. Norn: An SMT solver for string constraints. In *Proc. Computer Aided Verification (CAV)*, volume 9206 of *Lecture Notes in Computer Science (LNCS)*, pages 462–469, 2015.

2 M. Alkhalaf, T. Bultan, and F. Yu. STRANGER: An automata-based string analysis tool for PHP. In *Proc. Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 6015 of *Lecture Notes in Computer Science (LNCS)*, 2010.

3 D. Angluin. Finding patterns common to a set of strings. *Journal of Computer and System Sciences*, 21:46–62, 1980.

4 C. Barrett, C. L. Conway, M. Deters, L. Hadarean, D. Jovanović, T. King, A. Reynolds, and C. Tinelli. CVC4. In *Proc. Computer Aided Verification (CAV)*, volume 6806 of *Lecture Notes in Computer Science (LNCS)*, pages 171–177, 2011.

5 D. Berwanger, A. Dawar, P. Hunter, S. Kreutzer, and J. Obdrzálek. The DAG-width of directed graphs. *Journal of Combinatorial Theory, Series B*, 102(4):900–923, 2012.

6 M. Berzish, V. Ganesh, and Y. Zheng. Z3str3: A string solver with theory-aware heuristics. In *Proc. Formal Methods in Computer-Aided Design (FMCAD)*, pages 55–59. IEEE, 2017.

7 J. D. Day, V. Ganesh, P.l He, F. Manea, and D. Nowotka. The satisfiability of word equations: Decidable and undecidable theories. In I. Potapov and P. Reynier, editors, *In Proc. 12th International Conference on Reachability Problems, RP 2018*, volume 11123 of *Lecture Notes in Computer Science (LNCS)*, pages 15–29, 2018.

8 J. D. Day, F. Manea, and D. Nowotka. The hardness of solving simple word equations. In *Proc. Mathematical Foundations of Computer Science (MFCS)*, volume 83 of *LIPIcs*, pages 18:1–18:14, 2017.

9 J. D. Day, F. Manea, and D. Nowotka. Upper bounds on the length of minimal solutions to certain quadratic word equations. In *Proc. Mathematical Foundations of Computer Science (MFCS)*, volume 138 of *LIPIcs*, pages 44:1–44:15, 2019.

10 V. Diekert, A. Jeż, and W. Plandowski. Finding all solutions of equations in free groups and monoids with involution. *Information and Computation*, 251:263–286, 2016.

11 V. Diekert and J. M. Robson. On quadratic word equations. In *Proc. 16th Annual Symposium on Theoretical Aspects of Computer Science, STACS*, volume 1563 of *Lecture Notes in Computer Science (LNCS)*, pages 217–226, 1999.

12 A. Ehrenfeucht and G. Rozenberg. Finding a homomorphism between two words is NP-complete. *Information Processing Letters*, 9:86–88, 1979.

13 D. D. Freydenberger. A logic for document spanners. *Theory of Computing Systems*, 63(7):1679–1754, 2019.

14 D. D. Freydenberger and M. Holldack. Document spanners: From expressive power to decision problems. *Theory of Computing Systems*, 62(4):854–898, 2018.

15 A. Jeż. Recompression: A simple and powerful technique for word equations. *Journal of the ACM*, 63, 2016.

16 A. Jeż. Word equations in nondeterministic linear space. In *Proc. International Colloquium on Automata, Languages and Programming (ICALP)*, volume 80 of *LIPIcs*, pages 95:1–95:13, 2017.

17 J. Karhumäki, F. Mignosi, and W. Plandowski. The expressibility of languages and relations by word equations. *Journal of the ACM*, 47:483–505, 2000.

18 A. Kiezun, V. Ganesh, P. J. Guo, P. Hooimeijer, and M. D. Ernst. HAMPI: a solver for string constraints. In *Proc. ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, pages 105–116. ACM, 2009.

19 A. W. Lin and P. Barceló. String solving with word equations and transducers: towards a logic for analysing mutation xss. In *ACM SIGPLAN Notices*, volume 51, pages 123–136. ACM, 2016.

20 A. W. Lin and R. Majumdar. Quadratic word equations with length constraints, counter systems, and Presburger arithmetic with divisibility. In S. K. Lahiri and C. Wang, editors, *In Proc. 16th International Symposium on Automated Technology for Verification and Analysis (ATVA)*, volume 11138 of *Lecture Notes in Computer Science (LNCS)*, pages 352–369. Springer, 2018.

21 M. Lothaire. *Algebraic Combinatorics on Words.* Cambridge University Press, Cambridge, New York, 2002.

22 G. S. Makanin. The problem of solvability of equations in a free semigroup. *Sbornik: Mathematics*, 32(2):129–198, 1977.

23 F. Manea, D. Nowotka, and M. L. Schmid. On the complexity of solving restricted word equations. *International Journal of Foundations of Computer Science*, 29(5):893–909, 2018.

24 E. Petre. An elementary proof for the non-parametrizability of the equation $xyz = zvx$. In *Proc. 29th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 3153 of *Lecture Notes in Computer Science (LNCS)*, pages 807–817, 2004.

25 W. Plandowski. Satisfiability of word equations with constants is in PSPACE. In *Proc. Foundations of Computer Science (FOCS)*, pages 495–500. IEEE, 1999.

26 W. Plandowski and W. Rytter. Application of Lempel-Ziv encodings to the solution of words equations. In *Proc. International Colloquium on Automata, Languages and Programming (ICALP)*, volume 1443 of *Lecture Notes in Computer Science (LNCS)*, pages 731–742, 1998.

27 K. U. Schulz. Makanin's algorithm for word equations-two improvements and a generalization. In *International Workshop on Word Equations and Related Topics*, pages 85–150. Springer, 1990.

From Linear to Additive Cellular Automata

Alberto Dennunzio[1]
Dipartimento di Informatica, Sistemistica e Comunicazione,
Università degli Studi di Milano-Bicocca, Milano, Italy
dennunzio@disco.unimib.it

Enrico Formenti
Université Côte d'Azur, CNRS, I3S, Nice, France
enrico.formenti@univ-cotedazur.fr

Darij Grinberg
Mathematisches Forschungsinstitut Oberwolfach, Oberwolfach-Walke, Germany
darijgrinberg@gmail.com

Luciano Margara
Department of Computer Science and Engineering, University of Bologna, Cesena, Italy
luciano.margara@unibo.it

Abstract

This paper proves the decidability of several important properties of additive cellular automata over finite abelian groups. First of all, we prove that equicontinuity and sensitivity to initial conditions are decidable for a nontrivial subclass of additive cellular automata, namely, the linear cellular automata over $\mathbb{K}^n$, where $\mathbb{K}$ is the ring $\mathbb{Z}/m\mathbb{Z}$. The proof of this last result has required to prove a general result on the powers of matrices over a commutative ring which is of interest in its own.

Then, we extend the decidability result concerning sensitivity and equicontinuity to the whole class of additive cellular automata over a finite abelian group and for such a class we also prove the decidability of topological transitivity and all the properties (as, for instance, ergodicity) that are equivalent to it. Finally, a decidable characterization of injectivity and surjectivity for additive cellular automata over a finite abelian group is provided in terms of injectivity and surjectivity of an associated linear cellular automata over $\mathbb{K}^n$.

2012 ACM Subject Classification Theory of computation → Models of computation

Keywords and phrases Cellular Automata, Decidability, Symbolic Dynamics

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.125

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

1 Introduction

Cellular automata (CA) are widely known formal models for studying and simulating complex systems (for recent results, an up-to date bibliography on CA, and simulations of complex systems, see for instance [18, 1, 8, 9, 22, 5]). They are used in many disciplines ranging from physics to biology, stepping through sociology, ecology and many others. In computer science they are used for designing security schemes, random number generation, image processing, etc. This extensive use is essentially due to three main ingredients: the huge variety of distinct dynamical behaviors; the emergence of complex behaviors from local interactions; the ease of implementation (even at a hardware level). In practical applications one needs to know if the CA used for modelling a system has or not some specific property and this can be an issue. Indeed, Jarkko Kari proved a strong result stating (roughly speaking) that all non-trivial dynamical behaviors are undecidable [27]. From this seminal result, a long sequence followed (see [2, 21, 25], just to cite some of them).

[1] corresponding author

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 125; pp. 125:1–125:13
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The undecidability issue can be tackled by adding more constraints on the model. These constraints may consist of conservation laws over the evolutions [24, 20, 23, 3, 34] or superposition principles induced by imposing a rich algebraic structure over the CA alphabet [26, 31, 30, 28, 7, 6, 19] (in both cases the literature is huge and only a very small excerpt is cited here).

In this paper we follow the latter trend: the alphabet of the CA is a finite abelian group G and its global update map is an additive function, i.e., an endomorphism of $G^{\mathbb{Z}}$. This pretty broad requirement provides a class of CA generalizing those with linear local rule defined by $n \times n$ matrices (see the previous citations for $n = 1$ and [28, 4] for $n > 1$).

Even if the superposition principle still allows us to prove deep and interesting results on the asymptotic behavior of linear CA over $(\mathbb{Z}/m\mathbb{Z})^n$ (for some integers $m, n > 1$), their dynamics is definitely more interesting and expressive than that of linear CA over $\mathbb{Z}/m\mathbb{Z}$ (the classical linear CA setting) and exhibits much more complex features.

In [13, 10], we proved that ergodicity coincides with topological transitivity (and many other properties) for additive CA over finite abelian groups and in [12] we proved the decidability of those properties for the restricted case of linear CA over $(\mathbb{Z}/m\mathbb{Z})^n$.

The present paper adds the following important results to the panorama of the existing ones for additive CA over finite abelian groups:

- a lifting of the decidability of topological transitivity, ergodicity, and all the related properties from linear CA to the general case of additive CA over finite abelian groups;
- a decidable characterization of sensitivity to initial conditions for linear CA over $(\mathbb{Z}/m\mathbb{Z})^n$ which is then lifted to additive CA over finite abelian groups;
- a dichotomy property of sensitivity to initial conditions vs. equicontinuity;
- a characterization of surjectivity and injectivity properties extending the known results for linear CA given in [28, 4].

The above results are important features of the dynamics of additive CA over finite abelian groups which are involved in the most complex CA behaviors. Two main tools were used in the proofs:

- an embedding of an additive CA over a finite abelian group into a linear CA over a commutative ring;
- a deep result about commutative algebras defined over a commutative ring which is of interest in its own;

The paper is structured as follows. The next section introduces all the necessary background and formal definitions. Section 3 recalls the known results about linear CA over $(\mathbb{Z}/m\mathbb{Z})^n$ and proves the new ones, including the non trivial algebra result about powers of matrix over commutative rings. Section 4 explains the embedding allowing to lift results from linear CA over $(\mathbb{Z}/m\mathbb{Z})^n$ to generic additive CA over abelian groups. It also contains all the main results. In the last section we draw our conclusion and provide some perspectives.

2 Background on DTDS and Cellular Automata

We begin by reviewing some general notions about discrete time dynamical systems and cellular automata.

A *discrete time dynamical system* (DTDS) is a pair $(\mathcal{X}, \mathcal{F})$, where $\mathcal{X}$ is any set equipped with a distance function d (i.e., $(\mathcal{X}, d)$ is a metric space) and $\mathcal{F} : \mathcal{X} \to \mathcal{X}$ is a map that is continuous on $\mathcal{X}$ according to the topology induced by d.

Let $(\mathcal{X}, \mathcal{F})$ be a DTDS. We say that it is *surjective*, resp., *injective*, if $\mathcal{F}$ is *surjective*, resp., *injective*. The DTDS $(\mathcal{X}, \mathcal{F})$ is *sensitive to the initial conditions* (or simply *sensitive*) if there exists $\varepsilon > 0$ such that for any $x \in \mathcal{X}$ and any $\delta > 0$ there is an element $y \in \mathcal{X}$ such that $0 < d(y, x) < \delta$ and $d(\mathcal{F}^k(y), \mathcal{F}^k(x)) > \varepsilon$ for some $k \in \mathbb{N}$. The system $(\mathcal{X}, \mathcal{F})$ is said to be *equicontinuous* if $\forall \varepsilon > 0$ there exists $\delta > 0$ such that for all $x, y \in \mathcal{X}$, $d(x, y) < \delta$ implies that $\forall k \in \mathbb{N}$, $d(\mathcal{F}^k(x), \mathcal{F}^k(y)) < \varepsilon$. As dynamical properties, sensitivity and equicontinuity represent the main features of unstable and stable dynamical systems, respectively. The former is the well-known basic component and essence of the chaotic behavior of discrete time dynamical systems, while the latter is a strong form of stability.

The DTDS $(\mathcal{X}, \mathcal{F})$ is *topologically transitive* (or, simply, *transitive*) if for all nonempty open subsets U and V of $\mathcal{X}$ there exists a natural number h such that $\mathcal{F}^h(U) \cap V \neq \emptyset$, while it is said to be *topologically mixing* if for all nonempty open subsets U and V of $\mathcal{X}$ there exists a natural number h_0 such that the previous intersection condition holds for every $h \geq h_0$. Clearly, topological mixing is a stronger condition than transitivity. Moreover, $(\mathcal{X}, \mathcal{F})$ is *topologically weakly mixing* if the DTDS $(\mathcal{X} \times \mathcal{X}, \mathcal{F} \times \mathcal{F})$ is topologically transitive, while it is *totally transitive* if $(\mathcal{X}, \mathcal{F}^h)$ is topologically transitive for all $h \in \mathbb{N}$.

Let $(\mathcal{X}, \mathcal{M}, \mu)$ be a probability space and let $(\mathcal{X}, \mathcal{F})$ be a DTDS where $\mathcal{F}$ is a measurable map which preserves μ, i.e., $\mu(E) = \mu(\mathcal{F}^{-1}(E))$ for every $E \in \mathcal{M}$. The DTDS $(\mathcal{X}, \mathcal{F})$, or, the map $\mathcal{F}$, is *ergodic* with respect to μ if for every $E \in \mathcal{M}$

$$\big(E = \mathcal{F}^{-1}(E)\big) \Rightarrow \mu(E)(1 - \mu(E)) = 0$$

It is well known that $\mathcal{F}$ is ergodic iff for any pair of sets $A, B \in \mathcal{M}$ it holds that

$$\lim_{h \to \infty} \frac{1}{h} \sum_{i=0}^{h-1} \mu(\mathcal{F}^{-i}(A) \cap B) = \mu(A)\mu(B)$$

The DTDS $(\mathcal{X}, \mathcal{F})$ is *(ergodic) mixing*, if for any pair of sets $A, B \in \mathcal{M}$ it holds that

$$\lim_{h \to \infty} \mu(\mathcal{F}^{-h}(A) \cap B) = \mu(A)\mu(B) \ ,$$

while it is *(ergodic) weak mixing*, if for any pair of sets $A, B \in \mathcal{M}$ it holds that

$$\lim_{h \to \infty} \frac{1}{h} \sum_{i=0}^{h-1} |\mu(\mathcal{F}^{-i}(A) \cap B) - \mu(A)\mu(B)| = 0$$

We now recall some general notions about cellular automata.

Let S be a finite set. A configuration over S is a map from $\mathbb{Z}$ to S. We consider the following *space of configurations* $S^{\mathbb{Z}} = \{\boldsymbol{c} | \ \boldsymbol{c} \colon \mathbb{Z} \to S\}$. Each element $\boldsymbol{c} \in S^{\mathbb{Z}}$ can be visualized as an infinite one-dimensional cell lattice in which each cell $i \in \mathbb{Z}$ contains the element $\boldsymbol{c}_i \in S$.

Let $r \in \mathbb{N}$ and $\delta \colon S^{2r+1} \to S$ be any map. We say that r is the radius of δ.

▶ **Definition 1** (Cellular Automaton). *A one-dimensional CA based on a radius r local rule δ is a pair $(S^{\mathbb{Z}}, F)$, where $F \colon S^{\mathbb{Z}} \to S^{\mathbb{Z}}$ is the* global transition map *defined as follows:*

$$\forall \boldsymbol{c} \in S^{\mathbb{Z}}, \forall i \in \mathbb{Z}, \quad F(\boldsymbol{c})_i = \delta\left(\boldsymbol{c}_{i-r}, \ldots, \boldsymbol{c}_{i+r}\right). \tag{1}$$

We stress that the local rule δ completely determines the global rule F of a CA.

In order to study the dynamical properties of one-dimensional CA, we introduce a distance over the space of the configurations. Namely, $S^{\mathbb{Z}}$ is equipped with the Tychonoff distance d defined as follows

$$\forall \boldsymbol{c}, \boldsymbol{c}' \in S^{\mathbb{Z}}, \quad d(\boldsymbol{c}, \boldsymbol{c}') = \begin{cases} 0, & \text{if} \quad \boldsymbol{c} = \boldsymbol{c}', \\ 2^{-\min\{i \in \mathbb{N} \,:\, \boldsymbol{c}_i \neq \boldsymbol{c}'_i \text{ or } \boldsymbol{c}_{-i} \neq \boldsymbol{c}'_{-i}\}} & \text{otherwise} \end{cases} .$$

It is easy to verify that metric topology induced by d coincides with the product topology induced by the discrete topology on $S^{\mathbb{Z}}$. With this topology, $S^{\mathbb{Z}}$ is a compact and totally disconnected space and the global transition map F of any CA $(S^{\mathbb{Z}}, F)$ turns out to be (uniformly) continuous. Therefore, any CA itself is also a discrete time dynamical system. Moreover, any map $F : S^{\mathbb{Z}} \to S^{\mathbb{Z}}$ is the global transition rule of a CA if and only if F is (uniformly) continuous and $F \circ \sigma = \sigma \circ F$, where $\sigma : S^{\mathbb{Z}} \to S^{\mathbb{Z}}$ is the *shift map* defined as $\forall \boldsymbol{c} \in S^{\mathbb{Z}}$, $\forall i \in \mathbb{Z}$, $\sigma(\boldsymbol{c})_i = \boldsymbol{c}_{i+1}$. From now, when no misunderstanding is possible, we identify a CA with its global rule. Moreover, whenever an ergodic property is considered for CA, μ is the well-known Haar measure over the collection $\mathcal{M}$ of measurable subsets of $S^{\mathbb{Z}}$, i.e., the one defined as the product measure induced by the uniform probability distribution over S.

2.1 Additive and Linear Cellular Automata

Let us introduce the background of additive CA. The alphabet S will be a finite abelian group G, with group operation $+$, neutral element 0, and inverse operation $-$. In this way, the configuration space $G^{\mathbb{Z}}$ turns out to be a finite abelian group, too, where the group operation of $G^{\mathbb{Z}}$ is the componentwise extension of $+$ to $G^{\mathbb{Z}}$. With an abuse of notation, we denote by the same symbols $+$, 0, and $-$ the group operation, the neutral element, and the inverse operation, respectively, both of G and $G^{\mathbb{Z}}$. Observe that $+$ and $-$ are continuous functions in the topology induced by the metric d. A configuration $\boldsymbol{c} \in G^{\mathbb{Z}}$ is said to be *finite* if the number of positions $i \in \mathbb{Z}$ with $\boldsymbol{c}_i \neq 0$ is finite.

▶ **Definition 2** (Additive Cellular Automata). *An* additive CA *over a abelian finite group G is a CA $(G^{\mathbb{Z}}, F)$ where the global transition map $F : G^{\mathbb{Z}} \to G^{\mathbb{Z}}$ is an endomorphism of $G^{\mathbb{Z}}$.*

The *sum of two additive CA* F_1 and F_2 over G is naturally defined as the map on $G^{\mathbb{Z}}$ denoted by $F_1 + F_2$ and such that

$$\forall \boldsymbol{c} \in G^{\mathbb{Z}}, \quad (F_1 + F_2)(\boldsymbol{c}) = F_1(\boldsymbol{c}) + F_2(\boldsymbol{c})$$

Clearly, $F_1 + F_2$ is an additive CA over G.

We now recall the notion of linear CA, an important subclass of additive CA. We stress that, whenever the term *linear* is involved, the alphabet S is $\mathbb{K}^n$, where $\mathbb{K} = \mathbb{Z}/m\mathbb{Z}$ for some positive integer m. Both $\mathbb{K}^n$ and $(\mathbb{K}^n)^{\mathbb{Z}}$ become $\mathbb{K}$-modules in the obvious (i.e., entrywise) way.

A local rule $\delta \colon (\mathbb{K}^n)^{2r+1} \to \mathbb{K}^n$ of radius r is said to be linear if it is defined by $2r + 1$ matrices $A_{-r}, \ldots, A_0, \ldots, A_r \in \mathbb{K}^{n \times n}$ as follows:

$$\forall (x_{-r}, \ldots, x_0, \ldots, x_r) \in (\mathbb{K}^n)^{2r+1}, \quad \delta(x_{-r}, \ldots, x_0, \ldots, x_r) = \sum_{i=-r}^{r} A_i \cdot x_i \ .$$

▶ **Definition 3** (Linear Cellular Automata (LCA)). *A* linear CA (LCA) *over $\mathbb{K}^n$ is a CA based on a linear local rule.*

Let $\mathbb{K}^n[X, X^{-1}]$ and $\mathbb{K}^n[[X, X^{-1}]]$ denote the set of *Laurent polynomials* and the set of *Laurent series*, respectively, with coefficients in $\mathbb{K}^n$. Before proceeding, let us recall that such formalisms have been successfully used to study the dynamical behaviour of LCA in the case $n = 1$ [26, 31]. Indeed, global rules and configurations are represented by Laurent polynomials and Laurent series, respectively, and the application of a global rule turns into a polynomial-series multiplication. In the more general case of LCA over $\mathbb{K}^n$, a configuration $\boldsymbol{c} \in (\mathbb{K}^n)^{\mathbb{Z}}$ can be associated with the Laurent series

$$\boldsymbol{P_c}(X) = \sum_{i\in\mathbb{Z}} \boldsymbol{c}_i X^i = \begin{bmatrix} c^1(X) \\ \vdots \\ c^n(X) \end{bmatrix} = \begin{bmatrix} \sum_{i\in\mathbb{Z}} c^1_i X^i \\ \vdots \\ \sum_{i\in\mathbb{Z}} c^n_i X^i \end{bmatrix} \in \left(\mathbb{K}[[X, X^{-1}]]\right)^n \cong \mathbb{K}^n[[X, X^{-1}]] \ .$$

Then, if F is the global rule of a LCA defined by $A_{-r}, \ldots, A_0, \ldots, A_r$, one finds

$$\boldsymbol{P}_{F(\boldsymbol{c})}(X) = A \cdot \boldsymbol{P_c}(X)$$

where

$$A = \sum_{i=-r}^{r} A_i X^{-i} \in \mathbb{K}[X, X^{-1}]^{n\times n}$$

is the *the matrix associated with the LCA F*. In this way, for any integer $k > 0$ the matrix associated with F^k is A^k, and then $\boldsymbol{P}_{F^k(\boldsymbol{c})}(X) = A^k \cdot \boldsymbol{P_c}(X)$.

A matrix $A \in \mathbb{K}[X, X^{-1}]^{n\times n}$ is in *Frobenius normal form* if

$$A = \begin{bmatrix} 0 & 1 & 0 & \ldots & 0 & 0 \\ 0 & 0 & 1 & \ddots & 0 & 0 \\ 0 & 0 & 0 & \ddots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & 0 & 1 \\ \mathbb{a}_0 & \mathbb{a}_1 & \mathbb{a}_2 & \ldots & \mathbb{a}_{n-2} & \mathbb{a}_{n-1} \end{bmatrix} \tag{2}$$

where each $\mathbb{a}_i \in \mathbb{K}[X, X^{-1}]$. Recall that the coefficients of the characteristic polynomial of A are just the elements $\mathbb{a}_i$ of the n-th row of A (up to sign).

▶ **Definition 4** (Frobenius LCA). *A LCA $((\mathbb{K}^n)^{\mathbb{Z}}, F)$ is said to be a* Frobenius LCA *if the matrix $A \in \mathbb{K}[X, X^{-1}]^{n\times n}$ associated with F is in Frobenius normal form.*

3 Decidability Results about Linear CA

We now deal with sensitivity and equicontinuity for LCA over $\mathbb{K}^n$. First of all, we remind that a dichotomy between sensitivity and equicontinuity holds for LCA. Moreover, these properties are characterized by the behavior of the powers of the matrix associated with a LCA.

▶ **Proposition 5** ([14]). *Let $\left((\mathbb{K}^n)^{\mathbb{Z}}, F\right)$ be a LCA over $\mathbb{K}^n$ and let A be the matrix associated with F. The following statements are equivalent:*

1. *F is sensitive to the initial conditions;*
2. *F is not equicontinuous;*
3. $\left|\{A^1, A^2, A^3, \ldots\}\right| = \infty$.

An immediate consequence of Proposition 5 is that any decidable characterization of sensitivity to the initial conditions in terms of the matrices defining LCA over $\mathbb{K}^n$ would also provide a characterization of equicontinuity. In the sequel, we are going to show that such a characterization actually exists. First of all, we remind that a decidable characterization of sensitivity and equicontinuity was provided for the class of Frobenius LCA in [14]. In particular, the following result holds.

▶ **Theorem 6** (Theorem 31 in [14]). *Sensitivity and equicontinuity are decidable for Frobenius LCA over $\mathbb{K}^n$.*

In order to prove that equicontinuity and sensitivity are decidable for the whole class of LCA over $\mathbb{K}^n$, we need to prove the following result whose proof is strongly far from trivial and, for a lack of space, is omitted (the proof can be found in [11]).

Notation. Let $\mathbb{K}$ be a commutative ring. Let $n \in \mathbb{N}$. Let A be an $n \times n$-matrix over $\mathbb{K}$. We denote by χ_A the *characteristic polynomial* of A which is as usual defined as the polynomial $\det(tI_n - A) \in \mathbb{K}[t]$, where I_n stands for the $n \times n$ identity matrix and $tI_n - A$ is considered as an $n \times n$-matrix over the polynomial ring $\mathbb{K}[t]$.

▶ **Theorem 7.** *Let $\mathbb{K}$ be a finite commutative ring. Let $\mathbb{L}$ be a commutative $\mathbb{K}$-algebra. Let $n \in \mathbb{N}$. Let A and B be two $n \times n$-matrices over $\mathbb{L}$ such that $\chi_A = \chi_B$. Then, the set $\{A^0, A^1, A^2, \ldots\}$ is finite if and only if the set $\{B^0, B^1, B^2, \ldots\}$ is finite.*

We are now able to prove the following

▶ **Theorem 8.** *Sensitivity and equicontinuity are decidable for LCA over $\mathbb{K}^n$.*

Proof. Let $\left((\mathbb{K}^n)^{\mathbb{Z}}, G\right)$ be any LCA over $\mathbb{K}^n$ and let A be the matrix associated with G. Consider the Frobenius LCA $\left((\mathbb{K}^n)^{\mathbb{Z}}, F\right)$ such that $\chi_A = \chi_B$, where B is the matrix (in Frobenius normal form) associated with F. By Theorem 7 and Proposition 5 the former LCA is equicontinuous if and only if the latter is. Theorem 6 concludes the proof. ◀

For a sake of completeness, we recall that injectivity and surjectivity are decidable for LCA over $\mathbb{K}^n$. This result follows from a characterization of these properties in terms of the determinant of the matrix associated with a LCA and from the fact that injectivity and surjectivity are decidable for LCA over $\mathbb{K}$ (for the latter, see [26]).

▶ **Theorem 9** ([4, 28]). *Injectivity and surjectivity are decidable for LCA over $\mathbb{K}^n$. In particular, a LCA over $\mathbb{K}^n$ is injective (resp., surjective) if and only if the determinant of the matrix associated with it is the Laurent series associated with an injective (resp., surjective) LCA over $\mathbb{K}$.*

The decidability of topologically transitivity, ergodicity, and other mixing and ergodic properties for LCA over $\mathbb{K}^n$ has been recently proved in [13]. In particular, authors showed the equivalence of all the mixing and ergodic properties for additive CA over a finite abelian group and the decidability for LCA over $\mathbb{K}^n$ (see also [10]).

▶ **Theorem 10** ([13, 10]). *Let F be any additive CA over a finite abelian group. The following statements are equivalent: (1) F is topologically transitive; (2) F is ergodic; (3) F is surjective and for every $k \in \mathbb{N}$ it holds that $F^k - I$ is surjective; (4) F is topologically mixing; (5) F is weak topologically transitive; (6) F is totally transitive; (7) F is weakly ergodic mixing; (8) F is ergodic mixing. Moreover, all the previously mentioned properties are decidable for LCA over $\mathbb{K}^n$.*

4 From Linear to Additive CA

In this section we are going to prove that sensitivity, equicontinuity, topological transitivity, and all the properties equivalent to the latter are decidable also for additive CA over a finite abelian group. For each of them we will reach the decidability result by extending the analogous one obtained for LCA to the wide class of additive CA over a finite abelian group. In a similar way, we provide a decidable characterization of injectivity and surjectivity for additive CA over a finite abelian group.

We recall that the local rule $\delta : G^{2r+1} \to G$ of an additive CA of radius r over a finite abelian group G can be written as

$$\forall (x_{-r}, \dots, x_r) \in G^{2r+1}, \qquad \delta(x_{-r}, \dots, x_r) = \sum_{i=-r}^{r} \delta_i(x_i) \tag{3}$$

where the functions δ_i are endomorphisms of G.

The fundamental theorem of finite abelian groups states that every finite abelian group G is isomorphic to $\bigoplus_{i=1}^{h} \mathbb{Z}/k_i\mathbb{Z}$ where the numbers $k_1, \dots, k_h$ are powers of (not necessarily distinct) primes and $\oplus$ is the direct sum operation. Hence, the global rule F of an additive CA over G splits into the direct sum of a suitable number h' of additive CA over subgroups $G_1, \dots, G_{h'}$ with $h' \le h$ and such that $\gcd(|G_i|, |G_j|) = 1$ for each pair of distinct $i, j \in \{1, \dots, h'\}$. Each of them can be studied separately and then the analysis of the dynamical behavior of F can be carried out by combining together the results obtained for each component.

In order to make things clearer, consider the following example. If F is an additive CA over $G \cong \mathbb{Z}/4\mathbb{Z} \times \mathbb{Z}/8\mathbb{Z} \times \mathbb{Z}/3\mathbb{Z} \times \mathbb{Z}/3\mathbb{Z} \times \mathbb{Z}/25\mathbb{Z}$ then F splits into the direct sum of 3 additive CA F_1, F_2, and F_3 over $\mathbb{Z}/4\mathbb{Z} \times \mathbb{Z}/8\mathbb{Z}$, $\mathbb{Z}/3\mathbb{Z} \times \mathbb{Z}/3\mathbb{Z}$ and $\mathbb{Z}/25\mathbb{Z}$, respectively. Therefore, F will be sensitive to initial conditions iff at least one F_i is sensitive to the initial conditions, while F will be topological transitive iff every F_i is topological transitive.

The above considerations lead us to three distinct scenarios:

1) $\boldsymbol{G \cong \mathbb{Z}/p^k\mathbb{Z}}$**.** Then, G is cyclic and we can define each δ_i simply assigning the value of δ_i applied to the unique generator of G. Moreover, every pair δ_i, δ_j commutes, i.e., $\delta_i \circ \delta_j = \delta_j \circ \delta_i$, and this makes it possible a detailed analysis of the global behavior of F. Indeed, additive cellular automata over $\mathbb{Z}/p^k\mathbb{Z}$ are nothing but LCA over $\mathbb{Z}/p^k\mathbb{Z}$ and almost all dynamical properties, including sensitivity to the initial conditions, equicontinutity, injectivity, surjectivity, topological transitivity and so on are well understood and characterized (see [31]).

2) $\boldsymbol{G \cong (\mathbb{Z}/p^k\mathbb{Z})^n}$**.** In this case, G is not cyclic anymore and has n generators. We can define each δ_i assigning the value of δ_i for each generator of G. This gives rise to the class of linear CA over $(\mathbb{Z}/p^k\mathbb{Z})^n$. Now, δ_i and δ_j do not commute in general and this makes the analysis of the dynamical behavior much harder. Nevertheless, in Section 3 we have proved that sensitivity and equicontinuity are decidable by exploiting Theorem 7. As pointed out in [14], we also recall that linear CA over $(\mathbb{Z}/p^k\mathbb{Z})^n$ allow the investigation of some classes of non-uniform CA over $\mathbb{Z}/p^k\mathbb{Z}$ ((for these latter see [15, 16, 17]).

3) $\boldsymbol{G \cong \bigoplus_{i=1}^{n} \mathbb{Z}/p^{k_i}\mathbb{Z}}$**.** In this case ($\mathbb{Z}/4\mathbb{Z} \times \mathbb{Z}/8\mathbb{Z}$ in the example), G is again not cyclic and F turns out to be a subsystem of a suitable LCA. Then, the analysis of the dynamical behavior of F is even more complex than in **2)**. We do not even know easy checkable characterizations of basic properties like surjectivity or injectivity so far. We will provide them in the sequel as we stated at the beginning of this section.

Therefore, without loss of generality, in the sequel we can assume that $G = \mathbb{Z}/p^{k_1}\mathbb{Z} \times \ldots \times \mathbb{Z}/p^{k_n}\mathbb{Z}$ with $k_1 \ge k_2 \ge \ldots \ge k_n$ in order to reach our goal.

For any $i \in \{1, \dots, n\}$ let us denote by $\boldsymbol{e}^{(i)} \in G^{\mathbb{Z}}$ the bi-infinite configuration such that $\boldsymbol{e}^{(i)}_0 = e_i$ and $\boldsymbol{e}^{(i)}_j = 0$ for every integer $j \neq 0$.

▶ **Definition 11.** *Let $(G^{\mathbb{Z}}, F)$ be an additive CA over G. We say that $\boldsymbol{e}^{(i)} \in G^{\mathbb{Z}}$* spreads under F *if for every $\ell \in \mathbb{N}$ there exists $k \in \mathbb{N}$ such that $F^k(\boldsymbol{e}^{(i)})_j \neq 0$ for some integer j with $|j| > \ell$.*

▶ Remark 12. Whenever we consider $\boldsymbol{P}_{\boldsymbol{e}^{(i)}}(X) \in G[X, X^{-1}]$, we will say that $\boldsymbol{P}_{\boldsymbol{e}^{(i)}}(X)$ spreads under F if for every $\ell \in \mathbb{N}$ there exists $k \in \mathbb{N}$ such that $\boldsymbol{P}_{F^k(\boldsymbol{e}^{(i)})}(X)$ has at least one component with a non null monomial of degree which is greater than ℓ in absolute value. Clearly, $\boldsymbol{P}_{\boldsymbol{e}^{(i)}}(X)$ spreads under F if and only if $\boldsymbol{e}^{(i)}$ spreads under F.

Let $\hat{G} = (\mathbb{Z}/p^{k_1}\mathbb{Z})^n$. Define the map $\psi : G \to \hat{G}$ as follows

$$\forall h \in G, \quad \forall i = 1, \dots, n, \quad \psi(h)^i = h^i \, p^{k_1 - k_i} \ ,$$

where, for a sake of clarity, we stress that h^i denotes the i-th component of h, while $p^{k_1-k_i}$ is just the $(k_1 - k_i)$-th power of p.

▶ **Definition 13.** *We define the function $\Psi : G^{\mathbb{Z}} \to \hat{G}^{\mathbb{Z}}$ as the componentwise extension of ψ, i.e.,*

$$\forall \boldsymbol{c} \in G^{\mathbb{Z}}, \quad \forall j \in \mathbb{Z}, \quad \Psi(\boldsymbol{c})_j = \psi(\boldsymbol{c}_j) \ .$$

It is easy to check that Ψ is continuous and injective. Since every configuration $\boldsymbol{c} \in G^{\mathbb{Z}}$ (or $\hat{G}^{\mathbb{Z}}$) is associated with the Laurent series $\boldsymbol{P}_{\boldsymbol{c}}(X) \in G[[X, X^{-1}]]$ (or $\hat{G}[[X, X^{-1}]]$), with an abuse of notation we will sometimes consider Ψ as map from $G[[X, X^{-1}]]$ to $\hat{G}[[X, X^{-1}]]$ with the obvious meaning.

For any additive CA over G, we are now going to define a LCA over $(\mathbb{Z}/p^{k_1}\mathbb{Z})^n$ associated to it. With a further abuse of notation, in the sequel we will write p^{-m} with $m \in \mathbb{N}$ even if this quantity might not exist in $\mathbb{Z}/p^k\mathbb{Z}$. However, we will use it only when it multiplies $p^{m'}$ for some integer $m' > m$. In such a way $p^{m'-m}$ is well-defined in $\mathbb{Z}/p^k\mathbb{Z}$ and we will note it as product $p^{-m} \cdot p^{m'}$.

▶ **Definition 14.** *Let $(G^{\mathbb{Z}}, F)$ be any additive CA and let $\delta : G^{2r+1} \to G$ be its local rule defined, according to* (3)*, by $2r+1$ endomorphisms $\delta_{-r}, \dots, \delta_r$ of G . For each $z \in \{-r, \dots, r\}$, we define the matrix $A_z = (a^{(z)}_{i,j})_{1 \le i \le n, 1 \le j \le n} \in (\mathbb{Z}/p^{k_1}\mathbb{Z})^{n \times n}$ as*

$$\forall i, j \in \{1, \dots, n\}, \qquad a^{(z)}_{i,j} = p^{k_j - k_i} \cdot \delta_z(e_j)^i$$

The LCA *associated with the additive* CA *$(G^{\mathbb{Z}}, F)$ is $(\hat{G}^{\mathbb{Z}}, L)$, where L is defined by $A_{-r}, \dots, A_r$ or, equivalently, by $A = \sum_{z=-r}^{r} A_z X^{-z} \in \hat{G}[X, X^{-1}]^{n \times n}$.*

▶ Remark 15. Since every δ_z is an endomorphism of G, by construction A turns out to be well-defined.

▶ Remark 16. The following diagram commutes

$$\begin{tikzcd}
G^{\mathbb{Z}} \arrow[r, "F"] \arrow[d, "\Psi"'] & G^{\mathbb{Z}} \arrow[d, "\Psi"] \\
\hat{G}^{\mathbb{Z}} \arrow[r, "L"'] & \hat{G}^{\mathbb{Z}}
\end{tikzcd} ,$$

i.e., $L \circ \Psi = \Psi \circ F$. Therefore we say that $(\hat{G}^{\mathbb{Z}}, L)$ is the LCA associated with $(G^{\mathbb{Z}}, F)$ *via the embedding* Ψ.

4.1 Sensitivity and Equicontinuity for Additive Cellular Automata

Let us start with the decidability of sensitivity and equicontinuity.

▶ **Lemma 17.** *Let $(G^{\mathbb{Z}}, F)$ be any additive CA. If for some $i \in \{1, \ldots, n\}$ the configuration $\boldsymbol{e}^{(i)} \in G^{\mathbb{Z}}$ spreads under F then $(G^{\mathbb{Z}}, F)$ is sensitive to the initial conditions.*

Proof. We prove that F is sensitive with constant $\varepsilon = 1$. Let $\boldsymbol{e}^{(i)} \in G^{\mathbb{Z}}$ be the configuration spreading under F. Choose arbitrarily an integer $\ell \in \mathbb{N}$ and a configuration $\boldsymbol{c} \in G^{\mathbb{Z}}$. Let $t \in \mathbb{N}$ and $j \notin \{-\ell, \ldots, \ell\}$ be the integers such that $F^t(\boldsymbol{e}^{(i)})_j \neq 0$. Consider the configuration $\boldsymbol{c}' = \boldsymbol{c} + \sigma^j(\boldsymbol{e}^{(i)})$. Clearly, it holds that $d(\boldsymbol{c}, \boldsymbol{c}') < 2^{-\ell}$ and $F^t(\boldsymbol{c}') = F^t(\boldsymbol{c}) + F^t(\sigma^j(\boldsymbol{e}^{(i)})) = F^t(\boldsymbol{c}) + \sigma^j(F^t(\boldsymbol{e}^{(i)}))$. So, we get $d(F^t(\boldsymbol{c}'), F^t(\boldsymbol{c})) = 1$ and this concludes the proof. ◀

In order to prove the decidability of sensitivity, we need to deal with the following notions about Laurent polynomials.

▶ **Definition 18.** *Given any polynomial $\mathbb{p}(X) \in \mathbb{Z}/p^{k_1}\mathbb{Z}\left[X, X^{-1}\right]$, the* positive *(resp.,* negative*)* degree *of $\mathbb{p}(X)$, denoted by $deg^+[\mathbb{p}(X)]$ (resp., $deg^-[\mathbb{p}(X)]$) is the maximum (resp., minimum) degree among those of the monomials having both positive (resp., negative) degree and coefficient which is not multiple of p. If there is no monomial satisfying both the required conditions, then $deg^+[\mathbb{p}(X)] = 0$ (resp., $deg^-[\mathbb{p}(X)]$=0).*

▶ **Lemma 19.** *Let $(\hat{G}^{\mathbb{Z}}, L)$ be a LCA and let $A \in \mathbb{Z}/p^{k_1}\mathbb{Z}\left[X, X^{-1}\right]^{n \times n}$ be the matrix associated to it. If $(\hat{G}^{\mathbb{Z}}, L)$ is sensitive then for every integer $m \geq 1$ there exists an integer $k \geq 1$ such that at least one entry of A^k has either positive or negative degree with absolute value which is greater than m.*

Proof. We can write $A = B + p \cdot C$ for some $B, C \in \mathbb{Z}/p^{k_1}\mathbb{Z}\left[X, X^{-1}\right]^{n \times n}$, where the monomials of all entries of B have coefficient which is not multiple of p. Assume that there exists a bound $b \geq 1$ such that for every $k \geq 1$ all entries of A^k have degree less than b in absolute value. Therefore, it holds that $\left|\{A^k, k \geq 1\}\right| < \infty$ and so, by Proposition 5, $(\hat{G}^{\mathbb{Z}}, L)$ is not sensitive. ◀

We are now able to prove the following important result.

▶ **Theorem 20.** *Let $(G^{\mathbb{Z}}, F)$ be any additive CA over G and let $(\hat{G}^{\mathbb{Z}}, L)$ be the LCA associated to it via the embedding Ψ. Then, the CA $(G^{\mathbb{Z}}, F)$ is sensitive to the initial conditions if and only if $(\hat{G}^{\mathbb{Z}}, L)$ is. Moreover, the CA $(G^{\mathbb{Z}}, F)$ is equicontinuous if and only if $(\hat{G}^{\mathbb{Z}}, L)$ is.*

Proof. Let us start with the equivalence between sensitivity of $(G^{\mathbb{Z}}, F)$ and sensitivity of $(\hat{G}^{\mathbb{Z}}, L)$.

$\Longrightarrow$: Assume that $(\hat{G}^{\mathbb{Z}}, L)$ is not sensitive. Then, by Proposition 5, there exist two integers $k \in \mathbb{N}$ and $m > 0$ such that $L^{k+m} = L^k$. Therefore, we get $\Psi \circ F^{k+m} = L^{k+m} \circ \Psi = L^k \circ \Psi = \Psi \circ F^k$. Since Ψ is injective, it holds that $F^{k+m} = F^k$ and so $(G^{\mathbb{Z}}, F)$ is not sensitive.

$\Longleftarrow$: Assume that $(\hat{G}^{\mathbb{Z}}, L)$ is sensitive and for any natural k let $A^k = (a^{(k)}_{i,j})_{1 \leq i \leq n, 1 \leq j \leq n}$ be the k-th power of $A \in \mathbb{Z}/p^{k_1}\mathbb{Z}\left[X, X^{-1}\right]^{n \times n}$, where A is the matrix associated to $(\hat{G}^{\mathbb{Z}}, L)$. We are going to show that at least one configuration among $\boldsymbol{e}^{(1)}, \ldots, \boldsymbol{e}^{(n)}$ spreads under F. Choose arbitrarily $\ell \in \mathbb{N}$. By Lemma 19, there exist an integer $m \geq 1$ and one entry (i, j) such that either $deg^-[a^{(m)}_{i,j}] < -\ell$ or $deg^+[a^{(m)}_{i,j}] > \ell$. Without loss of generality suppose that $deg^+[a^{(m)}_{i,j}] > \ell$. The i–th component of $\boldsymbol{P}_{F^m(\boldsymbol{e}^{(j)})}(X)$ is the well defined polynomial $p^{k_i - k_1} \cdot p^{k_1 - k_j} \cdot a^{(m)}_{i,j}$. Since $deg^+[a^{(m)}_{i,j}] > \ell$, we can state that $\boldsymbol{e}^{(j)}$ spreads under F. By Lemma 17, it follows that $(G^{\mathbb{Z}}, F)$ is sensitive.

As to the equicontinuity equivalence, the above first implication also proves that if $(\hat{G}^{\mathbb{Z}}, L)$ is equicontinuous (i.e., by Proposition 5, it is not sensitive) then $F^{k+m} = F^k$, i.e., by [29], $(G^{\mathbb{Z}}, F)$ is equicontinuous. Conversely, if $(G^{\mathbb{Z}}, F)$ is equicontinuous then it trivially follows that it is not sensitive, i.e., by the above second implication, $(\hat{G}^{\mathbb{Z}}, L)$ is not sensitive, i.e., by Proposition 5, $(\hat{G}^{\mathbb{Z}}, L)$ is equicontinuous. ◀

As immediate consequence of Theorem 20 we can state that the dichotomy between sensitivity and equicontinuity also holds for additive CA.

▶ **Corollary 21.** *Any additive CA over a finite abelian group is sensitive to the initial conditions if and only if it is not equicontinuous.*

The following decidability result follows from Theorem 20 and the decidability of sensitivity for LCA.

▶ **Corollary 22.** *Equicontinuity and sensitivity to the initial conditions are decidable for additive CA over a finite abelian group.*

Proof. Use Theorem 8 and 20. ◀

4.2 Surjectivity and Injectivity for Additive Cellular Automata

We now study injectivity and surjectivity for additive CA.

▶ **Lemma 23.** *Let $(\hat{G}^{\mathbb{Z}}, L)$ be any LCA over $\hat{G}$. If there exists a configuration $\boldsymbol{b} \in \hat{G}^{\mathbb{Z}}$ with $\boldsymbol{b} \neq 0$ and $L(\boldsymbol{b}) = 0$, then there exists a configuration $\boldsymbol{b}' \in \Psi(G^{\mathbb{Z}})$ such that $\boldsymbol{b}' \neq 0$ and $L(\boldsymbol{b}') = 0$. In particular, if $\boldsymbol{b}$ is finite then $\boldsymbol{b}'$ is finite too.*

Proof. Let $\boldsymbol{b} \in \hat{G}^{\mathbb{Z}}$ any configuration with $\boldsymbol{b} \neq 0$ and $L(\boldsymbol{b}) = 0$. Set $\boldsymbol{b}^{(1)} = p \cdot \boldsymbol{b}$. If $\boldsymbol{b}^{(1)} = 0$ then for every $i \in \mathbb{Z}$ each component of $\boldsymbol{b}_i$ has $p^{k_1 - 1}$ as factor. So, $\boldsymbol{b} \in \Psi(G^{\mathbb{Z}})$ and $\boldsymbol{b}' = \boldsymbol{b}$ is just one possible configuration the thesis requires to exhibit. Otherwise, by repeating the same argument, set $\boldsymbol{b}^{(2)} = p \cdot \boldsymbol{b}^{(1)}$. If $\boldsymbol{b}^{(2)} = 0$ then, for every $i \in \mathbb{Z}$, each component of $\boldsymbol{b}_i^{(1)}$ has p^{k_1-1} as factor and so $\boldsymbol{b}^{(1)} \in \Psi(G^{\mathbb{Z}})$. Since $L(\boldsymbol{b}^{(1)}) = 0$, a configuration we are looking for is $\boldsymbol{b}' = \boldsymbol{b}^{(1)}$. After $k_1 - 1$ iterations, i.e., once we get $\boldsymbol{b}^{(k_1-1)} = p \cdot \boldsymbol{b}^{(k-2)}$ (with $\boldsymbol{b}^{(k-2)} \neq 0$), if $\boldsymbol{b}^{(k_1-1)} = 0$ holds we conclude that $\boldsymbol{b}' = \boldsymbol{b}^{(k_1-2)}$ by using the same argument of the previous steps. Otherwise, by definition, for every $i \in \mathbb{Z}$ each component of $\boldsymbol{b}_i^{(k_1-1)}$ itself certainly contains p^{k_1-1} as factor. Therefore, $\boldsymbol{b}^{(k_1-1)} \in \Psi(G^{\mathbb{Z}})$. Moreover, $L(\boldsymbol{b}^{(k_1-1)}) = 0$. Hence, we can set $\boldsymbol{b}' = \boldsymbol{b}^{(k_1-1)}$ and this concludes the proof. ◀

The following lemma will be useful for studying both surjectivity and other properties.

▶ **Lemma 24.** *Let $(G^{\mathbb{Z}}, F)$ and $(\hat{G}^{\mathbb{Z}}, L)$ be any additive CA over G and any LCA over $\hat{G}$, respectively, such that $L \circ \Psi = \Psi \circ F$. Then, the CA $(G^{\mathbb{Z}}, F)$ is surjective if and only if $(\hat{G}^{\mathbb{Z}}, L)$ is.*

Proof. $\Longleftarrow$: Assume that F is not surjective. Then, by the Garden of Eden theorem [32, 33], F is not injective on the finite configurations, i.e., there exist two distinct and finite configurations $\boldsymbol{c}', \boldsymbol{c}'' \in G^{\mathbb{Z}}$ with $F(\boldsymbol{c}') = F(\boldsymbol{c}'')$. Therefore, the element $\boldsymbol{c} = \boldsymbol{c}' - \boldsymbol{c}'' \in G^{\mathbb{Z}}$ is a finite configuration such that $\boldsymbol{c} \neq 0$ and $F(\boldsymbol{c}) = 0$. So, we get both $\Psi(\boldsymbol{c}) \neq 0$ and $L(\Psi(\boldsymbol{c})) = \Psi(F(\boldsymbol{c})) = 0$. Since $\Psi(\boldsymbol{c}) \neq 0$, it follows that L is not surjective.
$\Longrightarrow$: Assume that L is not surjective. Then it is not injective on the finite configurations. Thus, there exist a finite configuration $\boldsymbol{b} \neq 0$ with $L(\boldsymbol{b}) = 0$. By Lemma 23, there exists a finite configuration $\boldsymbol{b}' \in \Psi(G^{\mathbb{Z}})$ such that $\boldsymbol{b}' \neq 0$ and $L(\boldsymbol{b}') = 0$. Let $\boldsymbol{c} \in G^{\mathbb{Z}}$ be the finite

configuration such that $\Psi(\boldsymbol{c}) = \boldsymbol{b}'$. Clearly, it holds that $\boldsymbol{c} \neq 0$. We get $\Psi(F(\boldsymbol{c})) = L(\Psi(\boldsymbol{c})) = 0$. Since Ψ is injective, it follows that $F(\boldsymbol{c}) = 0$. Therefore, we conclude that F is not surjective. ◀

Next two theorems state that surjectivity and injectivity behave as sensitivity when looking at an additive CA over G and the associated LCA via the embedding Ψ.

▶ **Theorem 25.** *Let $(G^{\mathbb{Z}}, F)$ be any additive CA over G and let $(\hat{G}^{\mathbb{Z}}, L)$ be the LCA associated with it via the embedding Ψ. Then, the CA $(G^{\mathbb{Z}}, F)$ is surjective if and only if $(\hat{G}^{\mathbb{Z}}, L)$ is.*

Proof. Use Lemma 24. ◀

▶ **Theorem 26.** *Let $(G^{\mathbb{Z}}, F)$ be any additive CA and let $(\hat{G}^{\mathbb{Z}}, L)$ be the LCA associated with it via the embedding Ψ. Then, the CA $(G^{\mathbb{Z}}, F)$ is injective if and only if $(\hat{G}^{\mathbb{Z}}, L)$ is.*

Proof. $\Longleftarrow$: Assume that F is not injective. Then, there exist two distinct configurations $\boldsymbol{c}, \boldsymbol{c}' \in G^{\mathbb{Z}}$ with $F(\boldsymbol{c}) = F(\boldsymbol{c}')$. We get $L(\Psi(\boldsymbol{c})) = \Psi(F(\boldsymbol{c})) = \Psi(F(\boldsymbol{c}')) = L(\Psi(\boldsymbol{c}'))$ and, since Ψ is injective, it follows that L is not injective.
$\Longrightarrow$: Assume that L is not injective. Then, there exists a configuration $\boldsymbol{b} \in \hat{G}^{\mathbb{Z}}$ such that $\boldsymbol{b} \neq 0$ and $L(\boldsymbol{b}) = 0$. By Lemma 23, there exists a configuration $\boldsymbol{b}' \in \Psi(G^{\mathbb{Z}})$ such that $\boldsymbol{b}' \neq 0$ and $L(\boldsymbol{b}') = 0$. Let $\boldsymbol{c} \in G^{\mathbb{Z}}$ be the configuration such that $\Psi(\boldsymbol{c}) = \boldsymbol{b}'$. Clearly, it holds that $\boldsymbol{c} \neq 0$. We get $\Psi(F(\boldsymbol{c})) = L(\Psi(\boldsymbol{c})) = 0$. Since Ψ is injective, it follows that $F(\boldsymbol{c}) = 0$. Since $F(0) = 0$, we conclude that F is not injective. ◀

4.3 Topological transitivity and ergodicity

We start by proving that the embedding Ψ also preserves topological transitivity between an additive CA over G and the associated LCA.

▶ **Theorem 27.** *Let $(G^{\mathbb{Z}}, F)$ be any additive CA over G and let $(\hat{G}^{\mathbb{Z}}, L)$ be the LCA associated with it via the embedding Ψ. Then, the CA $(G^{\mathbb{Z}}, F)$ is topologically transitive if and only if $(\hat{G}^{\mathbb{Z}}, L)$ is.*

Proof. Since $\Psi \circ F = L \circ \Psi$, for every $k \in \mathbb{N}$ it holds that $\Psi \circ (F^k - I) = \Psi \circ F^k - \Psi = L^k \circ \Psi - \Psi = (L^k - I) \circ \Psi$. By Lemma 24 , $F^k - I$ is surjective iff $L^k - I$ is. Theorem 25 and 10 conclude the proof. ◀

As a final result, we get the decidability of many mixing and ergodic properties for additive CA over any finite abelian group, including topological transitivity and ergodicity.

▶ **Corollary 28.** *All the following properties are decidable for additive CA over any finite abelian group: (1) topological transitivity; (2) ergodicity; (3) topological mixing; (4) weak topological transitivity; (5) total transitivity; (6) weak ergodic mixing; (7) ergodic mixing.*

Proof. It is an immediate consequence of Theorem 10 and 27. ◀

5 Conclusions

In this paper we have provided many decidability and characterization results about the dynamical behavior of additive CA over finite abelian groups. These results were obtained using an embedding of linear CA over $(\mathbb{Z}/m\mathbb{Z})^n$ to additive CA over finite abelian groups and a deep algebra result about powers of matrices over commutative rings.

The are at least three main research directions that are worth investigating. First, one might ask which results and characterizations are still true when considering non-abelian groups. Second, it would be very interesting to find characterizations or decidability results about positive expansivity and strong transitivity for the case of additive CA over finite abelian groups. Finally, an important research direction consists in generalizing our results to higher dimensions (see [18] for recent results about D-dimensional CA).

References

1 Luigi Acerbi, Alberto Dennunzio, and Enrico Formenti. Conservation of some dynamical properties for operations on cellular automata. *Theoretical Computer Science*, 410(38-40):3685–3693, 2009.

2 Vincent Bernardi, Bruno Durand, Enrico Formenti, and Jarkko Kari. A new dimension sensitive property for cellular automata. In Jirí Fiala, Václav Koubek, and Jan Kratochvíl, editors, *Mathematical Foundations of Computer Science 2004, 29th International Symposium, MFCS 2004, Prague, Czech Republic, August 22-27, 2004, Proceedings*, volume 3153 of *Lecture Notes in Computer Science*, pages 416–426. Springer, 2004.

3 Nino Boccara and Henryk Fuks. Number-conserving cellular automaton rules. *Fundam. Inform.*, 52(1-3):1–13, 2002.

4 Lieven Le Bruyn and Michel Van den Bergh. Algebraic properties of linear cellular automata. *Linear algebra and its applications*, 157:217–234, 1991.

5 Gianpiero Cattaneo, Alberto Dennunzio, and Fabio Farina. A full cellular automaton to simulate predator-prey systems. In Samira El Yacoubi, Bastien Chopard, and Stefania Bandini, editors, *Cellular Automata, 7th International Conference on Cellular Automata, for Research and Industry, ACRI 2006, Perpignan, France, September 20-23, 2006, Proceedings*, volume 4173 of *Lecture Notes in Computer Science*, pages 446–451. Springer, 2006.

6 Gianpiero Cattaneo, Alberto Dennunzio, and Luciano Margara. Solution of some conjectures about topological properties of linear cellular automata. *Theoretical Computer Science*, 325(2):249–271, 2004.

7 Gianpiero Cattaneo, Enrico Formenti, Giovanni Manzini, and Luciano Margara. Ergodicity, transitivity, and regularity for linear cellular automata over $\mathbb{Z}_m$. *Theoretical Computer Science*, 233(1-2):147–164, 2000.

8 Alberto Dennunzio. From one-dimensional to two-dimensional cellular automata. *Fundamenta Informaticae*, 115(1):87–105, 2012.

9 Alberto Dennunzio, Pietro Di Lena, Enrico Formenti, and Luciano Margara. Periodic orbits and dynamical complexity in cellular automata. *Fundamenta Informaticae*, 126(2-3):183–199, 2013.

10 Alberto Dennunzio, Enrico Formenti, Darij Grinberg, and Luciano Margara. Additive cellular automata over finite abelian groups: Topological and measure theoretic properties. In *44th International Symposium on Mathematical Foundations of Computer Science, MFCS 2019, August 26-30, 2019, Aachen, Germany*, volume 138 of *LIPIcs*, pages 68:1–68:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.MFCS.2019.68.

11 Alberto Dennunzio, Enrico Formenti, Darij Grinberg, and Luciano Margara. Integrality of matrices, finiteness of matrix semigroups, and dynamics of linear and additive cellular automata. *Preprint available on arXiv*, 2019. arXiv:1907.08565.

12 Alberto Dennunzio, Enrico Formenti, Darij Grinberg, and Luciano Margara. Chaos and ergodicity are decidable for linear cellular automata over $(\mathbb{Z}/m\mathbb{Z})^n$. *Information Sciences*, 2020. doi:10.1016/j.ins.2020.05.123.

13 Alberto Dennunzio, Enrico Formenti, Darij Grinberg, and Luciano Margara. Dynamical behavior of additive cellular automata over finite abelian groups. *Theoretical Computer Science*, 2020. doi:10.1016/j.tcs.2020.06.021.

14 Alberto Dennunzio, Enrico Formenti, Luca Manzoni, Luciano Margara, and Antonio E. Porreca. On the dynamical behaviour of linear higher-order cellular automata and its decidability. *Information Sciences*, 486:73–87, 2019.

15 Alberto Dennunzio, Enrico Formenti, and Julien Provillard. Non-uniform cellular automata: Classes, dynamics, and decidability. *Information and Computation*, 215:32–46, 2012.

16 Alberto Dennunzio, Enrico Formenti, and Julien Provillard. Local rule distributions, language complexity and non-uniform cellular automata. *Theoretical Computer Science*, 504:38–51, 2013.

17 Alberto Dennunzio, Enrico Formenti, and Julien Provillard. Three research directions in non-uniform cellular automata. *Theoretical Computer Science*, 559:73–90, 2014.

18 Alberto Dennunzio, Enrico Formenti, and Michael Weiss. Multidimensional cellular automata: closing property, quasi-expansivity, and (un)decidability issues. *Theoretical Computer Science*, 516:40–59, 2014.

19 Alberto Dennunzio, Pietro Di Lena, Enrico Formenti, and Luciano Margara. On the directional dynamics of additive cellular automata. *Theoretical Computer Science*, 410(47-49):4823–4833, 2009.

20 Bruno Durand, Enrico Formenti, and Zsuzsanna Róka. Number-conserving cellular automata I: decidability. *Theoretical Computer Science*, 299(1-3):523–535, 2003.

21 Bruno Durand, Enrico Formenti, and Georges Varouchas. On undecidability of equicontinuity classification for cellular automata. In *Discrete Models for Complex Systems, DMCS'03, Lyon, France, June 16-19, 2003*, volume AB of *Discrete Mathematics and Theoretical Computer Science Proceedings*, pages 117–128. DMTCS, 2003.

22 Fabio Farina and Alberto Dennunzio. A predator-prey cellular automaton with parasitic interactions and environmental effects. *Fundamenta Informaticae*, 83(4):337–353, 2008.

23 Enrico Formenti and Aristide Grange. Number conserving cellular automata II: dynamics. *Theoretical Compututer Science*, 304(1-3):269–290, 2003.

24 Enrico Formenti, Jarkko Kari, and Siamak Taati. On the hierarchy of conservation laws in a cellular automaton. *Natural Computing*, 10(4):1275–1294, 2011.

25 Pierre Guillon and Gaétan Richard. Revisiting the Rice theorem of cellular automata. In Jean-Yves Marion and Thomas Schwentick, editors, *27th International Symposium on Theoretical Aspects of Computer Science, STACS 2010, March 4-6, 2010, Nancy, France*, volume 5 of *LIPIcs*, pages 441–452. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010.

26 Masanobu Ito, Nobuyasu Osato, and Masakazu Nasu. Linear cellular automata over $\mathbb{Z}_m$. *Journal of Computer and Systems Sciences*, 27:125–140, 1983.

27 Jarkko Kari. Rice's theorem for the limit sets of cellular automata. *Theoretical Computer Science*, 127(2):229–254, 1994.

28 Jarkko Kari. Linear cellular automata with multiple state variables. In Horst Reichel and Sophie Tison, editors, *STACS 2000*, volume 1770 of *LNCS*, pages 110–121. Springer-Verlag, 2000.

29 Petr Kůrka. Languages, equicontinuity and attractors in cellular automata. *Ergodic Theory and Dynamical Systems*, 17(2):417–433, 1997.

30 Giovanni Manzini and Luciano Margara. Attractors of linear cellular automata. *Journal of Computer & System Sciences*, 58(3):597–610, 1999.

31 Giovanni Manzini and Luciano Margara. A complete and efficiently computable topological classification of d-dimensional linear cellular automata over $\mathbb{Z}_m$. *Theoretical Computer Science*, 221(1-2):157–177, 1999.

32 Edward Forrest Moore. Machine models of self-reproduction. *Proceedings of Symposia in Applied Mathematics*, 14:13–33, 1962.

33 John Myhill. The converse to Moore's garden-of-eden theorem. *Proceedings of the American Mathematical Society*, 14:685–686, 1963.

34 Shinji Takesue. Staggered invariants in cellular automata. *Complex Systems*, 9:149–168, 1995.

The Complexity of Knapsack Problems in Wreath Products

Michael Figelius
Universität Siegen, Germany

Moses Ganardi
Universität Siegen, Germany

Markus Lohrey
Universität Siegen, Germany

Georg Zetzsche
Max Planck Institute for Software Systems (MPI-SWS), Kaiserslautern, Germany

Abstract

We prove new complexity results for computational problems in certain wreath products of groups and (as an application) for free solvable groups. For a finitely generated group we study the so-called power word problem (does a given expression $u_1^{k_1} \dots u_d^{k_d}$, where $u_1, \dots, u_d$ are words over the group generators and $k_1, \dots, k_d$ are binary encoded integers, evaluate to the group identity?) and knapsack problem (does a given equation $u_1^{x_1} \dots u_d^{x_d} = v$, where $u_1, \dots, u_d, v$ are words over the group generators and $x_1, \dots, x_d$ are variables, have a solution in the natural numbers). We prove that the power word problem for wreath products of the form $G \wr \mathbb{Z}$ with G nilpotent and iterated wreath products of free abelian groups belongs to TC^0. As an application of the latter, the power word problem for free solvable groups is in TC^0. On the other hand we show that for wreath products $G \wr \mathbb{Z}$, where G is a so called uniformly strongly efficiently non-solvable group (which form a large subclass of non-solvable groups), the power word problem is coNP-hard. For the knapsack problem we show NP-completeness for iterated wreath products of free abelian groups and hence free solvable groups. Moreover, the knapsack problem for every wreath product $G \wr \mathbb{Z}$, where G is uniformly efficiently non-solvable, is Σ_2^p-hard.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness

Keywords and phrases algorithmic group theory, knapsack, wreath product

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.126

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

Related Version A full version of the paper is available at `https://arxiv.org/abs/2002.08086` [9].

Funding *Michael Figelius*: Funded by DFG project LO 748/12-1.
Markus Lohrey: Funded by DFG project LO 748/12-1.

1 Introduction

Since the seminal work of Dehn [7] on the word and conjugacy problem in surface groups, the area of combinatorial group theory [31] is tightly connected to algorithmic questions. The famous Novikov-Boone result [4, 40] on the existence of finitely presented groups with undecidable word problem was one of the first undecidability results that touched real mathematics. Since this pioneering work, the area of algorithmic group theory has been extended in many different directions. More general algorithmic problems have been studied and also the computational complexity of group theoretic problems has been investigated. In this paper, we focus on the decidability/complexity of two specific problems in group theory that have received considerable attention in recent years: the knapsack problem and the power word problem.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 126; pp. 126:1–126:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Knapsack problems. There exist several variants of the classical knapsack problem over the integers [21]. In the variant that is particularly relevant for this paper, it is asked whether a linear equation $x_1 \cdot a_1 + \cdots + x_d \cdot a_d = b$, with $a_1, \ldots, a_d, b \in \mathbb{Z}$, has a solution $(x_1, \ldots, x_d) \in \mathbb{N}^d$. A proof for the NP-completeness of this problem for binary encoded integers $a_1, \ldots, a_d, b$ can be found in [15]. In contrast, if the numbers a_i, b are given in unary notation then the problem falls down into the circuit complexity class TC^0 [8]. In the course of a systematic investigation of classical commutative discrete optimization problems in non-commutative group theory, Myasnikov, Nikolaev, and Ushakov [33] generalized the above definition of knapsack to any f.g. group G: The input for the knapsack problem for G (KP(G) for short) is an equation of the form $g_1^{x_1} \cdots g_d^{x_d} = h$ for group elements $g_1, \ldots, g_d, h \in G$ (specified by finite words over the generators of G) and pairwise different variables $x_1, \ldots, x_d$ that take values in $\mathbb{N}$ and it is asked whether this equation has a solution (in Section 3.2, we formulate this problem in a slightly more general but equivalent way). In this form, KP($\mathbb{Z}$) is exactly the above knapsack problem for unary encoded integers studied in [8] (a unary encoded integer can be viewed as a word over a generating set $\{t, t^{-1}\}$ of $\mathbb{Z}$). For the case where $g_1, \ldots, g_d, h$ are commuting matrices over an algebraic number field, the knapsack problem has been studied in [1]. Let us emphasize that we are looking for solutions of knapsack equations in the natural numbers. One might also consider the variant, where the variables $x_1, \ldots, x_d$ take values in $\mathbb{Z}$. This latter version can be easily reduced to our knapsack version (with solutions in $\mathbb{N}$), but we are not aware of a reduction in the opposite direction.[1] Let us also mention that the knapsack problem is a special case of the more general rational subset membership problem [26].

We also consider a generalization of KP(G): An exponent equation is an equation of the form $g_1^{x_1} \cdots g_d^{x_d} = h$ as in the specification of KP(G), except that the variables $x_1, \ldots, x_d$ are not required to be pairwise different. *Solvability of exponent equations* for G (ExpEq(G) for short) is the problem where the input is a conjunction of exponent equations (possibly with shared variables) and the question is whether there is a joint solution for these equations in the natural numbers.

Let us briefly survey the results about knapsack obtained in [33] and subsequent papers:

- Knapsack can be solved in polynomial time for every hyperbolic group [33]. Some extensions of this result can be found in [11, 25].
- There are nilpotent groups of class 2 for which knapsack is undecidable. Examples are direct products of sufficiently many copies of the discrete Heisenberg group $H_3(\mathbb{Z})$ [22], and free nilpotent groups of class 2 and sufficiently high rank [37]. In contrast, knapsack for $H_3(\mathbb{Z})$ is decidable [22]. Thus, direct products to not preserve decidability of knapsack.
- Knapsack is decidable for every co-context-free group [22], i.e., groups where the set of all words over the generators that do not represent the identity is a context-free language. Lehnert and Schweitzer [23] have shown that the Higman-Thompson groups are co-context-free.
- Knapsack belongs to NP for all virtually special groups (finite extensions of subgroups of graph groups) [28]. The class of virtually special groups is very rich. It contains all Coxeter groups, one-relator groups with torsion, fully residually free groups, and fundamental groups of hyperbolic 3-manifolds. For graph groups (a.k.a. right-angled Artin groups) a

[1] Note that the problem whether a given system of linear equations has a solution in $\mathbb{N}$ is NP-complete, whereas the problem can be solved in polynomial time (using the Smith normal form) if we ask for a solution in $\mathbb{Z}$. In other words, if we consider the knapsack problem for $\mathbb{Z}^n$ with n part of the input, then looking for solutions in $\mathbb{N}$ seems to be more difficult than looking for solutions in $\mathbb{Z}$.

complete classification of the complexity was obtained in [29]: If the underlying graph contains an induced path or cycle on 4 nodes, then knapsack is NP-complete; in all other cases knapsack can be solved in polynomial time (even in LogCFL).

- Knapsack is NP-complete for every wreath product $A \wr \mathbb{Z}$ with $A \neq 1$ f.g. abelian [12] (wreath products are formally defined in Section 3.1).
- Decidability of knapsack is preserved under finite extensions, HNN-extensions over finite associated subgroups and amalgamated free products over finite subgroups [28].

For a knapsack equation $g_1^{x_1} \cdots g_d^{x_d} = h$ we may consider the set of all solutions $\{(n_1, \ldots, n_d) \in \mathbb{N}^d \mid g_1^{n_1} \cdots g_d^{n_d} = g \text{ in } G\}$. In the papers [25, 22, 29] it turned out that in many groups the solution set of every knapsack equation is a *semilinear set* (see Section 2 for a definition). We say that a group is *knapsack-semilinear* if for every knapsack equation the set of all solutions is semilinear and a semilinear representation can be computed effectively (the same holds then also for exponent equations). Note that in any group G the set of solutions on an equation $g^x = h$ is periodic and hence semilinear. This result generalizes to solution sets of knapsack instances of the for $g_1^x g_2^y = h$ (see Lemma 9), but there are examples of knapsack instances with three variables where solutions sets (in certain groups) are not semilinear. Examples of knapsack-semilinear groups are graph groups [29] (which include free groups and free abelian groups), hyperbolic groups [25], and co-context free groups [22].[2] Moreover, the class of knapsack-semilinear groups is closed under finite extensions, graph products, amalgamated free products with finite amalgamated subgroups, HNN-extensions with finite associated subgroups (see [10] for these closure properties) and wreath products [12].

Power word problems. In the power word problem for a f.g. group G (PowerWP(G) for short) the input consists of an expression $u_1^{n_1} u_2^{n_2} \cdots u_d^{n_d}$, where $u_1, \ldots, u_d$ are words over the group generators and $n_1, \ldots, n_d$ are binary encoded integers. The problem is then to decide whether the expression $u_1^{n_1} u_2^{n_2} \cdots u_d^{n_d}$ evaluates to the identity in G. The power word problem arises very naturally in the context of the knapsack problem: it allows us to verify a proposed solution for a knapsack equation with binary encoded numbers. The power word problem has been first studied in [27], where it was shown that the power word problem for f.g. free groups has the same complexity as the word problem and hence can be solved in logarithmic space. Other groups with easy power word problems are f.g. nilpotent groups and wreath products $A \wr \mathbb{Z}$ with A f.g. abelian [27]. In contrast it is shown in [27] that the power word problem for wreath products $G \wr \mathbb{Z}$, where G is either finite non-solvable or f.g. free, is coNP-complete. Implicitly, the power word problem appeared also in the work of Ge [13], where it was shown that one can verify in polynomial time an identity $\alpha_1^{n_1} \alpha_2^{n_2} \cdots \alpha_d^{n_d} = 1$, where the α_i are elements of an algebraic number field and the n_i are binary encoded integers. The power word problem is a special case of the compressed word problem [24], which asks whether a grammar-compressed word over the group generators evaluates to the group identity.

Main results. Our main focus is on the problems PowerWP(G), KP(G) and ExpEq(G) for the case where G is a wreath product. We start with the following result:

▶ **Theorem 1.** *PowerWP($G \wr \mathbb{Z}$) is in TC^0 for every f.g. nilpotent group G.*

[2] Knapsack-semilinearity of co-context free groups is not stated in [22] but follows immediately from the proof for the decidability of knapsack.

Theorem 1 generalizes the above mentioned result from [27] (for G abelian) in a nontrivial way. Our proof analyzes periodic infinite words over a nilpotent group G. Roughly speaking, we show that one can check in TC^0, whether a given list of such periodic infinite words pointwise multiplies to the identity of G. We believe that this is a result of independent interest. We use this result also in the proof of the following theorem:

▶ **Theorem 2.** *KP*$(G \wr \mathbb{Z})$ *is* NP*-complete for every finite nilpotent group* $G \neq 1$*.*

Next, we consider iterated wreath products. Fix $r \geq 1$ and define the iterated wreath products $W_{0,r} = \mathbb{Z}^r$ and $W_{m+1,r} = \mathbb{Z}^r \wr W_{m,r}$. By a famous result of Magnus [32] the free solvable group $S_{m,r}$ of derived length r and rank m embeds into $W_{m,r}$. Our main results for these groups are:

▶ **Theorem 3.** *PowerWP*$(W_{m,r})$ *and hence PowerWP*$(S_{m,r})$ *is in* TC^0 *for* $m \geq 0$*,* $r \geq 1$*.*

It was only recently shown in [35] that the word problem (and the conjugacy problem) for every free solvable group belongs to TC^0. Theorem 3 generalizes TC^0 membership of the word problem.

▶ **Theorem 4.** *ExpEq*$(W_{m,r})$ *and hence ExpEq*$(S_{m,r})$ *is* NP*-complete for* $m \geq 0$*,* $r \geq 1$*.*

For the proof of Theorem 4 we show that if a given knapsack equation over $W_{m,r}$ has a solution then it has a solution where all numbers are exponentially bounded in the length of the knapsack instance. Theorem 4 then follows easily from Theorem 3. For some other algorithmic results for free solvable groups see [34].

Finally, we show new hardness results for the power word problem and knapsack problem. For this we make use so-called *uniformly strongly efficiently non-solvable* groups (uniformly SENS groups) that were recently defined in [3]. Roughly speaking, a group G is uniformly SENS if there exists nontrivial nested commutators of arbitrary depth that moreover, are efficiently computable in a certain sense (see Section 6 for the precise definition). The essence of these groups is that they allow to carry out Barrington's argument showing the NC^1-hardness of the word problem for a finite solvable group [2]. We prove the following:

▶ **Theorem 5.** *PowerWP*$(G \wr \mathbb{Z})$ *is* coNP*-hard for every f.g. uniformly SENS group* G*.*

This result generalizes a result from [27] saying that PowerWP$(G \wr \mathbb{Z})$ is coNP-hard for the case that G is f.g. free or finite non-solvable.

▶ **Theorem 6.** *KP*$(G \wr \mathbb{Z})$ *is* Σ_2^p*-hard for every f.g. uniformly SENS group* G*.*

Recall that for every nontrivial group G, KP$(G \wr \mathbb{Z})$ is NP-hard [12]. We also show several corollaries of Theorems 5 and 6. For instance, we show that for the famous Thompson's group F, PowerWP(F) is coNP-complete and KP(F) is Σ_2^p-hard.

2 Preliminaries

Complexity theory. We assume some knowledge in complexity theory; in particular the reader should be familiar with the classes P, NP, and coNP. The class Σ_2^p (second existential level of the polynomial time hierarchy) contains all languages $L \subseteq \Sigma^*$ for which there exists a polynomial p and a language $K \subseteq \Sigma^*\#\{0,1\}^*\#\{0,1\}^*$ in P (for a symbol $\# \notin \Sigma \cup \{0,1\}$) such that $x \in L$ if and only if $\exists y \in \{0,1\}^{\leq p(|x|)} \forall z \in \{0,1\}^{\leq p(|x|)} : x\#y\#z \in K$.

The class TC^0 contains all problems that can be solved by a family of threshold circuits of polynomial size and constant depth. In this paper, TC^0 will always refer to the $\mathsf{DLOGTIME}$-uniform version of TC^0. A precise definition is not needed for our work; see [42] for details.

All we need is that the following arithmetic operations on binary encoded integers belong to TC^0: iterated addition and multiplication (i.e., addition and multiplication of n many n-bit numbers) and division with remainder.

For languages (or computational problems) $A, B_1, \ldots, B_k \subseteq \{0,1\}^*$ we write $A \in \mathsf{TC}^0(B_1, \ldots, B_k)$ (A is TC^0-Turing-reducible to $B_1, \ldots, B_k$) if A can be solved by a family of threshold circuits of polynomial size and constant depth that in addition may also use oracle gates for the languages $B_1, \ldots, B_k$ (an oracle gate for B_i yields the output 1 if and only if the string of input bits belongs to B_i).

Semilinear sets. Fix a dimension $d \geq 1$. All vectors will be column vectors. For a vector $\boldsymbol{v} = (v_1, \ldots, v_d)^\mathsf{T} \in \mathbb{Z}^d$ we define its norm $\|\boldsymbol{v}\| := \max\{|v_i| \mid 1 \leq i \leq d\}$ and for a matrix $M \in \mathbb{Z}^{c \times d}$ with entries $m_{i,j}$ ($1 \leq i \leq c$, $1 \leq j \leq d$) we define the norm $\|M\| = \max\{|m_{i,j}| \mid 1 \leq i \leq c, 1 \leq j \leq d\}$. Finally, for a finite set of vectors $A \subseteq \mathbb{N}^d$ let $\|A\| = \max\{\|\boldsymbol{a}\| \mid \boldsymbol{a} \in A\}$. We extend the operations of vector addition and multiplication of a vector by a matrix to sets of vectors in the obvious way. A *linear subset* of $\mathbb{N}^d$ is a set of the form $L = L(\boldsymbol{b}, P) := \boldsymbol{b} + P \cdot \mathbb{N}^k$, where $\boldsymbol{b} \in \mathbb{N}^d$ and $P \in \mathbb{N}^{d \times k}$. A set $S \subseteq \mathbb{N}^d$ is called *semilinear* if it is a finite union of linear sets. Semilinear sets play an important role in automata theory, logic, and other areas. They are precisely the sets definable in Presburger arithmetic, i.e. first-order logic over the structure $(\mathbb{N}, +)$, and thus form a Boolean algebra.

For a semilinear set $S = \bigcup_{i=1}^k L(\boldsymbol{b}_i, P_i)$, we call the tuple $(\boldsymbol{b}_1, P_1, \ldots, \boldsymbol{b}_k, P_k)$ a *semilinear representation* of S. The magnitude of the semilinear representation $(\boldsymbol{b}_1, P_1, \ldots, \boldsymbol{b}_k, P_k)$ is $\max\{\|\boldsymbol{b}_1\|, \|P_1\| \ldots, \|\boldsymbol{b}_k\|, \|P_k\|\}$. The *magnitude* $\|S\|$ of a semilinear set S is the minimal magnitude of all semilinear representations for S.

It is often convenient to treat mappings $\nu\colon \{x_1, \ldots, x_d\} \to \mathbb{N}$, where $X = \{x_1, \ldots, x_d\}$ is a finite set of variables, as vectors. To this end, we identify ν with the vector $(\nu(x_1), \ldots, \nu(x_d))^\mathsf{T}$. This way, vector operations (e.g. addition and scalar multiplication) and the notion of semilinearity carry over to the set $\mathbb{N}^X$ of all mappings from X to $\mathbb{N}$.

3 Groups

We assume that the reader is familiar with the basics of group theory. Let G be a group. We always write 1 for the group identity element. For $g, h \in G$ we write $[g, h] := g^{-1}h^{-1}gh$ for the commutator of g and h and g^h for $h^{-1}gh$. For subgroups A, B of G we write $[A, B]$ for the subgroup generated by all commutators $[a, b]$ with $a \in A$ and $b \in B$. The order of an element $g \in G$ is the smallest number $z > 0$ with $g^z = 1$ and ∞ if such a z does not exist. The group G is torsion-free, if every $g \in G \setminus \{1\}$ has infinite order.

We say that G is *finitely generated (f.g.)* if there is a finite subset $\Sigma \subseteq G$ such that every element of G can be written as a product of elements from Σ; such a Σ is called a *finite generating set* for G. We also write $G = \langle \Sigma \rangle$. We then have a canonical morphism $h\colon \Sigma^* \to G$ that maps a word over Σ to its product in G. If $h(w) = 1$ we also say that $w = 1$ in G. For $g \in G$ we write $|g|$ for the length of a shortest word $w \in \Sigma^*$ such that $h(w) = g$. This notation depends on the generating set Σ. We always assume that the generating set Σ is symmetric in the sense that $a \in \Sigma$ implies $a^{-1} \in \Sigma$. Then, we can define on Σ^* a natural involution $\cdot^{-1}$ by $(a_1 a_2 \cdots a_n)^{-1} = a_n^{-1} \cdots a_2^{-1} a_1^{-1}$ for $a_1, a_2, \ldots, a_n \in \Sigma$. This allows to use the notations $[g, h] = g^{-1}h^{-1}gh$ and $g^h = h^{-1}gh$ in the case $g, h \in \Sigma^*$. By *computing a homomorphism* $h\colon G_1 = \langle \Sigma_1 \rangle \to G_2 = \langle \Sigma_2 \rangle$, we mean computing the images $h(a)$ for $a \in \Sigma_1$.

A group G is called *orderable* if there exists a linear order $\leq$ on G such that $g \leq h$ implies $xgy \leq xhy$ for all $g, h, x, y \in G$ [39, 38]. Every orderable group is torsion-free (this follows directly from the definition) and has the unique roots property [41], i.e., $g^n = h^n$ implies $g = h$. The are numerous examples of orderable groups: for instance, torsion-free nilpotent groups, right-angled Artin groups, and diagram groups are all orderable.

Two elements $g, h \in G$ in a group G are called *commensurable* if $g^x = h^y$ for some $x, y \in \mathbb{Z} \setminus \{0\}$. This defines an equivalence relation on G, in which the elements with finite order form an equivalence class. By [39, Corollary 1.2] commensurable elements in an orderable group commute.

3.1 Wreath products

Let G and H be groups. Consider the direct sum $K = \bigoplus_{h \in H} G_h$, where G_h is a copy of G. We view K as the set $G^{(H)}$ of all mappings $f \colon H \to G$ such that $\mathrm{supp}(f) := \{h \in H \mid f(h) \neq 1\}$ is finite, together with pointwise multiplication as the group operation. The set $\mathrm{supp}(f) \subseteq H$ is called the *support* of f. The group H has a natural left action on $G^{(H)}$ given by $hf(a) = f(h^{-1}a)$, where $f \in G^{(H)}$ and $h, a \in H$. The corresponding semidirect product $G^{(H)} \rtimes H$ is the (restricted) *wreath product* $G \wr H$. In other words:

- Elements of $G \wr H$ are pairs (f, h), where $h \in H$ and $f \in G^{(H)}$.
- The multiplication in $G \wr H$ is defined as follows: Let $(f_1, h_1), (f_2, h_2) \in G \wr H$. Then $(f_1, h_1)(f_2, h_2) = (f, h_1h_2)$, where $f(a) = f_1(a)f_2(h_1^{-1}a)$.

There are canonical mappings

- $\sigma \colon G \wr H \to H$ with $\sigma(f, h) = h$ and
- $\tau \colon G \wr H \to G^{(H)}$ with $\tau(f, h) = f$

In other words: $g = (\tau(g), \sigma(g))$ for $g \in G \wr H$. Note that σ is a homomorphism whereas τ is in general not a homomorphism. Throughout this paper, the letters σ and τ will have the above meaning, which of course depends on the underlying wreath product $G \wr H$, but the latter will be always clear from the context.

The following intuition might be helpful: An element $(f, h) \in G \wr H$ can be thought of as a finite multiset of elements of $G \setminus \{1_G\}$ that are sitting at certain elements of H (the mapping f) together with the distinguished element $h \in H$, which can be thought of as a cursor moving in H. If we want to compute the product $(f_1, h_1)(f_2, h_2)$, we do this as follows: First, we shift the finite collection of G-elements that corresponds to the mapping f_2 by h_1: If the element $g \in G \setminus \{1_G\}$ is sitting at $a \in H$ (i.e., $f_2(a) = g$), then we remove g from a and put it to the new location $h_1a \in H$. This new collection corresponds to the mapping $f_2' \colon a \mapsto f_2(h_1^{-1}a)$. After this shift, we multiply the two collections of G-elements pointwise: If in $a \in H$ the elements g_1 and g_2 are sitting (i.e., $f_1(a) = g_1$ and $f_2'(a) = g_2$), then we put the product g_1g_2 into the location a. Finally, the new distinguished H-element (the new cursor position) becomes h_1h_2.

Clearly, H is a subgroup of $G \wr H$. We also regard G as a subgroup of $G \wr H$ by identifying G with the set of all $f \in G^{(H)}$ with $\mathrm{supp}(f) \subseteq \{1\}$. This copy of G together with H generates $G \wr H$. In particular, if $G = \langle \Sigma \rangle$ and $H = \langle \Gamma \rangle$ with $\Sigma \cap \Gamma = \emptyset$ then $G \wr H$ is generated by $\Sigma \cup \Gamma$. In this situation, we will also apply the above mappings σ and τ to words over $\Sigma \cup \Gamma$.

In [34] it was shown that the word problem of a wreath product $G \wr H$ is TC^0-reducible to the word problems for G and H. Let us briefly sketch the argument. Assume that $G = \langle \Sigma \rangle$ and $H = \langle \Gamma \rangle$. Given a word $w \in (\Sigma \cup \Gamma)^*$ one has to check whether $\sigma(w) = 1$ in H and $\tau(w)(h) = 1$ in H for all h in the support of $\tau(w)$. One can compute in TC^0 the word $\sigma(w)$ by projecting w onto the alphabet Γ. Moreover, one can enumerate the support of $\tau(w)$ by going over all prefixes of w and checking which σ-values are the same. Similarly, one produces for a given $h \in \mathrm{supp}(\tau(w))$ a word over Σ that represents $\tau(w)(h)$.

We will need the following result from [30] (which holds only for the so-called restricted wreath product that we consider in this paper):

▶ **Theorem 7** ([30]). *If G and H are orderable then also $G \wr H$ is orderable.*

3.2 Knapsack problem

Let $G = \langle \Sigma \rangle$ be a f.g. group. An *exponent expression* over G is an expression of the form $E = v_0 u_1^{x_1} v_1 u_2^{x_2} v_2 \cdots u_d^{x_d} v_d$ with $d \geq 1$, words $v_0, \ldots, v_d \in \Sigma^*$, non-empty words $u_1, \ldots, u_d \in \Sigma^*$, and variables $x_1, \ldots, x_d$. Here, we allow $x_i = x_j$ for $i \neq j$. If every variable x_i occurs at most once, then E is called a *knapsack expression*. Let $X = \{x_1, \ldots, x_d\}$ be the set of variables that occur in E. For a homomorphism $h \colon G \to G' = \langle \Sigma' \rangle$ (that is specified by a mapping from Σ to $(\Sigma' \cup \Sigma'^{-1})^*$), we denote with $h(E)$ the exponent expression $h(v_0)h(u_1)^{x_1}h(v_1)h(u_2)^{x_2}h(v_2) \cdots h(u_d)^{x_d}h(v_d)$. For a mapping $\nu \in \mathbb{N}^X$, we define $\nu(E) = v_0 u_1^{\nu(x_1)} v_1 u_2^{\nu(x_2)} v_2 \cdots u_d^{\nu(x_d)} v_d \in \Sigma^*$. We say that ν is a *G-solution* for E if $\nu(E) = 1$ in G. With $\mathrm{sol}_G(E)$ we denote the set of all G-solutions of E. The *length* of E is defined as $|E| = \sum_{i=1}^{d} |u_i| + |v_i|$. We define *solvability of exponent equations over G*, $\textsc{ExpEq}(G)$ for short, as the following decision problem:

Input A finite list of exponent expressions $E_1, \ldots, E_n$ over G.

Question Is $\bigcap_{i=1}^{n} \mathrm{sol}_G(E_i)$ non-empty?

The *knapsack problem for G*, $\mathrm{KP}(G)$ for short, is the following decision problem:

Input A single knapsack expression E over G.

Question Is $\mathrm{sol}_G(E)$ non-empty?

It is an easy observation that the choice of the generating set Σ has no influence on the decidability or complexity of these problems. For the knapsack problem in wreath products the following result has been shown in [12]:

▶ **Theorem 8** ([12]). *For every nontrivial group G, $KP(G \wr \mathbb{Z})$ is* NP*-hard.*

3.3 Knapsack-semilinear groups

The group G is called *knapsack-semilinear* if for every knapsack expression E over Σ, the set $\mathrm{sol}_G(E)$ is a semilinear set of vectors and a semilinear representation can be effectively computed from E. Since semilinear sets are effectively closed under intersection, it follows that for every exponent expression E over Σ, the set $\mathrm{sol}_G(E)$ is semilinear and a semilinear representation can be effectively computed from E. Moreover, solvability of exponent equations is decidable for every knapsack-semilinear group. As mentioned above, the class of knapsack-semilinear groups is very rich. An example of a group G, where knapsack is decidable but solvability of exponent equations is undecidable is the Heisenberg group $H_3(\mathbb{Z})$ (which consists of all upper triangular (3×3)-matrices over the integers, where all diagonal entries are 1), see [22]. In particular, $H_3(\mathbb{Z})$ is not knapsack-semilinear. A non-semilinear solution set can be achieved with a three-variable knapsack instance over $H_3(\mathbb{Z})$. For two variables, the solutions sets are semilinear for any group. In fact, they have a particularly simple structure:

▶ **Lemma 9.** *Let G be a group and $g_1, g_2, h \in G$ be elements.*

(i) *The solution set $S_1 = \{(x, y) \in \mathbb{Z}^2 \mid g_1^x g_2^y = 1\}$ is a subgroup of $\mathbb{Z}^2$. If G is torsion-free and $\{g_1, g_2\} \neq \{1\}$ then S_1 is cyclic.*

(ii) *The solution set $S = \{(x, y) \in \mathbb{Z}^2 \mid g_1^x g_2^y = h\}$ is either empty or a coset $(a, b) + S_1$ of S_1 where $(a, b) \in S$ is any solution.*

For a knapsack-semilinear group G and a finite generating set Σ for G we define a growth function. For $n \in \mathbb{N}$ let $\mathsf{Knap}(n)$ (resp., $\mathsf{Exp}(n)$) be the finite set of all knapsack expressions (resp., exponent expression) E over Σ such that $\mathrm{sol}_G(E) \neq \emptyset$ and $|E| \leq n$. We define the mapping $\mathsf{K}_{G,\Sigma} \colon \mathbb{N} \to \mathbb{N}$ and $\mathsf{E}_{G,\Sigma} \colon \mathbb{N} \to \mathbb{N}$ as follows:

$$\mathsf{K}_{G,\Sigma}(n) = \max\{\|\mathrm{sol}_G(E)\| \mid E \in \mathsf{Knap}(n)\}, \quad (1)$$

$$\mathsf{E}_{G,\Sigma}(n) = \max\{\|\mathrm{sol}_G(E)\| \mid E \in \mathsf{Exp}(n)\}. \quad (2)$$

Clearly, if $\mathrm{sol}_G(E) \neq \emptyset$ and $\|\mathrm{sol}_G(E)\| \leq N$ then E has a G-solution ν such that $\nu(x) \leq N$ for all variables x that occur in E. Thus, if G has a decidable word problem and a computable bound on the function $\mathsf{K}_{G,\Sigma}$, then we can solve $\mathrm{KP}(G)$ non-deterministically: given a knapsack expression E with variables from X, we guess $\nu \colon X \to \mathbb{N}$ with $\sigma(x) \leq N$ for all variables x and then check (using an algorithm for the word problem) whether ν is a solution.

Let Σ and Σ' be two generating sets for the group G. Then there is a constant c such that $\mathsf{K}_{G,\Sigma}(n) \leq \mathsf{K}_{G,\Sigma'}(cn)$, and similarly for $\mathsf{E}_{G,\Sigma}(n)$. To see this, note that for every $a \in \Sigma'$ there is a word $w_a \in \Sigma^*$ such that a and w_a represent the same element in G. Then we can choose $c = \max\{|w_a| \mid a \in \Sigma'\}$. Due to this fact, we do not have to specify the generating set Σ when we say that $\mathsf{K}_{G,\Sigma}$ (resp., $\mathsf{E}_{G,\Sigma}$) is polynomially/exponentially bounded.

Important for us is also the following result from [12]:

▶ **Theorem 10** ([12]). *If G and H are knapsack-semilinear then so is $G \wr H$.*

The proof of this result in [12] does not yield a good bound of $\mathsf{K}_{G\wr H}(n)$ in terms of $\mathsf{K}_G(n)$ and $\mathsf{K}_H(n)$ (and similarly for the E-function). One of our main achievements is such a bound for the case that the left factor G is f.g. abelian. For $\mathsf{E}_G(n)$ we then have the following bound, which follows from well-known bounds on solutions of linear Diophantine equations [43]:

▶ **Lemma 11.** *If G is a f.g. abelian group then $\mathsf{E}_G(n) \leq 2^{n^{\mathcal{O}(1)}}$.*

3.4 Power word problem

A *power word* (over Σ) is a tuple $(u_1, k_1, u_2, k_2, \ldots, u_d, k_d)$ where $u_1, \ldots, u_d \in \Sigma^*$ are words over the group generators (called the periods of the power word) and $k_1, \ldots, k_d \in \mathbb{Z}$ are integers that are given in binary notation. Such a power word represents the word $u_1^{k_1} u_2^{k_2} \cdots u_d^{k_d}$. We will often identify the power word $(u_1, k_1, u_2, k_2, \ldots, u_d, k_d)$ with the word $u_1^{k_1} u_2^{k_2} \cdots u_d^{k_d}$. Moreover, if $k_i = 1$, then we usually omit the exponent 1 in a power word. The *power word problem* for the f.g. group G, PowerWP(G) for short, is the following:

Input A power word $(u_1, k_1, u_2, k_2, \ldots, u_d, k_d)$.

Question Does $u_1^{k_1} u_2^{k_2} \cdots u_d^{k_d} = 1$ hold in G?

Due to the binary encoded exponents, a power word can be seen as a succinct description of an ordinary word. We have the following simple lemma.

▶ **Lemma 12.** *If the f.g. group G is knapsack-semilinear, $\mathsf{E}_G(n)$ is exponentially bounded, and* PowerWP(G) *belongs to* NP *then* ExpEq(G) *belongs to* NP.

4 Wreath products of nilpotent groups and the integers

Nilpotent groups. The *lower central series* of a group G is the sequence of groups $(G_i)_{i\geq 0}$ with $G_0 = G$ and $G_{i+1} = [G_i, G]$. The group G is *nilpotent* if there is a $c \geq 0$ with $G_c = 1$; in this case the minimal c with $G_c = 1$ is called the *nilpotency class* of G. In this section we prove Theorems 1 and 2. Our main tool are periodic words over G as introduced in [12].

Periodic words over groups. Let $G = \langle \Sigma \rangle$ be a f.g. group. Let G^ω be the set of all functions $f\colon \mathbb{N} \to G$, which forms a group by pointwise multiplication $(fg)(t) = f(t) \cdot g(t)$. A function $f \in G^\omega$ is *periodic* if there exists a number $d \geq 1$ such that $f(t) = f(t+d)$ for all $t \geq 0$. The smallest such d is called the *period* of f. If $f \in G^\omega$ has period d and $g \in G^\omega$ has period e then fg has period at most $\mathrm{lcm}(d,e)$. A periodic function $f \in G^\omega$ with period d can be specified by its initial d elements $f(0), \dots, f(d-1)$ where each element $f(t)$ is given as a word over the generating set Σ. The *periodic words problem* PERIODIC(G) over G is the following:

Input Periodic functions $f_1, \dots, f_m \in G^\omega$ and a binary encoded number T.

Question Does the product $f = \prod_{i=1}^m f_i$ satisfy $f(t) = 1$ for all $t \leq T$?

We shall derive Theorems 1 and 2 from the following result:

▶ **Theorem 13.** *If G is a f.g. nilpotent group then* PERIODIC(G) *belongs to* TC^0.

Previously it was proven that PERIODIC(G) belongs to TC^0 if G is abelian [12]. As an introduction let us reprove this result.

Let $\rho\colon G^\omega \to G^\omega$ be the *shift*-operator, i.e. $(\rho(f))(t) = f(t+1)$, which is a group homomorphism. For a subgroup H of G^ω, we denote by $H^{(n)}$ the smallest subgroup of G^ω that contains $\rho^0(H), \rho^1(H), \dots, \rho^n(H)$. Note that $(H^{(m)})^{(n)} = H^{(m+n)}$ for any $m, n \in \mathbb{N}$. A function $f \in G^\omega$ *satisfies a recurrence of order* $d \geq 1$ if $\rho^d(f)$ is contained in the subgroup $\langle f \rangle^{(d-1)}$ of G^ω. If f has period d then f clearly satisfies a recurrence of order d.

Let us now consider the case that G is abelian. Then, also G^ω is abelian and we use the additive notation for G^ω. The following lemma is folklore:

▶ **Lemma 14** (cf. [17]). *Let G be a f.g. abelian group. If $f_1, \dots, f_m \in G^\omega$ satisfy recurrences of order $d_1, \dots, d_m \geq 1$ respectively, then $\sum_{i=1}^m f_i$ satisfies a recurrence of order $\sum_{i=1}^m d_i$.*

Proof. Observe that G^ω is a $\mathbb{Z}[x]$-module with scalar multiplication

$$\sum_{i=0}^{d} a_i x^i \cdot f \mapsto \sum_{i=0}^{d} a_i \rho^i(f). \tag{3}$$

Then $f \in G^\omega$ satisfies a recurrence of order $d \geq 1$ if and only if there exists a monic polynomial $p \in \mathbb{Z}[x]$ of degree d (where monic means that the leading coefficient is one) such that $pf = 0$. Therefore, if $p_1, \dots, p_m \in \mathbb{Z}[x]$ such that $\deg(p_i) = d_i \geq 1$ and $p_i f_i = 0$ then $\prod_{i=1}^m p_i \sum_{j=1}^m f_j = \sum_{j=1}^m (\prod_{i=1}^m p_i) f_j = 0$. Since $\prod_{i=1}^m p_i$ is a monic polynomial of degree $d := \sum_{i=1}^m d_i$, $\sum_{i=1}^m f_i$ satisfies a recurrence of order d. ◀

The above lemma implies that $\sum_{i=1}^m f_i = 0$ if and only if $\sum_{i=1}^m f_i(t) = 0$ for all $0 \leq t \leq d-1$, where d is the sum of the periods of the f_i.

Let us now turn to the nilpotent case. For $n \in \mathbb{N}$, let $G^{\omega,n}$ be the subgroup of G^ω generated by all elements with period at most n. Then $G^{\omega,n}$ is closed under shift. The key fact for showing Theorem 13 is the following.

▶ **Proposition 15.** *If G is a f.g. nilpotent group, then there is a polynomial p such that every element of $G^{\omega,n}$ satisfies a recurrence of order $p(n)$.*

Let $H \leq G^\omega$ be a subgroup which is closed under shifting, i.e. $\rho(H) \subseteq H$. Since the shift is a homomorphism, the commutator subgroup $[H,H]$ is closed under shifting as well. We will work in the abelianization $H' = H/[H,H]$ where we write $\bar{f}$ for the coset $f[H,H]$. We also define $\rho\colon H' \to H'$ by $\rho(\bar{f}) = \overline{\rho(f)}$. This is well-defined since $fg^{-1} \in [H,H]$ implies $\rho(f)\rho(g)^{-1} = \rho(fg^{-1}) \in [H,H]$ and hence $\overline{\rho(f)} = \overline{\rho(g)}$. As an abelian group H' is a $\mathbb{Z}$-module and, in fact, H' forms a $\mathbb{Z}[x]$-module using the shift-operator. By the above remark (see (3)) we have the following (where we use the multiplicative notation for H'):

▶ **Lemma 16.** *H' is a $\mathbb{Z}[x]$-module with the scalar multiplication $\sum_{i=0}^{d} a_i x^i \cdot \bar{f} \mapsto \prod_{i=0}^{d} \rho^i(\bar{f})^{a_i}$.*

Our first step for proving Proposition 15 is to show that every element of $G^{\omega,n}$ satisfies a polynomial-order recurrence, modulo some element in $[G^{\omega,n}, G^{\omega,n}]$.

▶ **Lemma 17.** *For every $f \in G^{\omega,n}$, we have $\rho^d(f) \in \langle f\rangle^{(d-1)}[G^{\omega,n}, G^{\omega,n}]$ for $d = n(n+1)/2$.*

Proof. Suppose $f = f_1 \cdots f_m$ such that $f_1, \dots, f_m \in G^{\omega}$ are elements of period $\leq n$. According to Lemma 16, we consider $G^{\omega,n}/[G^{\omega,n}, G^{\omega,n}]$ as a $\mathbb{Z}[x]$-module.

If $g \in G^{\omega}$ has period q then $\rho^q(g)g^{-1} = 1$ and thus $(x^q - 1)\bar{g} = \rho^q(\bar{g})\bar{g}^{-1} = 1$. Define the polynomial $p(x) = \prod_{i=1}^{n}(x^i - 1) = \sum_{i=0}^{d} a_i x^i$ of degree $d = n(n+1)/2$ satisfying $a_d = 1$. Since all functions $f_1, \dots, f_m$ have period at most n, we have $p\bar{f} = 1$. Explicitly, this means $1 = p\bar{f} = \rho^0(\bar{f})^{a_0} \cdot \rho^1(\bar{f})^{a_1} \cdots \rho^d(\bar{f})^{a_d} = \overline{\rho^0(f)^{a_0} \cdots \rho^d(f)^{a_d}}$. Noticing that $a_d = 1$, we can write $\rho^d(f) = gh$ for some $g \in \langle f\rangle^{(d-1)}$ and $h \in [G^{\omega,n}, G^{\omega,n}]$, which has the desired form. ◀

The following lemma gives us control over the remaining factor from $[G^{\omega,n}, G^{\omega,n}]$.

▶ **Lemma 18.** *Let G be a group with nilpotency class c. Then $[G^{\omega,n}, G^{\omega,n}] \subseteq [G,G]^{\omega,n^{2c}}$.*

Proof. We need the fact that the commutator subgroup $[F,F]$ of a group F with generating set Γ is generated by all left-normed commutators $[g_1, \dots, g_k] := [[\dots[[g_1, g_2], g_3], \dots], g_k]$, where $g_1, \dots, g_k \in \Gamma \cup \Gamma^{-1}$ and $k \geq 2$, cf. [6, Lemma 2.6]. Therefore $[G^{\omega,n}, G^{\omega,n}]$ is generated by all left-normed commutators $[g_1, \dots, g_k]$ where $k \geq 2$ and $g_1, \dots, g_k \in G^{\omega}$ have period at most n. Furthermore, we can bound k by c since any left-normed commutator $[g_1, \dots, g_{c+1}]$ is trivial (recall that G is nilpotent of class c). A left-normed commutator $[g_1, \dots, g_k]$ with $2 \leq k \leq c$ and $g_1, \dots, g_k$ periodic with period at most n is a product containing at most $2k \leq 2c$ distinct functions of period at most n (namely, the $g_1, \dots, g_k$ and their inverses). Hence $[G^{\omega,n}, G^{\omega,n}]$ is generated by functions $g \in [G,G]^{\omega}$ of period at most n^{2c}. ◀

Proof of Proposition 15. We proceed by induction on the nilpotency class of G. If G has nilpotency class 0, then G is trivial and the claim is vacuous. Now suppose that G has nilpotency class $c \geq 1$. According to Lemma 17, we have $\rho^d(f) \in \langle f\rangle^{(d-1)}h$ for some $h \in [G^{\omega,n}, G^{\omega,n}]$. By Lemma 18, we have $[G^{\omega,n}, G^{\omega,n}] \subseteq [G,G]^{\omega,n^{2c}}$. Since the group $[G,G]$ has nilpotency class at most $c-1$ (we included a proof for this in the full version [9]), we may apply induction. Thus, we know that $\rho^e(h) \in \langle h\rangle^{(e-1)}$ for some $e = e(n^{2c})$. We claim that then $\rho^{d+e}(f) \in \langle f\rangle^{(d+e-1)}$. Note that $\rho^{d+e}(f) \in \rho^e(\langle f\rangle^{(d-1)}h) \subseteq \rho^e(\langle f\rangle^{(d-1)})\rho^e(h) \subseteq \langle f\rangle^{(d+e-1)} \cdot \rho^e(h)$. Therefore, it suffices to show that $\rho^e(h) \in \langle f\rangle^{(d+e-1)}$. Since $\rho^d(f) \in \langle f\rangle^{(d-1)}h$ we have $h \in \langle f\rangle^{(d)}$ and thus $\rho^e(h) \in \langle h\rangle^{(e-1)} \subseteq (\langle f\rangle^{(d)})^{(e-1)} = \langle f\rangle^{(d+e-1)}$. ◀

Proof of Theorem 13. Given periodic functions $f_1, \dots, f_m \in G^{\omega}$ with maximum period n, and a number $T \in \mathbb{N}$. By Proposition 15 the product $f = f_1 \cdots f_m$ satisfies a recurrence of order d, where d is bounded polynomially in n. Notice that $f = 1$ if and only if $f(t) = 1$ for all $t \leq d-1$. Hence, it suffices to verify that $f_1(t) \cdots f_m(t) = 1$ for all $t \leq \min\{d, T\}$. This can be accomplished by solving in parallel a polynomial number of instances of the word problem over G, which is contained in TC^0 by [36]. ◀

Proof of Theorem 1. In [27] it is shown that for every f.g. group G, PowerWP($G \wr \mathbb{Z}$) belongs to TC^0(Periodic(G), PowerWP(G)). By [27] the power word problem for a f.g. nilpotent group belongs to TC^0 and by Theorem 13, Periodic(G) belongs to TC^0. ◀

Proof of Theorem 2. By Theorem 8, KP($G \wr \mathbb{Z}$) is NP-hard. For the upper bound we use the following result from [12] that holds for every f.g. group G: There is a non-deterministic polynomial time Turing machine M that takes as input a knapsack expression E over $G \wr \mathbb{Z}$ and outputs in each leaf of the computation tree the following data: (i) an instance of $\textsc{ExpEq}(G)$ and (ii) a finite list of instances of $\textsc{Periodic}(G)$. Moreover, the input expression E has a $(G \wr \mathbb{Z})$-solution if and only if the computation tree has a leaf in which all $\textsc{Periodic}(G)$ instances are positive. If G is finite and nilpotent, then $\textsc{Periodic}(G)$ belongs to TC^0 and $\textsc{ExpEq}(G)$ belongs to NP (this holds for every finite group). The theorem follows. ◀

5 Wreath products with abelian left factors

In this section we consider wreath products $A \wr H$ where A is f.g. abelian and H is a f.g. torsion-free group. We study for which groups H, the complexity of the power word/knapsack problem in H is passed on to $A \wr H$. As applications, we obtain Theorems 3 and 4.

Power word problem over $A \wr H$. As a first step, we *normalize* a given power word $u_1^{k_1} \dots u_d^{k_d}$, i.e. ensure that $u_1, \dots, u_d \in AH$, say $u_i = a_i h_i$ for some $a_i \in A$ and $h_i \in H$ for $1 \le i \le d$. Intuitively, the computation of the power word can be described by finite progressions in the Cayley graph of H, which are labelled with elements a_i from A. The goal is to determine whether the labels on each point cancel out in the abelian group A. Here, a *progression* in H is a sequence $\boldsymbol{p} = (gh^k)_{0 \le k \le \ell}$ with *offset* $g \in H$ and *period* $h \in H$. If $h \neq 1$ then $\boldsymbol{p}$ is a *ray*. For all $1 \le i \le d$ the power word writes the element a_i into the Cayley graph of H along the progression $\boldsymbol{p}_i = (h_1^{k_1} \dots h_{i-1}^{k_{i-1}} h_i^k)_{0 \le k \le k_i}$. Notice that the offset of $\boldsymbol{p}_i$ is given as a power word for $h_1^{k_1} \dots h_{i-1}^{k_{i-1}}$ and the period is given explicitly as a word for the group element h; we call such a progression *power-compressed*.

To solve the power word problem over $A \wr H$ it seems inevitable to compute the intersection set $\{(i,j) \in [0,k] \times [0,\ell] \mid ab^i = gh^j\}$ of two given power-compressed progressions $\boldsymbol{p} = (ab^i)_{0 \le i \le k}, \boldsymbol{q} = (gh^j)_{0 \le j \le \ell}$, for any pair of progressions appearing in the power word. Such a intersection set is always a finite progression in $\mathbb{N}^2$ (c.f. Lemma 9).

However, the key insight of Theorem 3 is that it essentially suffices to compute the intersection of *parallel* rays, i.e. rays with commensurable periods. This is because two non-parallel rays can intersect at most once. Therefore, the number of points in H that cancel to zero with the help of intersections between non-parallel rays can be at most polynomial.

Therefore, roughly speaking, we proceed as follows. Consider a class C of parallel rays from the progressions $\boldsymbol{p}_1, \dots, \boldsymbol{p}_d$. First, we compute the intersection sets of all rays in C. Second, we decide whether the number of points in the support of C which do not cancel to 0 in A exceeds a polynomial bound. In order to count such non-cancelling points, we use Lemma 14 to limit the search to (polynomially many) polynomial-length rays. If our bound on such non-cancelling points is exceeded, then we can reject the entire power word: As mentioned above, non-parallel rays $\boldsymbol{p}_i$ can only intersect at a polynomial number of points in C. If, however, our bound is obeyed, we can explicitly compute the non-cancelling points (as power compressed words) for each parallelity class C and verify that they do evaluate to 0 in the entire set of progressions $\boldsymbol{p}_i$.

In order to (i) compute the intersection set of two parallel power-compressed rays and (ii) count non-cancelling points, we need to solve a generalization of the power word problem in the group H, which we explain next. For a f.g. group $G = \langle \Sigma \rangle$ we define the *power compressed power problem* $\textsc{PowerPP}(G)$:

Input A word $u \in \Sigma^*$ and a power word $(v_1, k_1, \dots, v_d, k_d)$ over Σ.

Output A binary encoded number $z \in \mathbb{Z}$ with $u^z = v$ where $v = v_1^{k_1} \dots v_d^{k_d}$, or **no** if $u^z = v$ has no solution.

Note that the word u in the input of PowerPP is uncompressed. In order to guarantee that we have small uncompressed inputs to PowerPP, we need to show another property of our groups. Specifically, we prove that the intersection set of parallel rays has a small period: A group $G = \langle \Sigma \rangle$ is *tame with respect to commensurability*, or short *c-tame*, if there exists a number $d \in \mathbb{N}$ such that for all commensurable elements $g, h \in G$ having infinite order there exist numbers $s, t \in \mathbb{Z} \setminus \{0\}$ such that $g^s = h^t$ and $|s|, |t| \leq \mathcal{O}((|g| + |h|)^d)$.

Our algorithm for the power word problem sketched above yields the following:

▶ **Proposition 19.** *If the group H is c-tame and torsion-free then PowerWP$(A \wr H)$ is TC^0-reducible to PowerPP(H).*

This means, in order to solve the power word problem for groups $W_{m,r}$ and $S_{m,r}$ in TC^0, we also need to solve the power compressed power problem in TC^0. To this end, we first establish TC^0 membership of PowerPP in groups $W_{m,r}$ in the following transfer result.

▶ **Theorem 20.** *Let H and A be f.g. groups where A is abelian and H is c-tame and torsion-free. Then PowerPP$(A \wr H)$ is TC^0-reducible to PowerPP(H).*

To show Theorem 20, we provide an elementary (but still somewhat involved) TC^0-reduction from PowerPP$(A \wr H)$ to PowerWP$(A \wr H)$ and PowerPP(H) and apply Proposition 19.

Finally, we need to show that all the groups $W_{m,r}$ and $S_{m,r}$ are c-tame.

▶ **Proposition 21.** *For all $r \geq 1$, $m \geq 0$ the groups $W_{m,r}$ and $S_{m,r}$ are c-tame.*

For Proposition 21, we use elementary arguments and the unique roots property of $W_{m,r}$. The preceding ingredients now yield Theorem 3.

Proof of Theorem 3. We will prove by induction on $m \in \mathbb{N}$ that PowerPP$(W_{m,r})$ and hence also PowerWP$(W_{m,r})$ belongs to TC^0. If $m = 0$ then PowerPP$(W_{0,r})$ is the problem of solving a system of r linear equations $a_i x = b_i$ where a_i is given in unary encoding and b_i is given in binary encoding for $1 \leq i \leq r$. Since integer division belongs to TC^0 (here, we only have to divide by the unary encoded integers a_i) this problem can be solved in TC^0. The inductive step follows from Theorem 20 and the fact that all groups $W_{m,r}$ are c-tame (Proposition 21) and torsion-free. ◀

Knapsack problem over $A \wr H$. For the knapsack problem we prove the following transfer theorem (recall the definition of an orderable group from Section 3 and the definition of the function $\mathsf{E}_G(n)$ from (2) in Section 3.3):

▶ **Theorem 22.** *Let H and A be f.g. groups where A is abelian and H is orderable and knapsack-semilinear. If $\mathsf{E}_H(n)$ is exponentially bounded then so is $\mathsf{E}_{A \wr H}(n)$.*

The proof of Theorem 22 follows a similar pattern as Theorem 20. The condition that H is orderable ensures that parallel rays in H are contained in cosets of a common cyclic subgroup. We describe the solution set of an exponent equation over $A \wr H$ as a disjunction of polynomially large existential Presburger formulas, which use exponent equations over H and inequalities as atomic formulas. Here, we do not need to algorithmically construct the formula: Its mere existence yields an exponential bound on the size of a solution.

Using Theorem 3 and 22 we can prove Theorem 4: let us fix an iterated wreath product $W = W_{m,r}$ for some $m \geq 0$, $r \geq 1$ (recall that $W_{0,r} = \mathbb{Z}^r$ and $W_{m+1,r} = \mathbb{Z}^r \wr W_{m,r}$). Since $\mathbb{Z}^m$ is orderable, Theorem 7 implies that W is orderable. Moreover, by Theorem 10, W is also knapsack-semilinear. Since by Lemma 11, $\mathsf{E}_A(n)$ is exponentially bounded for every f.g. abelian group A, it follows from Theorem 22 that $\mathsf{E}_W(n)$ is exponentially bounded as well. By Theorem 3 and Lemma 12, $\textsc{ExpEq}(W)$ belongs to NP. Finally, NP-hardness of $\textsc{ExpEq}(W)$ follows from the fact that the question whether a given system of linear Diophantine equations with unary encoded numbers has a solution in $\mathbb{N}$ is NP-hard.

6 Wreath products with difficult knapsack and power word problems

In this section we provide additional details concerning Theorems 5 and 6. We start with a formal definition of uniformly SENS groups [3].

Strongly efficiently non-solvable groups. Let us fix a f.g. group $G = \langle \Sigma \rangle$. Following [3] we need the additional assumption that the generating set Σ contains the group identity 1. This allows to pad words over Σ to any larger length without changing the group element represented by the word. One also says that Σ is a *standard generating set* for G. The group G is called *strongly efficiently non-solvable (SENS)* if there is a constant $\mu \in \mathbb{N}$ such that for every $d \in \mathbb{N}$ and $v \in \{0,1\}^{\leq d}$ there is a word $w_{d,v} \in \Sigma^*$ with the following properties:

- $|w_{d,v}| = 2^{\mu d}$ for all $v \in \{0,1\}^d$,
- $w_{d,v} = [w_{d,v0}, w_{d,v1}]$ for all $v \in \{0,1\}^{<d}$ (here we take the commutator of words),
- $w_{d,\varepsilon} \neq 1$ in G.

The group G is called *uniformly strongly efficiently non-solvable* if, moreover,

- given $v \in \{0,1\}^d$, a binary number i with μd bits, and $a \in \Sigma$ one can decide in linear time on a random access Turing-machine whether the i-th letter of $w_{d,v}$ is a.

In [3] the authors defines also the weaker condition of being (uniformly) efficiently non-solvable. The definition is more technical and it is not clear whether it really leads to a larger class of groups. Examples for uniformly SENS groups are: finite non-solvable groups (more generally, every f.g. group that has a finite non-solvable quotient), f.g. non-abelian free groups, Thompson's group F, and weakly branched self-similar groups with a f.g. branching subgroup (this includes several famous self-similar groups like the Grigorchuk group, the Gupta-Sidki groups and the Tower of Hanoi groups); see [3] for details.

Wreath products with difficult knapsack problems. Recall that Theorem 6 states that $\mathrm{KP}(G \wr \mathbb{Z})$ is Σ_2^p-hard for every uniformly SENS group G. For the proof we consider G-programs. A *G-program* is a sequence of instructions (X, a, b) where X is a boolean variable and a, b are generators of G. Given an assignment for the boolean variables, one can evaluate the G-program in the natural way: If X is set to 1 (resp., 0) then the instruction (X, a, b) evaluates to a (resp. b). The resulting sequence of group generators evaluates to an element of G and this is the evaluation of the G-program under the given assignment. We consider now the following computational problem $\exists\forall\text{-}\textsc{Sat}(G)$: Given a G-program P, whose variables are split into two sets $\overline{X}$ and $\overline{Y}$, does there exist an assignment $\alpha : \overline{X} \to \{0,1\}$ such that for every assignment $\beta : \overline{Y} \to \{0,1\}$ the program P evaluates to the group identity under the combined assignment $\alpha \cup \beta$?

We prove Theorem 6 in two steps. The first is Σ_2^p-hardness of $\exists\forall\text{-}\textsc{Sat}(G)$.

▶ **Lemma 23.** *Let the f.g. group $G = \langle \Sigma \rangle$ be uniformly SENS. Then, $\exists\forall\text{-}\textsc{Sat}(G)$ is Σ_2^p-hard.*

Proof. We prove the lemma by a reduction from the following Σ_2^p-complete problem: given a boolean formula $F = F(\overline{X}, \overline{Y})$ in disjunctive normal form, where $\overline{X}$ and $\overline{Y}$ are disjoint tuples of boolean variables, does the quantified boolean formula $\exists \overline{X} \forall \overline{Y} : F$ hold? Let us fix such a formula $F(\overline{X}, \overline{Y})$. We can write F as a fan-in two boolean circuit of depth $\mathcal{O}(\log |F|)$.

By [3, Remark 34] we can compute in logspace from F a G-program P over the variables $\overline{X} \cup \overline{Y}$ of length polynomial in $|F|$ such that for every assignment $\gamma : \overline{X} \cup \overline{Y} \to \{0,1\}$ the following two statements are equivalent:

- $F(\gamma(\overline{X}), \gamma(\overline{Y}))$ holds.
- $P(\gamma) = 1$ in G.

Hence, $\exists\overline{X}\forall\overline{Y} : F$ holds if and only if $\exists\overline{X}\forall\overline{Y} : P = 1$ holds. ◀

The second step is to reduce $\exists\forall$-SAT(G) to KP($G \wr \mathbb{Z}$). In fact, this reduction works for any f.g. group G.

▶ **Lemma 24.** *For every f.g. nontrivial group G, $\exists\forall$-SAT(G) is logspace many-one reducible to KP($G \wr \mathbb{Z}$).*

Proof sketch. Let us fix a G-program

$$P = (Z_1, a_1, b_1)(Z_2, a_2, b_2) \cdots (Z_\ell, a_\ell, b_\ell) \in ((\overline{X} \cup \overline{Y}) \times \Sigma \times \Sigma)^*$$

where $\overline{X}$ and $\overline{Y}$ are disjoint sets of variables. Let $m = |\overline{X}|$ and $n = |\overline{Y}|$. We want to construct a knapsack expression E over $G \wr \mathbb{Z}$ which has a solution if and only if there is an assignment $\alpha : \overline{X} \to \{0,1\}$ such that $P(\alpha \cup \beta) = 1$ for every assignment $\beta : \overline{Y} \to \{0,1\}$. Let us choose a generator t for $\mathbb{Z}$. Then $\Sigma \cup \{t, t^{-1}\}$ generates the wreath product $G \wr \mathbb{Z}$. First, we compute in logspace the $m+n$ first primes $p_1, \ldots, p_{m+n}$ and fix a bijection $p : \overline{X} \cup \overline{Y} \to \{p_1, \ldots, p_{m+n}\}$. Moreover, let $M = \prod_{i=1}^{m+n} p_i$.

Roughly speaking, the idea is as follows. Each assignment $\alpha : \overline{X} \to \{0,1\}$ will correspond to a valuation ν for our expression E. The resulting element $\nu(E) \in G \wr \mathbb{Z}$ then encodes the value $P(\alpha \cup \beta)$ for each $\beta \colon \overline{Y} \to \{0,1\}$ in some position $s \in [0, M-1]$. To be precise, to each $s \in [0, M-1]$, we associate the assignment $\beta_s \colon \overline{Y} \to \{0,1\}$ where $\beta_s(Y) = 1$ if and only if $s \equiv 0 \bmod p(Y)$. Then, $\tau(\nu(E))(s)$ will be $P(\alpha \cup \beta_s)$. This means, $\nu(E) = 1$ implies that $P(\alpha \cup \beta) = 1$ for all assignments $\beta \colon \overline{Y} \to \{0,1\}$.

Our expression implements this as follows. For each $i = 1, \ldots, \ell$, it walks to the right to some position $M' \geq M$ and then walks back to the origin. On the way to the right, the behavior depends on whether Z_i is an existential or a universal variable. If Z_i is existential, we either place a_i at every position (if $\alpha(Z_i) = 1$) or b_i at every position (if $\alpha(Z_i) = 0$). If Z_i is universal, we place a_i in the positions divisible by $p(Z_i)$; and we place b_i in the others. That way, in position $s \in [0, M-1]$, the accumulated element will be $P(\alpha \cup \beta_s)$. The complete proof can be found in the full version [9]. ◀

Let us now show some applications of Theorem 6:

▶ **Corollary 25.** *KP($G \wr \mathbb{Z}$) is Σ_2^p-complete for G finite non-solvable or f.g.non-abelian free.*

Proof. Finite non-solvable groups and f.g. non-abelian free groups are uniformly SENS [3]. By Theorem 6, KP($G \wr \mathbb{Z}$) is Σ_2^p-hard. It remains to show that KP($G \wr \mathbb{Z}$) belongs to Σ_2^p. According to [12] (see also the proof of Theorem 2) it suffices to show that PERIODIC(G) and EXPEQ(G) both belong to Σ_2^p. The problem PERIODIC(G) belongs to coNP (since the word problem for G can be solved in polynomial time) and EXPEQ(G) belongs to NP. For a finite group this is clear and for a free group one can use [29]. ◀

Theorem 6 can be also applied to Thompson's group F. This is one of the most well studied groups in (infinite) group theory due to its unusual properties, see e.g. [5]. It can be defined in several ways; let us just mention the following finite presentation: $F = \langle x_0, x_1 \mid [x_0x_1^{-1}, x_0^{-1}x_1x_0], [x_0x_1^{-1}, x_0^{-2}x_1x_0^2] \rangle$. Thompson's group F is uniformly SENS [3] and contains a copy of $F \wr \mathbb{Z}$ [14]. Theorem 6 yields:

▶ **Corollary 26.** *The knapsack problem for Thompson's group F is Σ_2^p-hard.*

We conjecture Σ_2^p-completeness. Since F is co-context-free [23], KP(F) is decidable [22].

Wreath product with difficult power word problems. In [27] it was shown that the problem PowerWP($G \wr \mathbb{Z}$) is coNP-complete in case G is a finite non-solvable group or a f.g. free group. The proof in [27] immediately generalizes to the case were G is uniformly SENS. This yields Theorem 5. Alternatively, one can prove Theorem 5 by showing that

- $\forall$-Sat(G) (the question whether a given G-program P evaluates to the group identity for all assignment) is coNP-hard if G is uniformly SENS, and
- $\forall$-Sat(G) is logspace many-one reducible to PowerWP($G \wr \mathbb{Z}$).

This can be shown with the same reductions as in Lemmas 23 and 24.

Fix a f.g. group $G = \langle \Sigma \rangle$. With WP$(G, \Sigma)$ we denote the set of all words $w \in \Sigma^*$ such that $w = 1$ in G (the word problem for G with respect to Σ). We say that G is *co-context-free* if $\Sigma^* \setminus \mathrm{WP}(G, \Sigma)$ is context-free (the choice of Σ is not relevant for this) [18, Section 14.2].

▶ **Theorem 27.** *The power word problem for a co-context-free group G belongs to* coNP*.*

Proof. The following argument is similar to the decidability proof for knapsack in co-context-free groups in [22]. Let $G = \langle \Sigma \rangle$ and let $(u_1, k_1, u_2, k_2, \dots, u_d, k_d)$ be the input power word, where $u_i \in \Sigma^*$. We can assume that all k_i are positive. We have to check whether $u_1^{k_1} u_2^{k_2} \cdots u_d^{k_d}$ is trivial in G. Let L be the complement of WP(G, Σ), which is context-free. Take the alphabet $\{a_1, \dots, a_d\}$ and define the morphism $h : \{a_1, \dots, a_d\}^* \to \Sigma^*$ by $h(a_i) = u_i$. Consider the language $K = h^{-1}(L) \cap a_1^* a_2^* \cdots a_d^*$. Since the context-free languages are closed under inverse morphisms and intersections with regular languages, K is context-free too. Moreover, from the tuple $(u_1, u_2, \dots, u_d)$ we can compute in polynomial time a context-free grammar for K: Start with a push-down automaton M for L (since L is a fixed language, this is an object of constant size). From M one can compute in polynomial time a push-down automaton M' for $h^{-1}(L)$: when reading the symbol a_i, M' has to simulate (using ε-transitions) M on $h(a_i)$. Next, we construct in polynomial time a push-down automaton M'' for $h^{-1}(L) \cap a_1^* a_2^* \cdots a_d^*$ using a product construction. Finally, we transform M'' back into a context-free grammar. This is again possible in polynomial time using the standard triple construction. It remains to check whether $a_1^{k_1} a_2^{k_2} \cdots a_d^{k_d} \notin L(G)$. This is equivalent to $(k_1, k_2, \dots, k_d) \notin \Psi(L(G))$, where $\Psi(L(G))$ denotes the Parikh image of $L(G)$. Checking $(k_1, k_2, \dots, k_d) \in \Psi(L(G))$ is an instance of the uniform membership problem for commutative context-free languages, which can be solved in NP according to [19]. This implies that the power word problem for G belongs to coNP. ◀

▶ **Theorem 28.** *For Thompson's group F, the power word problem is* coNP*-complete.*

Proof. Since F is co-context-free [23], Theorem 27 yields the upper bound. The lower bound follows from Theorem 5 and the facts that F is uniformly SENS and that $F \wr \mathbb{Z} \leq F$. ◀

7 Open problems

Our results naturally lead to several open research problems:

- Theorems 1 and 5 leave some room for further improvements. In this context, a particularly interesting problem is the power word problem for a wreath product $G \wr \mathbb{Z}$, where G is finite solvable but not nilpotent. Recall that for Theorem 5 we reduced $\forall$-Sat(G) to PowerWP($G \wr \mathbb{Z}$). This reduction works for every non-trivial f.g. group. Moreover, the problem whether a given equation $u = v$ with variables holds in G for all assignments of the variables to elements of G (called EqnId(G) in [44]) can be easily reduced to ∀-Sat(G). This allows us to apply recent results from [44], where the author constructs finite solvable groups G for which EqnId(G) cannot be solved in polynomial time assuming the exponential time hypothesis (this holds for instance for all finite solvable groups of Fitting length at least 4). Hence, there is no hope to find a polynomial time algorithm for the power word problem for $G \wr \mathbb{Z}$ for every finite solvable group G, but one can still look at restricted classes of solvable groups.

- We believe that in Theorem 22, the assumption that H is orderable is not needed. In other words, we conjecture the following: Let H and A be f.g. groups where A is abelian and H is knapsack-semilinear. If $\mathsf{E}_H(n)$ is exponentially bounded then so is $\mathsf{E}_{A \wr H}(n)$.
- Recall the we proved that knapsack for Thompson's group F is Σ_2^p-hard. Decidability of knapsack for Thompson's group F follows from [22] and the fact that F is co-context-free. It is shown in [22] that for every co-context-free group the knapsack problem reduces to checking non-universality of the Parikh image of a bounded context-free language. The latter problem belongs to $\mathsf{NEXPTIME}$ [20, Theorem 2.10] (see also [16, Corollary 1]). It would be interesting to find better complexity bounds for this problem.

References

1 Lazlo Babai, Robert Beals, Jin-Yi Cai, Gábor Ivanyos, and Eugene M. Luks. Multiplicative equations over commuting matrices. In *Proceedings of SODA 1996*, pages 498–507. ACM/SIAM, 1996.

2 David A. Mix Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *Journal of Computer and System Sciences*, 38:150–164, 1989.

3 Laurent Bartholdi, Michael Figelius, Markus Lohrey, and Armin Weiß. Groups with ALOGTIME-hard word problems and PSPACE-complete compressed word problems. Technical report, arXiv.org, 2020. arXiv:1909.13781.

4 William Boone. The word problem. *Annals of Mathematics. Second Series*, 70:207–265, 1959.

5 John W. Cannon, William J. Floyd, and Walter R. Parry. Introductory notes on Richard Thompson's groups. *L'Enseignement Mathématique*, 42(3):215–256, 1996.

6 Anthony E. Clement, Stephen Majewicz, and Marcos Zyman. *The Theory of Nilpotent Groups*. Springer, 2017.

7 Max Dehn. Über unendliche diskontinuierliche Gruppen. *Mathematische Annalen*, 71:116–144, 1911. In German.

8 Michael Elberfeld, Andreas Jakoby, and Till Tantau. Algorithmic meta theorems for circuit classes of constant and logarithmic depth. *Electronic Colloquium on Computational Complexity (ECCC)*, 18:128, 2011.

9 Michael Figelius, Moses Ganardi, Markus Lohrey, and Georg Zetzsche. The complexity of knapsack problems in wreath products. *CoRR*, abs/2002.08086, 2020. arXiv:2002.08086.

10 Michael Figelius, Markus Lohrey, and Georg Zetzsche. Closure properties of knapsack semilinear groups. *CoRR*, abs/1911.12857, 2019. arXiv:1911.12857.

11 Elizaveta Frenkel, Andrey Nikolaev, and Alexander Ushakov. Knapsack problems in products of groups. *Journal of Symbolic Computation*, 74:96–108, 2016.

12 Moses Ganardi, Daniel König, Markus Lohrey, and Georg Zetzsche. Knapsack problems for wreath products. In *Proceedings of STACS 2018*, volume 96 of *LIPIcs*, pages 32:1–32:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

13 Guoqiang Ge. Testing equalities of multiplicative representations in polynomial time (extended abstract). In *Proceedings of the 34th Annual Symposium on Foundations of Computer Science, FOCS 1993*, pages 422–426. IEEE Computer Society, 1993.

14 Victor S. Guba and Mark V. Sapir. On subgroups of the R. Thompson group F and other diagram groups. *Mat. Sb.*, 190(8):3–60, 1999.

15 Christoph Haase. *On the complexity of model checking counter automata*. PhD thesis, University of Oxford, St Catherine's College, 2011.

16 Christoph Haase. Subclasses of Presburger arithmetic and the weak EXP hierarchy. In *Proceedings of CSL-LICS 2014*, pages 47:1–47:10. ACM, 2014.

17 Tore Herlestam. On functions of linear shift register sequences. In *Proceedings of EUROCRYPT '85*, volume 219 of *Lecture Notes in Computer Science*, pages 119–129. Springer, 1986.

18 Derek F. Holt, Sarah Rees, and Claas E. Röver. *Groups, Languages and Automata*, volume 88 of *London Mathematical Society Student Texts*. Cambridge University Press, 2017.

19 Dung T. Huynh. Commutative grammars: The complexity of uniform word problems. *Information and Control*, 57:21–39, 1983.

20 Dung T. Huynh. The complexity of equivalence problems for commutative grammars. *Information and Control*, 66(1/2):103–121, 1985.

21 Richard M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

22 Daniel König, Markus Lohrey, and Georg Zetzsche. Knapsack and subset sum problems in nilpotent, polycyclic, and co-context-free groups. In *Algebra and Computer Science*, volume 677 of *Contemporary Mathematics*, pages 138–153. American Mathematical Society, 2016.

23 Jörg Lehnert and Pascal Schweitzer. The co-word problem for the Higman-Thompson group is context-free. *Bulletin of the London Mathematical Society*, 39(2):235–241, 2007.

24 Markus Lohrey. *The Compressed Word Problem for Groups*. SpringerBriefs in Mathematics. Springer, 2014.

25 Markus Lohrey. Knapsack in hyperbolic groups. *Journal of Algebra*, 545:390–415, 2020.

26 Markus Lohrey, Benjamin Steinberg, and Georg Zetzsche. Rational subsets and submonoids of wreath products. *Information and Computation*, 243:191–204, 2015.

27 Markus Lohrey and Armin Weiß. The power word problem. In *Proceedings of MFCS 2019*, volume 138 of *LIPIcs*, pages 43:1–43:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

28 Markus Lohrey and Georg Zetzsche. Knapsack in graph groups, HNN-extensions and amalgamated products. In *Proceedings of STACS 2016*, volume 47 of *LIPIcs*, pages 50:1–50:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

29 Markus Lohrey and Georg Zetzsche. Knapsack in graph groups. *Theory of Computing Systems*, 62(1):192–246, 2018.

30 Patrizia Longobardi and Mercede Maj. On some classes of orderable groups. *Milan Journal of Mathematics*, 68(1):203–216, 1998.

31 R. C. Lyndon and Paul E. Schupp. *Combinatorial Group Theory*. Springer, 1977.

32 Wilhelm Magnus. On a theorem of Marshall Hall. *Annals of Mathematics. Second Series*, 40:764–768, 1939.

33 Alexei Miasnikov, Andrey Nikolaev, and Alexander Ushakov. Knapsack problems in groups. *Mathematics of Computation*, 84:987–1016, 2015.

34 Alexei Miasnikov, Vitaly Roman'kov, Alexander Ushakov, and Anatoly Vershik. The word and geodesic problems in free solvable groups. *Transactions of the American Mathematical Society*, 362(9):4655–4682, 2010.

35 Alexei Miasnikov, Svetla Vassileva, and Armin Weiß. The conjugacy problem in free solvable groups and wreath products of abelian groups is in TC^0. *Theory of Computing Systems*, 63(4):809–832, 2019.

36 Alexei Miasnikov and Armin Weiß. TC^0 circuits for algorithmic problems in nilpotent groups. In *Proceedings of MFCS 2017*, volume 83 of *LIPIcs*, pages 23:1–23:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

37 Alexei Mishchenko and Alexander Treier. Knapsack problem for nilpotent groups. *Groups Complexity Cryptology*, 9(1):87–98, 2017.

38 Roberta Mura and Akbar H. Rhemtulla. *Orderable groups*. Marcel Dekker, 1977.

39 Bernhard Hermann Neumann. On ordered groups. *American Journal of Mathematics*, 71(1):1–18, 1949.

40 Pyotr S. Novikov. On the algorithmic unsolvability of the word problem in group theory. *American Mathematical Society, Translations, II. Series*, 9:1–122, 1958.

41 Dale Rolfsen. Low-dimensional topology and ordering groups. *Mathematica Slovaca*, 64(3):579–600, 2014.

42 Heribert Vollmer. *Introduction to Circuit Complexity.* Springer, 1999.

43 Joachim von zur Gathen and Malte Sieveking. A bound on solutions of linear integer equalities and inequalities. *Proceedings of the American Mathematical Society*, 72(1):155–158, 1978.

44 Armin Weiß. Hardness of equations over finite solvable groups under the exponential time hypothesis. *CoRR*, abs/2002.10145, 2020. `arXiv:2002.10145`.

The Adversarial Stackelberg Value in Quantitative Games

Emmanuel Filiot
Université libre de Bruxelles (ULB), Belgium

Raffaella Gentilini
University of Perugia, Italy

Jean-François Raskin
Université libre de Bruxelles (ULB), Belgium

Abstract

In this paper, we study the notion of adversarial Stackelberg value for two-player non-zero sum games played on bi-weighted graphs with the mean-payoff and the discounted sum functions. The adversarial Stackelberg value of Player 0 is the largest value that Player 0 can obtain when announcing her strategy to Player 1 which in turn responds with any of his best response. For the mean-payoff function, we show that the adversarial Stackelberg value is not always achievable but ϵ-optimal strategies exist. We show how to compute this value and prove that the associated threshold problem is in NP. For the discounted sum payoff function, we draw a link with the target discounted sum problem which explains why the problem is difficult to solve for this payoff function. We also provide solutions to related gap problems.

2012 ACM Subject Classification Theory of computation → Solution concepts in game theory; Theory of computation → Logic and verification

Keywords and phrases Non-zero sum games, reactive synthesis, adversarial Stackelberg

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.127

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

Related Version A full version of the paper is available at `http://arxiv.org/abs/2004.12918`.

Funding This work is partially supported by the PDR project Subgame perfection in graph games (F.R.S.-FNRS), the MIS project F451019F (F.R.S.-FNRS), the ARC project Non-Zero Sum Game Graphs: Applications to Reactive Synthesis and Beyond (Fédération Wallonie-Bruxelles), the EOS project Verifying Learning Artificial Intelligence Systems (F.R.S.-FNRS & FWO), and the COST Action 16228 GAMENET (European Cooperation in Science and Technology). Emmanuel Filiot is associate researcher at F.R.S.-FNRS.

Acknowledgements We thank anonymous reviewers, Dr. Shibashis Guha and Ms. Mrudula Balachander for useful comments on a preliminary version of this paper.

1 Introduction

In this paper, we study two-player non-zero sum infinite duration quantitative games played on graph games. In non-zero sum games, the notion of worst-case value is not rich enough to reason about the (rational) behavior of players. More elaborate solution concepts have been proposed in game theory to reason about non-zero sum games: Nash equilibria, subgames perfect equilibria, admissibility, and Stackelberg equilibria are important examples of such solution concepts, see e.g. [18] and [19].

Let us first recall the abstract setting underlying the notion of Stackelberg equilibria and explain the variant that is the focus of this paper. Stackelberg games are strategic games played by two players. We note Σ_0 the set of strategies of Player 0, also called

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 127; pp. 127:1–127:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

the *leader*, and Σ_1 the set of strategies of Player 1, also called the *follower*. Additionally, the game comes with two (usually $\mathbb{R}$-valued) payoff functions, Payoff_0 and Payoff_1, that determine the payoff each player receives: if $\sigma_0 \in \Sigma_0$ and $\sigma_1 \in \Sigma_1$ are chosen then Player 0 receives the payoff $\mathsf{Payoff}_0(\sigma_0, \sigma_1)$ while Player 1 receives the payoff $\mathsf{Payoff}_1(\sigma_0, \sigma_1)$. Both players aim at maximizing their respective payoffs, and in a Stackelberg game, players play sequentially as follows. ① Player 0, the leader, announces her choice of strategy $\sigma_0 \in \Sigma_0$. ② Player 1, the follower, announces his choice of strategy $\sigma_1 \in \Sigma_1$. ③ Both players receive their respective payoffs: $\mathsf{Payoff}_0(\sigma_0, \sigma_1)$ and $\mathsf{Payoff}_1(\sigma_0, \sigma_1)$. Due to the sequential nature of the game, Player 1 knows the strategy σ_0, and so to act rationally (s)he should choose a strategy σ_1 that maximizes the payoff $\mathsf{Payoff}_1(\sigma_0, \sigma_1)$. If such a strategy σ_1 exists, it is called a *best-response* [1] to the strategy $\sigma_0 \in \Sigma_0$. In turn, if the leader assumes a rational response of the follower to her strategy, this should guide the leader when choosing $\sigma_0 \in \Sigma_0$. Indeed, the leader should choose a strategy $\sigma_0 \in \Sigma_0$ such that the value $\mathsf{Payoff}_0(\sigma_0, \sigma_1)$ is as large as possible when σ_1 is a best-response of the follower.

Two different scenarios can be considered in this setting: either the best-response $\sigma_1 \in \Sigma_1$ is imposed by the leader (or equivalently chosen *cooperatively* by the two players), or the best-response is chosen *adversarially* by Player 1. In classical results from game theory and most of the close related works on games played on graphs [13, 15], with the exception of [17], only the cooperative scenario has been investigated. But, the adversarial case is interesting because it allows us to model the situation in which the leader chooses $\sigma_0 \in \Sigma_0$ only and must be prepared to face any rational response of Player 1, i.e. if Player 1 has several possible best responses then σ_0 should be designed to face all of them. In this paper, our main contribution is to investigate the second route. As already noted in [17], this route is particularly interesting for applications in *automatic synthesis*. Indeed, when designing a program, and this is especially true for reactive programs [20, 2], we aim for robust solutions that works for multiple rational usages, e.g. all the usages that respect some specification or that maximize some measure for the user.

To reflect the two scenarios above, there are two notions of *Stackelberg values*. First, the *cooperative Stackelberg value* is the largest value that Player 0 can secure by proposing a strategy σ_0 and a strategy σ_1 to the follower with the constraint that σ_1 is a best-response for the follower to σ_0. Second, the *adversarial Stackelberg value* is the largest value that Player 0 can secure by proposing a strategy σ_0 and facing any best response σ_1 of the follower to the strategy σ_0. In this paper, we mostly concentrate on the *adversarial* Stackelberg value, for infinite duration games played on bi-weighted game graphs for the mean-payoff function and the discounted sum function. The cooperative case has been studied in [13, 15] and we only provide some additional results when relevant for that case (see also related works below).

Main contributions. First, we consider the mean-payoff function. For this payoff function, best responses of Player 1 to a strategy $\sigma_0 \in \Sigma_0$ not always exist (Lemma 3). As a consequence, the *cooperative* (CSV) and *adversarial* (ASV) Stackelberg values are defined using ϵ-best responses. While strategies of Player 0 to achieve CSV always exist as shown in [13], we show that it is not the case for ASV (Theorem 4). The ASV can only be approached in general and memory may be necessary to play optimally or ϵ-optimally in adversarial Stackelberg games for the mean-payoff function (Theorem 4). We also provide results for related algorithmic

[1] As we will see later in the paper, sometimes, best-responses are not guaranteed to exist. In such cases, we need to resort to weaker notions such as ϵ-best-responses. We leave those technical details for later in the paper.

problems. We provide a notion of witness for proving that the ASV is (strictly) above some threshold (Theorem 5), and it is the basis for an NP algorithm to solve the threshold problem (Theorem 7). Finally, we show how the ASV can be computed effectively (Theorem 12).

Second, we consider the discounted sum function. In that case, best responses of Player 1 to strategies $\sigma_0 \in \Sigma_0$ of Player 0 always exist (Lemma 13). The CSV and ASV are directly based on best-responses in that case. Then we draw a link between the *target discounted sum problem* and the CSV threshold problem (Lemma 15). The target discounted sum problem has been studied recently in [1], left open there for the general case and shown to be related to other open problems in piecewise affine maps and the representation of numbers in nonintegral bases. As a consequence, we introduce a relaxation of the threshold problems for both CSV and ASV in the form of gap problems (or promised problems as defined in [12]). We provide algorithms to solve those gap problems (Theorem 17) both for CSV and ASV. Finally, we prove NP-hardness for the gap problems both for CSV and ASV (Theorem 18).

Closely related work. The notions of cooperative and adversarial synthesis have been introduced in [11, 17], and further studied in [7, 10]. Those two notions are closely related to our notion of cooperative and adversarial Stackelberg value respectively. The games that are considered in those papers are infinite duration games played on graphs but they consider Boolean ω-regular payoff functions or finite range ω-regular payoff functions. Neither the mean-payoff function nor the discounted sum payoff function are ω-regular, and thus they are not considered in [11, 17]. The ω-regularity of the payoff functions that they consider is central to their techniques: they show how to reduce their problems to problems on tree automata and strategy logic. Those reductions cannot be used for payoff functions that are *not* ω-regular functions and we need specific new techniques to solve our problems.

In [13, 15], the cooperative scenario for Stackelberg game is studied for mean-payoff and discounted sum respectively. Their results are sufficient to solve most of the relevant questions on the CSV but not for ASV. Indeed, the techniques that are used for CSV are closely related to the techniques that are used to reason on Nash equilibria and build on previous works [4] which in turn reduce to algorithmic solutions for zero-sum one dimensional mean-payoff (or discounted sum games). For the ASV in the context of the mean-payoff function, we have to use more elaborate multi-dim. mean-payoff games and a notion of Pareto curve adapted from [3]. Additionally, we provide new results on the CSV for the discounted sum function. First, our reduction that relates the target discounted sum problem to the CSV is new and gives additional explanations why the CSV is difficult to solve and not solved in the general case in [15]. Second, while we also leave the general problem open here, we show how to solve the gap problems related to both CSV and ASV. Finally, the authors of [14] study *incentive equilibria* for multi-player mean-payoff games. This work is an extension of their previous work [13] and again concentrates on CSV and does not consider ASV.

Structure of the paper. In Sect. 2, we introduce the necessary preliminaries for our definitions and developments. In Sect. 3, we consider the adversarial Stackelberg value for the mean-payoff function. In Sect. 4, we present our results for the discounted sum function.

2 Preliminaries and notations

Arenas. A (bi-weighted) *arena* $\mathcal{A} = (V, E, \langle V_0, V_1 \rangle, w_0, w_1)$ consists of a finite set V of vertices, a set $E \subseteq V \times V$ of edges such that for all $v \in V$ there exists $v' \in V$ such that $(v, v') \in E$, a partition $\langle V_0, V_1 \rangle$ of V, where V_0 (resp. V_1) is the set of vertices for Player 0 (resp. Player 1), and two edge weight functions $w_0 : E \mapsto \mathbb{Z}, w_1 : E \mapsto \mathbb{Z}$. In the sequel, we denote the maximum absolute value of a weight in $\mathcal{A}$ by W. As arenas are directed weighted graphs, we use, sometimes without recalling the details, the classical vocabulary for directed graphs. E.g., a set of vertices $S \subseteq V$ is a strongly connected component of the arena (SCC for short), if for all $s_1, s_2 \in S$, there exists a path from s_1 to s_2 and a path from s_2 to s_1.

Plays and histories. A *play* in $\mathcal{A}$ is an infinite sequence of vertices $\pi = \pi_0\pi_1 \cdots \in V^\omega$ such that for all $k \in \mathbb{N}, (\pi_k, \pi_{k+1}) \in E$. We denote by $\mathsf{Plays}_\mathcal{A}$ the set of plays in $\mathcal{A}$, omitting the subscript $\mathcal{A}$ when the underlying arena is clear from the context. Given $\pi = \pi_0\pi_1 \cdots \in \mathsf{Plays}_\mathcal{A}$ and $k \in \mathbb{N}$, the prefix $\pi_0\pi_1 \dots \pi_k$ of π (resp. suffix $\pi_k\pi_{k+1} \dots$ of π) is denoted by $\pi_{\leq k}$ (resp. $\pi_{\geq k}$). An *history* in $\mathcal{A}$ is a (non-empty) prefix of a play in $\mathcal{A}$. The length $|h|$ of an history $h = \pi_{\leq k}$ is the number $|h| = k$ of its edges. We denote by $\mathsf{Hist}_\mathcal{A}$ the set of histories in $\mathcal{A}$, $\mathcal{A}$ is omitted when clear from the context. Given $i \in \{0, 1\}$, the set $\mathsf{Hist}^i_\mathcal{A}$ denotes the set of histories such that their last vertex belongs to V_i. We denote the last vertex of a history h by $\mathsf{last}(h)$. We write $h \leq \pi$ whenever h is a prefix of π. A play π is called a lasso if it is obtained as the concatenation of a history h concatenated with the infinite repetition of another history l, i.e. $\pi = h \cdot l^\omega$ with $h, l \in \mathsf{Hist}_\mathcal{A}$ (notice that l is not necessary a simple cycle). The *size* of a lasso $h \cdot l^\omega$ is defined as $|h \cdot l|$. Given a vertex $v \in V$ in the arena $\mathcal{A}$, we denote by $Succ(v) = \{v'|(v, v') \in E\}$ the set of successors of v and by $Succ^*$ its transitive closure.

Games. A *game* $\mathcal{G} = (\mathcal{A}, \langle \mathsf{Val}_0, \mathsf{Val}_1 \rangle)$ consists of a bi-weighted arena $\mathcal{A}$, a value function $\mathsf{Val}_0 : \mathsf{Plays}_\mathcal{A} \mapsto \mathbb{R}$ for Player 0 and a value function $\mathsf{Val}_1 : \mathsf{Plays}_\mathcal{A} \mapsto \mathbb{R}$ for Player 1. In this paper, we consider the classical *mean-payoff* and *discounted-sum* value functions. Both are played in bi-weighted arenas.

In a *mean-payoff* game $\mathcal{G} = (\mathcal{A}, \langle \mathsf{MP}_0, \mathsf{MP}_1 \rangle)$ the payoff functions $\mathsf{MP}_0, \mathsf{MP}_1$ are defined as follows. Given a play $\pi \in \mathsf{Plays}_\mathcal{A}$ and $i \in \{0, 1\}$, the payoff $\underline{\mathsf{MP}}_i(\pi)$ is given by $\underline{\mathsf{MP}}_i(\pi) = \liminf_{k\to\infty} \frac{1}{k} w_i(\pi_{\leq k})$, where the weight $w_i(h)$ of an history $h \in \mathsf{Hist}$ is the sum of the weights assigned by w_i to its edges. In our definition of the mean-payoff, we have used $\liminf$, we will also need the $\limsup$ case for technical reasons. Here is the formal definition together with its notation: $\overline{\mathsf{MP}}_i(\pi) = \limsup_{k\to\infty} \frac{1}{k} w_i(\pi_{\leq k})$

For a given discount factor $0 < \lambda < 1$, a *discounted sum* game is a game $\mathcal{G} = (\mathcal{A}, \langle \mathsf{DS}^\lambda_0, \mathsf{DS}^\lambda_1 \rangle)$ where the payoff functions $\mathsf{DS}^\lambda_0, \mathsf{DS}^\lambda_1$ are defined as follows. Given a play $\pi \in \mathsf{Plays}_\mathcal{A}$ and $i \in \{0, 1\}$, the payoff $\mathsf{DS}^\lambda_\mathsf{i}(\pi)$ is defined as $\mathsf{DS}^\lambda_i(\pi) = \sum_{k=0}^{\infty} \lambda^k w_i(\pi_k, \pi_{k+1})$.

Strategies and payoffs. A strategy for Player $i \in \{0, 1\}$ in a game $\mathcal{G} = (\mathcal{A}, \langle \mathsf{Val}_0, \mathsf{Val}_1 \rangle)$ is a function $\sigma : \mathsf{Hist}^i_\mathcal{A} \mapsto V$ that maps histories ending with a vertex $v \in V_i$ to a successor of v. The set of all strategies of Player $i \in \{0, 1\}$ in the game $\mathcal{G}$ is denoted $\Sigma_i(\mathcal{G})$, or Σ_i when $\mathcal{G}$ is clear from the context.

A strategy has memory M if it can be realized as the output of a finite state machine with M states. A memoryless (or positional) strategy is a strategy with memory 1, that is, a function that only depends on the last element of the given partial play. We note Σ^{ML}_i the set of memoryless strategies of Player i, and Σ^{FM}_i its set of finite memory strategies. A *profile* is a pair of strategies $\bar{\sigma} = (\sigma_0, \sigma_1)$, where $\sigma_0 \in \Sigma_0(\mathcal{G})$ and $\sigma_1 \in \Sigma_1(\mathcal{G})$. As we consider games with perfect information and deterministic transitions, any profile $\bar{\sigma}$ yields, from any history h, a unique *play* or *outcome*, denoted $\mathsf{Out}_h(\mathcal{G}, \bar{\sigma})$. Formally, $\mathsf{Out}_h(\mathcal{G}, \bar{\sigma})$ is the play π such that $\pi_{\leq |h|-1} = h$ and $\forall k \geq |h| - 1$ it holds that $\pi_{k+1} = \sigma_i(\pi_{\leq k})$ if $\pi_k \in V_i$. The set of outcomes (resp. histories) compatible with a strategy $\sigma \in \Sigma_{i\in\{0,1\}}(\mathcal{G})$ after a history h is $\mathsf{Out}_h(\mathcal{G}, \sigma) = \{\pi \mid \exists \sigma' \in \Sigma_{1-i}(\mathcal{G})$ such that $\pi = \mathsf{Out}_h(\mathcal{G}, (\sigma, \sigma'))\}$ (resp. $\mathsf{Hist}_h(\sigma) = \{h' \in \mathsf{Hist}(\mathcal{G}) \mid \exists \pi \in \mathsf{Out}_h(\mathcal{G}, \sigma), n \in \mathbb{N} : h' = \pi_{\leq n}\}$.

Each outcome π in $\mathcal{G} = (\mathcal{A}, \langle \mathsf{Val}_0, \mathsf{Val}_1\rangle)$ yields a payoff $\mathsf{Val}(\pi) = (\mathsf{Val}_0(\pi), \mathsf{Val}_1(\pi))$, where $\mathsf{Val}_0(\pi)$ is the payoff for Player 0 and $\mathsf{Val}_1(\pi)$ is the payoff for Player 1. We denote by $\mathsf{Val}(h, \bar{\sigma}) = \mathsf{Val}(\mathsf{Out}_h(\mathcal{G}, \bar{\sigma}))$ the payoff of a profile of strategies $\bar{\sigma}$ after a history h.

Usually, we consider game instances such that players start to play at a fixed vertex v_0. Thus, we call an initialized game a pair $(\mathcal{G}, v_0)$, where $\mathcal{G}$ is a game and $v_0 \in V$ is the initial vertex. When the initial vertex v_0 is clear from context, we speak directly of $\mathcal{G}, \mathsf{Out}(\mathcal{G}, \bar{\sigma}), \mathsf{Out}(\mathcal{G}, \sigma)$, $\mathsf{Val}(\bar{\sigma})$ instead of $(\mathcal{G}, v_0)$, $\mathsf{Out}_{v_0}(\mathcal{G}, \bar{\sigma})$, $\mathsf{Out}_{v_0}(\mathcal{G}, \sigma)$,$\mathsf{Val}(v_0, \bar{\sigma})$. We sometimes simplify further the notation omitting also $\mathcal{G}$, when the latter is clear from the context.

Best-responses and adversarial value in zero-sum games. Let $\mathcal{G} = (\mathcal{A}, \langle \mathsf{Val}_0, \mathsf{Val}_1\rangle)$ be a $(\mathsf{Val}_0, \mathsf{Val}_1)$-game on the bi-weighted arena $\mathcal{A}$. Given a strategy σ_0 for Player 0, we define two sets of strategies for Player 1. His *best-responses* to σ_0, noted $\mathsf{BR}_1(\sigma_0)$, and defined as:

$$\{\sigma_1 \in \Sigma_1 \mid \forall v \in V \cdot \forall \sigma_1' \in \Sigma_1 : \mathsf{Val}_1(\mathsf{Out}_v(\sigma_0, \sigma_1)) \geq \mathsf{Val}_1(\mathsf{Out}_v(\sigma_0, \sigma_1'))\}\,.$$

And his *ϵ-best-responses* to σ_0, for $\epsilon > 0$, noted $\mathsf{BR}_1^{\epsilon}(\sigma_0)$, and defined as:

$$\{\sigma_1 \in \Sigma_1 \mid \forall v \in V \cdot \forall \sigma_1' \in \Sigma_1 : \mathsf{Val}_1(\mathsf{Out}_v(\sigma_0, \sigma_1) \geq \mathsf{Val}_1(\mathsf{Out}_v(\sigma_0, \sigma_1')) - \epsilon\}\,.$$

We also introduce notations for zero-sum games (that are needed as intermediary steps in our algorithms). The adversarial value that Player 1 can enforce in the game $\mathcal{G}$ from vertex v as: $\mathsf{WCV}\ _1(v) = \sup_{\sigma_1 \in \Sigma_1} \inf_{\sigma_0 \in \Sigma_0} \mathsf{Val}_1(\mathsf{Out}_v(\sigma_0, \sigma_1))$. Let $\mathcal{A}$ be an arena, $v \in V$ one of its states, and $\mathcal{O} \subseteq \mathsf{Plays}_{\mathcal{A}}$ be a set of plays (called objective), then we write $\mathcal{A}, v \models \ll i \gg \mathcal{O}$, if $\exists \sigma_i \in \Sigma_i \cdot \forall \sigma_{1-i} \in \Sigma_{1-i} : \mathsf{Out}_v(\mathcal{A}, (\sigma, \sigma')) \in \mathcal{O}$, for $i \in \{0, 1\}$. Here the underlying interpretation is zero-sum: Player i wants to force an outcome in $\mathcal{O}$ and Player $1 - i$ has the opposite goal. All the zero-sum games we consider in this paper are *determined* meaning that for all $\mathcal{A}$, for all objectives $\mathcal{O} \subseteq \mathsf{Plays}_{\mathcal{A}}$ we have that: $\mathcal{A}, v \models \ll i \gg \mathcal{O}$ iff $\mathcal{A}, v \not\models \ll 1 - i \gg \mathsf{Plays}_{\mathcal{A}} \setminus \mathcal{O}$.

Convex hull and F_{min}. First, we need som e additional notations and vocabulary related to linear algebra. Given a finite set of d-dim. vectors $X \subset \mathbb{R}^d$, we note the set of all their convex combinations as $\mathsf{CH}(X) = \{v \mid v = \sum_{x \in X} \alpha_x \cdot x \wedge \forall x \in X : \alpha_x \in [0, 1] \wedge \sum_{x \in X} \alpha_x = 1\}$, this set is called the *convex hull* of X. We also need the following additional, and less standard notions, introduced in [5]. Given a finite set of d-dim. vectors $X \subset \mathbb{R}^d$, let $f_{\mathsf{min}}(X)$ be the vector $v = (v_1, v_2, \ldots, v_d)$ where $v_i = \min\ \{c \mid \exists x \in X : x_i = c\}$, i.e. the vector v is the pointwise minimum of the vectors in X. Let $S \subseteq \mathbb{R}^d$, then $F_{\mathsf{min}}(S) = \{f_{\mathsf{min}}(P) \mid P$ is a finite subset of $S\}$. The following proposition expresses properties of the $F_{\mathsf{min}}(S)$ operator that are useful for us in the sequel. The interested reader will find more results about the F_{min} operator in [5].

▶ **Proposition 1.** *For all sets $S \subseteq \mathbb{R}^d$, for all $x \in F_{min}(S)$, there exists $y \in S$ such that $x \leq y$. If S is a closed bounded set then $F_{min}(S)$ is also a closed bounded set.*

In the sequel, we also use formulas of the theory of the reals with addition and order, noted $\langle \mathbb{R}, +, \leq \rangle$, in order to define subsets of $\mathbb{R}^n$. This theory is decidable and admits effective quantifier elimination [8].

3 Adversarial Stackelberg value for mean-payoff games

Mean-payoffs induced by simple cycles. Given a play $\pi \in \mathsf{Plays}_{\mathcal{A}}$, we note $\mathsf{inf}(\pi)$ the set of vertices v that appear infinitely many times along π, i.e. $\mathsf{inf}(\pi) = \{v \mid \forall i \in \mathbb{N} \cdot \exists j \geq i : v = \pi_j\}$. It is easy to see that $\mathsf{inf}(\pi)$ is an SCC in the underlying graph of the arena $\mathcal{A}$. A *cycle* c is a sequence of edges that starts and stops in a given vertex v, it is *simple* if it does not contain any other repetition of vertices. Given an SCC S, we write $\mathbb{C}(S)$ for the set of simple cycles inside S. Given a simple cycle c, for $i \in \{0,1\}$, let $\mathsf{MP}_i(c) = \frac{w_i(c)}{|c|}$ be the mean of w_i weights along edges in the simple cycle c, and we call the pair $(\mathsf{MP}_0(c), \mathsf{MP}_1(c))$ the *mean-payoff coordinate* of the cycle c. We write $\mathsf{CH}(\mathbb{C}(S))$ for the convex-hull of the set of mean-payoff coordinates of simple cycles of S. The following theorem relates the d-dim. mean-payoff values of infinite plays and the d-dim. mean-payoff of simple cycles in the arena.

▶ **Theorem 2** ([5]). *Let S be an SCC in the arena $\mathcal{A}$, the following two properties hold: (i) for all $\pi \in \mathsf{Plays}_{\mathcal{A}}$, if $\mathit{inf}(\pi) \subseteq S$ then $(\underline{\mathit{MP}}_0(\pi), \underline{\mathit{MP}}_1(\pi)) \in F_{\mathit{min}}(\mathit{CH}(\mathbb{C}(S)))$ (ii) for all $(x,y) \in F_{\mathit{min}}(\mathit{CH}(\mathbb{C}(S)))$, there exists $\pi \in \mathsf{Plays}_{\mathcal{A}}$ such that $\mathit{inf}(\pi) = S$ and $(\underline{\mathit{MP}}_0(\pi), \underline{\mathit{MP}}_1(\pi)) = (x,y)$. Furthermore, the set $F_{\mathit{min}}(\mathit{CH}(\mathbb{C}(S)))$ is effectively expressible in $\langle \mathbb{R}, +, \leq \rangle$.*

In the sequel, we denote by $\Phi_S(x,y)$ the formula with two free variables in $\langle \mathbb{R}, +, \leq \rangle$ such that for all $(u,v) \in \mathbb{R}^2$, $(u,v) \in F_{\mathsf{min}}(\mathsf{CH}(\mathbb{C}(S)))$ if and only if $\Phi_S(x,y)[x/u, y/v]$ is true.

On the existence of best-responses for MP. We start the study of mean-payoff games with some considerations about the existence of best-responses and ϵ-best-responses for Player 1 to strategies of Player 0.

▶ **Lemma 3.** *There is a mean-payoff game $\mathcal{G}$ and a strategy $\sigma_0 \in \Sigma_0(\mathcal{G})$ such that $\mathit{BR}_1(\sigma_0) = \varnothing$. For all mean-payoff games $\mathcal{G}$ and finite memory strategies $\sigma_0 \in \Sigma_0^{\mathit{FM}}(\mathcal{G})$, $\mathit{BR}_1(\sigma_0) \neq \varnothing$. For all mean-payoff games $\mathcal{G}$, for all strategies $\sigma_0 \in \Sigma_0(\mathcal{G})$, for all $\epsilon > 0$, $\mathit{BR}_1^{\epsilon}(\sigma_0) \neq \varnothing$.*

Proof sketch - full proof in the full version [9]. First, in the arena of Fig. 1, we consider the strategy of Player 0 that plays the actions c and d with a frequency that is equal to $1 - \frac{1}{k}$ for c and $\frac{1}{k}$ for d where k is the number of times that Player 1 has played a in state 1 before sending the game to state 2. We claim that there is no best response of Player 1 to this strategy of Player 0. Indeed, taking a one more time before going to state 2 is better for Player 1.

Second, if Player 0 plays a finite memory strategy, then a best response for Player 1 is an optimal path for the mean-payoff of Player 1 in the finite graph obtained as the product of the original game arena with the finite state strategy of Player 0. Optimal mean-payoff paths are guaranteed to exist [16].

Finally, the existence of ϵ-best responses for $\epsilon > 0$, is guaranteed by an analysis of the infinite tree obtained as the unfolding of the game arena with the (potentially infinite memory) strategy of Player 1. Branches of this tree witness responses of Player 1 to the strategy of Player 0. The supremum of the values of those branches for Player 1 is always approachable to any $\epsilon > 0$. ◀

According to Lemma 3, the set of best-responses of Player 1 to a strategy of Player 0 can be empty. As a consequence, we need to use the notion of ϵ-best-responses (which are always guaranteed to exist) when we define the adversarial Stackelberg value:

$$\mathsf{ASV}(\sigma_0)(v) = \sup_{\epsilon \geq 0 \ \mid \ \mathsf{BR}_1^{\epsilon}(\sigma_0) \neq \varnothing} \ \inf_{\sigma_1 \in \mathsf{BR}_1^{\epsilon}(\sigma_0)} \underline{\mathsf{MP}}_0(\mathsf{Out}_v(\sigma_0, \sigma_1)) \text{ and } \mathsf{ASV}(v) = \sup_{\sigma_0 \in \Sigma_0} \mathsf{ASV}(\sigma_0)(v)$$

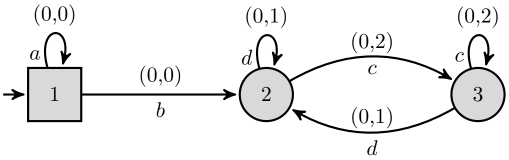


Figure 1 A mean-payoff game in which there exists a Player 0's strategy σ_0 such that $\mathsf{BR}_1(\sigma_0) = \varnothing$.

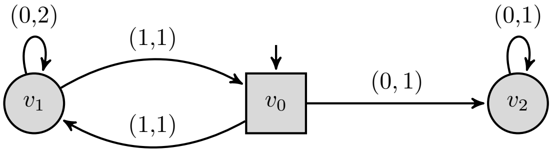


Figure 2 In this game, $\mathsf{ASV}(v_0) = 1$ but there is no Player 0 strategy to achieve this value.

We note that when best-responses to a strategy σ_0 exist, then as expected the following equality holds, because $\mathsf{BR}_1(\sigma_0) = \mathsf{BR}_1^0(\sigma_0)$ and $\mathsf{BR}_1^{\epsilon_1}(\sigma_0) \subseteq \mathsf{BR}_1^{\epsilon_2}(\sigma_0)$ for all $\epsilon_1 \leq \epsilon_2$, ϵ should be taken equal to 0:

$$\mathsf{ASV}(\sigma_0)(v) = \sup_{\epsilon \geq 0 \ \mid \ \mathsf{BR}_1^{\epsilon}(\sigma_0) \neq \varnothing} \ \inf_{\sigma_1 \in \mathsf{BR}_1^{\epsilon}(\sigma_0)} \underline{\mathsf{MP}}_0(\mathsf{Out}_v(\sigma_0, \sigma_1)) = \inf_{\sigma_1 \in \mathsf{BR}_1(\sigma_0)} \underline{\mathsf{MP}}_0(\mathsf{Out}_v(\sigma_0, \sigma_1))$$

Finally, we note that changing the sup over ϵ into an inf in our definition, we get the classical notion of worst-case value in which the rationality of Player 1 and his payoff are ignored. We also recall the definition of CSV, the cooperative Stackelberg value:

$$\mathsf{CSV}(\sigma_0)(v) = \sup_{\epsilon \geq 0 \ \mid \ \mathsf{BR}_1^{\epsilon}(\sigma_0) \neq \varnothing} \ \sup_{\sigma_1 \in \mathsf{BR}_1^{\epsilon}(\sigma_0)} \underline{\mathsf{MP}}_0(\mathsf{Out}_v(\sigma_0, \sigma_1)) \text{ and } \mathsf{CSV}(v) = \sup_{\sigma_0 \in \Sigma_0} \mathsf{CSV}(\sigma_0)(v)$$

The interest reader is referred to [13] for an in-depth treatment of this value.

The adversarial Stackelberg value may not be achievable. In contrast with results in [13] that show that CSV can always be achieved, the following statement expresses the fact that the adversarial Stackelberg value may not be achievable but it can always be approximated by a strategy of Player 0.

▶ **Theorem 4.** *There exists a mean-payoff game $\mathcal{G}$ in which Player 0 has no strategy which enforces the adversarial Stackelberg value. Furthermore, for all mean-payoff games $\mathcal{G}$, for all vertices $v \in V$, for all $\epsilon > 0$, there exists a strategy $\sigma_0 \in \Sigma_0$ such that $\mathsf{ASV}(\sigma_0)(v) > \mathsf{ASV}(v) - \epsilon$. Memory is needed to achieve high ASV.*

Proof sketch - full proof in the full version [9]. First, consider the game depicted in Fig 2. In this game, $\mathsf{ASV}(v_0) = 1$ and it is not achievable. Player 0 needs to ensure that Player 1 does not take the transition from v_0 to v_2 otherwise she gets a payoff of 0. To ensure this, Player 0 needs to choose a strategy (that cycles within $\{v_0, v_1\}$) and that gives to Player 1 at least $1 + \epsilon$ with $\epsilon > 0$. Such strategies gives $1 - \epsilon$ to Player 0, and the value 1 cannot be reached.

Second, by definition of the ASV, the value is obtained as the sup over all strategies of Player 0. As a consequence, ϵ-optimal strategies (for $\epsilon > 0$) exist. ◀

Witnesses for the ASV. Given a mean-payoff game $\mathcal{G}$, we associate with each vertex v, the following set of pairs of real numbers: $\Lambda(v) = \{(c,d) \in \mathbb{R}^2 \mid v \models \ll 1 \gg \underline{\mathsf{MP}}_0 \leq c \wedge \underline{\mathsf{MP}}_1 \geq d\}$. We say that v is (c,d)-bad if $(c,d) \in \Lambda(v)$. Let $c' \in \mathbb{R}$. A play π in $\mathcal{G}$ is called a (c',d)-witness of $\mathsf{ASV}(v) > c$ if it starts from v, $(\underline{\mathsf{MP}}_0(\pi), \underline{\mathsf{MP}}_1(\pi)) = (c', d)$, $c' > c$ and π does not contain any (c,d)-bad vertex. A play π is called a witness of $\mathsf{ASV}(v) > c$ if it is a (c',d)-witness of $\mathsf{ASV}(v) > c$ for some c', d. The following theorem justifies the name witness.

▶ **Theorem 5.** *Let $\mathcal{G}$ be a mean-payoff game and v be one of its vertices. $\mathsf{ASV}(v) > c$ iff there exists a play π in $\mathcal{G}$ such that π is a witness of $\mathsf{ASV}(v) > c$.*

Proof. *From right to left.* Assume the existence of a (c',d)-witness π and let us show that there exists a strategy σ_0 which forces $\mathsf{ASV}(\sigma_0)(v) > c$. We define σ_0 as follows:

1. for all histories $h \leq \pi$ such that *last* (h) belongs to Player 0, $\sigma_0(h)$ follows π.
2. for all histories $h \not\leq \pi$ where there has been a deviation from π by Player 1, we assume that Player 0 switches to a strategy that we call *punishing*. This strategy is defined as follows. In the subgame after history h' where h' is a first deviation by Player 1 from π, we know that Player 0 has a strategy to enforce the objective: $\underline{\mathsf{MP}}_0 > c \vee \underline{\mathsf{MP}}_1 < d$. This is true because π does not cross any (c,d)-bad vertex. So, we know that $h' \not\models \ll 1 \gg \underline{\mathsf{MP}}_0 \leq c \wedge \underline{\mathsf{MP}}_1 \geq d$ which entails the previous statement by determinacy of n-dimension mean-payoff games [21] (here $n = 2$).
3. for all other histories h, Player 0 can behave arbitrarily as those histories are never reached when Player 0 plays as defined in point 1 and 2 above.

Let us now establish that the strategy σ_0 satisfies $\mathsf{ASV}(\sigma_0)(v) > c$. We have to show the existence of some $\epsilon \geq 0$ such that $\mathsf{BR}_1^{\epsilon}(\sigma_0) \neq \varnothing$ and for all $\sigma_1 \in \mathsf{BR}_1^{\epsilon}(\sigma_0)$, $\underline{\mathsf{MP}}_0(\mathsf{Out}_v(\sigma_0,\sigma_1)) > c$ holds. For that, we consider two subcases:

1. $\sup_{\sigma_1} \underline{\mathsf{MP}}_1(\mathsf{Out}_v(\sigma_0,\sigma_1)) = d = \underline{\mathsf{MP}}_1(\pi)$. This means that any strategy σ_1 of Player 1 that follows π is for $\epsilon = 0$ a best-response to σ_0. Now let us consider any strategy $\sigma_1 \in \mathsf{BR}_1^0(\sigma_0)$. Clearly, $\pi' = \mathsf{Out}_v(\sigma_0,\sigma_1)$ is such that $\underline{\mathsf{MP}}_1(\pi') \geq d$. If $\pi' = \pi$, we have that $\underline{\mathsf{MP}}_0(\pi') = c' > c$. If $\pi' \neq \pi$, then when π' deviates from π, we know that σ_0 behaves as the punishing strategy and so we have that $\underline{\mathsf{MP}}_0(\pi') > c \vee \underline{\mathsf{MP}}_1(\pi') < d$. But as $\sigma_1 \in \mathsf{BR}_1^0(\sigma_0)$, we conclude that $\underline{\mathsf{MP}}_1(\pi') \geq d$, and so in turn, we obtain that $\underline{\mathsf{MP}}_0(\pi') > c$.
2. $\sup_{\sigma_1} \underline{\mathsf{MP}}_1(\mathsf{Out}_v(\sigma_0,\sigma_1)) = d' > d$. Let $\epsilon > 0$ be such that $d' - \epsilon > d$. By Lemma 3, $\mathsf{BR}_1^{\epsilon}(\sigma_0) \neq \varnothing$. Let us now characterize the value that Player 0 receives against any strategy $\sigma_1 \in \mathsf{BR}_1^{\epsilon}(\sigma_0)$. First, if σ_1 follows π then Player 0 receives $c' > c$. Second, if σ_1 deviates from π, Player 1 receives at least $d' - \epsilon > d$. But by definition of σ_0, we know that if the play deviates from π then Player 0 applies her punishing strategy. Then we know that the outcome satisfies $\underline{\mathsf{MP}}_0 > c \vee \underline{\mathsf{MP}}_1 < d$. But as $d' - \epsilon > d$, we must conclude that the outcome π' is such that $\underline{\mathsf{MP}}_0(\pi') > c$.

From left to right. Let σ_0 such that $\mathsf{ASV}(\sigma_0)(v) > c$. Then by the equivalence shown in the proof of Theorem 4, we know that

$$\exists \epsilon \geq 0 : \mathsf{BR}_1^{\epsilon}(\sigma_0) \neq \varnothing \wedge \forall \sigma_1 \in \mathsf{BR}_1^{\epsilon}(\sigma_0) : \mathsf{Out}_v(\sigma_0,\sigma_1) > c \tag{1}$$

Let ϵ^* be a value for ϵ that makes eq. (1) true. Take any $\sigma_1 \in \mathsf{BR}_1^{\epsilon^*}(\sigma_0)$ and consider $\pi = \mathsf{Out}_v(\sigma_0,\sigma_1)$. We will show that π is a witness for $\mathsf{ASV}(v) > c$.

We have that $\underline{\mathsf{MP}}_0(\pi) > c$. Let $d_1 = \underline{\mathsf{MP}}_1(\pi)$ and consider any $\pi' \in \mathsf{Out}_v(\sigma_0)$. Clearly if $\underline{\mathsf{MP}}_1(\pi') \geq d_1$ then there exists $\sigma_1' \in \mathsf{BR}_1^{\epsilon^*}(\sigma_0)$ such that $\pi' = \mathsf{Out}_{v_0}(\sigma_0,\sigma_1')$ and we conclude that $\underline{\mathsf{MP}}_0(\pi') > c$. So all deviations of Player 1 w.r.t. π against σ_0 are either giving him a $\underline{\mathsf{MP}}_1$ which is less than d_1 or it gives to Player 0 a $\underline{\mathsf{MP}}_0$ which is larger than c. So π is a $(\underline{\mathsf{MP}}_0(\pi), \underline{\mathsf{MP}}_1(\pi))$-witness for $\mathsf{ASV}(v) > c$ as we have shown that π never crosses an $(c, \underline{\mathsf{MP}}_1(\pi))$-bad vertex, and we are done. ◀

The following statement is a direct consequence of the proof of the previous theorem.

▶ **Corollary 6.** *If π is a witness for $\mathsf{ASV}(v) > c$ then all π' such that: $\pi'(0) = v$, the set of vertices visited along π and π' are the same, and $\underline{\mathsf{MP}}_0(\pi') \geq \underline{\mathsf{MP}}_0(\pi)$ and $\underline{\mathsf{MP}}_1(\pi') \geq \underline{\mathsf{MP}}_1(\pi)$, are also witnesses for $\mathsf{ASV}(v) > c$.*

Small witnesses and NP membership. Here, we refine Theorem 5 to establish membership of the threshold problem to NP.

▶ **Theorem 7.** *Given a mean-payoff game $\mathcal{G}$, a vertex v and a rational value $c \in \mathbb{Q}$, it can be decided in nondeterministic polynomial time if $\mathsf{ASV}(v) > c$.*

Proof of Thm. 7 relies on the existence of small witnesses established in the following lemma:

▶ **Lemma 8.** *Given a mean-payoff game $\mathcal{G}$, a vertex v and $c \in \mathbb{Q}$, $\mathsf{ASV}(v) > c$ if and only if there exists an SCC reachable from v that contains two simple cycles ℓ_1, ℓ_2 such that: (i) there exist $\alpha, \beta \in \mathbb{Q}$ such that $\alpha \cdot w_0(\ell_1) + \beta \cdot w_0(\ell_2) = c' > c$, and $\alpha \cdot w_1(\ell_1) + \beta \cdot w_1(\ell_2) = d$ (ii) there is no (c, d)-bad vertex v' along the path from v to ℓ_1, the path from ℓ_1 to ℓ_2, and the path from ℓ_2 to ℓ_1.*

Proof sketch - full proof in the full version [9]. Theorem 5 establishes the existence of a witness π for $\mathsf{ASV}(v) > c$. In turn, we show here that the existence of such a π can be established by a polynomially checkable witness composed of the following elements. First, a simple path from v to the SCC in which π gets trapped in the long run, (ii) two simple cycles (that can produce the value (c', d) of π) by looping at the right frequencies along the two cycles. Indeed, $(\underline{\mathsf{MP}}_0(\pi), \underline{\mathsf{MP}}_1(\pi))$ only depends on the suffix in the SCC in which it gets trapped. Furthermore, by Theorem 2, Proposition 1 and Corollary 6, we know that the mean-payoff of witnesses can be obtained as the convex combination of the mean-payoff coordinates of simple cycles, and 3 such simple cycles are sufficient by the Carathéodory baricenter theorem. A finer analysis of the geometry of the sets allows us to go to 2 cycles only (see the full proof in [9]). ◀

Proof of Theorem 7. According to Lemma 8, the nondeterministic algorithm that establishes the membership to NP guesses a reachable SCC together with the two simple cycles ℓ_1 and ℓ_2, and parameters α and β. Additionally, for each vertex v' that appears along the paths to reach the SCC, on the simple cycles ℓ_1 and ℓ_2, and to connect those simple cycles, the algorithm guesses a memoryless strategy $\sigma_0^{v'}$ for Player 0 that establishes $v' \not\models \ll 1 \gg \underline{\mathsf{MP}}_0 \leq c \wedge \underline{\mathsf{MP}}_1 \geq d$ which means by determinacy of multi-dimensional mean-payoff games, that $v' \models \ll 0 \gg \underline{\mathsf{MP}}_0 > c \vee \underline{\mathsf{MP}}_1 < d$. The existence of those memoryless strategy is established in Propositions 20 and 21 in the full version [9] (in turn those propositions rely on results from [21]). Those memoryless strategies are checkable in PTime [16]. ◀

Computing the ASV in mean-payoff games. The previous theorems establish the existence of a notion of witness for the adversarial Stackelberg value in non zero-sum two-player mean-payoff games. This notion of witness can be used to decide the threshold problem in NPTIME. We now show how to use this notion to effectively compute the ASV. This algorithm is also based on the computation of an effective representation, for each vertex v of the game graph, of the infinite set of pairs $\Lambda(v)$. The following lemma expresses that a symbolic representation of this set of pairs can be constructed effectively. This result is using techniques that have been introduced in [3].

▶ **Lemma 9.** *Given a bi-weighted game graph $\mathcal{G}$ and a vertex $v \in V$, we can effectively construct a formula $\Psi_v(x,y)$ of $\langle \mathbb{R}, +, \leq \rangle$ with two free variables such that $(c,d) \in \Lambda(v)$ if and only if the formula $\Psi_v(x,y)[x/c, y/d]$ is true.*

Extended graph game. From the graph game $\mathcal{G} = (V, E, w_0, w_1)$, we construct the extended graph game $\mathcal{G}^{\mathsf{ext}} = (V^{\mathsf{ext}}, E^{\mathsf{ext}}, w_0^{\mathsf{ext}}, w_1^{\mathsf{ext}})$, whose vertices and edges are defined as follows. The set of vertices is $V^{\mathsf{ext}} = V \times 2^V$. With an history h in $\mathcal{G}$, we associate a vertex in $\mathcal{G}^{\mathsf{ext}}$ which is a pair (v, P), where $v = \mathit{last}\,(h)$ and P is the set of the vertices traversed along h. Accordingly the set of edges and the weight functions are defined as $E^{\mathsf{ext}} = \{((v,P),(v',P')) \mid (v,v') \in E \wedge P' = P \cup \{v'\}\}$ and $w_i^{\mathsf{ext}}((v,P),(v',P')) = w_i((v,v'))$, for $i \in \{0,1\}$. Clearly, there exists a bijection between the plays π in $\mathcal{G}$ and the plays π^{ext} in $\mathcal{G}^{\mathsf{ext}}$ which start in vertices of the form $(v,\{v\})$, i.e. π^{ext} is mapped to the play π in $\mathcal{G}$ that is obtained by erasing the second dimension of its vertices.

▶ **Proposition 10.** *For all game graph $\mathcal{G}$, the following holds:*

1. *Let π^{ext} be an infinite play in the extended graph and π be its projection into the original graph $\mathcal{G}$ (over the first component of each vertex) , the following properties hold: (i) For all $i < j$: if $\pi^{\mathsf{ext}}(i) = (v_i, P_i)$ and $\pi^{\mathsf{ext}}(j) = (v_j, P_j)$ then $P_i \subseteq P_j$. (ii) $\underline{\mathsf{MP}}_i(\pi^{\mathsf{ext}}) = \underline{\mathsf{MP}}_i(\pi)$, for $i \in \{0,1\}$.*
2. *The unfolding of $\mathcal{G}$ from v and the unfolding of $\mathcal{G}^{\mathsf{ext}}$ from $(v,\{v\})$ are isomorphic, and so $\mathsf{ASV}(v) = \mathsf{ASV}(v,\{v\})$.*

By the first point of the latter proposition and since the set of vertices of the graph is finite, the second component of any play π^{ext} stabilises into a set of vertices of $\mathcal{G}$ which we denote by $V^*(\pi^{\mathsf{ext}})$.

We now show how to characterize $\mathsf{ASV}(v)$ with the notion of witness introduced above and the decomposition of G^{ext} into SCC. This is formalized in the following lemma:

▶ **Lemma 11.** *For all mean-payoff games $\mathcal{G}$, for all vertices $v \in V$, let $\mathsf{SCC}^{\mathsf{ext}}(v)$ be the set of strongly-connected components in $\mathcal{G}^{\mathsf{ext}}$ which are reachable from $(v,\{v\})$, then we have*

$$\mathsf{ASV}(v) = \max_{S \in \mathsf{SCC}^{\mathsf{ext}}(v)} \sup\{c \in \mathbb{R} \mid \exists \pi^{\mathsf{ext}} : \pi^{\mathsf{ext}} \textit{ is a witness for } \mathsf{ASV}(v,\{v\}) > c \textit{ and } V^*(\pi^{\mathsf{ext}}) = S\}$$

Proof. First, we note the following sequence of equalities:

$$\begin{aligned}
&\mathsf{ASV}(v)\\
&= \sup\{c \in \mathbb{R} \mid \mathsf{ASV}(v) \geq c\}\\
&= \sup\{c \in \mathbb{R} \mid \mathsf{ASV}(v) > c\}\\
&= \sup\{c \in \mathbb{R} \mid \exists \pi : \pi \text{ is a witness for } \mathsf{ASV}(v) > c\}\\
&= \sup\{c \in \mathbb{R} \mid \exists \pi^{\mathsf{ext}} : \pi^{\mathsf{ext}} \text{ is a witness for } \mathsf{ASV}(v,\{v\}) > c\}\\
&= \max\nolimits_{S \in \mathsf{SCC}^{\mathsf{ext}}(v)} \sup\{c \in \mathbb{R} \mid \exists \pi^{\mathsf{ext}} : \pi^{\mathsf{ext}} \text{ is a witness for } \mathsf{ASV}(v,\{v\}) > c \text{ and } V^*(\pi^{\mathsf{ext}}) = S\}
\end{aligned}$$

The first two equalities are direct consequences of the definition of the supremum and that $\mathsf{ASV}(v) \in \mathbb{R}$. The third is a consequence of Theorem 5 that guarantees the existence of witnesses for strict inequalities. The fourth equality is a consequence of point 2 in Proposition 10. The last equality is the consequence of point 1 in Proposition 10. ◀

By definition of $\mathcal{G}^{\mathsf{ext}}$, for all SCC S of $\mathcal{G}^{\mathsf{ext}}$, there exists a set of vertices of $\mathcal{G}$ which we also denote by $V^*(S)$ such that any vertex of S is of the form $(v, V^*(S))$. The set of bad thresholds for S is then defined as $\Lambda^{\mathsf{ext}}(S) = \bigcup_{v \in V^*(S)} \Lambda(v)$. Applying Lemma 9, we can construct a formula $\Psi_S(x,y)$ which symbolic encodes the set $\Lambda^{\mathsf{ext}}(S)$.

Now, we are equipped to prove that $\mathsf{ASV}(v)$ is effectively computable. This is expressed by the following theorem and established in its proof.

▶ **Theorem 12.** *For all mean-payoff games $\mathcal{G}$, for all vertices $v \in V$, the value $\mathit{ASV}(v)$ can be effectively expressed by a formula ρ_v in $\langle \mathbb{R}, +, \leq \rangle$ and explicitly computed from this formula.*

Proof. To establish this theorem, we show how to build the formula $\rho_v(z)$ that is true iff $\mathsf{ASV}(v) = z$. We use Lemma 11, to reduce this to the construction of a formula that expresses the existence of witnesses for $\mathsf{ASV}(v)$ from $(v, \{v\})$:

$$\mathsf{ASV}(v) = \max_{S \in \mathsf{SCC}^{\mathsf{ext}}(v)} \sup\{c \in \mathbb{R} \mid \exists \pi^{\mathsf{ext}} : \pi^{\mathsf{ext}} \text{ is a witness for } \mathsf{ASV}(v, \{v\}) > c \text{ and } V^*(\pi^{\mathsf{ext}}) = S\}$$

As $\max_{S \in \mathsf{SCC}^{\mathsf{ext}}(v)}$ is easily expressed in $\langle \mathbb{R}, +, \leq \rangle$, we concentrate on one SCC S reachable from $(v, \{v\})$ and we show how to express

$$\sup\{c \in \mathbb{R} \mid \exists \pi^{\mathsf{ext}} : \pi^{\mathsf{ext}} \text{ is a witness for } \mathsf{ASV}(v, \{v\}) > c \text{ and } V^*(\pi^{\mathsf{ext}}) = S\}$$

First, we define a formula that express the existence of a witness for $\mathsf{ASV}(v) > c$. This is done by the following formula:

$$\rho_{v_0}^S(c) \equiv \exists x, y \cdot x > c \wedge \Phi_S(x, y) \wedge \neg \Psi_S(c, y)$$

Where $\Phi_S(x, y)$ is the symbolic encoding of $F_{\mathsf{min}}(\mathsf{CH}(\mathbb{C}(S)))$ as defined in Theorem 2. This ensures that the values (x, y) are the mean-payoff values realizable by some path in S. By Lemma 9, $\neg \Psi_S(c, y)$ expresses that the path does not cross a (c, y)-bad vertex. So the conjunction $\exists x, y \cdot x > c \wedge \Phi_S(x, y) \wedge \neg \Psi_S(c, y)$ establishes the existence of a witness with mean-payoff values (x, y) for the threshold c. From this formula, we can compute the ASV by quantifier elimination in:

$$\exists z \cdot \forall e > 0 \cdot (\rho_{v_0}^S(z - e) \wedge (\forall y \cdot \rho_{v_0}^S(y) \implies y \leq z))$$

and obtain the unique value of z that makes the formula true. ◀

4 Stackelberg values for discounted-sum games

In this section, we study the notion of Stackelberg value in the case of discounted sum measures. Beside the adversarial setting considered so far, we also refer to a *cooperative* framework for discounted sum-games, since we add some results to [15], where the cooperative Stackelberg value for discounted-sum measures has been previously introduced and studied.

On the existence of best-responses for DS. First, we show that the set of best-responses for Player 1 to strategies of Player 0 is guaranteed to be nonempty for discounted sum games, while this was not the case in mean-payoff games.

▶ **Lemma 13.** *For all discounted sum games $\mathcal{G}$ and strategies $\sigma_0 \in \Sigma_0(\mathcal{G})$, $\mathit{BR}_1(\sigma_0) \neq \varnothing$.*

Proof. Given $\sigma \in \Sigma_0(\mathcal{G})$, consider $S = \{\mathsf{DS}_1(\mathsf{Out}(\sigma, \tau)) \mid \tau \in \Sigma_1(\mathcal{G})\}$. S is a non empty limited subset of $\mathbb{R}$, since for each $\tau \in \Sigma_1(\mathcal{G})$ it holds $\mathsf{DS}_1(\mathsf{Out}(\sigma, \tau)) \leq \dfrac{W}{1 - \lambda}$, where W is the maximum absolute value of a weight in $\mathcal{G}$. Hence, S admits a unique superior extreme $s = \mathit{sup}(S)$. By definition of superior extreme, for each $\epsilon > 0$, there exists $v_\epsilon \in S$ such that $s \geq v_\epsilon > s - \epsilon$. Therefore, for each $\epsilon > 0$ there exists $\tau_\epsilon \in \Sigma_1(\mathcal{G})$ such that $s \geq \mathsf{DS}_1(\mathsf{Out}(\sigma, \tau_\epsilon)) > \mathsf{s} - \epsilon$, i.e.:

$$0 \leq s - \mathsf{DS}_1(\mathsf{Out}(\sigma, \tau_\epsilon)) < \epsilon \qquad (2)$$

We show that this implies that $\mathsf{Out}(\sigma)$ contains a play π^* such that $\mathsf{DS}_1(\pi^*) = s$, which leads to $BR_1(\sigma) \neq \varnothing$, since Player 1 has a strategy to achieve $s = sup(\{\mathsf{DS}_1(\mathsf{Out}(\sigma, \tau)) \mid \tau \in \Sigma_1(\mathcal{G})\})$. By contradiction, suppose that $\mathsf{Out}(\sigma)$ does not contain any play π such that $\mathsf{DS}_1(\pi) = s$. Hence, for each $\pi \in \mathsf{Out}(\sigma)$, it holds that $\mathsf{DS}_1(\pi) < s$ and π admits a prefix $\pi_{\leq k}$ such that:

$$\mathsf{DS}_1(\pi_{\leq k}) + W\frac{\lambda^k}{1-\lambda} < s \tag{3}$$

Hence, we can cut each play in $\mathsf{Out}(\sigma)$ as soon as Equation 3 is accomplished, leading to a finite tree T (by Konig lemma, since $\mathsf{Out}(\sigma)$ is finitely branching). Let $\pi^*_T = \pi_0 \dots \pi_k$ be a branch in the finite tree T such that the value $v(\pi^*_T) = s - (\mathsf{DS}_1(\pi^*_T) + W\frac{\lambda^k}{1-\lambda})$ is minimal. Note that, by Equation 3, $v(\pi^*_T) > 0$ since $v(\pi^*_T) = s - (\mathsf{DS}_1(\pi^*_T) + W\frac{\lambda^k}{1-\lambda}) > s - s = 0$.

Then, for each play π, let $\pi_{\leq p}$ be the longest prefix of π which is also a branch in the finite tree T. By definition of π^*_T, we have:

$$s - \mathsf{DS}_1(\pi) \geq s - (\mathsf{DS}_1(\pi_{\leq p}) + W\frac{\lambda^p}{1-\lambda}) \geq v(\pi^*_T) > 0 \tag{4}$$

This leads to a contradiction to the fact that for all $\epsilon > 0$ there exists $\tau \in \Sigma_1(\mathcal{G})$ such that $s - \mathsf{DS}_1(\mathsf{Out}(\sigma, \tau_\epsilon)) < \epsilon$, established within Equation 2. ◀

Stackelberg values for DS in the adversarial and cooperative settings. The existence of best-responses allows us to simplify the notion of Stackelberg value for discounted sum measures, avoiding the parameter ϵ used for mean-payoff games. In particular, the adversarial Stackelberg value $\mathsf{ASV}(v)$ for discounted sum games is defined for all $\sigma_0 \in \Sigma_0(\mathcal{G})$ as:

$$\mathsf{ASV}(\sigma_0)(v) = \inf_{\sigma_1 \in \mathsf{BR}_1(\sigma_0)} \mathsf{DS}^\lambda_0(\mathsf{Out}_v(\sigma_0, \sigma_1)) \text{ and } \mathsf{ASV}(v) = \sup_{\sigma_0 \in \Sigma_0} \mathsf{ASV}(\sigma_0)(v)$$

As previously announced, we also consider the notion of Stackelberg value for discounted sum measures in the cooperative setting, where Player 0 suggests a profile of strategies (σ_0, σ_1) and Player 1 agrees to play σ_1 if the latter strategy is a best response to σ_0. Formally, the cooperative Stackelberg value $\mathsf{CSV}(v)$ for discounted sum games is defined as:

$$\mathsf{CSV}(\sigma_0)(v) = \sup_{\sigma_1 \in \mathsf{BR}_1(\sigma_0)} \mathsf{DS}^\lambda_0(\mathsf{Out}_v(\sigma_0, \sigma_1)) \text{ and } \mathsf{CSV}(v) = \sup_{\sigma_0 \in \Sigma_0} \mathsf{CSV}(\sigma_0)(v)$$

Lemma 15 below links the cooperative Stackelberg value for discounted-sum measures to the *target discounted-sum problem* [1] (cfr. Definition 14), whose decidability is notoriously hard to solve and relates to several open questions in mathematics and computer science [1].

▶ **Definition 14** (Target Discount Sum Problem [1] (TDS)). *Given a rational discount factor $0 < \lambda < 1$ and three rationals a, b, t does there exist an infinite sequence $w \in \{a, b\}^\omega$ such that $\sum_{i=0}^{\infty} w(i)\lambda^i = t$?*

In particular, given an instance $I = (a, b, t, \lambda)$ of the TDS problem, Figure 3 depicts a discounted sum game $\mathcal{G}^I$ such that I admits a solution iff $\mathsf{CSV}(v) \geq \lambda \cdot t$.

▶ **Lemma 15.** *The target discounted-sum problem reduces to the problem of deciding if $\mathsf{CSV}(v) \geq c$ in discounted-sum games.*

Proof. Let $I = (a, b, t, \lambda)$ be an instance of the target discounted sum problem and consider the game $\mathcal{G}^I$ depicted in Figure 3. We prove that I admits a solution iff $\mathsf{CSV}(v) \geq \lambda \cdot t$.

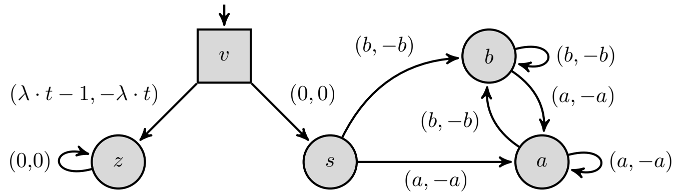


Figure 3 The instance of TDS $I = (a, b, \lambda, t)$ admits a solution iff $\mathsf{CSV}(v) \geq \lambda \cdot t$.

Suppose that I admits a solution and let $w \in \{a,b\}^\omega$ such that $\sum_{i=0}^{\infty} w(i)\lambda^i = t$. Consider the following strategy σ for Player 0: for all $\alpha \in \{a,b\}^*$, $\sigma(vs\alpha) = x$ if $w(|\alpha|) = x$, where $x \in \{a,b\}$. We prove that if τ is a best response to σ, then $\mathsf{DS}_0(\mathsf{Out}(\sigma,\tau)) = \lambda \cdot t$. In fact, Player 1 has two choices from v. Let us denote τ_s (resp. τ_z) the strategy that prescribes to Player 1 to proceed to vertex s (resp. z) out from v. We have that $\mathsf{DS}_1(\mathsf{Out}(\sigma,\tau_\mathsf{s})) = \mathsf{DS}_1(\mathsf{Out}(\sigma,\tau_\mathsf{z})) = -\lambda \cdot t$, by definition of σ and $\mathcal{G}^I$. Hence, τ_s is a best response to σ which guarantees to Player 0 a payoff $\mathsf{DS}_0(\mathsf{Out}(\sigma,\tau_\mathsf{s})) = \lambda \cdot t$.

In the other direction, suppose that I does not admit any solution, i.e. there does not exist an infinite sequence $w \in \{a,b\}^\omega$ such that $\sum_{i=0}^{\infty} w(i)\lambda^i = t$. We prove that for any strategy σ for Player 0, if τ is a best response of Player 1 to σ then $\mathsf{DS}_0(\mathsf{Out}(\sigma,\tau)) < \lambda \cdot t$. Let σ be an arbitrary strategy for Player 0 and consider the strategy τ_z for Player 1.

We have two cases to consider depending on wether τ_z is a best response to σ or not. In the first case, we have that $\mathsf{DS}(\mathsf{Out}(\sigma,\tau_\mathsf{z})) = (\lambda \cdot t - 1, -\lambda \cdot t)$ and, since τ_z is a best response to σ, we need to have $\mathsf{DS}_1(\mathsf{Out}(\sigma,\tau_\mathsf{s})) \leq -\lambda \cdot t$. We can not have that $\mathsf{DS}_1(\mathsf{Out}(\sigma,\tau_\mathsf{s})) = -\lambda \cdot t$, since this would imply $\mathsf{DS}_0(\mathsf{Out}(\sigma,\tau_\mathsf{i})) = -\mathsf{DS}_1(\mathsf{Out}(\sigma,\tau_\mathsf{s})) = \lambda \cdot t$ contradicting our hypothesis that I does not admit any solution. Therefore, $\mathsf{DS}_1(\mathsf{Out}(\sigma,\tau_\mathsf{s})) < -\lambda \cdot t$, meaning that τ_s is not a best response to σ and $\mathsf{CSV}(v) = \lambda \cdot t - 1 < \lambda \cdot t$.

In the second case, where σ_z is not a best response to σ, we have that $\mathsf{DS}_1(\mathsf{Out}(\sigma,\tau_\mathsf{s})) > -\lambda \cdot t$ which implies that $\mathsf{CSV}(v) = \mathsf{DS}_0(\mathsf{Out}(\sigma,\tau_\mathsf{s})) = -\mathsf{DS}_1(\mathsf{Out}(\sigma,\tau_\mathsf{s})) < \lambda \cdot t$. ◀

The construction used to link the cooperative Stackelberg value to the target discounted sum problem can be properly modified[2] to prove that infinite memory may be necessary to allow Player 0 to achieve her CSV, recovering a result originally proved in [15]. In the same paper, the authors show that in 3-player discounted sum games the cooperative Stackelberg value cannot be approximated by considering strategies with bounded memory only. In the next section, we show that this is not the case for 2-player discounted sum games.

Gap problems and their algorithmic solutions. We consider a gap approximation of the Stackelberg value problem in both the cooperative and the adversarial settings. Given $\epsilon > 0$ and $c \in \mathbb{Q}$, and $\mathsf{VAL} \in \{\mathsf{CSV}, \mathsf{ASV}\}$, let us define the sets of games:

- $\mathsf{Yes}^{\epsilon,c}_{\mathsf{VAL}} = \{(\mathcal{G}, v) \mid \mathcal{G} \text{ is a game with } \mathsf{VAL}(v) > c + \epsilon\}$
- $\mathsf{No}^{\epsilon,c}_{\mathsf{VAL}} = \{(\mathcal{G}, v) \mid \mathcal{G} \text{ is a game with } \mathsf{VAL}(v) < c - \epsilon\}$

The (ϵ, c)-CSV-gap (resp. (ϵ, c)-ASV-gap) problem (also referred to as (ϵ, c)-gap problems or just gap problems when ϵ and c are clear from the context) consists in determining if a given game $\mathcal{G}$ and an initial vertex v belong to $\mathsf{Yes}^{\epsilon,c}_{\mathsf{CSV}}$ or $\mathsf{No}^{\epsilon,c}_{\mathsf{CSV}}$ (resp. $\mathsf{Yes}^{\epsilon,c}_{\mathsf{ASV}}$ or $\mathsf{No}^{\epsilon,c}_{\mathsf{ASV}}$). More

[2] Consider the game $\mathcal{G}^I$ depicted in Figure 3 for $a = 0, b = 1, \lambda = \frac{2}{3}, t = \frac{3}{2}$. By Proposition 1 in [6], Player 0 can achieve $\frac{3}{2}$ from s – and therefore CSV(v)=1 – only with infinite memory.

precisely, solving the Stackelberg value gap problem in e.g. the cooperative setting amounts to answer Yes if the instance of the game belongs to $\mathsf{Yes}^{\epsilon,c}_{\mathsf{CSV}}$, answer No if the instance belongs to $\mathsf{No}^{\epsilon,c}_{\mathsf{CSV}}$, never answer or answer arbitrarily otherwise.

Theorem 17 below uses the results in Lemma 16 to provide an algorithm that solves the Stackelberg value gap problem in the cooperative and adversarial settings, for games with discounted sum objectives. In particular, Lemma 16, shows that finite memory strategies are sufficient to witness Stackelberg values strictly greater than a threshold $c \in \mathbb{Q}$.

▶ **Lemma 16.** *Let $\mathcal{G}$ be a discounted-sum game and consider $c \in \mathbb{Q}$ and $\epsilon > 0$. If Player 0 has a strategy σ_0 such that $\mathit{CSV}(\sigma_0)(v) > c + \epsilon$ (resp. $\mathit{ASV}(\sigma_0)(v) > c + \epsilon$), then Player 0 has a strategy σ_0^* with finite memory $M(\epsilon)$ such that $\mathit{CSV}(\sigma_0^*)(v) > c$ (resp. $\mathit{ASV}(\sigma_0^*)(v) > c$). Moreover, $M(\epsilon)$ is computable given ϵ.*

Proof. Let $\sigma^{\mathsf{DS}_1}_{min} \in \Sigma_0$ a memoryless strategy for Player 0 minimizing $sup_{\tau \in \Sigma_1} \mathsf{DS}_1(\mathsf{Out}(\sigma, \tau))$. Let $\sigma^{\mathsf{DS}_1}_{max} \in \Sigma_0$ be a memoryless strategy for Player 0 that maximizes $sup_{\tau \in \Sigma_1} \mathsf{DS}_1(\mathsf{Out}(\sigma, \tau))$. Such memoryless strategies exist since 2-player (single-valued) discounted-sum games are memoryless determined. In particular, $\sigma^{\mathsf{DS}_1}_{min} \in \Sigma_0$ can be obtained by using standard algorithms for two players (single-valued) discounted-sum games. In turn, $\sigma^{\mathsf{DS}_1}_{max} \in \Sigma_0$ can be computed by solving a single player (single valued) discounted-sum game, in which all the nodes are controlled by the maximizer who aims at maximizing DS_1.

Cooperative Setting: Let $\sigma^* \in \Sigma_0(\mathcal{G})$ be a strategy for Player 0 s.t. $\mathsf{DS}_0(\mathsf{Out}(\sigma^*, \tau)) > \mathsf{c} + \epsilon$ for some strategy $\tau \in BR_1(\sigma^*)$. Denote by π^* the play $\pi^* = \mathsf{Out}(\sigma^*, \tau)$ and let N such that $\lambda^N \dfrac{W}{1-\lambda} < \dfrac{\epsilon}{2}$. Given the above premises, consider the finite memory strategy $\sigma' \in \Sigma_0$ for Player 0 that follows σ^* for the first N steps and then either apply the memoryless strategy $\sigma^{\mathsf{DS}_1}_{min} \in \Sigma_0$ or the memoryless strategy $\sigma^{\mathsf{DS}_1}_{max} \in \Sigma_0$, depending on the history h followed up to N. In particular, if $h = \pi^*_{\leq N}$, then the strategy σ' prescribes to Player 0 to follow $\sigma^{\mathsf{DS}_1}_{max} \in \Sigma_0$, cooperating with Player 1 at maximizing DS_1. Otherwise ($h \neq \pi^*_{\leq N}$), the strategy σ' prescribes to Player 0 to follow $\sigma^{\mathsf{DS}_1}_{min} \in \Sigma_0$, minimizing the payoff of the adversary. We show that a best response τ' for Player 1 to σ' consists in following π^* up to N and then applying the memoryless strategy $\tau^{\mathsf{DS}_1}_{max} \in \Sigma_1$, i.e. maximizing $sup_{\sigma \in \Sigma_0} \mathsf{DS}_1(\mathsf{Out}(\sigma, \tau))$. In fact, by definition of σ' and τ' we have that:

- $\mathsf{DS}_1(\mathsf{Out}(\sigma', \tau')) \geq \mathsf{DS}_1(\pi^*)$
- for any other strategy $\tau'' \neq \tau'$ for Player 1:
  - if $\mathsf{Out}(\sigma', \tau'')_{\leq N} = x \neq \pi^*_{\leq N}$, then:

$$\mathsf{DS}_1(\mathsf{Out}(\sigma', \tau'')) = \mathsf{DS}_1(x) + \lambda^N \mathsf{DS}_1(\mathsf{Out}_x(\sigma^{\mathsf{DS}_1}_{min}, \tau'')) \leq$$
$$\leq \mathsf{DS}_1(x) + \lambda^N (sup_{\tau \in \Sigma_1}(\mathsf{DS}_1(\mathsf{Out}_x(\sigma^{\mathsf{DS}_1}_{min}, \tau))) \leq$$
$$\leq \mathsf{DS}_1(x) + \lambda^N (sup_{\tau \in \Sigma_1}(\mathsf{DS}_1(\mathsf{Out}_x(\sigma^*, \tau))) = \mathsf{DS}_1(\pi^*) \leq \mathsf{DS}_1(\mathsf{Out}(\sigma', \tau'))$$

  since $\mathsf{DS}_1(\pi^*)$ is the payoff (for player 1) of a best response of Player 1 to σ^*.
  - if $\mathsf{Out}(\sigma', \tau'')_{\leq N} = x = \pi^*_{\leq N}$, then:

$$\mathsf{DS}_1(\mathsf{Out}(\sigma', \tau'')) \leq \mathsf{DS}_1(x) + \lambda^N \cdot sup\{\mathsf{DS}_1(\pi) \mid \pi \in \mathsf{Plays}(\mathcal{G}) \wedge \pi \text{ starts at } last(x)\} =$$
$$= \mathsf{DS}_1(x) + \lambda^N \cdot \mathsf{DS}_1(\mathsf{Out}_x(\sigma^{\mathsf{DS}_1}_{max}, \tau^{\mathsf{DS}_1}_{max})) = \mathsf{DS}_1(\mathsf{Out}(\sigma', \tau'))$$

Finally, we show that the best response π' of Player 1 to σ' guarantees to Player 0 a payoff greater than c. In fact, $\mathsf{DS}_0(\mathsf{Out}(\sigma', \tau')) > \mathsf{DS}_0(\pi^*_{\leq N}) - \dfrac{\epsilon}{2} > c + \dfrac{\epsilon}{2} - \dfrac{\epsilon}{2} = c$, since $\mathsf{DS}_0(\pi^*_{\leq N}) > c + \dfrac{\epsilon}{2}$. Due to the choice of N, having $\mathsf{DS}_0(\pi^*_{\leq N}) \leq c + \dfrac{\epsilon}{2}$ would lead in fact to the following contradiction: $\mathsf{DS}_0(\pi^*) \leq \mathsf{DS}_0(\pi^*_{\leq N}) + \lambda^N \dfrac{W}{1-\lambda} < \mathsf{DS}_0(\pi^*_{\leq N}) + \dfrac{\epsilon}{2} \leq c + \dfrac{\epsilon}{2} + \dfrac{\epsilon}{2} = c + \epsilon$, i.e. $\mathsf{DS}_0(\pi^*) \leq c + \epsilon$.

Adversarial Setting: Let $\sigma \in \Sigma_0$ be a strategy for Player 0 such that for all $\tau \in BR_1(\sigma)$ it holds $\mathsf{DS}_0(\mathsf{Out}(\sigma, \tau)) > \mathsf{c} + \epsilon$. Let N such that $\lambda^N \dfrac{W}{1-\lambda} < \dfrac{\epsilon}{2}$ and consider the unfolding T of $\mathsf{Out}(\sigma)$ up to N. For each maximal root-to-leaf branch b of T, color its leaf $last(b)$ green if b is the prefix $\pi_{\leq N}$ of some play $\pi = \mathsf{Out}(\sigma, \tau)$ such that $\tau \in BR_1(\sigma)$. Otherwise, let the leaf $last(b)$ of b be colored by red. We show that the finite memory strategy $\sigma^* \in \Sigma_0$ that prescribes to Player 0 to follow σ up to N and then:

- from each green node apply the memoryless strategy $\sigma^{\mathsf{DS}_1}_{max} \in \Sigma_0$ (i.e. cooperate with Player 1 to maximize the payoff DS_1)
- from each red node apply the memoryless strategy $\sigma^{\mathsf{DS}_1}_{min} \in \Sigma_0$ (i.e. minimize the payoff DS_1 of the adversary)

is such that $ASV(\sigma^*) > c$. Let $d = sup\{\mathsf{DS}_1(\mathsf{Out}(\sigma, \tau)) | \tau \in \mathsf{\Sigma}_1(\mathcal{G})\}$ and consider $\pi \in \mathsf{Out}(\sigma^*)$.

First, we show that if π contains a green node then $\mathsf{DS}_0(\pi_{\leq \mathsf{N}}) > \mathsf{c}$. In fact, $\mathsf{DS}_0(\pi_{\leq N}) > \mathsf{DS}_0(\pi_{\leq N}) - \lambda^N \dfrac{W}{1-\lambda} > c + \dfrac{\epsilon}{2} - \dfrac{\epsilon}{2} = c$, since $\lambda^N \dfrac{W}{1-\lambda} < \dfrac{\epsilon}{2}$ by definition of N and since $\mathsf{DS}_0(\pi_{\leq N}) > c + \dfrac{\epsilon}{2}$ being $last(\pi_{\leq N})$ a green node (witnessing that $\pi_{\leq N}$ is the prefix of a play π' compatible with a best response of Player 1 to σ^*, for which $\mathsf{DS}_0(\pi') > \mathsf{c} + \epsilon$).

Moreover, there is a play $\pi \in \mathsf{Out}(\sigma^*)$ containing a green node for which $\mathsf{DS}_1(\pi) \geq d$. This is because of two reasons. First, a play in $\mathsf{Out}(\sigma)$ compatible with a best response to σ by Player 1 is of the form $hv\pi'$, where hv is a maximal root-to-leaf branch b of T with $last(b) = v$ green (by definition of green nodes). Second, for each hystory hv such that hv is a maximal root-to-leaf branch b of T with $last(b) = v$ green, $\mathsf{Out}(\sigma^*)$ contains a play $hv\bar{\pi}$, where $\bar{\pi}$ is a play starting in v maximizing DS_1. Therefore $\mathsf{DS}_1(hv\bar{\pi}) = \mathsf{DS}_1(hv) + \lambda^N \mathsf{DS}_1(\bar{\pi}) \geq \mathsf{DS}_1(hv) + \lambda^N \mathsf{DS}_1(\pi')) = d$, where $hv\pi'$ is a play compatible with a best response of Player 1 to σ. To conclude our proof, we need just to show that each play $\pi \in \mathsf{Out}(\sigma^*)$ containing a red node is such that $\mathsf{DS}_1(\pi) < d$. In fact, being $last(\pi_{\leq N})$ red, the history $\pi_{\leq N}$ can not be a prefix of any play in $\mathsf{Out}(\sigma)$ compatible with a best response of Player 1 to σ. In other words, by playing σ Player 0 allows the adversary to gain a payoff that is at most $r < d$ on each play $\pi = hv\pi'$ with v red. Therefore, switching her strategy from σ to σ^* (i.e. playing σ for the first N turns and then switching to the memoryless strategy $\sigma^{\mathsf{DS}_1}_{min} \in \Sigma_0$) Player 0 is sure to let Player 1 gain a payoff that is at most $r' \leq r < d$ on each play $\pi = hv\pi'$ with v red.

As a conclusion, against σ^* Player 1 can achieve at least a value d. Hence, each best response to σ^* visits a green node (if it does not, then $\mathsf{DS}_1 < d$ which is a contradiction). This guarantees that $\mathsf{DS}_0 > c$. ◀

The approximation algorithm for solving the Stackelberg values gap problems introduced in Theorem 17 roughly works as follows. Given a discounted sum game $\mathcal{G}$, a rational threshold $c \in \mathbb{Q}$ and a tolerance rational value $\epsilon > 0$, the procedure checks whether there exists a strategy $\sigma_0 \in \Sigma_0(\mathcal{G})$ with finite memory $M(\epsilon)$ such that $\mathsf{ASV}(\sigma_0) > c$ (resp. $\mathsf{CSV}(\sigma_0) > c$). If such a strategy exists, the procedure answers Yes, otherwise it answers No. The correctness of the outlined procedure follows directly from Lemma 16.

▶ **Theorem 17.** *The (ϵ, c)-gap problems for both the CSV and ASV are decidable for games with discounted-sum objectives.*

We conclude this subsection by providing a reduction from the partition problem to our gap problems (for both CSV and ASV), showing NP-hardness for the corresponding problems.

▶ **Theorem 18.** *The (ϵ, c)-gap problems for both the CSV and ASV are* NP*-hard for games with discounted-sum objectives, where ϵ and c are given as input.*

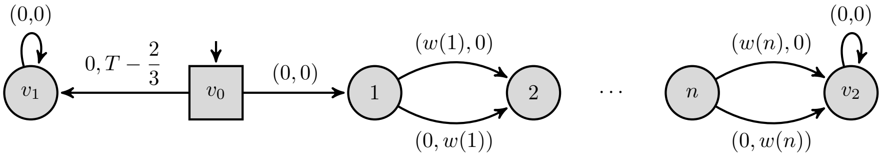


Figure 4 Arena for hardness proof of the gap problem.

Proof. We do a reduction from the Partition problem to our gap problems, working for both CSV and ASV. Let us consider an instance of the partition problem defined by a set $A = \{1, 2, \dots n\}$, a function $w : A \to \mathbb{N}_0$. The partition problem asks if there exists $B \subset A$ such that $\sum_{a \in B} w(a) = \sum_{a \in A \setminus B} w(a)$. W.l.o.g., we assume $\sum_{a \in A} w(a) = 2 \cdot T$ for some T.

To define our reduction, we first fix c to $T - \frac{1}{2}$, and the two parameters $\lambda \in (0,1)$ and $\epsilon > 0$ by choosing values that respect the following two constraints:

$$T \cdot \lambda^{n+1} > T - \frac{1}{2} + \epsilon \qquad\qquad (T-1) \cdot \lambda^{n+1} < T - \frac{1}{2} - \epsilon \tag{5}$$

It is not difficult to see that such values always exist and they can be computed in polynomial time from the description of the partition problem. Then, we construct the bi-weighted arena $\mathcal{A}$ depicted in Fig. 4. In this arena, Player 1 has only two choices in the starting state of the game v_0. There, he can either send the game to the state v_1, and get a payoff of $T - \frac{2}{3}$, or he can go to state 1.

From state 1, Player 0 can simulate a partition of the elements of A by choosing edges: left edges simulate the choice of putting the object corresponding to the state in the left class and right edges simulate the choice of putting the corresponding object in the right class. Let D_0 and D_1 be the discounted sum obtained by Player 0 and Player 1 when arriving in v_2. Because λ and ϵ have been chosen according to eq. (5) , we have that: $D_0 > T - \frac{1}{2} + \epsilon \wedge D_1 > T - \frac{1}{2} + \epsilon$ if and only if the choices of edges of Player 0 correspond to a valid partition of A.

Indeed, assume that $B \subseteq A$ is a solution to the partition problem. Assume that Player 0 follows the choices defined by B. Then when the game reaches state v_2, the discounted sum of rewards for both players is larger than $T \cdot \lambda^{n+1}$. This is because along the way to v_2, the discounted factor applied on the rewards obtained by both players has always been smaller than λ^{n+1} as they were equal to λ^{i+1} for all $i \leq n$. Additionally, we know that sum of (non-discounted) rewards for both players is equal to T as B is a correct partition. Now, it should be clear that both $\mathsf{ASV}(v_0)$ and $\mathsf{CSV}(v_0)$ are greater than $T - \frac{1}{2} + \epsilon$ as in the two cases, Player 1 has no incentive to deviate from the play that goes to v_1 as Player 1 would only get $T - \frac{2}{3}$ which is strictly smaller than D_1.

Now, assume that there is no solution to the partition problem. In that case, Player 0 cannot avoid to give less than $T - 1$ to herself or to Player 1 when going from v_0 to v_2. In the first case, its reward is less than $T - 1$ and in the second case, the reward of Player 1 is less than $T - 1$ and Player 1 has an incentive to deviate to state v_1. In the two cases, we have that both $\mathsf{ASV}(v_0)$ and $\mathsf{CSV}(v_0)$ are less than $T - \frac{1}{2} - \epsilon$. So, we have established that the answer to the gap problem is yes if the partition instance is positive, and the answer is no if the partition instance is negative. ◀

References

1 Udi Boker, Thomas A. Henzinger, and Jan Otop. The target discounted-sum problem. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*, pages 750–761, 2015. URL: https://ieeexplore.ieee.org/xpl/conhome/7174833/proceeding.

2 Romain Brenguier, Lorenzo Clemente, Paul Hunter, Guillermo A. Pérez, Mickael Randour, Jean-François Raskin, Ocan Sankur, and Mathieu Sassolas. Non-zero sum games for reactive synthesis. In *Language and Automata Theory and Applications - 10th International Conference, LATA 2016, Prague, Czech Republic, March 14-18, 2016, Proceedings*, volume 9618 of *Lecture Notes in Computer Science*, pages 3–23. Springer, 2016. doi:10.1007/978-3-319-30000-9.

3 Romain Brenguier and Jean-François Raskin. Pareto curves of multidimensional mean-payoff games. In *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part II*, volume 9207 of *Lecture Notes in Computer Science*, pages 251–267. Springer, 2015. doi:10.1007/978-3-319-21668-3.

4 Thomas Brihaye, Julie De Pril, and Sven Schewe. Multiplayer cost games with simple nash equilibria. In *Logical Foundations of Computer Science, International Symposium, LFCS 2013, San Diego, CA, USA, January 6-8, 2013. Proceedings*, volume 7734 of *Lecture Notes in Computer Science*, pages 59–73. Springer, 2013. doi:10.1007/978-3-642-35722-0.

5 Krishnendu Chatterjee, Laurent Doyen, Herbert Edelsbrunner, Thomas A. Henzinger, and Philippe Rannou. Mean-payoff automaton expressions. In *CONCUR 2010 - Concurrency Theory, 21th International Conference, CONCUR 2010, Paris, France, August 31-September 3, 2010. Proceedings*, volume 6269, pages 269–283. Springer, 2010. doi:10.1007/978-3-642-15375-4.

6 Krishnendu Chatterjee, Vojtěch Forejt, and Dominik Wojtczak. Multi-objective discounted reward verification in graphs and mdps. In Ken McMillan, Aart Middeldorp, and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning*, pages 228–242, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

7 Rodica Condurache, Emmanuel Filiot, Raffaella Gentilini, and Jean-François Raskin. The complexity of rational synthesis. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 121:1–121:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. URL: http://www.dagstuhl.de/dagpub/978-3-95977-013-2.

8 Jeanne Ferrante and Charles Rackoff. A decision procedure for the first order theory of real addition with order. *SIAM J. Comput.*, 4(1):69–76, 1975. doi:10.1137/0204006.

9 Emmanuel Filiot, Raffaella Gentilini, and Jean-François Raskin. The adversarial stackelberg value in quantitative games, 2020. arXiv:2004.12918.

10 Emmanuel Filiot, Raffaella Gentilini, and Jean-François Raskin. Rational synthesis under imperfect information. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 422–431. ACM, 2018. doi:10.1145/3209108.

11 Dana Fisman, Orna Kupferman, and Yoad Lustig. Rational synthesis. In *Tools and Algorithms for the Construction and Analysis of Systems, 16th International Conference, TACAS*, volume 6015 of *Lecture Notes in Computer Science*, pages 190–204. Springer, 2010. doi:10.1007/978-3-642-12002-2.

12 Oded Goldreich. On promise problems: A survey. In *Theoretical Computer Science, Essays in Memory of Shimon Even*, volume 3895 of *Lecture Notes in Computer Science*, pages 254–290. Springer, 2006. doi:10.1007/11685654.

13 Anshul Gupta and Sven Schewe. Quantitative verification in rational environments. In *21st International Symposium on Temporal Representation and Reasoning, TIME 2014, Verona, Italy, September 8-10, 2014*, pages 123–131. IEEE Computer Society, 2014. URL: https://ieeexplore.ieee.org/xpl/conhome/6933234/proceeding.

14 Anshul Gupta, Sven Schewe, Ashutosh Trivedi, Maram Sai Krishna Deepak, and Bharath Kumar Padarthi. Incentive stackelberg mean-payoff games. In *Software Engineering and Formal Methods - 14th International Conference, SEFM 2016, Held as Part of STAF 2016, Vienna, Austria, July 4-8, 2016, Proceedings*, volume 9763 of *Lecture Notes in Computer Science*, pages 304–320. Springer, 2016.

15 Anshul Gupta, Sven Schewe, and Dominik Wojtczak. Making the best of limited memory in multi-player discounted sum games. In *Proceedings Sixth International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2015, Genoa, Italy, 21-22nd September 2015*, volume 193 of *EPTCS*, pages 16–30, 2015. doi:10.4204/EPTCS.193.

16 Richard M. Karp. A characterization of the minimum cycle mean in a digraph. *Discrete Mathematics*, 23(3):309–311, 1978. doi:10.1016/0012-365X(78)90011-0.

17 Orna Kupferman, Giuseppe Perelli, and Moshe Y. Vardi. Synthesis with rational environments. *Ann. Math. Artif. Intell.*, 78(1):3–20, 2016. doi:10.1007/s10472-016-9508-8.

18 J. F. Nash. Equilibrium points in n-person games. In *PNAS*, volume 36, pages 48–49. National Academy of Sciences, 1950.

19 Martin J. Osborne. *An introduction to game theory*. Oxford Univ. Press, 2004.

20 Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *Conference Record of the Sixteenth Annual ACM Symposium on Principles of Programming Languages, Austin, Texas, USA, January 11-13, 1989*, pages 179–190, 1989. URL: http://dl.acm.org/citation.cfm?id=75277.

21 Yaron Velner, Krishnendu Chatterjee, Laurent Doyen, Thomas A. Henzinger, Alexander Moshe Rabinovich, and Jean-François Raskin. The complexity of multi-mean-payoff and multi-energy games. *Inf. Comput.*, 241:177–196, 2015. doi:10.1016/j.ic.2015.03.001.

The Topology of Local Computing in Networks

Pierre Fraigniaud
Institut de Recherche en Informatique Fondamentale, CNRS, Université de Paris, France

Ami Paz
Faculty of Computer Science, Universität Wien, Austria

Abstract

Modeling distributed computing in a way enabling the use of formal methods is a challenge that has been approached from different angles, among which two techniques emerged at the turn of the century: protocol complexes, and directed algebraic topology. In both cases, the considered computational model generally assumes communication via shared objects (typically a shared memory consisting of a collection of read-write registers), or message-passing enabling direct communication between any pair of processes. Our paper is concerned with network computing, where the processes are located at the nodes of a network, and communicate by exchanging messages along the edges of that network (only neighboring processes can communicate directly).

Applying the topological approach for verification in network computing is a considerable challenge, mainly because the presence of identifiers assigned to the nodes yields protocol complexes whose size grows exponentially with the size of the underlying network. However, many of the problems studied in this context are of local nature, and their definitions do not depend on the identifiers or on the size of the network. We leverage this independence in order to meet the above challenge, and present *local* protocol complexes, whose sizes do not depend on the size of the network. As an application of the design of "compacted" protocol complexes, we reformulate the celebrated lower bound of $\Omega(\log^* n)$ rounds for 3-coloring the n-node ring, in the algebraic topology framework.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Distributed computing, distributed graph algorithms, combinatorial topology

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.128

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

Related Version `https://arxiv.org/pdf/2003.03255.pdf`

Funding *Pierre Fraigniaud*: Additional support from ANR projects DESCARTES and FREDDA, and INRIA project GANG.
Ami Paz: This research was done during the stay of the second author at Institut de Recherche en Informatique Fondamentale (IRIF), supported by Fondation des Sciences Mathématiques de Paris (FSMP).

Acknowledgements Both authors are thankful to Juho Hirvonen for his help with the figures.

1 Context and Objective

Several techniques for formalizing distributed computing based on algebraic topology have emerged in the last decades, including the study of complexes capturing all possible global states of the systems at a given time [11], and the study of the (di)homotopy classes of directed paths representing the execution traces of concurrent programs [7]. We refer to [10] for a recent attempt to reconcile the two approaches. This paper is focusing on the approach based on the study of complexes.

A generic methodology for studying distributed computing through the lens of topology has been set by Herlihy and Shavit [12]. This methodology has played an important role in distributed computing, mostly for establishing lower bounds and impossibility results [5,12,18],

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 128; pp. 128:1–128:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

but also for the design of algorithms [6]. It is based on viewing distributed computation as a topological deformation of an input space. More specifically, recall that a *simplicial complex* $\mathcal{K}$ is a collection of non-empty subsets of a finite set V, downward closed under inclusion, i.e., for every $\sigma \in \mathcal{K}$, and every non-empty $\sigma' \subset \sigma$, it holds that $\sigma' \in \mathcal{K}$. Every $\sigma \in \mathcal{K}$ is called a *simplex*, and every $v \in V$ is called a *vertex*. For instance, a graph $G = (V, E)$ with $E \subseteq \binom{V}{2}$, can be viewed as the complex $\mathcal{K} = \{\{v\} : v \in V\} \cup E$ on the set V of vertices. A *sub-complex* of a complex $\mathcal{K}$ is a subset of simplices of $\mathcal{K}$ forming a complex. The dimension of a simplex is one less than the number of its elements. A *facet* of a complex $\mathcal{K}$ is a maximal simplex of $\mathcal{K}$, that is, a simplex not contained in any other simplex. E.g., the facets of a graph with no isolated nodes are its edges. We note that a set of facets uniquely defines a complex.

The set of all possible input (resp., output) configurations of a distributed system can be viewed as a simplicial complex, called *input complex* (resp., *output complex*), and denoted by $\mathcal{I}$ (resp., $\mathcal{O}$). A vertex of $\mathcal{I}$ (resp., $\mathcal{O}$) is a pair (p, x) where p is a process name, and x is an input (resp., output) value. For instance, the input complex of binary consensus in an n-process system with process names $p_1, \dots, p_n$ is:

$$\mathcal{I}_{\|} = \Big\{ \big\{ (p_i, x_i) : i \in I, x_i \in \{0,1\} \text{ for every } i \in I \big\} : I \subseteq [n], I \neq \varnothing \Big\},$$

with $[n] = \{1, \dots, n\}$, and the output complex is:

$$\mathcal{O}_{\|} = \Big\{ \big\{ (p_i, y) : i \in I \big\}, I \subseteq [n], I \neq \varnothing, y \in \{0,1\} \Big\}.$$

One can check that $\mathcal{I}_{\|}$ and $\mathcal{O}_{\|}$ are indeed collections of non-empty subsets of a finite set, downward closed under inclusion. A distributed computing *task* is then specified as a *carrier map* $\Delta : \mathcal{I} \to 2^{\mathcal{O}}$, i.e., a function Δ that maps every input simplex $\sigma \in \mathcal{I}$ to a sub-complex $\Delta(\sigma)$ of the output complex, satisfying that, for every $\sigma, \sigma' \in \mathcal{I}$, if $\sigma \subseteq \sigma'$ then $\Delta(\sigma)$ is a sub-complex of $\Delta(\sigma')$. The carrier map Δ is describing the output configurations that are legal with respect to the input configuration σ. For instance, the specification of consensus is, for every $\sigma = \{(p_i, x_i) : i \in I, x_i \in \{0,1\}\} \in \mathcal{I}_{\|}$,

$$\Delta_{\|}(\sigma) = \begin{cases} \{\{(p_i, 0) : i \in I\}, \{(p_i, 1) : i \in I\}\} & \text{if } \exists\, i, j \in I,\ x_i \neq x_j; \\ \{\{(p_i, y) : i \in I\}\} & \text{if } \forall\, i \in I,\ x_i = y. \end{cases}$$

Note that the specification of consensus given here is very general, i.e., Δ is specified for every simplex $\sigma \in \mathcal{I}_{\|}$. This enables, e.g., to handle crash failures. In absence of failures, the specification of a task can be done just by specifying Δ for the facets in the input complex.

In the topological framework, computation is modeled by a *protocol complex* that evolves with time, where the notion of "time" depends on the computational model at hand. The protocol complex at time t, denoted by $\mathcal{P}^{(t)}$, captures all possible states of the system at time t. Typically, a vertex of $\mathcal{P}^{(t)}$ is a pair (p, s) where p is a process name, and s is a possible state of p at time t. A set $\{(p_i, s_i) : i \in I\}$ of such vertices, for $\emptyset \neq I \subseteq [n]$, forms a simplex of $\mathcal{P}^{(t)}$ if the states s_i, $i \in I$, are mutually compatible, that is, if $\{s_i : i \in I\}$ forms a possible global state for the processes in the set $\{p_i : i \in I\}$ at time t.

A crucial point is that an algorithm that outputs in time t induces a mapping $\delta : \mathcal{P}^{(t)} \to \mathcal{O}$. Specifically, if the process p_i with state s_i at time t outputs y_i, then δ maps the vertex $(p_i, s_i) \in \mathcal{P}^{(t)}$ to the vertex $\delta(p_i, s_i) = (p_i, y_i)$ in $\mathcal{O}$. For the task to be correctly solved, the mapping δ must preserve the simplices of $\mathcal{P}^{(t)}$, and must agree with the specification Δ of the task. That is, δ must map simplices to simplices, and if the configuration $\{(p_i, s_i), i \in I\}$ of a distributed system is reachable at time t starting from the input configuration $\{(p_i, x_i), i \in I\}$, then it must be the case that $\{\delta(p_i, s_i), i \in I\} \in \Delta(\{(p_i, x_i), i \in I\})$. The set of configurations reachable in time t stating from an input configuration $\sigma \in \mathcal{I}$ is denoted by $\Xi_t(\sigma)$. In particular, $\Xi_t : \mathcal{I} \to 2^{\mathcal{P}^{(t)}}$ is a carrier map.

Fundamental Lemma. The framework defined by Herlihy and Shavit [12] enables to characterize the power and limitation of distributed computing, thanks to the following generic result, which can be viewed as the basis of distributed computing within the topological framework. Let us consider some (deterministic) distributed computing model, assumed to be *full information*, that is, every process communicates its entire history at each of its communication step. The following result connects solvability of a task by an algorithm in a given model with the existence of a mapping of a specific form between the topological complexes corresponding to this task and this model (see [4, 11, 12] for instantiations of this result for different computational models).

▶ **Lemma 1.** *A task $(\mathcal{I}, \mathcal{O}, \Delta)$ is solvable in time t if and only if there exists a simplicial map $\delta : \mathcal{P}^{(t)} \to \mathcal{O}$ such that, for every $\sigma \in \mathcal{I}$, $\delta(\Xi_t(\sigma)) \subseteq \Delta(\sigma)$.*

Again, beware that the notion of *time* in the above lemma depends on the computational model. The topology of the protocol complex $\mathcal{P}^{(t)}$, and the nature of the carrier map Ξ_t, depend on the input complex $\mathcal{I}$, and on the computing model at hand. For instance, wait-free computing in asynchronous shared memory systems induces protocol complexes by a deformation of the input complex, called *chromatic subdivisions* [11]. Similarly, t-resilient computing may introduce *holes* in the protocol complex, in addition to chromatic subdivisions. More generally, the topological deformation Ξ_t of the input complex caused by the execution of a full information protocol in the considered computing model entirely determines the existence of a decision map $\delta : \mathcal{P}^{(t)} \to \mathcal{O}$, which makes the task $(\mathcal{I}, \mathcal{O}, \Delta)$ solvable or not in that model.

Topological Invariants. The typical approach for determining whether a task (e.g., consensus) is solvable in t rounds consists of identifying topological *invariants*, i.e., properties of complexes that are preserved by simplicial maps. Specifically, the approach consists in:

1. Identifying a topological invariant, i.e., a property satisfied by the input complex $\mathcal{I}$, and preserved by Ξ_t;
2. Checking whether this invariant, which must be satisfied by the sub-complex $\delta(\mathcal{P}^{(t)})$ of the output complex $\mathcal{O}$, does not contradict the specification Δ of the task.

For instance, in the case of binary consensus, the input complex $\mathcal{I}_{||}$ is a *sphere*. One basic property of spheres is being *path-connected* (i.e., there is a path in $\mathcal{I}_{||}$ between any two vertices). As mentioned earlier, shared-memory wait-free computing corresponds to subdividing the input complex [11]. Therefore, independently from the length t of the execution, the protocol complex $\mathcal{P}^{(t)}$ is a chromatic subdivision of the sphere $\mathcal{I}_{||}$, and thus it remains path-connected. On the other hand, the output complex $\mathcal{O}_{||}$ of binary consensus is the disjoint union of two complexes $\mathcal{O}_0$ and $\mathcal{O}_1$, where $\mathcal{O}_y = \big\{\{(i, y) : i \in I\}, I \subseteq [n], I \neq \varnothing\big\}$ for $y \in \{0, 1\}$. Since simplicial maps preserve connectivity, it follows that $\delta(\mathcal{P}^{(t)}) \subseteq \mathcal{O}_0$ or $\delta(\mathcal{P}^{(t)}) \subseteq \mathcal{O}_1$. As a consequence, δ cannot agree with $\Delta_{||}$, as the latter maps the simplex $\{(i, 0), i \in [n]\}$ to $\mathcal{O}_0$, and the simplex $\{(i, 1), i \in [n]\}$ to $\mathcal{O}_1$. Therefore, consensus cannot be achieved wait-free, regardless of the number t of rounds.

The fact that connectivity plays a significant role in the inability to solve consensus in the presence of asynchrony and crash failures is known since the original proof of the FLP theorem [8] in the early 1980s. However, the relation between k-set agreement and higher dimensional forms of connectivity (i.e., the ability to contract high dimensional spheres) was only established ten years later [12, 18]. We refer to [11] for numerous applications of Lemma 1 to various models of distributed computing, including asynchronous crash-prone shared-memory or fully-connected message passing models. In particular, for tasks such as renaming, identifying the minimal number t of rounds enabling a simplicial map δ to exist is currently the only known technique for upper bounding their time complexities [1].

Network Computing. Recently, Castañeda et al. [4] applied Lemma 1 to synchronous fault-free computing in networks, that is, to the framework in which processes are located at the vertices of a simple (no multiple edges, no loops) n-node undirected graph G, and can exchange messages only along the edges of that graph. They mostly focus on *input-output tasks* such as consensus and set-agreement, in a simplified computing model, called KNOW-ALL, specifying that every process is initially aware of the name and the location of all the other processes in the network. As observed in [4], synchronous fault-free computing in the KNOW-ALL model preserves the *facets* of the input complex, and does not subdivide them. However, *scissor cuts* may occur between adjacent facets during the course of the computation, that is, the protocol complex $\mathcal{P}^{(t)}$ is obtained from the input complex $\mathcal{I}$ by partially separating facets that initially shared a simplex. Figure 1 illustrates two types of scissor cuts applied to the sphere, corresponding to two different communication networks. The positions of the cuts depend on the structure of the graph G in which the computation takes place, and determining the precise impact of the structure of G on the topology of the protocol complex is a nontrivial challenge, even in the KNOW-ALL model.

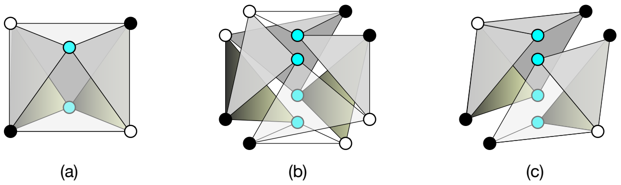


Figure 1 (a) The input complex of binary consensus for three processes; (b) The scissor cuts for the consistently directed 3-process cycle C_3 after one round; (c) The scissor cuts for the directed 3-process star S_3, where edges are directed from the center to the leaves, after one round.

Instead, we aim at analyzing classical *graph problems* (e.g., coloring, independent set, etc.) in the standard LOCAL model [17] of network computing, which is weaker than the KNOW-ALL model, and thus allows for more complicated topological deformations. In the LOCAL model, every node is initially aware of solely its identifier (which is unique in the network), and its input (e.g., for minimum weight vertex cover or for list-coloring), all nodes wake up synchronously, and compute in locksteps. The LOCAL model is an ideal model for studying *locality* in the context of network computing [17].

In addition to the fact that the topological deformations of the protocol complexes strongly depend on the structure of the network, another obstacle that makes applying the topological approach to the LOCAL model even more challenging is the presence of process *identifiers*. Indeed, the model typically assumes that the node IDs are taken in a range $[N]$ where $N = \text{poly}(n)$. As a consequence, independently from the potential presence of other input values, the size of the complexes (i.e., their number of vertices) may become as large as $\binom{N}{n}n!$, since there are $\binom{N}{n}$ ways of choosing n IDs, and $n!$ ways of assigning the n chosen IDs to the n nodes of G (unless G presents symmetries). For instance, Figure 1 assumes the KNOW-ALL model, hence fixed IDs. Redrawing these complexes assuming that the three processes can pick arbitrary distinct IDs as in the LOCAL model, even in the small domain $\{1,2,3,4\}$, would yield a cumbersome figure with 24 nodes. Note that the presence of IDs also results in input complexes that may be topologically more complicated than pseudospheres, even for tasks such as consensus.

Importantly, the fact that the IDs are not fixed a priori, and may even be taken in a range exceeding $[n]$, is inherent to distributed network computing. Indeed, this framework aims at understanding the power and limitation of computing in large networks, from LANs to the whole Internet, where the processing nodes are assigned arbitrary IDs taken from a range of values which may significantly exceed the number of nodes in the network.

Objective. To sum up, while the study of protocol complexes has found numerous applications in the context of fault-tolerant message-passing or shared-memory computing, extending this theory to network computing faces a difficulty caused by the presence of arbitrary IDs, which are often the only inputs to the processes [17]. The objective of this paper is to show how the combinatorial blowup caused by the presence of IDs in network computing can be bypassed, at least as far as local computing is concerned.

2 Our Results

We show how to bypass the aforementioned exponential blowup in the size of the complexes, that would result from a straightforward application of Lemma 1 for analyzing the complexity of tasks in networks. Our result holds for a variety of problems, including classical graph problems such as vertex and edge-coloring, maximal independent set (MIS), maximal matching, etc. More specifically, it holds for the large class of *locally checkable labeling* (LCL) tasks [16] on bounded-degree graphs. These are tasks for which it is possible to verify locally the correctness of a solution, and thus they are sometimes viewed as the analog of NP in the context of computing in networks. An LCL task is described by a finite set of labels, and a local description of how these labels can be legally assigned to the nodes of a network. Our local characterization theorem is strongly based on a seminal result by Naor and Stockmeyer [16] who showed that the *values* of the IDs do not actually matter much for solving LCL tasks in networks, but only their *relative order* matters.

We prove an analog of Lemma 1, but where the size of the complexes involved in the statement is independent of the size of the networks. Specifically, the size of the complexes in our characterization theorem depends solely on the maximum degree d of the network, the number of labels used for the description of the task, and the number of rounds of the considered algorithm for solving that task. In particular, the identifiers are taken from a bounded-size set, even if the theorem applies to tasks defined on n-node networks with arbitrarily large n, and for identifiers taken in an arbitrarily large range $[N]$. We denote by $\mathcal{K}_{x,[y]}$ the fact that the facets of $\mathcal{K}$ have dimension x, and that the IDs are taken in the set $\{1,\dots,y\}$, and we let $\mathcal{K}_x = \mathcal{K}_{x,\varnothing}$. Also $\pi : \mathcal{K}_{x,[y]} \to \mathcal{K}_x$ denotes the mapping that removes the IDs of the vertices. Every LCL task in networks with maximum degree d can be expressed topologically as a task $(\mathcal{I}_d, \mathcal{O}_d, \Delta)$ where $\mathcal{I}_d$ and $\mathcal{O}_d$ are complexes of dimension d. Our main result is the following.

▶ **Theorem 2** (A simplified version of Theorem 3). *For every LCL task $T = (\mathcal{I}_d, \mathcal{O}_d, \Delta)$ on graphs of maximum degree d, and for every $t \geq 0$, there exists $R \in \mathbb{N}$ such that the following holds. The task T is solvable in t rounds in the LOCAL model if and only if there is a simplicial map $\delta : \mathcal{P}^{(t)}_{d,[R]} \to \mathcal{O}_d$ such that, for every facet $\sigma \in \mathcal{I}_{d,[R]}$, $\delta(\Xi_t(\sigma)) \subseteq \Delta(\pi(\sigma))$.*

Figure 2 provides a rough description of the commutative diagram corresponding to the brute force application of Lemma 1 to LCL tasks, and of the commutative diagram corresponding to Theorem 2. Note that Lemma 1, which corresponds to the left diagram in Figure 2, involves *global* complexes with $(n-1)$-dimensional facets, whose vertices are labeled by IDs in an arbitrarily large set $[N]$. In contrast, the complexes corresponding to Theorem 2, which correspond to the right diagram, are *local* complexes, with facets of constant dimension, and vertices labeled with IDs in a finite set whose size is constant w.r.t. the number of nodes n in the network.

$$\begin{tikzcd}
\mathcal{I}_{n-1,[N]} \arrow[r, "\Xi_t"] \arrow[dr, "\Delta"'] & \mathcal{P}^{(t)}_{n-1,[N]} \arrow[d, "\delta"] \\
 & \mathcal{O}_{n-1,[N]}
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
\mathcal{I}_{d,[R]} \arrow[r, "\Xi_t"] \arrow[d, "\pi"'] & \mathcal{P}^{(t)}_{d,[R]} \arrow[d, "\delta"] \\
\mathcal{I}_d \arrow[r, "\Delta"] & \mathcal{O}_d
\end{tikzcd}$$

Figure 2 The commutative diagrams of Lemma 1 (left), and Theorem 2 (right).

As an application of Theorem 2, we reformulate the celebrated lower bound $\Omega(\log^* n)$ rounds for 3-coloring the n-node ring by Linial [15], in the algebraic topology framework (see Corollary 4).

3 Models and Definitions

The LOCAL model. The LOCAL model was introduced more than a quarter of a century ago (see, e.g., [15, 16]) for studying which tasks can be solved *locally* in networks, that is, which tasks can be solved when every node is bounded to collect information only from nodes in its vicinity. Specifically, the LOCAL model [17] states that the processors are located at the nodes of a simple connected graph $G = (V, E)$ modeling a network. All nodes are fault-free, they wake up simultaneously, and they execute the same algorithm. Computation proceeds in synchronous rounds, where a round consists of the following three steps performed by every node: (1) sending a message to each neighbor in G, (2) receiving the messages sent by the neighbors, and (3) performing local computation. There are no bounds on the size of the messages exchanged at every round between neighbors, and there are no limits on the individual computational power or memory of the nodes. These assumptions enable the design of unconditional lower bounds on the number of rounds required for performing some task (e.g., for providing the nodes with a proper coloring), while the vast majority of the algorithms solving these tasks do not abuse of these assumptions [19], that is, they exchange small (i.e., polylogarithmic size) messages, and perform efficient (i.e., poly-time) individual computations.

Every node in the network has an identifier (ID) which is supposed to be unique in the network. In n-node networks, the IDs are supposed to be in a range $1, \dots, N$ where $N \gg n$ typically holds (most often, $N = \text{poly}(n)$). The absence of limits on the amount of communication and computation that can be performed at every round implies that the LOCAL model enables *full-information* protocols, that is, protocols in which, at every round, every node sends all the information it acquired during the previous rounds to its neighbors. Therefore, for every $t \geq 0$, and every graph G, a t-round algorithm allows every node in G to acquires a local *view* of G, which is a ball in G centered at that node, and of radius t. A view includes the inputs and the IDs of the nodes in the corresponding ball. It follows that a t-round algorithm in the LOCAL model can be considered as a function from the set of views of radius t to the set of output values.

Locally Checkable Labelings (LCL). Let $d \geq 2$, and let $\mathcal{G}_d$ be the class of connected simple undirected d-regular graphs (all nodes have degree d). Recall that, for a positive integer c, c-coloring is the task consisting in providing each node with a color in $\{1, \dots, c\}$ in such a way that no two adjacent nodes are given the same color. Maximal independent set (MIS) is the closely related task consisting in providing each node with a boolean value (0 or 1) such that no two adjacent nodes are given the value 1, and every node with value 0 is adjacent to at least one node with value 1. Proper c-coloring in $\mathcal{G}_d$ can actually be described by the collection of *good stars* of degree d, and with nodes colored by labels in $\{1, \dots, c\}$, such that the center node has a color different from the color of each leaf. Similarly, maximal independent set (MIS) in $\mathcal{G}_d$ can be described by the collection of stars with degree d, and with each node colored by a label in $\{0, 1\}$, such that if the center node is labeled 1 then all the leaves are colored 0, and if the center node is labeled 0 then at least one leaf is colored 1. Other tasks such as variants of coloring, or $(2, 1)$-ruling set[1] can be described similarly, by a finite number of legal labeled stars.

More generally, given a finite set $\mathcal{L}$ of labels, we denote by $\mathbf{S}_d^{\mathcal{L}}$ the set of all labeled stars resulting from labeling each node of the $(d + 1)$-node star by some label in $\mathcal{L}$. A *locally checkable labeling* (LCL) [16] is then defined by a finite set $\mathcal{L}$ of labels, and a set $\mathcal{S} \subseteq \mathbf{S}_d^{\mathcal{L}}$. Every star in $\mathcal{S}$ is called a *good* star, and those in $\mathbf{S}_d^{\mathcal{L}} \setminus \mathcal{S}$ are *bad*. The computing task defined by an LCL $(\mathcal{L}, \mathcal{S})$ consists, for every node of every graph $G \in \mathcal{G}_d$, of computing a label in $\mathcal{L}$ such that every resulting labeled radius-1 star in G is isomorphic to a star in $\mathcal{S}$. In other words, the objective of every node is to compute a label in $\mathcal{L}$ such that every resulting labeled radius-1 star in G is good. It is undecidable, in general, whether a given LCL task has an algorithm performing in $O(1)$ rounds in the LOCAL model [16].

In their full generality, LCL tasks include tasks in which nodes have inputs, potentially of some restricted format. For instance, this is the case of the task consisting of reducing c-coloring to MIS in the n-node cycle C_n, studied in the next section. Hence, in its full generality, an LCL task is described by a quadruple $(\mathcal{L}_{in}, \mathcal{S}_{in}, \mathcal{L}_{out}, \mathcal{S}_{out})$ where $\mathcal{L}_{in}$ and $\mathcal{L}_{out}$ are the input and output labels, respectively. The set of stars $\mathcal{S}_{in}$ can often be simply viewed as a promise stating that every radius-1 star of the input graph G belongs to $\mathcal{S}_{in}$, and the set $\mathcal{S}_{out}$ is the target set of good radius-1 stars. LCL tasks also capture settings in which the legality of the output stars depends on the inputs. A typical example of such a setting is list-coloring where the output color of each node must belong to a list of colors given to this node as input. The framework of LCL tasks can be extended to balls of radius $r > 1$, and assuming radius 1 is not restrictive, up to increasing the size of the set of labels [3].

4 Warm Up: Coloring and MIS in the Ring

In this section, we exemplify our technique, in a way that resembles the proof of Theorem 2. We consider an LCL task on a ring, where the legal input stars define a proper 3-coloring, and the output stars define a maximal independent set (MIS). That is, we study the time complexity of reducing a 3-coloring to a MIS on a ring. It is known [15] that there is a 2-round algorithm for the problem in the LOCAL model, and we show that this is optimal using topological arguments. This toy example provides the basic concepts and arguments that we use later, when considering general LCL tasks and proving Theorem 2.

[1] Recall that an (α, β)-ruling set in a graph $G = (V, E)$ is a set $R \subseteq V$ such that, for any node $v \in V$ there is a node $u \in R$ in distance at most β from v, and any two nodes in R are at distance at least α from each other.

4.1 Reduction from 3-coloring to MIS

Let us consider three consecutive nodes of the n-node ring C_n, denoted by p_{-1}, p_0, and p_1, as displayed on Figure 3.

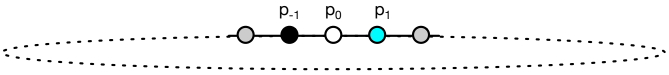


Figure 3 Three consecutive nodes in the n-node ring.

By the independence property, if p_0 is in the MIS, then neither p_{-1} nor p_1 can be in the MIS, and, by the maximality property, if p_0 is not in the MIS, then p_{-1} or p_1, or both, must be in the MIS. These constraints are captured by the complex $\mathcal{M}_2$ displayed on Figure 4, including six vertices (p_i, x), with $i \in \{-1, 0, 1\}$, and $x \in \{0, 1\}$, where $x = 1$ (resp., $x = 0$) indicates that p_i is in the MIS (resp., not in the MIS).

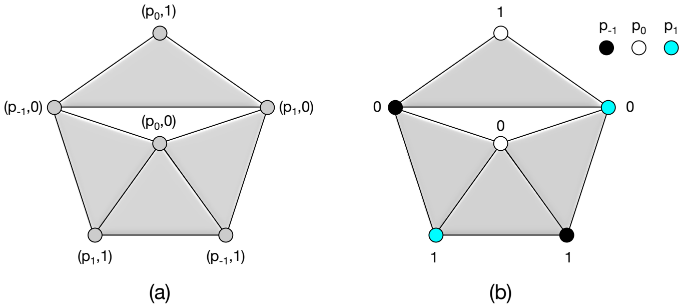


Figure 4 The local complex $\mathcal{M}_2$ of MIS in the ring. (a) the vertices are labeled with the index of the processes and the values; (b) the indexes of the processes are replaced by colors.

The complex $\mathcal{M}_2$ of Figure 4 has four facets of dimension 2: they are triangles. Some triangles intersect along an edge, while some others intersect only at a node. The complex $\mathcal{M}_2$ is called the *local* complex of MIS in the ring (the index 2 refers to the fact that rings have degree 2). Note that the sets $\{(p_{-1}, 0), (p_0, 0), (p_1, 0)\}$ and $\{(p_{-1}, 1), (p_0, 1), (p_1, 1)\}$ do not form simplices of $\mathcal{M}_2$. We call these two sets monochromatic. In the objective of reducing 3-coloring to MIS, $\mathcal{M}_2$ will be the output complex, corresponding to $\mathcal{O}_d$ with $d = 2$ in Figure 2 and in Theorem 2.

Similarly, let us focus on 3-coloring, with the same three processes p_{-1}, p_0, and p_1. The neighborhood of p_0 cannot include the same color as its own color, and thus there are twelve possible colorings of the nodes in the star centered at p_0. Each of these stars corresponds to a 2-dimensional simplex, forming the facets of the local complex $\mathcal{C}_2$ of 3-coloring in the ring, depicted in Figure 5. This complex contains nine vertices of the form (p_i, c), with $i \in \{-1, 0, 1\}$, and $c \in \{1, 2, 3\}$, and twelve facets. Note that the vertices $(p_{-1}, 3)$ and $(p_1, 3)$ appear twice in the figure, since the leftmost and rightmost edges are identified, but in opposite direction, forming a Möbius strip. $\mathcal{C}_2$ is a manifold (with boundary). When reducing 3-coloring to MIS, $\mathcal{C}_2$ will be the input complex, corresponding to $\mathcal{I}_d$ with $d = 2$ in Figure 2.

Remark. It is crucial to note that the complexes displayed in Figures 4 and 5 are not the ones used in the standard settings (e.g., [4, 11]), for which Lemma 1 would use vertices of the form (p, x) for $p \in [n]$, or even $p \in [N]$ assuming IDs in a range of N values. As

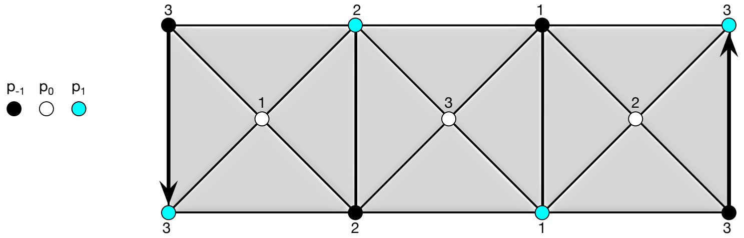


Figure 5 Local complex $\mathcal{C}_2$ of 3-coloring in the ring.

a consequence, these complexes have 6 vertices instead of $2n!\binom{N}{n}$ for MIS, and 9 vertices instead of $3n!\binom{N}{n}$ for coloring, where n can be arbitrarily large. Even if the IDs would have been fixed, the approach of Lemma 1 would yield complexes with a number of vertices linear in n, while the complexes of Figs. 4 and 5 are of constant size.

As it is well know since the early work by Linial [15], a properly 3-colored ring can be "recolored" into a MIS in just two rounds. First, the nodes colored 3 recolor themselves 1 if they have no neighbors originally colored 1. Then, the nodes colored 2 do the same, i.e., they recolor themselves 1 if they have no neighbors colored 1 (whether it be neighbors originally colored 1, or nodes that recolored themselves 1 during the first round). The nodes colored 1 output 1, and the other nodes output 0. The set of nodes colored 1 forms a MIS. Note that this algorithm is *ID-oblivious*, i.e., it can run in an anonymous network.

Task specification. The specification of reducing 3-coloring to MIS can be given by the trivial carrier map $\Delta : \mathcal{C}_2 \to 2^{\mathcal{M}_2}$ defined by $\Delta(F) = \{F' : F' \text{ is a facet of } \mathcal{M}_2\}$ for every facet F of $\mathcal{C}_2$. (As the LOCAL model is failure-free, it is enough to describe all maps at the level of facets.) Note that the initial coloring of a facet in $\mathcal{C}_2$ does not induce constraints on the facet of $\mathcal{M}_2$ to which it should be mapped. Figure 6 displays some of the various commutative diagrams that will be considered in this section. In all of them, Δ is the carrier map specifying reduction from 3-coloring to MIS in the ring, and none of the simplicial maps δ exist. Also recall that π is the map removing IDs.

$$\begin{tikzcd}
\mathcal{C}_{2,\varnothing} \arrow[r, "\Xi_0"] \arrow[d, "\pi"'] & \mathcal{C}_{2,\varnothing} \arrow[d, "\delta"'] \\
\mathcal{C}_2 \arrow[r, "\Delta"] & \mathcal{M}_2
\end{tikzcd}
\quad
\begin{tikzcd}
\mathcal{C}_{2,\varnothing} \arrow[r, "\Xi_1"] \arrow[d, "\pi"'] & \mathcal{P}^{(1)}_{2,\varnothing} \arrow[d, "\delta"'] \\
\mathcal{C}_2 \arrow[r, "\Delta"] & \mathcal{M}_2
\end{tikzcd}
\quad
\begin{tikzcd}
\mathcal{C}_{2,[24]} \arrow[r, "\Xi_0"] \arrow[d, "\pi"'] & \mathcal{C}_{2,[24]} \arrow[d, "\delta"'] \\
\mathcal{C}_2 \arrow[r, "\Delta"] & \mathcal{M}_2
\end{tikzcd}
\quad
\begin{tikzcd}
\mathcal{C}_{2,[R]} \arrow[r, "\Xi_1"] \arrow[d, "\pi"'] & \mathcal{P}^{(1)}_{2,[R]} \arrow[d, "\delta"'] \\
\mathcal{C}_2 \arrow[r, "\Delta"] & \mathcal{M}_2
\end{tikzcd}$$

Figure 6 Complexes corresponding to reduction from 3-coloring to MIS in the n-node ring. From left to right: 0 rounds without IDs, 1-round without IDs, 0 rounds with ID, and 1-round with IDs.

4.2 ID-Oblivious Algorithms

Impossibility in Zero Rounds. Let us consider an alleged ID-oblivious algorithm ALG which reduces 3-coloring to MIS in zero rounds. Such an algorithm sees only the node's color $c \in \{1, 2, 3\}$, and must map it to some $x \in \{0, 1\}$. This mapping can be extended to a mapping δ that maps every pair (p_i, c) with $i \in \{-1, 0, 1\}$ and $c \in \{1, 2, 3\}$ to a pair $\delta(p_i, c) = (p_j, x)$, $j \in \{-1, 0, 1\}$ and $x \in \{0, 1\}$, with the following properties.

- *Name-preservation.* The mapping δ must satisfy that $p_j = p_i$, i.e., δ is *name-preserving*. By the name-preserving property, the algorithm maps the vertices in Figure 5 to the vertices in Figure 4(b) while preserving the names p_{-1}, p_0, p_1 of these vertices. Therefore, the algorithm induces a *chromatic* simplicial map $\delta : \mathcal{C}_2 \to \mathcal{M}_2$. (The "color" of p, i.e., p's name, is preserved).
- *Name-independence.* In addition to name-preservation, the mapping δ must satisfy that, for every $i \neq j$, (p_i, c) and (p_j, c) are mapped to (p_i, x_i) and (p_j, x_j), respectively, with $x_i = x_j$, i.e., δ is *name-independent*. Indeed, the names p_{-1}, p_0, and p_1 given to the nodes are "external", i.e., they are not part of the input to the algorithm ALG.

We are therefore questioning the existence of a name-preserving name-independent simplicial map $\delta : \mathcal{C}_2 \to \mathcal{M}_2$. This is in correspondence to Figure 2 and Theorem 2, in the degenerated case where $t = 0$ and $[R] = \varnothing$, for which $\mathcal{C}_2 = \mathcal{I}_2$, and $\mathcal{C}_{2,\varnothing} = \mathcal{I}_{2,\varnothing} = \mathcal{P}^{(0)}_{2,\varnothing} = \mathcal{C}_2$ – see the leftmost diagram in Figure 6. There cannot exist a name-preserving name-independent simplicial map δ from the manifold $\mathcal{C}_2$ to $\mathcal{M}_2$ (from Figure 5 to Figure 4(b)), which we formally prove in the full version of the paper [9]. The intuition is that if some triangle of $\mathcal{C}_2$ is mapped to the triangle $\{(p_0, 1), (p_{-1}, 0), (p_1, 0)\}$ of $\mathcal{M}_2$ then *all* triangles of $\mathcal{C}_2$ must be mapped to that triangle of $\mathcal{M}_2$, from which it follows by name-independence that all processes output 1, or all processes output 0, which leads to contraction in both cases. The absence of a name-independent name-preserving simplicial map $\delta : \mathcal{C}_2 \to \mathcal{M}_2$ is a witness of the impossibility to construct a MIS from a 3-coloring of the ring in zero rounds, when using an ID-oblivious algorithm.

Impossibility in One Round. For analyzing 1-round algorithms, let us consider the local protocol complex $\mathcal{P}^{(1)}_{2,\varnothing}$, including the views of the three nodes p_{-1}, p_0, and p_1 after one round. The vertices of $\mathcal{P}^{(1)}_{2,\varnothing}$ are of the form (p_i, xyz) with $i \in \{-1, 0, 1\}$, and $x, y, z \in \{1, 2, 3\}$, $x \neq y$, and $y \neq z$. The vertex (p_i, xyz) is representing a process p_i starting with color y, and receiving the input colors x and z from its left and right neighbors, respectively. The facets of $\mathcal{P}^{(1)}_{2,\varnothing}$ are of the form $\{(p_{-1}, x'xy), (p_0, xyz), (p_1, yzz')\}$. Figure 7 displays that complex, which consists of three connected components $\mathcal{K}_1, \mathcal{K}_2$, and $\mathcal{K}_3$ where, for $y = 1, 2, 3$, $\mathcal{K}_y$ includes the four vertices (p_0, xyz) for $x, z \in \{1, 2, 3\} \smallsetminus \{y\}$, and all triangles that include these vertices. Each set of four triangles sharing a vertex (p_0, xyz) forms a cone (see Figure 8). These cones are displayed twisted on Figure 7 to emphasis the "circular structure" of the three components.

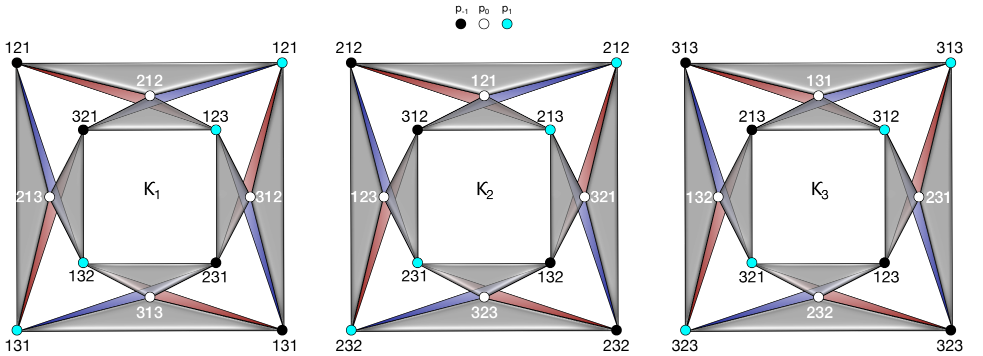


Figure 7 Local protocol complex $\mathcal{P}^{(1)}_{2,\varnothing}$ after 1 round starting from a 3-coloring of the ring.

Following the same reasoning as for 0-round algorithms, a 1-round algorithm ALG induces a chromatic (i.e., name-preserving) simplicial map $\delta : \mathcal{P}^{(1)}_{2,\varnothing} \to \mathcal{M}_2$, as in the second to left diagram in Figure 6. In the full version, we show that such a mapping cannot exist [9].

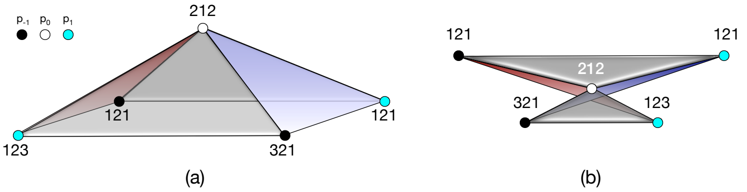


Figure 8 (a) A cone composed of four triangles; (b) The same cone "twisted".

The 2-Round Algorithm. The local protocol complex $\mathcal{P}^{(2)}_{2,\varnothing}$ includes the views of the three nodes p_{-1}, p_0, and p_1 after two rounds. The vertices of $\mathcal{P}^{(2)}_{2,\varnothing}$ are of the form $(p_i, c_1c_2c_3c_4c_5)$ with $i \in \{-1, 0, 1\}$, $c_j \in \{1, 2, 3\}$ for $1 \le j \le 5$, and $c_j \neq c_{j+1}$ for $1 \le j < 5$. Figure 9(a) displays one of the connected components of $\mathcal{P}^{(2)}_{2,\varnothing}$, denoted $\mathcal{K}_{323}$,which includes the four vertices $(p_0, c_1 323 c_5)$, $c_1, c_5 \in \{1, 2\}$. There are 12 disjoint isomorphic copies of this connected component in $\mathcal{P}^{(2)}_{2,\varnothing}$, one for each triplet $c_2, c_3, c_4 \in \{1, 2, 3\}$, $c_2 \neq c_3$, and $c_3 \neq c_4$.

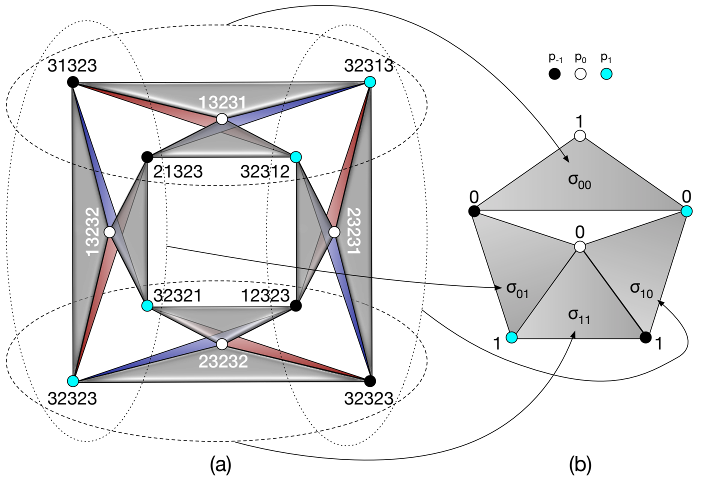


Figure 9 (a) The sub-complex $\mathcal{K}_{323}$ of the local protocol complex $\mathcal{P}^{(2)}_{2,\varnothing}$. (b) The facets of $\mathcal{M}_2$.

Interestingly, each connected component of $\mathcal{P}^{(2)}_{2,\varnothing}$ is isomorphic to each connected component of $\mathcal{P}^{(1)}_{2,\varnothing}$, while there are more connected components in $\mathcal{P}^{(2)}_{2,\varnothing}$ than in $\mathcal{P}^{(1)}_{2,\varnothing}$. However, the larger views of the processes provides more flexibility for the mapping from $\mathcal{P}^{(2)}_{2,\varnothing}$ to $\mathcal{M}_2$ than for the mapping from $\mathcal{P}^{(1)}_{2,\varnothing}$ to $\mathcal{M}_2$. And indeed, the 2-round anonymous algorithm presented at the end of Section 4.1 does induce a chromatic simplicial map $\delta : \mathcal{P}^{(2)}_{2,\varnothing} \to \mathcal{M}_2$. Specifically, the four sub-complexes $\mathcal{K}_{x1y}$, as well as the simplex $\mathcal{K}_{232}$ are entirely mapped to the simplex σ_{00} (see Figure 9(b) for the labeling of the four facets of $\mathcal{M}_2$). The two sub-complexes $\mathcal{K}_{1x1}$ are entirely mapped to the simplex σ_{11}. The two sub-complexes $\mathcal{K}_{321}$ and $\mathcal{K}_{231}$ are entirely mapped to the sub-complex $\sigma_{01} \cup \sigma_{11}$, and the two sub-complexes $\mathcal{K}_{123}$ and $\mathcal{K}_{132}$ are entirely mapped to the sub-complex $\sigma_{10} \cup \sigma_{11}$. The mapping of the remaining sub-complex $\mathcal{K}_{323}$ is more sophisticated, and illustrates that the simple algorithm showing reduction from 3-coloring to MIS in [15] is actually topologically non-trivial. Indeed, $\mathcal{K}_{323}$ is mapped by the algorithm so that it wraps around the hole in $\mathcal{M}_2$, as depicted in Figure 9.

4.3 General Case with IDs

The presence of IDs given to the nodes adds power to the distributed algorithms, as the output of a process is not only a function of the observed colors in its neighborhood, but also of the observed IDs. In particular, after one round, a process p is not only aware of a triplet of colors $(c_1c_2c_3)$, but also of a triplet of distinct IDs $(x_1x_2x_3)$.

Impossibility in Zero Rounds with IDs. Since the simplicial maps δ induced by the potential algorithms are name-preserving, they actually act on pairs (x, c) where x is an ID and c is a color, i.e., $\delta(p,(x,c)) = (p, \hat{\delta}(x,c))$ for some $\hat{\delta}$. For brevity, we identify $\hat{\delta}$ with δ. Let us assume that the IDs are from $\{1,\dots,R\}$, for some $R \geq 4$. That is, we consider now $\mathcal{C}_{2,[R]}$ for $R \geq 4$. By the pigeon-hole principle, there exists a set $I_1 \subseteq \{1,\dots,R\}$ with $|I_1| \geq R/2$ such that, for every $x, x' \in I_1$, $\delta(x,1) = \delta(x',1)$. Therefore, again by the pigeon-hole principle, there exists a set $I_2 \subseteq I_1$ with $|I_2| \geq |I_1|/2$ such that, for every $x, x' \in I_2$, $\delta(x,2) = \delta(x',2)$. Finally, there exists a set $I_3 \subseteq I_2$ with $|I_3| \geq |I_2|/2$ such that, for every $x, x' \in I_3$, $\delta(x,3) = \delta(x',3)$. Therefore, there exists a set $I \subseteq \{1,\dots,R\}$ with $|I| \geq R/8$ such that, for every $x, x' \in I$, $\delta(x,1) = \delta(x',1)$, $\delta(x,2) = \delta(x',2)$, and $\delta(x,3) = \delta(x',3)$. Therefore, whenever $R \geq 24$, the set I has size at least 3. Consider the sub-complex $\mathcal{C}'_{2,[R]}$ of $\mathcal{C}_{2,[R]}$ induced by the three smallest IDs in I – this sub-complex is isomorphic to $\mathcal{C}_{2,\varnothing}$ (Figure 5). More importantly, the mapping from $\mathcal{C}'_{2,[R]}$ to $\mathcal{M}_2$ depends only on the colors and not on the IDs, by the choice of I. Hence, if there was a mapping from $\mathcal{C}'_{2,[R]}$ to $\mathcal{M}_2$, then there would exist a mapping from $\mathcal{C}_{2,\varnothing}$ to $\mathcal{M}_2$, which we know does not exist.

It follows that there are no mappings from $\mathcal{C}_{2,[24]} = \mathcal{P}^{(0)}_{2,[24]}$ to $\mathcal{M}_2$ – see the second to right diagram in Figure 6. In other words, if the IDs are picked from a set of at least 24 values, then 3-coloring cannot be reduced to MIS in zero rounds.

Impossibility in One Rounds with IDs. We reduce the case with IDs to the case without IDs following the guideline introduced in [16]. We consider the 1-round protocol complex with IDs in a finite set X with at least 5 elements, denoted by $\mathcal{P}^{(1)}_{2,X}$. That is, $\mathcal{P}^{(1)}_{2,X} = \mathcal{P}^{(1)}_{2,[k]}$ with $k = |X|$. The vertices of this complex are of the form $(p_i,(xyz,abc))$ where $i \in \{-1,0,1\}$, $\{x,y,z\} \in \binom{X}{3}$, and $a,b,c \in \{1,2,3\}$ with $a \neq b$ and $b \neq c$. The facets of $\mathcal{P}^{(1)}_{2,X}$ are of the form $F = \{(p_{-1},(x'xy,a'ab)),(p_0,(xyz,abc)),(p_1,(yzz',bcc'))\}$. Let us assume the existence of a name-preserving name-independent simplicial map $\delta : \mathcal{P}^{(1)}_{2,X} \to \mathcal{M}_2$ (see the rightmost diagram in Figure 6). This map induces a labeling of the pairs (xyz,abc) with labels in $\{0,1\}$, where xyz is an ordered triplet of distinct IDs, and abc is an ordered triplet of colors in $\{1,2,3\}$. It follows that δ induces a labeling of the ordered triplets xyz of distinct IDs by labels in $\{0,1\}^{12}$, by applying δ to the 12 possible choices of color triplets. By Ramsey's Theorem [14], by taking the IDs in the set $X = \{1,\dots,R\}$ with R large enough, there exists a set Y of five IDs such that, for every two sets $\{x,y,z\}$ and $\{x,'y',z'\}$ of IDs in Y, with $x<y<z$ and $x'<y'<z'$, and for every ordered sequence abc of colors, $\delta(p_0,(xyz,abc)) = \delta(p_0,(x'y'z',abc))$. Let $\mathcal{P}^{(1)}_{2,Y}$ be the sub-complex of the 1-round protocol complex $\mathcal{P}^{(1)}_{2,X}$ induced by the vertices with IDs in Y ordered in increasing order. By construction of Y, δ is ID-oblivious on $\mathcal{P}^{(1)}_{2,Y}$. Now, let $\mathcal{P}^{(1)}_{2,\varnothing}$ as displayed on Figure 7. Let us define the map $\delta' : \mathcal{P}^{(1)}_{2,\varnothing} \to \mathcal{M}_2$ by $\delta'(p_i,abc) = \delta(p_i,(xyz,abc))$ where $\{x,y,z\} \subset Y$ and $x<y<z$. Note that δ' is well defined as δ is ID-oblivious on Y. Assuming $\delta : \mathcal{P}^{(1)}_{2,X} \to \mathcal{M}_2$ is simplicial yields that $\delta' : \mathcal{P}^{(1)}_{2,\varnothing} \to \mathcal{M}_2$ is simplicial as well. We have seen in Section 4.2 that such a simplicial mapping does not exist. It follows that there are no name-preserving name-independent simplicial maps $\delta : \mathcal{P}^{(1)}_{2,[R]} \to \mathcal{M}_2$ whenever R is large enough (see Figure 6).

5 Topology of LCL Tasks

Let S_d be the star of $d+1$ nodes, whose center node is named p_0, and the leaves are named p_i, for $i=1,\dots,d$. We consider algorithms for classes $\mathcal{G}\subseteq\mathcal{G}_d$ of graphs. Let $T=(\mathcal{L}_{in},\mathcal{S}_{in},\mathcal{L}_{out},\mathcal{S}_{out})$ be an LCL task for $\mathcal{G}\subseteq\mathcal{G}_d$. The input complex $\mathcal{I}_d$ (resp., output complex $\mathcal{O}_d$) associated with T is the complex where $\{(p_i,x_i):i\in\{0,\dots,d\}\}$ is a facet of $\mathcal{I}_d$ (resp., a facet of $\mathcal{O}_d$) if $x_i\in\mathcal{L}_{in}$ (resp., $\mathcal{L}_{out}$) for every $i\in\{0,\dots,d\}$, and the labeled star resulting from assigning label x_i to the node p_i of S_d for every $i\in\{0,\dots,d\}$ is in $\mathcal{S}_{in}$ (resp., $\mathcal{S}_{out}$). If the considered LCL task T imposes constraints on the correctness of the outputs as a function of the inputs, as in list-coloring, then the carrier map $\Delta:\mathcal{I}_d\to 2^{\mathcal{O}_d}$ specifies, for each facet $F\in\mathcal{I}_d$, the facets $\Delta(F)$ which are legal with respect to F. Otherwise, $\Delta(F)=\{\text{all facets of } \mathcal{O}_d\}$, for every facet F of $\mathcal{I}_d$.

Let $t\geq 0$, and let us fix a graph $G=(V,E)$ in $\mathcal{G}\subseteq\mathcal{G}_d$. In t rounds, every node in G acquires a *view* w, whose structure is isomorphic to a radius-t ball in G centered at that node, including the input labels and the IDs of the nodes in the ball. An ordered collection $w_0,\dots,w_d$ of views at distance t forms a collection of *mutually compatible* views for $\mathcal{G}$ if there exists a graph $G\in\mathcal{G}$, an assignment of input labels and IDs to the nodes of G, and a star S in G, with nodes $v_0,\dots,v_d$, centered at v_0, such that w_i is the view of v_i in G after t rounds, for $i=0,\dots,d$.

Let T be an LCL task for $\mathcal{G}\subseteq\mathcal{G}_d$, and let $t\geq 0$. The t-round protocol complex associated with T for a finite set X of IDs, is the complex $\mathcal{P}^{(t)}_{d,X}$ where $F=\{(p_i,w_i):i\in\{0,\dots,d\}\}$ is a facet of $\mathcal{P}^{(t)}_{d,X}$ if $w_0,\dots,w_d$ is an ordered collection of mutually compatible views at distance t for $\mathcal{G}$. The special case $t=0$ corresponds to $\mathcal{P}^{(0)}_{d,X}=\mathcal{I}_{d,X}$ where $\mathcal{I}_{d,X}$ in the input complex $\mathcal{I}_d$ extended with IDs in X. The set X must be large enough for all the nodes in the views w_i, $i=0,\dots,d$, to be provided with distinct IDs. Namely, $|X|\geq N(d,t+1)$, where $N(d,t+1)$ denotes the maximum number of nodes in the ball of radius t in a graph in $\mathcal{G}$.

Two mappings from $\mathcal{I}_{d,X}$ play a crucial role. The first is the simplicial map $\pi:\mathcal{I}_{d,X}\to\mathcal{I}_d$ defined by $\pi(p_i,(\mathsf{id},x))=(p_i,x)$ for every $i=0,\dots,d$, every $\mathsf{id}\in X$, and every $x\in\mathcal{L}_{in}$. The second is the carrier map $\Xi_t:\mathcal{I}_{d,X}\to 2^{\mathcal{P}^{(t)}_{d,X}}$ that specifies, for each facet $F\in\mathcal{I}_{d,X}$, the set $\Xi_t(F)$ of facets which may result from F after t rounds of computation in graphs in $\mathcal{G}$. Specifically, they are merely the facets of $\mathcal{P}^{(t)}_{d,X}$ for which the views $w_0,\dots,w_d$ are compatible with the IDs and inputs of $p_0,\dots,p_d$ in F.

Our main result is an analog of the generic lemma (see Lemma 1), but involving local complexes, even for tasks defined on arbitrarily large networks, and for arbitrarily large sets of IDs.

▶ **Theorem 3.** *Let $T=(\mathcal{I}_d,\mathcal{O}_d,\Delta)$ be an LCL task for $\mathcal{G}\subseteq\mathcal{G}_d$, and let $t\geq 0$.*

- *If there exists a distributed algorithm solving T in t rounds in the LOCAL model then, for every $R\geq N(d,t+1)$, there is a name-independent and name-preserving simplicial map $\delta:\mathcal{P}^{(t)}_{d,[R]}\to\mathcal{O}_d$ such that, for every facet $F\in\mathcal{I}_{d,[R]}$, $\delta(\Xi_t(F))\subseteq\Delta(\pi(F))$.*
- *There exists $R\geq N(d,t+1)$ satisfying that, if there is a name-independent and name-preserving simplicial map $\delta:\mathcal{P}^{(t)}_{d,[R]}\to\mathcal{O}_d$ such that, for every facet $F\in\mathcal{I}_{d,[R]}$, $\delta(\Xi_t(F))\subseteq\Delta(\pi(F))$, then there is a distributed algorithm solving T in t rounds in the LOCAL model.*

Proof. Let us fix an LCL task $T=(\mathcal{L}_{in},\mathcal{S}_{in},\mathcal{L}_{out},\mathcal{S}_{out})=(\mathcal{I}_d,\mathcal{O}_d,\Delta)$ for $\mathcal{G}$, and $t\geq 0$. Let ALG be a t-round algorithm for T. For any finite set X of IDs, let $\delta_X:\mathcal{P}^{(t)}_{d,X}\to\mathcal{O}_d$ defined by $\delta_X(p_i,w_i)=(p_i,\textsc{alg}(w_i))$, for every $i=0,\dots,d$. By construction, δ_X is name-independent, and name-preserving. To show that δ_X is simplicial, let $F'=\{(p_i,w_i):i\in\{0,\dots,d\}\}$ be a facet of the protocol complex $\mathcal{P}^{(t)}_{d,X}$. This facet is mapped to $\delta_X(F')=\{(p_i,\textsc{alg}(w_i)):$

$i \in \{0, \dots, d\}\}$. Since ALG solves T, every output $\text{ALG}(w_i)$ is in $\mathcal{L}_{out}$, and the labeled star resulting from assigning label $\text{ALG}(w_i)$ to the node p_i of the star S_d, for every $i \in \{0, \dots, d\}$, is in $\mathcal{S}_{out}$. It follows that $\delta_X(F')$ is a facet of $\mathcal{O}_d$, and thus δ_X is simplicial. Moreover, if the facet F' belongs to the image $\Xi_t(F)$ of a facet F of $\mathcal{I}_{d,X}$, since ALG solves T, it follows that $\delta_X(F') \in \Delta(\pi(F))$ as desired. So, the existence of an algorithm ALG guarantees the existence of a simplicial map δ_X satisfying the requirements of the theorem for *every* large enough set X of IDs.

We now show that, to guarantee the existence of an algorithm, it is sufficient to guarantee the existence of a simplicial map δ_X just for *one* specific set $X = [R]$. In order to identify R, we follow the same guideline as the specific impossibility proof in Section 4.3, using Ramsey's theorem. Note that the number of possible balls of radius t in graphs of $\mathcal{G}$ is finite, and depends only on t and d. Given such a ball B, there are finitely many ways of assigning input labels to the vertices of B. The number of assignments depends only on the structure of B, and on $|\mathcal{L}_{in}|$. (It may also depend on $\mathcal{S}_{in}$, but in the worst case, all assignments are possible.) Let us enumerate all the labeled balls in $\mathcal{G}$ as $B^{(1)}, \dots, B^{(k)}$. The number k of such labeled balls depends only on d, t, and $|\mathcal{L}_{in}|$. (It may also depend on $\mathcal{G}$, but it is upper bounded by a function of d, t, and $|\mathcal{L}_{in}|$.)

For every labeled ball $B^{(i)}$, $i = 1, \dots, k$, let $\nu_i = |B^{(i)}|$. Let us rank the vertices of $B^{(i)}$ arbitrarily from 1 to ν_i, and let Σ_i be the set of all permutations of $\{1, \dots, \nu_i\}$. To every $\pi \in \Sigma_i$ corresponds a labeled ball $B^{(i)}_\pi$ in which the rank of the vertices is determined by π. Now, let X be a finite set of IDs with $|X| \geq N(d, t+1)$. We consider all possible identity-assignments with IDs in X to the nodes of the labeled balls with ranked vertices, $B^{(i)}_\pi$, $i = 1, \dots, k$, $\pi \in \Sigma_i$, as follows. For every $S \subseteq X$ with $|S| = N(d, t)$, let us order the IDs in S in increasing order. Given a ranked labeled ball $B^{(i)}_\pi$, i.e., a labeled ball $B^{(i)}$ whose vertices are ranked by some permutation $\pi \in \Sigma_i$, the IDs in S are assigned to the nodes of $B^{(i)}_\pi$ by assigning the jth smallest ID in S to the node ranked $\pi(j)$ in $B^{(i)}_\pi$, for $j = 1, \dots, \nu_i$. By picking all $i = 1, \dots, k$, all $\pi \in \Sigma_i$, and all $S \subseteq X$, we obtain all possible views resulting from performing a t-round algorithm in $\mathcal{G}$ with IDs taken from X. Let us order these views as $w^{(1)}, \dots, w^{(h)}$, where the views induced by $B^{(1)}$ are listed first, then the views induced by $B^{(2)}$, etc., until the views induced by $B^{(k)}$. Moreover, for a given $i \in \{1, \dots, k\}$, the views corresponding to the labeled ball $B^{(i)}$ are listed according to the lexicographic order of the permutations in Σ_i. Note that the number h of views depends only on d, t, $|\mathcal{L}_{in}|$, and $|X|$. Each set S is then "colored" by

$$c(S) = (\delta_X(p_0, w^{(1)}), \dots, \delta_X(p_0, w^{(h)})) \in \{1, \dots, |\mathcal{L}_{out}|\}^h.$$

In this way, the set $\binom{X}{N(d,t)}$ is partitioned into $|\mathcal{L}_{out}|^h$ classes. Thanks to Ramsey's Theorem, by taking set

$$X = [R] \text{ with } R = R(a, b, c) \text{ for } a = |\mathcal{L}_{out}|^h, \text{ and } b = c = N(d, t+1),$$

we are guaranteed that there exists a set Y of at least $N(d, t+1)$ IDs such that every two sets S and S' of $N(d, t)$ IDs in Y are given the same color $c(S) = c(S')$. In other words, for any ball B of radius t in a graph from $\mathcal{G}$, and for every valid assignment of inputs values to the nodes of B, if one assigns the IDs in S and S' in the same manner (i.e., the ith smallest ID of S is assigned to the same node as the ith smallest ID of S'), then $\delta_X(p_0, w) = \delta_X(p_0, w')$, where w and w' are the views resulting from assigning IDs from S and S' to the nodes, respectively.

Now, let us define the following t-round algorithm ALG for T. Actually, this is precisely the order-invariant algorithm constructed in [16]. Every node v collects the data available in its centered ball $B = B_G(v,t)$ of radius t in the actual graph $G \in \mathcal{G}$. Note that B contains IDs, and input values. Node v reassigns the IDs to the nodes of B by using the $|B|$ smallest IDs in Y, and assigning these IDs to the nodes of B in the order respecting the order of the actual IDs assigned to the nodes of B. Then node v considers the view w after reassignment of the IDs, and outputs $\text{ALG}(w) = \delta_X(p_0, w)$. Note that δ_X returns values in $\mathcal{L}_{out}$, and thus ALG is well defined.

To show correctness, let us consider a star $v_0, \dots, v_d$ centered at v_0 in some graph $G \in \mathcal{G}$. Performing ALG in G, each of these $d+1$ nodes acquires a view of radius t. These views are mutually compatible. Let us reassign the IDs in the ball of radius $t+1$ centered at v_0 in G, using the at most $N(d, t+1)$ smallest IDs in Y, and assigning these IDs to the nodes of the ball B of radius $t+1$ centered at v_0, in the order respecting the order of the actual IDs assigned to the nodes of B. The resulting views $w_0, \dots, w_d$ of the $d+1$ nodes $v_0, \dots, v_d$ remain mutually compatible. It follows that if these $d+1$ nodes would output $\delta_X(p_0, w_0), \dots, \delta_X(p_d, w_d)$, respectively, then the resulting star would be good. We claim that this is exactly what occurs with ALG. Indeed, first, δ_X is name-independent, and thus $\delta_X(p_0, w) = \delta_X(p_i, w)$ for every $i = 1, \dots, d$. Second, and more importantly, by the construction of Y, the actual *values* of the IDs do not matter, but solely their *relative order*. The reassignment of IDs performed at each of the nodes $v_0, \dots, v_d$ is different from the reassignment of IDs in the ball B of radius $t+1$ around v_0, but the relative order of these IDs is preserved as it is governed by the relative order of the original IDs in B. As a consequence, the nodes of S_d correctly output $\delta_X(p_0, w_0), \dots, \delta_X(p_d, w_d)$ in ALG, as desired. ◀

To illustrate Theorem 3, we reprove the celebrated result by Linial [15] regarding 3-coloring the n-node ring in at least $\frac{1}{2}\log^* n - 1$ rounds (see also [2, 3, 13]), which can be obtained by iterating Corollary 4.

▶ **Corollary 4.** *Let $t \geq 1$, $k \geq 2$, $n \geq 1$, and $N \geq n$. If there is a t-round algorithm for k-coloring $C_n = (v_1, \dots, v_n)$ whenever the IDs in $[N]$ are assigned to consecutive nodes v_i, v_{i+1}, $i \in \{1, \dots, n-1\}$, in increasing order of their indices, then there is a $(t-1)$-round algorithm for 2^{2^k}-coloring C_n under the same constraints of the identity assignment.*

Proof. Our aim is to find $\delta_{t-1} : \mathcal{P}^{(t-1)}_{[R]} \to \mathcal{O}_{2^{2^k}}$ where $\mathcal{O}_{2^{2^k}}$ is the output complex for 2^{2^k}-coloring C_n. For this purpose, we follows the approach illustrated on Figure 10. That is, first, we identify a functor Φ on a category corresponding to a subclass of simplicial complexes. From the simplicial map $\delta_t : \mathcal{P}^{(t)}_{[R]} \to \mathcal{O}_k$, we derive the simplicial map $\Phi(\delta_t) : \Phi(\mathcal{P}^{(t)}_{[R]}) \to \Phi(\mathcal{O}_k)$. Then we show that $\Phi(\mathcal{O}_k) \subseteq \mathcal{O}_{2^{2^k}}$, and therefore $\Phi(\delta_t)$ maps $\Phi(\mathcal{P}^{(t)}_{[R]})$ to $\mathcal{O}_{2^{2^k}}$. Finally, we identify a simplicial map $f : \mathcal{P}^{(t-1)}_{[R]} \to \Phi(\mathcal{P}^{(t)}_{[R]})$ enabling to conclude that $\delta_{t-1} : \mathcal{P}^{(t-1)}_{[R]} \to \mathcal{O}_{2^{2^k}}$ defined by $\delta_{t-1} = \Phi(\delta_t) \circ f$ satisfies the hypotheses of Theorem 3, showing the existence of a $(t-1)$-round algorithm for 2^{2^k}-coloring C_n.

More specifically, given any complex $\mathcal{K}$ with vertices (p_i, v) with $i \in \{-1, 0, 1\}$, and $v \in V$ where V is a finite set of values, we define the functor Φ as follows. The complex $\Phi(\mathcal{K})$ is on the set of vertices $(p_i, \mathbf{S})$ where $\mathbf{S} = \{S_1, \dots, S_\ell\}$ for some $\ell \geq 0$, and $S_i \subseteq V$ for every $i = 1, \dots, \ell$. A set $\{(p_{-1}, \mathbf{S}_{-1}), (p_0, \mathbf{S}_0), (p_1, \mathbf{S}_1)\}$ forms a facet of $\Phi(\mathcal{K})$ if for every $i \in \{0, 1\}$,

$$\exists S \in \mathbf{S}_{i-1} \;\; \forall S' \in \mathbf{S}_i \;\; \exists v' \in S' \;\; \forall v \in S \;:\; \{(p_{i-1}, v), (p_i, s')\} \in \mathcal{K}. \qquad (1)$$

$$\begin{tikzcd}
\mathcal{P}^{(t)}_{[R]} \arrow[r, "\Phi"] \arrow[d, "\delta_t"'] & \Phi(\mathcal{P}^{(t)}_{[R]}) \arrow[d, "\Phi(\delta_t)"'] & \mathcal{P}^{(t-1)}_{[R]} \arrow[l, "f"'] \arrow[dl, "\delta_{t-1}"'] \\
\mathcal{O}_k \arrow[r, "\Phi"] & \Phi(\mathcal{O}_k) \subseteq \mathcal{O}_{2^{2^k}} &
\end{tikzcd}$$

Figure 10 Commutative diagrams in the proof of Corollary 4.

Given a simplicial map $\psi : \mathcal{A} \to \mathcal{B}$ the map $\Phi(\psi)$ is defined as

$$\Phi(\psi)(p_i, \mathbf{S}) = \Big(p_i, \Big\{\{\pi_2\circ\psi(p_i, v_{1,1}), \dots, \pi_2\circ\psi(v_{1,s_1})\}, \dots, \{\pi_2\circ\psi(v_{\ell,1}), \dots, \pi_2\circ\psi(v_{\ell,s_\ell})\}\Big\}\Big)$$

for every $i = \{-1, 0, 1\}$, and every $\mathbf{S} = \{S_1, \dots, S_\ell\}$ with $S_j = \{v_{j,1}, \dots, v_{j,s_j}\}$ and $s_j \geq 0$, where $\pi_2 : \mathcal{B} \to V$ is the mere projection $\pi_2(p_i, v) = v$ for every value v. By construction, $\Phi(\psi) : \Phi(\mathcal{A}) \to \Phi(\mathcal{B})$ is simplicial. Note that if ψ is name-invariant and name-preserving, then $\Phi(\psi)$ as well.

Next, we observe that $\Phi(\mathcal{O}_k)$ is a sub-complex of $\mathcal{O}_{2^{2^k}}$. To see why, note first that Φ maps vertices of $\mathcal{O}_k$ to vertices of $\mathcal{O}_{2^{2^k}}$. Moreover, a facet $F = \{(p_{-1}, \mathbf{S}_{-1}), (p_0, \mathbf{S}_0), (p_1, \mathbf{S}_1)\}$ of $\Phi(\mathcal{O}_k)$ is a facet of $\mathcal{O}_{2^{2^k}}$. Indeed, Eq. (1) guarantees that there exists a set S in $\mathbf{S}_{-1}$ such that for every set S' in $\mathbf{S}_0$, there exists a color v' in S' that is different from all the colors in S. It follows that $S \notin \mathbf{S}_0$, and therefore $\mathbf{S}_{-1} \neq \mathbf{S}_0$. By the same argument, $\mathbf{S}_0 \neq \mathbf{S}_1$, and thus F is a facet of $\mathcal{O}_{2^{2^k}}$, as claimed. Finally, we define the simplicial map $f : \mathcal{P}^{(t-1)}_{[R]} \to \Phi(\mathcal{P}^{(t)}_{[R]})$ as follows. Let us consider a vertex $(p_i, w) \in \mathcal{P}^{(t-1)}_{[R]}$, with $w = (z_{-(t-1)}, \dots, z_{-1}, z_0, z_1, \dots, z_{t-1}) \in [R]^{2t-1}$ with $z_{-(t-1)} < \cdots < z_{t-1}$. For every $b \in [R]$ with $b > z_{t-1}$, let $W_i^b = \{awb : a \in [R], a < z_{-(t-1)}\}$, and let $\mathbf{W}_i = \{W_i^b : b \in [R], b > z_{t-1}\}$. We set $f(p_i, w) = (p_i, \mathbf{W}_i)$. This mapping maps every vertex of $\mathcal{P}^{(t-1)}_{[R]}$ to a vertex of $\Phi(\mathcal{P}^{(t)}_{[R]})$. Let us show that f is simplicial. For this purpose, let us consider a facet

$$F = \{(p_{-1}, x'xw), (p_0, xwy), (p_1, wyy')\}$$

of $\mathcal{P}^{(t-1)}_{[R]}$. Here $w = (z_{-(t-2)}, \dots, z_{-1}, z_0, z_1, \dots, z_{t-2}) \in [R]^{2t-3}$ with $x' < x < z_{-(t-2)} < \cdots < z_{t-2} < y < y'$. Let us consider the two sets $W_{-1}^y \in \mathbf{W}_{-1}$ and $W_0^{y'} \in \mathbf{W}_0$. We claim that these are the two sets witnessing the validity of Eq. (1) for establishing the fact that $f(F)$ is a facet of $\Phi(\mathcal{P}^{(t)}_{[R]})$. To see why, let $W_0^b \in \mathbf{W}_0$, and let $x'xwyb \in W_0^b$. The view $ax'xwy$ for p_{-1} is compatible with the view $x'xwyb$ for p_0, for every $a < x'$. Therefore, for every set $W_0^b \in \mathbf{W}_0$, there exists a view $x'xwyb \in W_0^b$ such that, for every view $ax'xwy \in W_{-1}^y$,

$$\{(p_{-1}, ax'xwy), (p_0, x'xwyb)\} \in \mathcal{P}^{(t)}_{[R]}.$$

Hence Eq. (1) is satisfied for p_{-1} and p_0. By the same arguments, using $W_0^{y'}$ instead of W_{-1}^y, Eq. (1) is satisfied for p_{-1} and p_0, from which it follows that $f(F)$ is a facet of $\Phi(\mathcal{P}^{(t)}_{[R]})$. We conclude that f is simplicial. Since both f and $\Phi(\delta)$ are simplicial, the map $\delta' = \Phi(\delta) \circ f$ is simplicial too, which completes the proof by application of Theorem 3. ◀

6 Conclusion and Further Work

This paper shows that the study of algorithms for solving LCL tasks in the LOCAL model can be achieved by considering simplicial complexes whose sizes are independent of the number of nodes, and independent of the number of possible IDs that could be assigned to these nodes.

Two main directions for further work can be identified. A first direction is understanding topological properties of the carrier map $\Xi_t : \mathcal{I}_{d,X} \to \mathcal{P}^{(t)}_{d,X}$ occurring in the LOCAL model depending on the network. Another direction is understanding what governs the existence of the simplicial map $\delta : \mathcal{P}^{(t)}_{d,X} \to \mathcal{O}$ depending on the considered task.

References

1 Hagit Attiya, Armando Castañeda, Maurice Herlihy, and Ami Paz. Bounds on the step and namespace complexity of renaming. *SIAM J. Comput.*, 48(1):1–32, 2019. doi:10.1137/16M1081439.

2 Alkida Balliu, Sebastian Brandt, Juho Hirvonen, Dennis Olivetti, Mikaël Rabie, and Jukka Suomela. Lower bounds for maximal matchings and maximal independent sets. In *60th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 481–497, 2019. doi:10.1109/FOCS.2019.00037.

3 Sebastian Brandt. An automatic speedup theorem for distributed problems. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 379–388, 2019. doi:10.1145/3293611.3331611.

4 Armando Castañeda, Pierre Fraigniaud, Ami Paz, Sergio Rajsbaum, Matthieu Roy, and Corentin Travers. A topological perspective on distributed network algorithms. In *26th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, LNCS 11639, pages 3–18. Springer, 2019. doi:10.1007/978-3-030-24922-9_1.

5 Armando Castañeda and Sergio Rajsbaum. New combinatorial topology bounds for renaming: the lower bound. *Distributed Computing*, 22(5-6):287–301, 2010. doi:10.1007/s00446-010-0108-2.

6 Armando Castañeda and Sergio Rajsbaum. New combinatorial topology bounds for renaming: The upper bound. *J. ACM*, 59(1):3:1–3:49, 2012. doi:10.1145/2108242.2108245.

7 Lisbeth Fajstrup, Eric Goubault, Emmanuel Haucourt, Samuel Mimram, and Martin Raussen. *Directed Algebraic Topology and Concurrency.* Springer, 2016.

8 Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985. doi:10.1145/3149.214121.

9 Pierre Fraigniaud and Ami Paz. The topology of local computing in networks. Tech. Report arXiv abs/2003.03255, 2020. arXiv:2003.03255.

10 Éric Goubault, Samuel Mimram, and Christine Tasson. Geometric and combinatorial views on asynchronous computability. *Distributed Computing*, 31(4):289–316, 2018. doi:10.1007/s00446-018-0328-4.

11 Maurice Herlihy, Dmitry N. Kozlov, and Sergio Rajsbaum. *Distributed Computing Through Combinatorial Topology.* Morgan Kaufmann, 2013. URL: https://store.elsevier.com/product.jsp?isbn=9780124045781.

12 Maurice Herlihy and Nir Shavit. The topological structure of asynchronous computability. *J. ACM*, 46(6):858–923, 1999. doi:10.1145/331524.331529.

13 Juhana Laurinharju and Jukka Suomela. Brief announcement: Linial's lower bound made easy. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 377–378, 2014. doi:10.1145/2611462.2611505.

14 Ronald L.Graham, Bruce L. Rothschild, Joel H. Spencer, and József Solymosi. *Ramsey Theory.* John Wiley and Sons, 2015.

15 Nathan Linial. Locality in distributed graph algorithms. *SIAM J. Comput.*, 21(1):193–201, 1992. doi:10.1137/0221015.

16 Moni Naor and Larry J. Stockmeyer. What can be computed locally? *SIAM J. Comput.*, 24(6):1259–1277, 1995. doi:10.1137/S0097539793254571.

17 David Peleg. *Distributed Computing: A Locality-Sensitive Approach.* SIAM, 2001.

18 Michael E. Saks and Fotios Zaharoglou. Wait-free k-set agreement is impossible: The topology of public knowledge. *SIAM J. Comput.*, 29(5):1449–1483, 2000. doi:10.1137/S0097539796307698.

19 Jukka Suomela. Survey of local algorithms. *ACM Comput. Surv.*, 45(2):24:1–24:40, 2013. doi:10.1145/2431211.2431223.

The Complexity of Verifying Loop-Free Programs as Differentially Private

Marco Gaboardi
Boston University, MA, USA

Kobbi Nissim
Georgetown University, Washington, DC, USA

David Purser
University of Warwick, Coventry, UK
Max Planck Institute for Software Systems, Saarbrücken, Germany

Abstract

We study the problem of verifying differential privacy for loop-free programs with probabilistic choice. Programs in this class can be seen as randomized Boolean circuits, which we will use as a formal model to answer two different questions: first, deciding whether a program satisfies a prescribed level of privacy; second, approximating the privacy parameters a program realizes. We show that the problem of deciding whether a program satisfies ε-differential privacy is $\mathbf{coNP}^{\#\mathbf{P}}$-complete. In fact, this is the case when either the input domain or the output range of the program is large. Further, we show that deciding whether a program is (ε, δ)-differentially private is $\mathbf{coNP}^{\#\mathbf{P}}$-hard, and in $\mathbf{coNP}^{\#\mathbf{P}}$ for small output domains, but always in $\mathbf{coNP}^{\#\mathbf{P}^{\#\mathbf{P}}}$. Finally, we show that the problem of approximating the level of differential privacy is both **NP**-hard and **coNP**-hard. These results complement previous results by Murtagh and Vadhan [35] showing that deciding the optimal composition of differentially private components is #**P**-complete, and that approximating the optimal composition of differentially private components is in **P**.

2012 ACM Subject Classification Security and privacy; Theory of computation → Probabilistic computation

Keywords and phrases differential privacy, program verification, probabilistic programs

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.129

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

Related Version A full version of the paper is available at `https://arxiv.org/abs/1911.03272`.

Funding M. G. and K. N. were supported by NSF grant No. 1565387 TWC: Large: Collaborative: Computing Over Distributed Sensitive Data. D. P. was supported by the UK EPSRC Centre for Doctoral Training in Urban Science (EP/L016400/1).

Acknowledgements Research partially done while M.G. and K.N. participated in the "Data Privacy: Foundations and Applications" program held at the Simons Institute, UC Berkeley in spring 2019.

1 Introduction

Differential privacy [22] is currently making significant strides towards being used in large scale applications. Prominent real-world examples include the use of differentially private computations by the US Census' OnTheMap project[1], applications by companies such as Google and Apple [24, 36, 4, 18], and the US Census' plan to deploy differentially private releases in the upcoming 2020 Decennial [1].

[1] `https://onthemap.ces.census.gov`

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 129; pp. 129:1–129:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

More often than not, algorithms and their implementations are analyzed "on paper" to show that they provide differential privacy. This analysis – a proof that the outcome distribution of the algorithm is stable under the change in any single individual's information – is often intricate and may contain errors (see [32] for an illuminating discussion about several wrong versions of the sparse vector algorithm which appeared in the literature). Moreover, even if it is actually differentially private, an algorithm may be incorrectly implemented when used in practice, e.g., due to coding errors, or because the analysis makes assumptions which do not hold in finite computers, such as the ability to sample from continuous distributions (see [34] for a discussion about privacy attacks on naive implementations of continuous distributions). Verification tools may help validate, given the code of an implementation, that it would indeed provide the privacy guarantees it is intended to provide. However, despite the many verification efforts that have targeted differential privacy based on automated or interactive techniques (see, e.g.,[37, 9, 40, 25, 7, 44, 6, 2, 14, 15]), little is known about the complexity of some of the basic problems in this area. Our aim is to clarify the complexity of some of these problems.

In this paper, we consider the computational complexity of determining whether programs satisfy (ε, δ)-differential privacy. The problem is generally undecidable, and we hence restrict our attention to probabilistic loop-free programs, which are part of any reasonable programming language supporting random computations. To approach this question formally, we consider probabilistic circuits. The latter are Boolean circuits with input nodes corresponding both to input bits and to uniformly random bits ("coin flips") where the latter allow the circuit to behave probabilistically (see Figure 1). We consider both decision and approximation versions of the problem, where in the case of decision the input consists of a randomized circuit and parameters ε, δ and in the case of approximation the input is a randomized circuit, the desired approximation precision, and one of the two parameters ε, δ. In both cases, complexity is measured as function of the total input length in bits (circuit and parameters).

Previous works have studied the complexity of composing differentially private components. For any k differentially private algorithms with privacy parameters $(\varepsilon_1, \delta_1), \ldots, (\varepsilon_k, \delta_k)$, it is known that their composition is also differentially private [22, 23, 35], making composition a powerful design tool for differentially private programs. However, not all interesting differentially private programs are obtained by composing differentially private components, and a goal of our work is to understand what is the complexity of verifying that full programs are differentially private, and how this complexity differs from the one for programs which result of composing differentially private components.

Regarding the resulting parameters, the result of composing the k differentially private algorithms above results in $(\varepsilon_g, \delta_g)$-differentially private for a multitude of possible $(\varepsilon_g, \delta_g)$ pairs. Murtagh and Vadhan showed that determining the minimal ε_g given δ_g is $\#\mathbf{P}$-complete [35]. They also gave a polynomial time approximation algorithm that computes ε_g to arbitrary accuracy, giving hope that for "simple" programs deciding differential privacy or approximating of privacy parameters may be tractable. Unfortunately, our results show that this is not the case.

1.1 Contributions

Following the literature, we refer to the variant of differential privacy where $\delta = 0$ as *pure* differential privacy and to the variant where $\delta > 0$ as *approximate* differential privacy. We contribute in three directions.

- **Bounding pure differential privacy.** We show that determining whether a randomized circuit is ε-differentially private is $\mathbf{coNP}^{\#\mathbf{P}}$-complete.[2] To show hardness in $\mathbf{coNP}^{\#\mathbf{P}}$ we consider a complement to the problem E-Maj-Sat [31], which is complete for $\mathbf{NP}^{\#\mathbf{P}}$ [13]. In the complementary problem, All-Min-Sat, given a formula ϕ over $n+m$ variables the task is to determine if for all allocations $\boldsymbol{x} \in \{0,1\}^n$, $\phi(\boldsymbol{x}, \boldsymbol{y})$ evaluates to true on no more than $\frac{1}{2}$ of allocations to $\boldsymbol{y} \in \{0,1\}^m$.
- **Bounding approximate differential privacy.** Turning to the case where $\delta > 0$, we show that determining whether a randomized circuit is (ε, δ)-differentially private is $\mathbf{coNP}^{\#\mathbf{P}}$-complete when the number of output bits is small relative to the total size of the circuit and otherwise between $\mathbf{coNP}^{\#\mathbf{P}}$ and $\mathbf{coNP}^{\#\mathbf{P}^{\#\mathbf{P}}}$.
- **Approximating the parameters ε and δ.** Efficient approximation algorithms exist for optimal composition [35], and one might expect the existence of polynomial time algorithms to approximate ε or δ in randomized circuits. We show this is $\mathbf{NP}$-hard and $\mathbf{coNP}$-hard, and therefore an efficient algorithm does not exist (unless $\mathbf{P} = \mathbf{NP}$).

Our results show that for loop-free programs with probabilistic choice directly verifying whether a program is differentially private is intractable. These results apply to programs in any reasonable programming language supporting randomized computations. Hence, they set the limits on where to search for automated techniques for these tasks.

The relation to quantitative information flow

Differential privacy shares similarities with quantitative information flow [17, 27], which is an entropy-based theory measuring how secure a program is. Alvim et al. [3] showed that a bound on pure differential privacy implies a bound on quantitative information flow. So, one could hope that bounding differential privacy could be easier than bounding quantitative information flow. Yasuoka and Terauchi [42] have shown that bounding quantitative information flow for loop free boolean programs with probabilistic choice is $\mathbf{PP}$-hard (but in $\mathbf{PSPACE}$). In contrast, our results show that bounding pure differential privacy is $\mathbf{coNP}^{\#\mathbf{P}}$-complete. Chadha et al. [11] showed the problem to be $\mathbf{PSPACE}$-complete for boolean programs with loops and probabilistic choice (notice that this would be not true for programs with integers). We leave the analogous question for future works.

2 Preliminaries

Numbers

By a *number given as a rational* we mean a number of the form $\frac{x}{y}$ where x, y are given as binary integers.

2.1 Loop-free probabilistic programs

We consider a simple loop-free imperative programming language built over Booleans, and including probabilistic choice.

$$
\begin{array}{lcll}
x & ::= & [\mathrm{a{-}z}]^+ & \text{(variable identifiers)}\\
b & ::= & \texttt{true} \mid \texttt{false} \mid \texttt{random} \mid x \mid b \wedge b \mid b \vee b \mid \neg b & \text{(boolean expressions)}\\
c & ::= & \texttt{SKIP} \mid x := b \mid c;\ c \mid \texttt{if } b \texttt{ then } c \texttt{ else } c & \text{(commands)}\\
t & ::= & x \mid t, x & \text{(list of variables)}\\
p & ::= & \texttt{input}(t);\ c;\ \texttt{return}(t) & \text{(programs)}
\end{array}
$$

[2] The class $\mathbf{coNP}^{\#\mathbf{P}}$ is contained in $\mathbf{PSPACE}$ and contains the polynomial hierarchy (as, per Toda's Theorem, $\mathbf{PH} \subseteq \mathbf{P}^{\#\mathbf{P}}$).

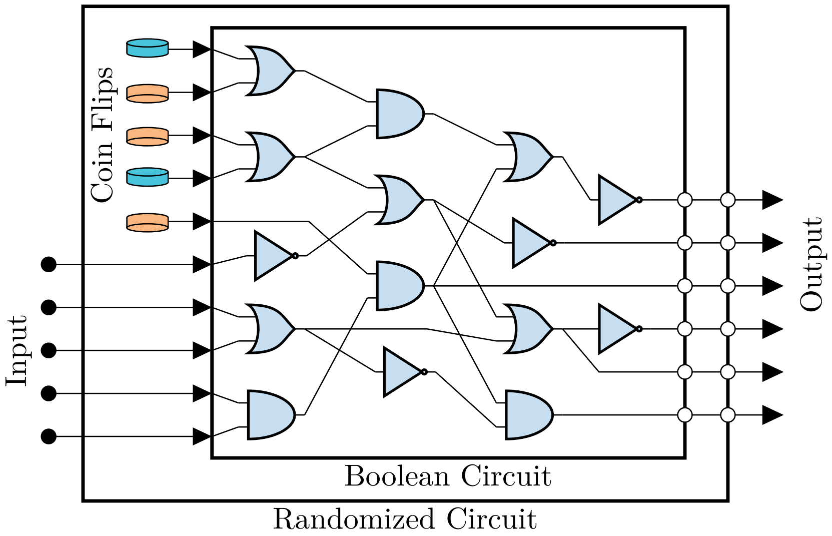


Figure 1 Example randomized circuit.

Probabilistic programs [30] extend standard programs with the addition of coin tosses; this is achieved by the probabilistic operation `random`, which returns either `true` or `false` with equal probability. A standard operation, sometimes denoted by $c \oplus c$, which computes one of the two expressions with probability $\frac{1}{2}$ each is achieved with `if random then` c `else` c. The notation $c \oplus c$ is avoided as $\oplus$ refers to *exclusive or* in this paper.

The semantics of the programming language are standard and straight forward. Without loss of generality, each variable assignment is final, that is, each assignment must go to a fresh variable. Looping behaviour is not permitted, although bounded looping can be encoded by unrolling the loop.

▶ Remark 1. Our results also hold when the language additionally supports integers and the associated operations (e.g. $+, \times, -, \geq, =$ etc.), providing the integers are of a bounded size. Such a language is equally expressive as the language presented here. Further details are given in the full version of the paper.

2.2 Probabilistic circuits

▶ **Definition 2.** *A Boolean circuit ψ with n inputs and ℓ outputs is a directed acyclic graph $\psi = (V, E)$ containing n input vertices with zero in-degree, labeled $X_1, \ldots, X_n$ and ℓ output vertices with zero out-degree, labeled $O_1, \ldots, O_\ell$. Other nodes are assigned a label in $\{\wedge, \vee, \neg\}$, with vertices labeled $\neg$ having in-degree one and all others having in-degree two. The size of ψ, denoted $|\psi|$, is defined to be $|V|$. A* randomized circuit *has m additional random input vertices labeled $R_1, \ldots, R_m$.*

Given an input string $\boldsymbol{x} = (x_1, \ldots, x_n) \in \{0,1\}^n$, the circuit is evaluated as follows. First, the values $x_1, \ldots, x_n$ are assigned to the nodes labeled $X_1, \ldots, X_n$. Then, m bits $\boldsymbol{r} = (r_1, \ldots, r_m)$ are sampled uniformly at random from $\{0,1\}^m$ and assigned to the nodes labeled $R_1, \ldots, R_m$. Then, the circuit is evaluated in topological order in the natural way. E.g., let v be a node labeled $\wedge$ with incoming edges $(u_1, v), (u_2, v)$ where u_1, u_2 were assigned values z_1, z_2 then v is assigned the value $z_1 \wedge z_2$. The outcome of ψ is $(o_1, \ldots, o_\ell)$, the concatenation of values assigned to the ℓ output vertices $O_1, \ldots, O_\ell$.

For input $\boldsymbol{x} \in \{0,1\}^n$ and event $E \subseteq \{0,1\}^\ell$ we have

$$\Pr[\psi(\boldsymbol{x}) \in E] = \frac{|\{\boldsymbol{r} \in \{0,1\}^m \ : \ \psi(\boldsymbol{x}, \boldsymbol{r}) \in E\}|}{2^m}.$$

▶ Remark 3. The operators, $\wedge, \vee$ and $\neg$ are functionally complete. However, we will also use $\oplus$ (exclusive or), such that $p \oplus q \iff (p \vee q) \wedge \neg(p \wedge q)$.

2.3 Equivalence of programs and circuits

▶ **Lemma 4.** *A loop-free probabilistic program can be converted into an equivalent probabilistic boolean circuit in linear time in the size of the program (and vice-versa).*

Proof sketch. It is clear that a probabilistic circuit can be expressed as a probabilistic program using just boolean operations by expressing a variable for each vertex after sorting the vertices in topological order.

To convert a probabilistic Boolean program into a probabilistic circuit, each of the commands can be handled using a fixed size sub-circuit, each of which can be composed together appropriately. ◀

Given the equivalence between loop-free probabilistic programs and probabilistic circuits, the remainder of the paper will use probabilistic circuits.

2.4 Differential privacy in probabilistic circuits

Let X be any input domain. An input to a differentially private analysis would generally be an array of elements from a data domain X, each corresponding to the information of an individual, i.e., $\boldsymbol{x} = (x_1, \ldots, x_n) \in X^n$.

The definition of differential privacy depends on adjacency between inputs, we define *neighboring* inputs.

▶ **Definition 5.** *Inputs $\boldsymbol{x} = (x_1, \ldots, x_n)$ and $\boldsymbol{x}' = (x'_1, \ldots, x'_n) \in X^n$ are called* neighboring *if there exist $i \in [n]$ s.t. if $j \neq i$ then $x_j = x'_j$.*

In this work, we will consider input domains with finite representation. Without loss of generality we set $X = \{0,1\}^k$ and hence an array $x = (x_1, \ldots, x_n)$ can be written as a sequence of nk bits, and given as input to a (randomized) circuit with nk inputs. Our lower bounds work already for for $k = 1$ and our upper bounds are presented using $k = 1$ but generalise to all k.

▶ **Definition 6** (Differential Privacy [22, 21])**.** *A probabilistic circuit ψ is (ε, δ)-differentially private if for all neighboring $\boldsymbol{x}, \boldsymbol{x}' \in X^n$ and for all $E \subseteq \{0,1\}^\ell$,*

$$\Pr[\psi(\boldsymbol{x}) \in E] \leq e^\varepsilon \cdot \Pr[\psi(\boldsymbol{x}') \in E] + \delta.$$

Following common use, we refer to the case where $\delta = 0$ as *pure* differential privacy and to the case where $\delta > 0$ as *approximate* differential privacy. When omitted, δ is understood to be zero.

2.5 Problems of deciding and approximating differential privacy

We formally define our three problems of interest.

▶ **Definition 7.** *The problem* Decide-ε-DP *asks, given ε and ψ, if ψ is ε-differentially private. We assume ε is given by the input e^ε as a rational number.*

▶ **Definition 8.** *The problem* Decide-ε,δ-DP *asks, given ε, δ and ψ, if ψ is (ε,δ)-differentially private. We assume ε is given by the input e^ε as a rational number.*

▶ **Definition 9.** *Given an approximation error $\gamma > 0$, the* Approximate-δ *problem and the* Approximate-ε *problem, respectively, ask:*

- *Given ε, find $\hat{\delta} \in [0,1]$, such that $0 \leq \hat{\delta} - \delta \leq \gamma$, where δ is the minimal value such that ψ is (ε,δ)-differentially private.*
- *Given δ, find $\hat{\varepsilon} \geq 0$, such that $0 \leq \hat{\varepsilon} - \varepsilon \leq \gamma$, where ε is the minimal value such that ψ is (ε,δ)-differentially private.*

2.6 The class $\mathbf{coNP}^{\#\mathbf{P}}$

The complexity class $\#\mathbf{P}$ is the counting analogue of **NP** problems. In particular #Sat, the problem of counting the number of satisfying assignments of a given a boolean formula ϕ on n variables, is complete for $\#\mathbf{P}$. Similarly #CircuitSat, the problem of counting the satisfying assignments of a circuit with a single output, is complete for $\#\mathbf{P}$.

A language L is in $\mathbf{coNP}^{\#\mathbf{P}}$ if membership in L can be refuted using a polynomial time non-deterministic Turing machine with access to a $\#\mathbf{P}$ oracle. It is easy to see that $\mathbf{coNP}^{\#\mathbf{P}} = \mathbf{coNP}^{\mathbf{PP}}$, and $\mathbf{PH} \subseteq \mathbf{coNP}^{\#\mathbf{P}} \subseteq \mathbf{PSPACE}$, where $\mathbf{PH} \subseteq \mathbf{coNP}^{\#\mathbf{P}}$ follows by Toda's theorem ($\mathbf{PH} \subseteq \mathbf{P}^{\#\mathbf{P}}$) [39].

The following decision problem is complete for $\mathbf{NP}^{\#\mathbf{P}}$ [13]:

▶ **Definition 10.** *E-Maj-Sat asks, given ϕ a quantifier free formula over $n+m$ variables if there exist an allocation $\boldsymbol{x} \in \{0,1\}^n$ such that there are strictly greater than $\frac{1}{2}$ of allocations to $\boldsymbol{y} \in \{0,1\}^m$ where $\phi(\boldsymbol{x},\boldsymbol{y})$ evaluates to true.*

The complementary problem All-Min-Sat, is complete for $\mathbf{coNP}^{\#\mathbf{P}}$: a formula ϕ is All-Min-Sat, if ϕ is not E-Maj-Sat. That is, ϕ a quantifier free formula over $n+m$ variables is All-Min-Sat if for all allocations $\boldsymbol{x} \in \{0,1\}^n$ there are no more than $\frac{1}{2}$ of allocations to $\boldsymbol{y} \in \{0,1\}^m$ where $\phi(\boldsymbol{x},\boldsymbol{y})$ evaluates to true.

3 The complexity of deciding pure differential privacy

In this section we classify the complexity of deciding ε-differential privacy, for which we show the following theorem:

▶ **Theorem 11.** Decide-ε-DP *is* $\mathbf{coNP}^{\#\mathbf{P}}$*-complete.*

It will be convenient to consider the well-known simpler reformulation of the definition of pure differential privacy in finite ranges to consider specific outcomes $\boldsymbol{o} \in \{0,1\}^\ell$ rather than events $E \subseteq \{0,1\}^\ell$.

▶ **Reformulation 12** (Pure differential privacy)**.** A probabilistic circuit ψ is ε-differentially private if and only if for all neighboring $\boldsymbol{x},\boldsymbol{x'} \in X^n$ and for all $\boldsymbol{o} \in \{0,1\}^\ell$,

$$\Pr[\psi(\boldsymbol{x}) = \boldsymbol{o}] \leq e^\varepsilon \cdot \Pr[\psi(\boldsymbol{x'}) = \boldsymbol{o}].$$

3.1 Decide-ε-DP is in $\mathbf{coNP}^{\#\mathbf{P}}$

We show a non-deterministic Turing machine which can "refute" ψ being ε-differentially private in (non-deterministic) polynomial time with a $\#\mathbf{P}$ oracle. A circuit ψ is shown not to be ε-differentially private by exhibiting a combination $\boldsymbol{x}, \boldsymbol{x'}, \boldsymbol{o}$ such that $\Pr[\psi(\boldsymbol{x}) = \boldsymbol{o}] > e^{\varepsilon} \cdot \Pr[\psi(\boldsymbol{x'}) = \boldsymbol{o}]$. The witness to the non-deterministic Turing machine would be a sequence of $2n$ bits parsed as neighboring inputs $\boldsymbol{x}, \boldsymbol{x'} \in \{0,1\}^n$ and ℓ bits describing an output $\boldsymbol{o} \in \{0,1\}^{\ell}$. The constraint can then be checked in polynomial time, using the $\#\mathbf{P}$ oracle to compute $\Pr[\psi(\boldsymbol{x}) = \boldsymbol{o}]$ and $\Pr[\psi(\boldsymbol{x'}) = \boldsymbol{o}]$.

To compute $\Pr[\psi(\boldsymbol{x}) = \boldsymbol{o}]$ in $\#\mathbf{P}$ we create an instance to #CircuitSat, which will count the number of allocations to the m probabilistic bits consistent with this output. We do this by extending ψ with additional gates reducing to a single output which is true only when the input is fixed to $\boldsymbol{x}$ and the output of ψ was $\boldsymbol{o}$.

3.2 $\mathbf{coNP}^{\#\mathbf{P}}$-hardness of Decide-$\varepsilon$-DP

To show $\mathbf{coNP}^{\#\mathbf{P}}$-hardness of Decide-$\varepsilon$-DP we show a reduction from All-Min-Sat in Lemma 14; together with the inclusion result above, this entails that Decide-ε-DP is $\mathbf{coNP}^{\#\mathbf{P}}$-complete (Theorem 11).

Randomized response [41] is a technique for answering sensitive Yes/No questions by flipping the answer with probability $p \leq 0.5$. Setting $p = \frac{1}{1+e^{\varepsilon}}$ gives ε-differential privacy. Thus $p = 0$ gives no privacy and $p = 0.5$ gives total privacy (albeit no utility).

▶ **Definition 13** (Randomized Response).

$$RR_{\varepsilon}(x) = \begin{cases} x & w.p. \quad \frac{e^{\varepsilon}}{1+e^{\varepsilon}} \\ \neg x & w.p. \quad \frac{1}{1+e^{\varepsilon}} \end{cases}$$

▶ **Lemma 14.** All-Min-Sat *reduces in polynomial time to* Decide-ε-DP.

Proof. We will reduce from All-Min-Sat to Decide-ε-DP using randomized response. We will take a boolean formula ϕ and create a probabilistic circuit that is ε-differentially private if and only if ϕ is All-Min-Sat.

Consider the circuit ψ which takes as input the value $z \in \{0,1\}$. It probabilistically chooses a value of $\boldsymbol{x} \in \{0,1\}^n$ and $\boldsymbol{y} \in \{0,1\}^m$ and one further random bit p_1 and computes $b = z \oplus \neg(p_1 \vee \phi(\boldsymbol{x}, \boldsymbol{y}))$. The circuit outputs $(\boldsymbol{x}, b)$.

▷ Claim 15. ψ is $\ln(3)$-differentially private if and only if ϕ is All-Min-Sat.

Suppose $\phi \in$ All-Min-Sat then, no matter the choice of $\boldsymbol{x}$,

$$0 \leq \Pr_{y}[\phi(\boldsymbol{x}, \boldsymbol{y}) = 1] \leq \frac{1}{2},$$

and hence

$$\frac{1}{4} \leq \Pr_{y,p_1}[\neg(p_1 \vee \phi(\boldsymbol{x}, \boldsymbol{y})) = 1] \leq \frac{1}{2}.$$

We conclude the true answer z is flipped between $\frac{1}{4}$ and $\frac{1}{2}$ of the time, observe this is exactly the region in which randomized response gives us the most privacy. In the worst case $p = \frac{1}{4} = \frac{1}{1+e^{\varepsilon}}$, gives $e^{\varepsilon} = 3$, so $\ln(3)$-differential privacy.

In the converse, suppose $\phi \in$ E-MAJ-SAT, then for some $\boldsymbol{x}$

$$\frac{1}{2} < \Pr_{y}[\phi(\boldsymbol{x}, \boldsymbol{y}) = 1] \leq 1,$$

and then

$$\Pr_{y,p_1}[\neg(p_1 \vee \phi(\boldsymbol{x}, \boldsymbol{y})) = 1] < \frac{1}{4},$$

in which case the randomized response does not provide $\ln(3)$-differential privacy. ◀

▶ Remark 16. We skew the result so that in the positive case (when $\phi \in$ ALL-MIN-SAT) the proportion of accepting allocations is between $\frac{1}{4}$ and $\frac{1}{2}$, resulting in the choice of $\ln(3)$-differentially privacy. Alternative skews, using more bits akin to p_1, shows hardness for other choices of ε.

Hardness by circuit shape

In our proof of the upper-bound we use **coNP** to resolve the non-deterministic choice of both input and output. We show this is necessary in the sense **coNP** is still required for either large input or large output. The hardness proof used in Lemma 14 shows that when $|\psi| = n$ the problem is hard for $\Omega(1)$-bit input and $\Omega(n)$-bit output.

We can also prove this is hard for $\Omega(n)$-bit input and $\Omega(1)$-bit output. Intuitively a counter example to differential privacy has two choices: a pair of adjacent input and a given output upon which the relevant inequality will hold. So to “refute” ALL-MIN-SAT the counterexample of the ALL choice (i.e. $\boldsymbol{x}$) can be selected in the input, rather than the output as in our case. Since the input is now non-trivial we must take care of what happens when the adjacent bit is in the choice of $\boldsymbol{x}$. Details are given in the full version.

Further the problem is in $\mathbf{P}^{\#\mathbf{P}}$ for $O(\log(n))$-bit input and $O(\log(n))$-bit output, as in this case, the choices made by **coNP** can instead be checked deterministically in polynomial time. In this case we show **PP**-hardness, which applies even when there is 1-bit input and 1-bit output.

4 On the complexity of deciding approximate differential privacy

It is less clear whether deciding (ε, δ)-differential privacy can be done in $\mathbf{coNP}^{\#\mathbf{P}}$. First we consider restrictions to the shape of the circuit so that $\mathbf{coNP}^{\#\mathbf{P}}$ can be recovered, and then show that in general the problem is in $\mathbf{coNP}^{\#\mathbf{P}^{\#\mathbf{P}}}$.

Recall that in the case of ε-differential privacy it was enough to consider singleton events $\{\boldsymbol{o}\}$ where $\boldsymbol{o} \in \{0,1\}^{\ell}$, however in the definition of (ε, δ)-differential privacy we must quantify over output events $E \subseteq \{0,1\}^{\ell}$. If we consider circuits with one output bit ($\ell = 1$), then the event space essentially reduces to $E \in \{\emptyset, \{0\}, \{1\}, \{0,1\}\}$ and we can apply the same technique.

We expand this to the case when the number of outputs bits is logarithmic $\ell \leq \log(|\psi|)$. To cater to this, rather than guessing a violating $E \in \{0,1\}^{\ell}$, we consider a violating subset of events $E \subseteq \{0,1\}^{\ell}$. Given such an event E we create a circuit ψ_E on ℓ inputs and a single output which indicates whether the input is in the event E. The size of this circuit is exponential in ℓ, thus polynomial in $|\psi|$. Composing $\psi_E \circ \psi$, we check the conditions hold for this event E, with just one bit of output.

▷ Claim 17. DECIDE-ε, δ-DP, restricted to circuits ψ with ℓ bit outputs where $\ell \leq \log(|\psi|)$, is in $\mathbf{coNP}^{\#\mathbf{P}}$ (and hence $\mathbf{coNP}^{\#\mathbf{P}}$-complete).

The claim trivially extends to $\ell \leq c \cdot \log(|\psi|)$ for any *fixed* $c > 0$.

4.1 Decide-$\boldsymbol{\varepsilon}$, $\boldsymbol{\delta}$-DP is in $\mathbf{coNP}^{\#\mathbf{P}^{\#\mathbf{P}}}$

We now show that Decide-ε,δ-DP in the most general case can be solved in $\mathbf{coNP}^{\#\mathbf{P}^{\#\mathbf{P}}}$. We will assume $e^\varepsilon = \alpha$ is given as a rational, with $\alpha = \frac{u}{v}$ for some integers u and v. Recall we use n, ℓ and m to refer to the number of input, output and random bits of a circuit respectively. While we will use non-determinism to choose inputs leading to a violating event, unlike in Section 3 it would not be used for finding a violating event E, as an (explicit) description of such an event may be of super-polynomial length. It would be useful for us to use a reformulation of approximate differential privacy, using a sum over potential individual outcomes.

▶ **Reformulation 18** (Pointwise differential privacy [7]). A probabilistic circuit ψ is (ε,δ)-differentially private if and only if for all neighboring $\boldsymbol{x}, \boldsymbol{x}' \in X^n$ and for all $\boldsymbol{o} \in \{0,1\}^\ell$,

$$\sum_{\boldsymbol{o}\in\{0,1\}^\ell} \delta_{\boldsymbol{x},\boldsymbol{x}'}(\boldsymbol{o}) \le \delta,$$

where $\delta_{\boldsymbol{x},\boldsymbol{x}'}(\boldsymbol{o}) = \max\left(\Pr[\psi(\boldsymbol{x}) = \boldsymbol{o}] - e^\varepsilon \cdot \Pr[\psi(\boldsymbol{x}') = \boldsymbol{o}], 0\right).$

We define $\mathcal{M}$, a non-deterministic Turing Machine with access to a $\#\mathbf{P}$-oracle, and where each execution branch runs in polynomial time. On inputs a probabilistic circuit ψ and neighboring $\boldsymbol{x}, \boldsymbol{x}' \in X^n$ the number of accepting executions of $\mathcal{M}$ would be proportional to $\sum_{\boldsymbol{o}\in\{0,1\}^\ell} \delta_{\boldsymbol{x},\boldsymbol{x}'}(\boldsymbol{o})$.

In more detail, on inputs ψ, $\boldsymbol{x}$ and $\boldsymbol{x}'$, $\mathcal{M}$ chooses $\boldsymbol{o} \in \{0,1\}^\ell$ and an integer $C \in \{1, 2, \dots, 2^{m+\lceil\log(v)\rceil}\}$ (this requires choosing $\ell + m + \lceil\log(v)\rceil$ bits). Through a call to the $\#\mathbf{P}$ oracle, $\mathcal{M}$ computes

$$a = |\{\boldsymbol{r} \in \{0,1\}^m : \psi(\boldsymbol{x}, \boldsymbol{r}) = \boldsymbol{o}\}|$$

and

$$b = |\{\boldsymbol{r} \in \{0,1\}^m : \psi(\boldsymbol{x}', \boldsymbol{r}) = \boldsymbol{o}\}|.$$

Finally, $\mathcal{M}$ accepts if $v \cdot a - u \cdot b \ge C$ and otherwise rejects.

▶ **Lemma 19.** *Given two inputs $\boldsymbol{x}, \boldsymbol{x}' \in X^n$, $\mathcal{M}(\psi, \boldsymbol{x}, \boldsymbol{x}')$ has exactly $v \cdot 2^m \sum_{\boldsymbol{o}\in\{0,1\}^\ell} \delta_{\boldsymbol{x},\boldsymbol{x}'}(\boldsymbol{o})$ accepting executions.*

Proof. Let $\mathbb{1}\{X\}$ be the indicator function, which is one if the predicate X holds and zero otherwise.

$$v \cdot 2^m \sum_{\boldsymbol{o}\in\{0,1\}^\ell} \delta_{\boldsymbol{x},\boldsymbol{x}'}(\boldsymbol{o}) = \sum_{\boldsymbol{o}\in\{0,1\}^\ell} v \cdot 2^m \max\Big(\Pr[\psi(\boldsymbol{x}) = \boldsymbol{o}] - \alpha \Pr[\psi(\boldsymbol{x}') = \boldsymbol{o}], 0\Big)$$

$$= \sum_{\boldsymbol{o}\in\{0,1\}^\ell} v2^m \max\left(\frac{1}{2^m} \sum_{\boldsymbol{r}\in\{0,1\}^m} \mathbb{1}\{\psi(\boldsymbol{x},\boldsymbol{r}) = \boldsymbol{o}\} - \alpha\frac{1}{2^m} \sum_{\boldsymbol{r}\in\{0,1\}^m} \mathbb{1}\big\{\psi(\boldsymbol{x}',\boldsymbol{r}) = \boldsymbol{o}\big\}, 0\right)$$

$$= \sum_{\boldsymbol{o}\in\{0,1\}^\ell} \max\left(v \sum_{\boldsymbol{r}\in\{0,1\}^m} \mathbb{1}\{\psi(\boldsymbol{x},\boldsymbol{r}) = \boldsymbol{o}\} - v\alpha \sum_{\boldsymbol{r}\in\{0,1\}^m} \mathbb{1}\big\{\psi(\boldsymbol{x}',\boldsymbol{r}) = \boldsymbol{o}\big\}, 0\right)$$

$$
\begin{aligned}
\dots &= \sum_{\boldsymbol{o}\in\{0,1\}^\ell} \max\left(v \sum_{\boldsymbol{r}\in\{0,1\}^m} \mathbb{1}\{\psi(\boldsymbol{x},\boldsymbol{r})=\boldsymbol{o}\} - u \sum_{\boldsymbol{r}\in\{0,1\}^m} \mathbb{1}\big\{\psi(\boldsymbol{x}',\boldsymbol{r})=\boldsymbol{o}\big\}, 0 \right)\\
&= \sum_{\boldsymbol{o}\in\{0,1\}^\ell} \sum_{C=1}^{2^{\lceil\log(v)\rceil+m}} \mathbb{1}\left\{ \max\left(v \sum_{\boldsymbol{r}\in\{0,1\}^m} \mathbb{1}\{\psi(\boldsymbol{x},\boldsymbol{r})=\boldsymbol{o}\} - u \sum_{\boldsymbol{r}\in\{0,1\}^m} \mathbb{1}\big\{\psi(\boldsymbol{x}',\boldsymbol{r})=\boldsymbol{o}\big\}, 0 \right) \geq C \right\}\\
&= \text{number of accepting executions in } \widehat{\mathcal{M}}
\end{aligned}
$$
◀

We can now describe our $\mathbf{coNP}^{\#\mathbf{P}^{\#\mathbf{P}}}$ procedure for Decide-ε,δ-DP. The procedure takes as input a probabilistic circuit ψ.

1. Non-deterministically choose neighboring $\boldsymbol{x}$ and $\boldsymbol{x}' \in \{0,1\}^n$ (i.e., $2n$ bits).
2. Let $\mathcal{M}$ be the non-deterministic Turing Machine with access to a $\#\mathbf{P}$-oracle as described above. Create a machine $\widehat{\mathcal{M}}$ with no input that executes $\mathcal{M}$ on $\psi, \boldsymbol{x}, \boldsymbol{x}'$.
3. Make an $\#\mathbf{P}^{\#\mathbf{P}}$ oracle call for the number of accepting executions $\widehat{\mathcal{M}}$ has.
4. Reject if the number of accepting executions is greater than $v \cdot 2^m \cdot \delta$ and otherwise accept.

By Lemma 19, there is a choice $\boldsymbol{x}, \boldsymbol{x}'$ on which the procedure rejects if and only if ψ is not (ε,δ)-differentially private.

4.2 Hardness

Theorem 11 shows that Decide-ε-DP is $\mathbf{coNP}^{\#\mathbf{P}}$-complete, in particular $\mathbf{coNP}^{\#\mathbf{P}}$-hard and since Decide-$\varepsilon$-DP is a special case of Decide-ε,δ-DP, this is also $\mathbf{coNP}^{\#\mathbf{P}}$-hard. Nevertheless the proof is based on particular values of ε and in the full version we provide an alternative proof of hardness based on δ. This proof result will apply for any ε (even for $\varepsilon = 0$) and for a large range of δ (but not $\delta = 0$).

The proof proceeds by first considering the generalisation of All-Min-Sat to the version where *minority*, i.e. less than $\frac{1}{2}$ of the assignments, is replaced with another threshold. This problem is also $\mathbf{coNP}^{\#\mathbf{P}}$-hard for a range of thresholds. Note however, if this threshold is exactly 1 the problem is true for all formulae, and if the threshold is 0 the problem is simply asks if the formula is unsatisfiable (a **coNP** problem).

This generalised problem can then be reduced to deciding Decide-ε,δ-DP, where the threshold corresponds exactly to δ. It will turn out in the resulting circuit ε does not change the status of differential privacy, i.e. it is (ε,δ)-differentially private for all ε, or not.

The proof shows hardness for $\Omega(n)$-input bits and 1-output bit; the case in which there also exists a $\mathbf{coNP}^{\#\mathbf{P}}$ upper-bound. Hence, showing hardness in a higher complexity class, e.g., $\mathbf{coNP}^{\#\mathbf{P}^{\#\mathbf{P}}}$, would require a reduction to a circuit with more output bits.

5 Inapproximability of the privacy parameters ε, δ

Given the difficulty of deciding if a circuit is differentially private, one might naturally consider whether approximating ε or δ could be efficient. We show that these tasks are both **NP**-hard and **coNP**-hard.

We show that distinguishing between (ε,δ), and (ε',δ')-differential privacy is **NP**-hard, by reduction from a problem we call Not-Constant which we also show is **NP**-hard. A boolean formula is in Not-Constant if it is satisfiable and not also a tautology.

▶ **Lemma 20.** *Not-Constant is* **NP**-*complete. (hence Constant is* **coNP**-*complete).*

Proof. Clearly, Not-Constant $\in$ **NP**, the witness being a pair of satisfying and non-satisfying assignments. We reduce 3-SAT to Not-Constant. Given a Boolean formula ϕ over variables $x_1, \ldots, x_n$ let $\phi'(x_1, \ldots, x_n, x_{n+1}) = \phi(x_1, \ldots, x_n) \wedge x_{n+1}$. Note that ϕ' is never a tautology as $\phi'(x_1, \ldots, x_n, 0) = 0$. Furthermore, ϕ' is satisfiable iff ϕ is. ◀

In Definition 13 we used randomized response in the pure differential privacy setting. We now consider the approximate differential privacy variant $RR_{\varepsilon,\delta} : \{0,1\} \to \{\top, \bot\} \times \{0,1\}$ defined as follows:

$$RR_{\varepsilon,\delta}(x) = \begin{cases} (\top, x) & \text{w.p. } \delta \\ (\bot, x) & \text{w.p. } (1-\delta)\frac{\alpha}{1+\alpha} \\ (\bot, \neg x) & \text{w.p. } (1-\delta)\frac{1}{1+\alpha} \end{cases} \quad \text{where} \quad \alpha = e^{\varepsilon}$$

I.e., with probability δ, $RR_{\varepsilon,\delta}(x)$ reveals x and otherwise it executes $RR_{\varepsilon}(x)$. The former is marked with "$\top$" and the latter with "$\bot$". This mechanism is equivalent to the one described in [35] and is (ε, δ)-differentially private.

▶ **Definition 21.** *Let $0 \leq \varepsilon \leq \varepsilon'$, $0 \leq \delta \leq \delta' \leq 1$, with either $\varepsilon < \varepsilon'$ or $\delta < \delta'$. The problem* Distinguish-$(\varepsilon, \delta), (\varepsilon', \delta')$-DP *takes as input a circuit ψ, guaranteed to be either (ε, δ)-differentially private, or (ε', δ')-differentially private. The problem asks whether ψ is (ε, δ)-differentially private or (ε', δ')-differentially private.*

▶ **Lemma 22.** *Distinguish-$(\varepsilon, \delta), (\varepsilon', \delta')$-DP is* **NP***-hard (and* **coNP***-hard).*

Proof. We reduce Not-Constant to Distinguish-$(\varepsilon, \delta), (\varepsilon', \delta')$-DP. Given the boolean formula $\phi(\boldsymbol{x})$ on n bits, we create a probabilistic circuit ψ. The input to ψ consists of the n bits $\boldsymbol{x}$ plus a single bit y. The circuit ψ has four output bits (o_1, o_2, o_3, o_4) such that $(o_1, o_2) = RR_{\varepsilon,\delta}(y)$ and $(o_3, o_4) = RR_{\varepsilon',\delta'}(\phi(\boldsymbol{x}))$.

Observe that $(o_1, o_2) = RR_{\varepsilon,\delta}(y)$ is always (ε, δ) differentially private. As for $(o_3, o_4) = RR_{\varepsilon',\delta'}(\phi(\boldsymbol{x}))$, if $\phi \in$ Not-Constant then there are adjacent $\boldsymbol{x}, \boldsymbol{x}'$ such that $\phi(\boldsymbol{x}) \neq \phi(\boldsymbol{x}')$. In this case, $(o_3, o_4) = RR_{\varepsilon',\delta'}(\phi(\boldsymbol{x}))$ is (ε', δ')-differentially private, and, because $(\varepsilon, \delta) < (\varepsilon', \delta')$, so is ψ . On the other hand, if $\phi \notin$ Not-Constant then $\phi(\boldsymbol{x})$ does not depend on $\boldsymbol{x}$ and hence (o_3, o_4) does not affect privacy, in which case we get that ψ is (ε, δ) differentially private.

The same argument also gives **coNP**-hardness. ◀

Notice that the above theorem holds when $\delta = \delta'$ and $\varepsilon < \varepsilon'$ (similarly, $\varepsilon = \varepsilon'$ and $\delta < \delta'$), which entails the following theorem:

▶ **Theorem 23.** *Assuming* **P** $\neq$ **NP***, for any approximation error $\gamma > 0$, there does not exist a polynomial time approximation algorithm that given a probabilistic circuit ψ and δ computes some $\hat{\varepsilon}$, where $|\hat{\varepsilon} - \varepsilon| \leq \gamma$ and ε is the minimal such that ψ is (ε, δ)-differentially private within error γ. Similarly, given ε, no such $\hat{\delta}$ can be computed polynomial time where $|\hat{\delta} - \delta| \leq \gamma$ and δ is minimal.*

▶ Remark 24. The result also applies when approximating within a given ratio $\rho > 1$ (e.g. in the case of approximating ε, to find $\hat{\varepsilon}$ such that $\frac{\hat{\varepsilon}}{\varepsilon} \leq \rho$). Moreover, the result also holds when approximating pure differential privacy, that is when $\delta = 0$.

6 Related work

Differential privacy was introduced in [22]. It is a definition of privacy in the context of data analysis capturing the intuition that information specific to an individuals is protected if every single user's input has a bounded influence on the computation's outcome distribution, where the bound is specified by two parameters, usually denoted by ε, δ. Intuitively, these parameters set an upperbound on privacy loss, where the parameter ε limits the loss and the parameter δ limits the probability in which the loss may exceed ε.

Extensive work has occurred in the computer-assisted or automated of verification of differential privacy. Early work includes, PINQ [33] and Airavat [38] which are systems that keep track of the privacy budgets (ε and δ) using trusted privacy primitives in SQL-like and MapReduce-like paradigms respectively. In other work, programming languages were developed, that use the type system to keep track of the sensitivity and ensure the correct level of noise is added [37, 9, 16, 8]. Another line of work uses proof assistants to help prove that an algorithm is differentially private [7]; although much of this work is not automated, recent work has gone in this direction [2, 44].

These techniques focuses on "soundness", rather than "completeness" thus are not amenable to complexity analysis. In the constrained case of verifying differential privacy on probabilistic automata and Markov chains there are bisimulation based techniques [40, 12]. Towards complexity analysis; [15] shows that computing the optimal value of δ for a finite labelled Markov chain is undecidable. Further [14] and [15] provides distances, which are (necessarily) not tight, but can be computed in polynomial time with an **NP** oracle and a weaker bound in polynomial time. Recent works have focused on developing techniques for finding violations of differential privacy [19, 10]. The methods proposed so far have been based on some form of testing. Our result limits also the tractability of these approaches. Finally, [5] proposes an automated technique for proving differential privacy or finding counterexamples. This paper studies a constrained class of programs extending the language we presented here, and provides a "complete" procedure for deciding differential privacy for them. The paper does not provide any complexity guarantee for the proposed method and we expect our results to apply also in their setting.

As we already discussed, Murtagh and Vadhan [35] showed that finding the optimal values for the privacy parameters when composing different algorithms in a black-box way is #**P**-complete, but also that approximating the optimal values can be done efficiently. In contrast, our results show that when one wants to consider programs as white-box, as often needed to achieve better privacy guarantees (e.g. in the case of the sparse vector technique), the complexity is higher.

Several works have explored different property testing related to differential privacy [20, 29, 26], including verification [26]. In the standard model used in property testing, a user has only black-box access to the function and the observable outputs are the ones provided by a privacy mechanism. In contrast, our work is based on the program description and aim to provide computational limits to the design of techniques for program analyses for differential privacy.

We already discussed some works on quantitative information flow. In addition to those, it was shown that comparing the quantitative information flow of two programs on inputs coming from the uniform distribution is #**P**-hard [43]. However, when quantifying over all distributions the question is **coNP**-complete [43].

As we remarked earlier, our language is equally expressive when integers of a fixed size are added. Recently Jacomme, Kremer and Barthe [28] show deciding equivalence of two such programs, operating over a fixed finite field, is $\mathbf{coNP}^{\mathbf{C_=P}}$-complete and the majority problem, which is similar to pure differential privacy, is $\mathbf{coNP}^{\mathbf{PP}}$-complete – matching the class we show for deciding ε-differential privacy. Further the universal equivalence problem, which shows the programs are equivalent over all field extensions, is decidable in 2-**EXP**; the universal majority problem is not know to be decidable.

7 Conclusions and future work

Verifying differential privacy of loop-free probabilistic boolean programs

We have shown the difficulty of verifying differential privacy in loop-free probabilistic boolean programs through their correspondence with probabilistic circuits. Deciding ε-differential privacy is $\mathbf{coNP}^{\#\mathbf{P}}$-complete and (ε, δ)-differential privacy is $\mathbf{coNP}^{\#\mathbf{P}}$-hard and in $\mathbf{coNP}^{\#\mathbf{P}^{\#\mathbf{P}}}$ (a gap that we leave for future work). Both problems are positioned in the counting hierarchy, in between the polynomial hierarchy **PH** and **PSPACE**.

Verifying differential privacy of probabilistic boolean programs

One interesting question that our work leaves open is the characterization of the complexity of deciding differential privacy problems for probabilistic boolean programs, including loops. Similarly to the works on quantitative information flow [11], we expect these problems to be decidable and we expect them to be in **PSPACE**. However, this question requires some further investigation that we leave for future work.

Solvers mixing non-determinism and counting

Returning to our motivation for this work – developing practical tools for verifying differential privacy – our results seem to point to a deficiency in available tools for model checking. The model checking toolkit includes well established SAT solvers for **NP** (and **coNP**) problems, solvers for further quantification in **PH**, solvers for #SAT (and hence for #**P** problems[3]). However to the best of our knowledge, there are currently no solvers that are specialized for mixing the polynomial hierarchy **PH** and counting problems #**P**, in particular $\mathbf{coNP}^{\#\mathbf{P}}$ and $\mathbf{coNP}^{\#\mathbf{P}^{\#\mathbf{P}}}$.

Approximating the differential privacy parameters

We show that distinguishing (ε, δ)-differential privacy from (ε', δ') differential privacy where $(\varepsilon, \delta) < (\varepsilon', \delta')$ is both **NP**- and **coNP**-hard. We leave refining the classification of this problem as an open problem.

References

1 John M. Abowd. The U.S. census bureau adopts differential privacy. In Yike Guo and Faisal Farooq, editors, *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018, London, UK, August 19-23, 2018*, page 2867. ACM, 2018. doi:10.1145/3219819.3226070.

2 Aws Albarghouthi and Justin Hsu. Synthesizing coupling proofs of differential privacy. *Proc. ACM Program. Lang.*, 2(POPL):58:1–58:30, 2018. doi:10.1145/3158146.

[3] See, for example, http://beyondnp.org/pages/solvers/, for a range of solvers.

3 Mário S. Alvim, Miguel E. Andrés, Konstantinos Chatzikokolakis, and Catuscia Palamidessi. On the relation between differential privacy and quantitative information flow. In Luca Aceto, Monika Henzinger, and Jirí Sgall, editors, *Automata, Languages and Programming - 38th International Colloquium, ICALP 2011, Zurich, Switzerland, July 4-8, 2011, Proceedings, Part II*, volume 6756 of *Lecture Notes in Computer Science*, pages 60–76. Springer, 2011. doi:10.1007/978-3-642-22012-8_4.

4 Apple. Apple differential privacy technical overview. URL: https://www.apple.com/privacy/docs/Differential_Privacy_Overview.pdf.

5 Gilles Barthe, Rohit Chadha, Vishal bibsource = self, Jagannath, A Prasad Sistla, and Mahesh Viswanathan. Deciding differential privacy for programs with finite inputs and outputs. In *LICS 2020 (to appear)*, 2020. arXiv preprint: arXiv:1910.04137.

6 Gilles Barthe, Marco Gaboardi, Emilio Jesús Gallego Arias, Justin Hsu, Aaron Roth, and Pierre-Yves Strub. Higher-order approximate relational refinement types for mechanism design and differential privacy. In Sriram K. Rajamani and David Walker, editors, *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015*, pages 55–68. ACM, 2015. doi:10.1145/2676726.2677000.

7 Gilles Barthe, Marco Gaboardi, Benjamin Grégoire, Justin Hsu, and Pierre-Yves Strub. Proving differential privacy via probabilistic couplings. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 749–758. ACM, 2016. doi:10.1145/2933575.2934554.

8 Gilles Barthe, Marco Gaboardi, Justin Hsu, and Benjamin C. Pierce. Programming language techniques for differential privacy. *SIGLOG News*, 3(1):34–53, 2016. URL: https://dl.acm.org/citation.cfm?id=2893591.

9 Gilles Barthe, Boris Köpf, Federico Olmedo, and Santiago Zanella Béguelin. Probabilistic relational reasoning for differential privacy. In John Field and Michael Hicks, editors, *Proceedings of the 39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2012, Philadelphia, Pennsylvania, USA, January 22-28, 2012*, pages 97–110. ACM, 2012. doi:10.1145/2103656.2103670.

10 Benjamin Bichsel, Timon Gehr, Dana Drachsler-Cohen, Petar Tsankov, and Martin T. Vechev. Dp-finder: Finding differential privacy violations by sampling and optimization. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 508–524. ACM, 2018. doi:10.1145/3243734.3243863.

11 Rohit Chadha, Dileep Kini, and Mahesh Viswanathan. Quantitative information flow in boolean programs. In Martín Abadi and Steve Kremer, editors, *Principles of Security and Trust - Third International Conference, POST 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings*, volume 8414 of *Lecture Notes in Computer Science*, pages 103–119. Springer, 2014. doi:10.1007/978-3-642-54792-8_6.

12 Konstantinos Chatzikokolakis, Daniel Gebler, Catuscia Palamidessi, and Lili Xu. Generalized bisimulation metrics. In Paolo Baldan and Daniele Gorla, editors, *CONCUR 2014 - Concurrency Theory - 25th International Conference, CONCUR 2014, Rome, Italy, September 2-5, 2014. Proceedings*, volume 8704 of *Lecture Notes in Computer Science*, pages 32–46. Springer, 2014. doi:10.1007/978-3-662-44584-6_4.

13 Dmitry Chistikov, Rayna Dimitrova, and Rupak Majumdar. Approximate counting in SMT and value estimation for probabilistic programs. *Acta Inf.*, 54(8):729–764, 2017. doi:10.1007/s00236-017-0297-2.

14 Dmitry Chistikov, Andrzej S. Murawski, and David Purser. Bisimilarity distances for approximate differential privacy. In Shuvendu K. Lahiri and Chao Wang, editors, *Automated Technology for Verification and Analysis - 16th International Symposium, ATVA 2018, Los Angeles, CA, USA, October 7-10, 2018, Proceedings*, volume 11138 of *Lecture Notes in Computer Science*, pages 194–210. Springer, 2018. doi:10.1007/978-3-030-01090-4_12.

15 Dmitry Chistikov, Andrzej S. Murawski, and David Purser. Asymmetric distances for approximate differential privacy. In Wan Fokkink and Rob van Glabbeek, editors, *30th International Conference on Concurrency Theory, CONCUR 2019, August 27-30, 2019, Amsterdam, the Netherlands*, volume 140 of *LIPIcs*, pages 10:1–10:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.CONCUR.2019.10.

16 Loris D'Antoni, Marco Gaboardi, Emilio Jesús Gallego Arias, Andreas Haeberlen, and Benjamin C. Pierce. Sensitivity analysis using type-based constraints. In Richard Lazarus, Assaf J. Kfoury, and Jacob Beal, editors, *Proceedings of the 1st annual workshop on Functional programming concepts in domain-specific languages, FPCDSL@ICFP 2013, Boston, Massachusetts, USA, September 22, 2013*, pages 43–50. ACM, 2013. doi:10.1145/2505351.2505353.

17 Dorothy E. Denning. *Cryptography and Data Security.* Addison-Wesley, 1982.

18 Differential Privacy Team, Apple. Learning with privacy at scale, 2017. URL: https://machinelearning.apple.com/docs/learning-with-privacy-at-scale/appledifferentialprivacysystem.pdf.

19 Zeyu Ding, Yuxin Wang, Guanhong Wang, Danfeng Zhang, and Daniel Kifer. Detecting violations of differential privacy. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 475–489. ACM, 2018. doi:10.1145/3243734.3243818.

20 Kashyap Dixit, Madhav Jha, Sofya Raskhodnikova, and Abhradeep Thakurta. Testing the lipschitz property over product distributions with applications to data privacy. In Amit Sahai, editor, *Theory of Cryptography - 10th Theory of Cryptography Conference, TCC 2013, Tokyo, Japan, March 3-6, 2013. Proceedings*, volume 7785 of *Lecture Notes in Computer Science*, pages 418–436. Springer, 2013. doi:10.1007/978-3-642-36594-2_24.

21 Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. Our data, ourselves: Privacy via distributed noise generation. In Serge Vaudenay, editor, *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*, volume 4004 of *Lecture Notes in Computer Science*, pages 486–503. Springer, 2006. doi:10.1007/11761679_29.

22 Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam D. Smith. Calibrating noise to sensitivity in private data analysis. In Shai Halevi and Tal Rabin, editors, *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006, Proceedings*, volume 3876 of *Lecture Notes in Computer Science*, pages 265–284. Springer, 2006. doi:10.1007/11681878_14.

23 Cynthia Dwork, Guy N. Rothblum, and Salil P. Vadhan. Boosting and differential privacy. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 51–60. IEEE Computer Society, 2010. doi:10.1109/FOCS.2010.12.

24 Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. RAPPOR: randomized aggregatable privacy-preserving ordinal response. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, pages 1054–1067. ACM, 2014. doi:10.1145/2660267.2660348.

25 Matthew Fredrikson and Somesh Jha. Satisfiability modulo counting: a new approach for analyzing privacy properties. In Thomas A. Henzinger and Dale Miller, editors, *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*, pages 42:1–42:10. ACM, 2014. doi:10.1145/2603088.2603097.

26 Anna C. Gilbert and Audra McMillan. Property testing for differential privacy. In *56th Annual Allerton Conference on Communication, Control, and Computing, Allerton 2018, Monticello, IL, USA, October 2-5, 2018*, pages 249–258. IEEE, 2018. doi:10.1109/ALLERTON.2018.8636068.

27 J. W. Gray. Probabilistic interference. In *Proceedings. 1990 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 170–179, May 1990. doi:10.1109/RISP.1990.63848.

28 Charlie Jacomme, Steve Kremer, and Gilles Barthe. Universal equivalence and majority on probabilistic programs over finite fields. In *LICS 2020 (to appear)*, 2020.

29 Madhav Jha and Sofya Raskhodnikova. Testing and reconstruction of lipschitz functions with applications to data privacy. *SIAM J. Comput.*, 42(2):700–731, 2013. doi:10.1137/110840741.

30 Dexter Kozen. Semantics of probabilistic programs. *J. Comput. Syst. Sci.*, 22(3):328–350, 1981. doi:10.1016/0022-0000(81)90036-2.

31 Michael L. Littman, Judy Goldsmith, and Martin Mundhenk. The computational complexity of probabilistic planning. *J. Artif. Intell. Res.*, 9:1–36, 1998. doi:10.1613/jair.505.

32 Min Lyu, Dong Su, and Ninghui Li. Understanding the sparse vector technique for differential privacy. *Proc. VLDB Endow.*, 10(6):637–648, 2017. doi:10.14778/3055330.3055331.

33 Frank McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In Ugur Çetintemel, Stanley B. Zdonik, Donald Kossmann, and Nesime Tatbul, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2009, Providence, Rhode Island, USA, June 29 - July 2, 2009*, pages 19–30. ACM, 2009. doi:10.1145/1559845.1559850.

34 Ilya Mironov. On significance of the least significant bits for differential privacy. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*, pages 650–661. ACM, 2012. doi:10.1145/2382196.2382264.

35 Jack Murtagh and Salil P. Vadhan. The complexity of computing the optimal composition of differential privacy. In Eyal Kushilevitz and Tal Malkin, editors, *Theory of Cryptography - 13th International Conference, TCC 2016-A, Tel Aviv, Israel, January 10-13, 2016, Proceedings, Part I*, volume 9562 of *Lecture Notes in Computer Science*, pages 157–175. Springer, 2016. doi:10.1007/978-3-662-49096-9_7.

36 Nicolas Papernot, Martín Abadi, Úlfar Erlingsson, Ian J. Goodfellow, and Kunal Talwar. Semi-supervised knowledge transfer for deep learning from private training data. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL: https://openreview.net/forum?id=HkwoSDPgg.

37 Jason Reed and Benjamin C. Pierce. Distance makes the types grow stronger: a calculus for differential privacy. In Paul Hudak and Stephanie Weirich, editors, *Proceeding of the 15th ACM SIGPLAN international conference on Functional programming, ICFP 2010, Baltimore, Maryland, USA, September 27-29, 2010*, pages 157–168. ACM, 2010. doi:10.1145/1863543.1863568.

38 Indrajit Roy, Srinath T. V. Setty, Ann Kilzer, Vitaly Shmatikov, and Emmett Witchel. Airavat: Security and privacy for mapreduce. In *Proceedings of the 7th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2010, April 28-30, 2010, San Jose, CA, USA*, pages 297–312. USENIX Association, 2010. URL: http://www.usenix.org/events/nsdi10/tech/full_papers/roy.pdf.

39 Seinosuke Toda. PP is as hard as the polynomial-time hierarchy. *SIAM J. Comput.*, 20(5):865–877, 1991. doi:10.1137/0220053.

40 Michael Carl Tschantz, Dilsun Kirli Kaynar, and Anupam Datta. Formal verification of differential privacy for interactive systems (extended abstract). In Michael W. Mislove and Joël Ouaknine, editors, *Twenty-seventh Conference on the Mathematical Foundations of Programming Semantics, MFPS 2011, Pittsburgh, PA, USA, May 25-28, 2011*, volume 276 of *Electronic Notes in Theoretical Computer Science*, pages 61–79. Elsevier, 2011. doi:10.1016/j.entcs.2011.09.015.

41 Stanley L Warner. Randomized response: A survey technique for eliminating evasive answer bias. *Journal of the American Statistical Association*, 60(309):63–69, 1965.

42 Hirotoshi Yasuoka and Tachio Terauchi. On bounding problems of quantitative information flow. In Dimitris Gritzalis, Bart Preneel, and Marianthi Theoharidou, editors, *Computer Security - ESORICS 2010, 15th European Symposium on Research in Computer Security, Athens, Greece, September 20-22, 2010. Proceedings*, volume 6345 of *Lecture Notes in Computer Science*, pages 357–372. Springer, 2010. doi:10.1007/978-3-642-15497-3_22.

43 Hirotoshi Yasuoka and Tachio Terauchi. Quantitative information flow - verification hardness and possibilities. In *Proceedings of the 23rd IEEE Computer Security Foundations Symposium, CSF 2010, Edinburgh, United Kingdom, July 17-19, 2010*, pages 15–27. IEEE Computer Society, 2010. doi:10.1109/CSF.2010.9.

44 Danfeng Zhang and Daniel Kifer. Lightdp: towards automating differential privacy proofs. In Giuseppe Castagna and Andrew D. Gordon, editors, *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*, pages 888–901. ACM, 2017. URL: http://dl.acm.org/citation.cfm?id=3009884.

Logical Characterisation of Hybrid Conformance

Maciej Gazda
Department of Computer Science, University of Sheffield, UK
m.gazda@sheffield.ac.uk

Mohammad Reza Mousavi
School of Informatics, University of Leicester, UK
mm789@le.ac.uk

Abstract

Logical characterisation of a behavioural equivalence relation precisely specifies the set of formulae that are preserved and reflected by the relation. Such characterisations have been studied extensively for exact semantics on discrete models such as bisimulations for labelled transition systems and Kripke structures, but to a much lesser extent for approximate relations, in particular in the context of hybrid systems. We present what is to our knowledge the first characterisation result for approximate notions of hybrid refinement and hybrid conformance involving tolerance thresholds in both time and value. Since the notion of conformance in this setting is approximate, any characterisation will unavoidably involve a notion of relaxation, denoting how the specification formulae should be relaxed in order to hold for the implementation. We also show that an existing relaxation scheme on Metric Temporal Logic used for preservation results in this setting is not tight enough for providing a characterisation of neither hybrid conformance nor refinement. The characterisation result, while interesting in its own right, paves the way to more applied research, as our notion of hybrid conformance underlies a formal model-based technique for the verification of cyber-physical systems.

2012 ACM Subject Classification Theory of computation → Logic and verification; Software and its engineering → Formal software verification

Keywords and phrases Logical Characterisation, Metric Temporal Logic, Conformance, Behavioural Equivalence, Hybrid Systems, Relaxation

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.130

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

Acknowledgements We thank Rayna Dimitrova for her helpful comments on an earlier version of this article. We also thank anonymous ICALP reviewers for their thorough feedback which helped us improve the paper.

1 Introduction

Cyber-physical systems integrate discrete aspects of computation, with continuous aspects of physical phenomena, and asynchronous aspects of communication protocols. To test cyber-physical systems against their discrete abstractions (also called discrete-event systems), several notions of conformance have been proposed [13, 28, 31]; we refer to the tutorial volume edited by Broy et al. [8] for an overview. Logical characterisations of conformance [21, 3] are of particular importance in this context, because they precisely specify the set of logical formulae that are preserved and reflected under conformance (we refer to [4] for an accessible introduction). Such logical characterisations provide a rigorous basis for design trajectories that involves subsequent conformance test at different layers of abstraction. Moreover, logical characterisations are stepping stones towards devising the notion of characterising formulae, which have been used in tools and algorithms for checking conformance [4, 10].

In the context of hybrid systems, i.e., abstractions of CPSs integrating both discrete and continuous aspects, some notions of conformance have been proposed in the recent literature [2, 1, 11, 16] (see [22] for an overview). However, not much is known about logical

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 130; pp. 130:1–130:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

characterisation of such notions; to our knowledge, the closest known results to a logical characterisation of hybrid conformance are the logical preservation results [16, 1] and the characterisation of metric bisimulation [12] and stochastic bisimulation for systems with rewards [17] (see the related work section for an in-depth discussion). This paper aims at bridging this gap and comes up with, to the best of our knowledge, the first logical characterisation of approximate conformance for hybrid systems [2, 1] in terms of Metric Temporal Logic [23, 5].

To this end, we study the hybrid conformance notion due to Abbas, Mittelmann and Fainekos [2, 1], as well as its its associated preorder which we call hybrid refinement (for both notions, we also study their extensions to non-deterministic hybrid-systems). We provide logical characterisations for each of theses notions in terms of Metric Temporal Logic (MTL) and suitable notions of relaxation. We also show that the notions of relaxation proposed in the preservation result by Abbas, Mittelmann and Fainekos [1] is insufficiently precise to lead to a logical characterisation. We formulate our results in a general semantic domain, called generalised timed traces, which encompasses both discretised hybrid systems (as studied by Abbas, Mittelmann, and Fainekos [1]) and their continuous variants that have not been given a logical characterisation so far, to the best of our knowledge. Moreover, we study a generalisation of these results for both bounded and unbounded nondeterministic systems.

The contributions of this paper have both theoretical and practical motivation and relevance. The theoretical motivation for logical characterisation is that it not only provides an idea about the logic that is preserved under conformance (subject to relaxation) such as – in our case – MTL, but also it specifies precise bounds on the relaxation required for such formulae to hold. The practical motivation is that firstly, it provides designers with a precisely specified set of properties that carry over from specification to implementation (while preservations results only provide a rough approximation of such properties) and moreover, logical characterisation sets the scene for developing algorithms for finding distinguishing formulae, and hence, provide an alternative means for checking hybrid conformance. Logical characterisations have also proven to be a versatile auxiliary tool in e.g. developing congruence formats for operational semantics [7], as well as providing approximations of hybrid systems [26].

The rest of this paper is organised as follows. In Section 2, we review the related work and position our contributions with respect to the state of the art. In Section 3, we define some preliminary notions, including our semantic domain, the notions of hybrid refinement and conformance [1] and Metric Temporal Logic [6]. Subsequently in Section 4, we define appropriate notions of relaxations to characterise these notions using Metric Temporal Logic. We compare our results to the past preservation results in Section 5, where we show that the existing relaxation scheme for Metric Temporal Logic are too lax to serve for a logical characterisation of hybrid refinement and conformance. Namely, we prove there is a class of non-conforming implementations that do satisfy all relaxed MTL formulae satisfied by the specification. In Section 6, we conclude the paper, and present the directions of our ongoing research in this domain.

2 Related work

Logical characterisations of conformance relations allow for identifying conforming systems by means of the logical formulae satisfied by them. They also facilitate the converse operation, important from a practical perspective, namely, distinguishing non-conforming systems with a formula that forms a succinct counterexample.

Characterisations using modal logic have been studied extensively in the setting of exact behavioural semantics on discrete models such as labelled transition systems [21, 30]. In this context, characterisations use direct comparison i.e. inclusion of sets of formulae satisfied by systems in question; distinguishing formulae are those belonging to a set difference of such sets. Our work differs from this line of work in that it deals with approximate behavioural semantics and hence, cannot use standard inclusion check between sets of satisfied formulae.

To our knowledge, the first notion of characterisation for approximate behavioural semantics has been offered in the context of Metric Transition Systems [12] for linear and branching distances based on Metric Bisimulation [20, 19].

On a general level, our semantic model and conformance relation are different from those in [12, 20] in that they involve separate time and value dimensions, both of which can be subject to perturbations. Our choices for the semantic model and the notion of conformance are motivated by the practical applications of hybrid conformance [2, 1] in testing cyber-physical systems, e.g., in the automotive- [29] and healthcare domain [27]. Moreover, from a technical perspective, we base our characterisation on a logic with a qualitative (binary) satisfaction relation, but with quantities embedded in its syntax, namely, the Metric Temporal Logic (MTL). However, our approach can be easily translated to a quantitative setting of [12], by defining an evaluation of a formulae as the least degree of relaxation after applying which the formula is satisfied by a system. Also in this case, the choice of Metric Temporal Logic [23, 5] (and its concrete instantiation with signal values for propositions: Signal Temporal Logic [24]) is motivated by its wide-spread use in the hybrid systems literature and in practice [1, 18, 15].

Prabhakar, Vladimerou, Viswanathan, and Dullerud [26] provide a characterisation theorem for approximate simulation [19]; the characterisation serves as an auxiliary tool for developing approximations of hybrid systems with polynomial flows. In terms of semantic domain and relation under consideration, their characterisation result is strongly related to [12]. One technical feature which makes that paper somewhat closer in style to ours than [12] is the use of a relaxation operator (called a shrink of a formula in [26]).

Desharnais, Gupta, Jagadeesan and Panangaden [14] provide an approximate characterisation of probabilistic bisimulation for labelled Markov processes. They do so using a quantitative extension of Hennessy-Milner logic. This work has led to several follow-up applications, e.g., to a logical characterisation of differential privacy by Castiglioni, Chatzikokolakis, and Palamidessi [9]. Gburek and Baier [17] have recently investigated characterisation of bisimulation for stochastic systems with actions and rewards with two probabilistic logics: a very expressive APCTL*, and simpler APCTL$_\circ$, that can provide succinct distinguishing formulae. Unlike their approach [17], our work is set in the context of standard hybrid systems.

The results that appear closest to ours in terms of underlying models, and conformance relations that allow for disturbances in both time and space values, are logical preservation results for hybrid conformance [1] and Skorokhod conformance [16]. Both papers define syntactical transformations on temporal logics yielding more relaxed formulae; they differ on the conformance relations and temporal logics investigated. We improve upon them by providing different relaxation schemes that are proven to be tight, i.e., are precisely sufficient for a characterisation. Moreover, we generalise their results to semantic models that can encompass both discrete and continuous behaviour and non-determinism. Our framework of generalised timed traces subsumes both discrete timed state sequences (TSSs) and continuous trajectories, e.g., allowing for a comparison of behaviours of different types (such as sampled discretised behaviour against continuous trajectories).

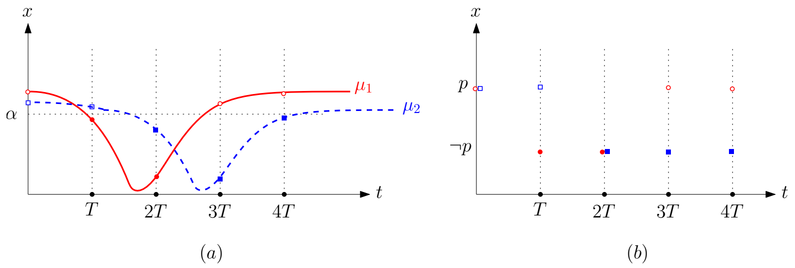


Figure 1 Examples of (a) continuous and (b) discretised GTTs.

3 Preliminaries

In this section, we define some preliminaries regarding our semantic domain, Metric Temporal Logic and notions of hybrid conformance and refinement.

Generalised timed traces and hybrid systems. In order for our theory to remain as general as possible, we define generalised timed traces, a notion that generalises both discrete semantic models, such as timed state sequences (TSSs) [1], and continuous-time trajectories [16]. A generalised timed trace is essentially a mapping from a discrete or continuous time domain to a set of values within some metric space.

▶ **Definition 1.** *Let $(\mathcal{Y}, d_{\mathcal{Y}})$ be a metric space. A $\mathcal{Y}$-valued generalised timed trace is a function $\mu : \mathcal{T} \to \mathcal{Y}$ such that $\mathcal{T} \subseteq \mathbb{R}_{\geq 0}$ is the time domain, and in addition $0 \in \mathcal{T}$ is the least element in $\mathcal{T}$. The set of all $\mathcal{Y}$-valued generalised timed traces is denoted by $GTT(\mathcal{Y})$.*

Observe that a timed state sequence (TSS) is simply a generalised timed trace with $\mathcal{T}$ being a finite subset of $\mathbb{R}_{\geq 0}$; moreover, in case $\mathcal{T}$ is an interval within $\mathbb{R}_{\geq 0}$, we obtain a standard continuous-time trajectory. We could generalise the domain of μ to any totally-ordered metric space, but we dispense with this generalisation here for the sake of simplicity. Likewise, the assumption that 0 is the least element of the time domain could be also dispensed with.

▶ **Example 2.** Consider trajectories μ_1 and μ_2 depicted in Figure 1.(a), where μ_1 represents the specification of a system and μ_2 its implementation. The mappings from the subset of reals in the domain of each trajectory to the value of x at the corresponding point form real-valued GTTs.

Consider the discretisation of these two trajectories where we sample the trajectories with a period T and we record whether the value of x at the sampling point is higher than α (denoted by $p \doteq x > \alpha$) or at most α (denoted by $\neg p \doteq x \leq \alpha$). The corresponding mappings from $\{0, T, 2T, 3T, 4T\}$ to $P = \{p, \neg p\}$ are discretised GTTs depicted in Figure 1.(b) are P-valued GTTs.

A hybrid system, defined below, is a mapping from initial conditions and inputs to sets of generalised (output) traces. We use the notation $\mathcal{P}(S)$ and $\mathcal{P}_{FIN}(S)$ denote, respectively, a powerset of S, and the powerset of S restricted to the finite subsets.

▶ **Definition 3.** *Given sets $\mathcal{C}$ and $\mathcal{I}$ of initial conditions and input space, the set of $\mathcal{Y}$-valued hybrid systems, denoted by $\mathcal{H}(\mathcal{C}, \mathcal{I}, \mathcal{Y})$ is the set of all functions of the type $\mathcal{C} \times \mathcal{I} \to \mathcal{P}(GTT(\mathcal{Y}))$. In addition, we distinguish the following two classes of hybrid systems: the class of* finitely branching hybrid systems *is defined as $\mathcal{H}_{FIN}(\mathcal{C}, \mathcal{I}, \mathcal{Y}) := \{H : \mathcal{C} \times \mathcal{I} \to \mathcal{P}_{FIN}(GTT(\mathcal{Y}))\}$; similarly, the class of* deterministic hybrid systems *is defined as $\mathcal{H}_{DET}(\mathcal{C}, \mathcal{I}, \mathcal{Y}) := \{H : \mathcal{C} \times \mathcal{I} \to \mathcal{P}(GTT(\mathcal{Y})) \,|\, \forall_{c \in \mathcal{C}, i \in \mathcal{I}} |H(c, i)| = 1\}$.*

Note that we intentionally left the nature of the initial conditions and input space implicit, as they play no role in the development of this paper. In reality, input conditions are typically constraints on input signals and the input space is typically a generalised timed trace with the same domain as the generalised timed trace for output. Also note that we focus mainly on finitely branching hybrid systems. When the parameters $\mathcal{I}, \mathcal{C}, \mathcal{Y}$ are not relevant or are clear from the context, we leave them out and refer to the set of hybrid systems with fixed parameters as $\mathcal{H}$.

3.1 Metric Temporal Logic

Metric Temporal Logic (MTL) [23, 5] is an extension of Linear Temporal Logic [25] with intervals; the introduction of intervals allows for reasoning about the real-time behaviour of dynamic systems once the propositions of the logic are interpreted over real-valued signals [24] (this interpretation of MTL is also called Signal Temporal Logic, or STL in the literature). MTL serves as an intuitive formalism for reasoning about hybrid systems [24, 1, 18, 15].

We work with the following language MTL^+ of MTL formulas in the negation-normal form

$$\phi ::= \mathsf{T} \mid \mathsf{F} \mid p \mid \neg p \mid \phi \wedge \phi \mid \phi \vee \phi \mid \phi \, \mathcal{U}_I \, \phi \mid \phi \, \mathcal{R}_I \, \phi$$

where p ranges over a collection of atomic propositions AP, and I ranges over intervals, $\mathcal{U}_I$ denotes the until operator and $\mathcal{R}_I$ denotes the release operator (both annotated with interval I).

For the purpose of relaxation, we shall also use the slightly extended language MTL^+_{ext} that in addition includes $p^+(\epsilon)$ and $p^-(\epsilon)$ constructs. Intuitively, they denote, respectively, the expansion- and contraction of the domain of validity of proposition p by ϵ.

$$\phi ::= \quad \mathsf{T} \mid \mathsf{F} \mid p \mid \neg p \mid \phi \wedge \phi \mid \phi \vee \phi \mid \phi \, \mathcal{U}_I \, \phi \mid \phi \, \mathcal{R}_I \, \phi \mid p^+(\epsilon) \mid p^-(\epsilon) \qquad (\epsilon \in \mathbb{R}_{\geq 0})$$

▶ **Example 4.** To illustrate the intuitive meaning of $p^+(\epsilon)$ and $p^-(\epsilon)$ consider the predicate $p := x > \alpha$ in Example 2. $p^+(\epsilon)$ relaxes p into $x > \alpha - \epsilon$; in other words $p^+(\epsilon)$ allows for an error margin of ϵ when checking p, while $p^-(\epsilon)$ shrinks p into $x > \alpha + \epsilon$. The latter is helpful for defining the relaxation of negated propositions.

In order to provide the formal semantics for MTL^+, we need two auxiliary definitions of δ-expansion and δ-contraction. Below, we assume the context of some metric space $(\mathcal{Y}, d_\mathcal{Y})$, and S ranges over subsets of $\mathcal{Y}$.

- $E(S, \delta) := \{x \in \mathcal{Y} \mid \exists y \in S : d_\mathcal{Y}(x, y) \leq \delta\}$ (δ-expansion)
- $C(S, \delta) := \mathcal{Y} \setminus E(\mathcal{Y} \setminus S, \delta)$ (δ-contraction)

Note that our definitions slightly differ from [1]. In particular, for any $y_0 \in \mathcal{Y}$, and the set $\overline{B_\epsilon}(y_0) = \{y \in \mathcal{Y} |, d_\mathcal{Y}(y, y_0) > \epsilon\}$ (complement of an ϵ-ball of point y_0), we have $E(\overline{B_\epsilon}(y_0), \epsilon) = \{y_0\}$ (rather than $\emptyset$ which the expansion of [1] would yield).

We also remark that the semantics of MTL^+_{ext} is provided in the context of an interpretation function $\mathcal{O} : AP \to \mathcal{P}(\mathcal{Y})$. This is a standard approach, similar to e.g. [1], but also to Signal Temporal Logic [24]. Note that the nature of the interpretation function restricts the expressive power of the logic, as the propositions are interpreted over the domain of values only (excluding time domain), which precludes expressing more powerful properties such as signal tracking (which is possible in Freeze LTL [16]).

▶ **Definition 5.** *Let $\mu : \mathcal{T} \to \mathcal{Y}$ be a generalised timed trace, $t \in \mathbb{R}$, and $\mathcal{O} : AP \to \mathcal{P}(\mathcal{Y})$ be an interpretation mapping for atomic propositions. The semantics of MTL^+_{ext} formulas is defined as follows:*
$(\mu, t) \models \mathsf{T} \quad (\mu, t) \not\models \mathsf{F}$
$(\mu, t) \models p$ *iff* $t \in \mathcal{T}$ *and* $\mu(t) \in \mathcal{O}(p)$
$(\mu, t) \models \neg p$ *iff* $t \in \mathcal{T}$ *and* $\mu(t) \notin \mathcal{O}(p)$
$(\mu, t) \models p^+(\epsilon)$ *iff* $t \in \mathcal{T}$ *and* $\mu(t) \in E(\mathcal{O}(p), \epsilon)$
$(\mu, t) \models p^-(\epsilon)$ *iff* $t \in \mathcal{T}$ *and* $\mu(t) \notin C(\mathcal{O}(p), \epsilon)$
$(\mu, t) \models \phi \wedge \psi$ *iff* $(\mu, t) \models \phi$ *and* $(\mu, t) \models \psi$
$(\mu, t) \models \phi \vee \psi$ *iff* $(\mu, t) \models \phi$ *or* $(\mu, t) \models \psi$
$(\mu, t) \models \phi \mathcal{U}_I \psi$ *iff* $\exists t' \in \mathcal{T}.\ t' - t \in I. (\mu, t') \models \psi$
$\wedge \forall t'' \in \mathcal{T}. t'' \in [t, t') \implies ((\mu, t'') \models \phi \vee (t'' - t \in I \wedge (\mu, t'') \models \psi))$
$(\mu, t) \models \psi \mathcal{R}_I \phi$ *iff* $\forall t' \in \mathcal{T}. (t' - t \in I \ \wedge\ (\mu, t') \not\models \phi) \implies (\exists t_1 \in \mathcal{T}. t_1 \in [t, t') \ \wedge\ (\mu, t_1) \models \psi)$

We say that a generalised timed trace $\mu : \mathcal{T} \to \mathcal{Y}$ satisfies an MTL^+ formula ϕ, notation $\mu \models \phi$ iff $(\mu, 0) \models \phi$. The satisfaction relation is lifted to hybrid systems in the standard manner, i.e., $H(c, i) \models \phi \iff \forall \mu \in H(c, i). \mu \models \phi$.

In the remainder of this paper, we use the common shorthand notation for eventually and always, defined as: $\Diamond_I \phi := \mathsf{T} \mathcal{U}_I \phi \quad \Box_I \phi := \mathsf{F} \mathcal{R}_I \phi$.

We remark that the semantics of the until operator slightly differs from the standard one used e.g. for MTL over discrete-time models. There, one simply requires the safety formula ϕ to hold in every time point before the “ultimate” formula ψ holds. In order to cater for dense-time domains where there may be no “earliest” time point satisfying ψ, we require that in all the preceding time points either ϕ, *or* ψ holds. A similar kind of semantics can be found in [16].

We also remark that the semantics of until operator makes it possible for the “ultimate” formula ψ to hold *before* the current state (time point); this is because we allow formulae to be annotated with arbitrary intervals, in particular those with negative endpoints.

Furthermore, note that the semantics allows for certain “ambiguous” cases where neither a formula nor its negation (which can be syntactically obtained by an appropriate transformation) is satisfied by a given state. This happens in case of (negated) propositions, and tuples of the form (μ, t), where t does not belong to the time domain $\mathcal{T}$. For instance, in case of a generalised timed trace $\mu : \{0, 1, 2, 3\} \to \mathbb{R}$ corresponding to a small sampling of a real-valued signal, and proposition pos such that $\mathcal{O}(\mathsf{pos}) = \mathbb{R}_{>0}$ we have $(\mu, \sqrt{2}) \not\models \mathsf{pos}$, and $(\mu, \sqrt{2}) \not\models \neg\,\mathsf{pos}$, regardless of the actual values of μ for the sampling points in the time domain.

However, if all occurrences of propositions in a formula are guarded by an until or release operator, the satisfaction status of a formula is never ambiguous – this is because semantics of those operators refer only to time points within the time domain. Throughout the rest of the paper, we work with propositions that are guarded with until or release and hence, in our context, the ambiguity is never an issue in the context of our theory.

3.2 Hybrid Conformance

Next, we provide the definition of hybrid conformance, due to Abbas and Fainekos [2, 1], in the context of our generalised semantic domain. Intuitively, hybrid conformance allows for conforming signal to differ up to τ in time and up to ϵ in the value. In addition to the “standard” hybrid conformance, which is a symmetric relation on traces, we also define its one-directional variant which we call hybrid refinement.

▶ **Definition 6.** *Let $\mu_1 : \mathcal{T}_1 \to \mathcal{Y}$ and $\mu_2 : \mathcal{T}_2 \to \mathcal{Y}$ be $\mathcal{Y}$-valued generalised timed traces. A trace μ_1 is a (τ, ϵ)-refinement of μ_2, notation $\mu_1 \sqsubseteq_{\tau,\epsilon} \mu_2$, iff:*

$$\forall t_1 \in dom(\mu_1).\ \exists t_2 \in dom(\mu_2).\ |t_2 - t_1| \leq \tau \wedge d_{\mathcal{Y}}(\mu_2(t_2), \mu_1(t_1)) \leq \epsilon$$

In the above definition, μ_2 can match any value in μ_1 within a sufficiently small time interval, but can potentially contain some other signal values that cannot be matched by μ_1. We know at least that the "behaviour"of μ_1 in terms of signal values does not go beyond those of μ_2 (up to the (τ, ϵ)-window).

By requiring two traces to be mutually conforming, we obtain the standard notion of hybrid conformance [2, 1] for individual traces:

▶ **Definition 7.** *Let $\mu_1 : \mathcal{T}_1 \to \mathcal{Y}$ and $\mu_2 : \mathcal{T}_2 \to \mathcal{Y}$ be $\mathcal{Y}$-valued generalised timed traces. μ_1 and μ_2 are (τ, ϵ)-close, denoted by $\mu_1 \sim_{\tau,\epsilon} \mu_2$, whenever $\mu_1 \sqsubseteq_{\tau,\epsilon} \mu_2$ and $\mu_2 \sqsubseteq_{\tau,\epsilon} \mu_1$.*

When the precise value of τ and ϵ is not relevant, we refer to (τ, ϵ)-refinement, and (τ, ϵ)-closeness, as respectively, hybrid refinement, and hybrid conformance. The two notions can be lifted to hybrid systems in the following manner:

▶ **Definition 8.**

1. *A system H_1 is a (τ, ϵ)-refinement of H_2, notation $H_1 \sqsubseteq_{\tau,\epsilon} H_2$, if for all $c \in \mathcal{C}$ and $i \in \mathcal{I}$, it holds that:*

$$\forall \mu_1 \in H_1(c, i).\ \exists \mu_2 \in H_2(c, i).\ \mu_1 \sqsubseteq_{\tau,\epsilon} \mu_2$$

2. *Two hybrid systems H_1, H_2 are (τ, ϵ)-close, denoted by $H_1 \sim_{\tau,\epsilon} H_2$, if and only if for all $c \in \mathcal{C}$ and $i \in \mathcal{I}$, it holds that*

$$\forall \mu_1 \in H_1(c, i).\ \exists \mu_2 \in H_2(c, i). \mu_1 \sim_{\tau,\epsilon} \mu_2$$

$$\forall \mu_2 \in H_2(c, i).\ \exists \mu_1 \in H_1(c, i). \mu_1 \sim_{\tau,\epsilon} \mu_2$$

4 Logical Characterisation of Hybrid Refinement and Hybrid Conformance

4.1 Logical Characterisation via Relaxation

Logical characterisation of a relation provides means to uniquely identify classes of related systems by sets of formulae in a certain logic. In case of non-exact relations involving some tolerance thresholds for disturbances, such as hybrid conformance or refinement, one cannot directly compare sets of formulae satisfied by systems in question.

Our approach to characterisation involves the notion of relaxation of logical formulae, that has been used in the context of hybrid systems [1, 16, 26]. It involves a syntactical transformation of a formula to a weaker one, which is supposed to be also satisfied by at least one trace of a conforming system.

For the purpose of logical characterisation, we introduce the following relation.

▶ **Definition 9.** *We say that a system potentially* exhibits property ϕ, *notation $H(c, i) \models_{\exists} \phi$, whenever* there exists $\mu \in H(c, i)$ *such that $\mu \models \phi$.*

The relation $\models_{\exists}$ can be seen as a variant of satisfaction relation for nondeterministic systems that has existential, rather than universal interpretation, the latter being the traditional interpretation in LTL literature. This alternative view on satisfaction is similar to one that is used in the context of Hennessy-Milner logic and its variations for behavioural models [21, 30], where a logical formula represents a (potentially) observable behaviour of a system. This approach is more suitable for the purpose of logical chracterisation.

Assume a logic (a collection of formulae) $\mathcal{L}$ and a notion of relaxation $\mathsf{rlx} : \mathcal{L} \to \mathcal{L}$. Our notion of characterisation can now be defined as follows

▶ **Definition 10.** *A logic $\mathcal{L}$ and a notion of relaxation $\mathit{rlx} : \mathcal{L} \to \mathcal{L}$ characterise a relation $R \subseteq \mathcal{H} \times \mathcal{H}$ if and only if, for any two systems H and H' we have:*

$$H\,R\,H' \iff \forall \phi \in \mathcal{L}.\ H \models_\exists \phi \implies H' \models_\exists \mathit{rlx}(\phi)$$

The implication from left to right is called preservation; in our context, there already exist some preservation results in the literature [1, 16]; the implication from right to left (called reflection) has not been studied for hybrid conformance and MTL to the best of our knowledge.

We remark that for certain classes of “well-behaved” relations, the implication under the existential interpretation in definition 10, namely $H \models_\exists \phi \implies H' \models_\exists \mathsf{rlx}(\phi)$, is equivalent to a dual one under the more common universal interpretation, i.e. $H' \models \phi \implies H \models \mathsf{rlx}(\phi)$. Regarding the two relations considered in our work, only hybrid conformance has this property on all systems, while hybrid refinement does not. This is because the underlying relation on individual traces is not symmetric, and moreover allows the presence of considerably different values on the side of the “larger” trace (as long as it also matches all the required values on other timepoints within the relevant time interval).

In this section, we define two novel (and in our view, very natural) relaxation operators on MTL which, as we subsequently show, precisely serve this purpose.

4.2 Characterisation of hybrid refinement

Relaxation operator $\mathsf{rlx}^{\sqsubseteq}_{\tau,\epsilon}$. We shall now introduce the first relaxation operator on MTL, which (as we subsequently prove) gives rise to the characterisation of hybrid refinement. Syntactically, it has a very simple structure: the actual relaxation is performed on the level of propositions only.

▶ **Definition 11.** *Let $\tau, \epsilon \geq 0$. The relaxation operator $\mathit{rlx}^{\sqsubseteq}_{\tau,\epsilon} : \mathit{MTL}^{+} \to \mathit{MTL}^{+}_{ext}$ is defined as follows:*

$$\begin{array}{lcl}
\mathit{rlx}^{\sqsubseteq}_{\tau,\epsilon}(\mathit{T}) = \mathit{T} & , & \mathit{rlx}^{\sqsubseteq}_{\tau,\epsilon}(\mathit{F}) = \mathit{F} \\
\mathit{rlx}^{\sqsubseteq}_{\tau,\epsilon}(p) = \Diamond_{[-\tau,\tau]}\, p^{+}(\epsilon) & , & \mathit{rlx}^{\sqsubseteq}_{\tau,\epsilon}(\neg p) = \Diamond_{[-\tau,\tau]}\, p^{-}(\epsilon) \\
\mathit{rlx}^{\sqsubseteq}_{\tau,\epsilon}(\phi_1 \wedge \phi_2) & = & \mathit{rlx}^{\sqsubseteq}_{\tau,\epsilon}(\phi_1) \wedge \mathit{rlx}^{\sqsubseteq}_{\tau,\epsilon}(\phi_2) \\
\mathit{rlx}^{\sqsubseteq}_{\tau,\epsilon}(\phi_1 \vee \phi_2) & = & \mathit{rlx}^{\sqsubseteq}_{\tau,\epsilon}(\phi_1) \vee \mathit{rlx}^{\sqsubseteq}_{\tau,\epsilon}(\phi_2) \\
\mathit{rlx}^{\sqsubseteq}_{\tau,\epsilon}(\phi\, \mathcal{U}_I\, \psi) & = & \mathit{rlx}^{\sqsubseteq}_{\tau,\epsilon}(\phi)\, \mathcal{U}_I\, \mathit{rlx}^{\sqsubseteq}_{\tau,\epsilon}(\psi) \\
\mathit{rlx}^{\sqsubseteq}_{\tau,\epsilon}(\phi\, \mathcal{R}_I\, \psi) & = & \mathit{rlx}^{\sqsubseteq}_{\tau,\epsilon}(\phi)\, \mathcal{R}_I\, \mathit{rlx}^{\sqsubseteq}_{\tau,\epsilon}(\psi)
\end{array}$$

Note that each relaxation of a formula different than T and F is guarded by either release or until formulae, and hence its satisfaction status is always unambiguous.

4.2.1 Characterisation of traces

We proceed to show that the introduced relaxation operator can be used to characterise the (τ, ϵ)-refinement, starting with the individual timed traces. Note that since the results below concern arbitrary generalised timed traces, they apply also to the setting with two traces of different kind, e.g., a discrete TSS against a continuous trajectory.

4.2.1.1 Preservation modulo relaxation

We start by proving that the satisfaction of MTL^+ formulae is preserved by the refinement relation $\sqsubseteq_{\tau,\epsilon}$ on timed traces modulo $\mathsf{rlx}^{\sqsubseteq}_{\tau,\epsilon}$ relaxation.

▶ **Proposition 12.** *Let $\mu_1 : \mathcal{T}_1 \to \mathcal{Y}$, $\mu_2 : \mathcal{T}_2 \to \mathcal{Y}$ be two $\mathcal{Y}$-valued generalised timed traces, and ϕ be an MTL formula. If $\mu_1 \sqsubseteq_{\tau,\epsilon} \mu_2$, then, for any $t \in \mathbb{R}$:*

$$(\mu_1, t) \models \phi \implies (\mu_2, t) \models \mathsf{rlx}^{\sqsubseteq}_{\tau,\epsilon}(\phi)$$

Proof. The proof proceeds by structural induction on the formula ϕ.

- $\phi = p$: since $(\mu_1, t) \models p$, we have $t \in \mathcal{T}_1$ and $\mu_1(t) \in \mathcal{O}(p)$. Furthermore, since $\mu_1 \sqsubseteq_{\tau,\epsilon} \mu_2$, we know that there is some t' such that $|t' - t| \le \tau$ and $d(\mu_1(t), \mu_2(t')) \le \epsilon$. We have thus $\mu_2(t') \in \mathcal{O}(p^+(\epsilon))$, and hence $(\mu_2, t') \models p^+(\epsilon)$. Moreover, since $|t' - t| \le \tau$, we obtain $(\mu_2, t) \models \Diamond_{[-\tau,\tau]}\, p^+(\epsilon) = \mathsf{rlx}^{\sqsubseteq}_{\tau,\epsilon}(p)$.
- $\phi = \neg p$: since $(\mu_1, t) \models \neg p$, we have $t \in \mathcal{T}_1$ and $\mu_1(t) \notin \mathcal{O}(p)$. Furthermore, since $\mu_1 \sqsubseteq_{\tau,\epsilon} \mu_2$, we know that there is some t' such that $|t' - t| \le \tau$ and $d(\mu_1(t), \mu_2(t')) \le \epsilon$. From the latter and $\mu_1(t) \in \mathcal{Y} \backslash \mathcal{O}(p)$, we obtain $\mu_2(t') \in E(\mathcal{Y} \backslash \mathcal{O}(p), \epsilon)$, which is equivalent to $\mu_2(t') \notin C(\mathcal{O}(p), \epsilon)$. Hence $(\mu_2, t) \models \Diamond_{[-\tau,\tau]}\, p^-(\epsilon) = \mathsf{rlx}^{\sqsubseteq}_{\tau,\epsilon}(\neg p)$
- $\phi = \phi \,\mathcal{U}_I\, \psi$: since $(\mu_1, t) \models \phi \,\mathcal{U}_I\, \psi$, there is some $t_1 \in \mathcal{T}_1$ such that $t_1 - t \in I$ and $(\mu_1, t_1) \models \psi$, and moreover for any $t_0 \in [t, t_1)$ we have $(\mu_1, t_0) \models \phi \vee (\mu_1, t_0) \models \psi$. By applying the inductive hypothesis, we obtain that $(\mu_2, t_1) \models \mathsf{rlx}^{\sqsubseteq}_{\tau,\epsilon}(\psi)$, and for any $t_0 \in [t, t_1)$ we have $(\mu_2, t_0) \models \mathsf{rlx}^{\sqsubseteq}_{\tau,\epsilon}(\phi)$ or $(\mu_2, t_0) \models \mathsf{rlx}^{\sqsubseteq}_{\tau,\epsilon}(\psi)$. We thus have $(\mu_2, t) \models \mathsf{rlx}^{\sqsubseteq}_{\tau,\epsilon}(\phi)\,\mathcal{U}_I\, \mathsf{rlx}^{\sqsubseteq}_{\tau,\epsilon}(\psi)$, and from the definition of relaxation we immediately obtain $(\mu_2, t) \models \mathsf{rlx}^{\sqsubseteq}_{\tau,\epsilon}(\phi \,\mathcal{U}_I\, \psi)$.
- $\phi = \phi \,\mathcal{R}_I\, \psi$: take any $t' \in \mathcal{T}_2$ such that $t' - t \in I$ and $(\mu_2, t') \not\models \mathsf{rlx}^{\sqsubseteq}_{\tau,\epsilon}(\psi)$. From the inductive hypothesis, we have $(\mu_1, t') \not\models \psi$, and since $(\mu_1, t) \models \phi \,\mathcal{R}_I\, \psi$, we know that there is some $t_1 \in \mathcal{T}_1$ such that $t_1 \in [t, t')$, and $(\mu_1, t_1) \models \phi$. By applying the inductive hypothesis again, we obtain $(\mu_2, t_1) \models \mathsf{rlx}^{\sqsubseteq}_{\tau,\epsilon}(\phi)$. From the statements obtained above we can now infer that $(\mu_2, t) \models \mathsf{rlx}^{\sqsubseteq}_{\tau,\epsilon}(\phi \,\mathcal{R}_I\, \psi)$. ◀

4.2.1.2 Existence of distinguishing formula

We shall now prove that the converse of the preceding theorem holds as well: whenever a timed trace is not a (τ, ϵ)-refinement of another, we can always find an MTL formula that witnesses this, that is, preservation modulo $\mathsf{rlx}^{\sqsubseteq}_{\tau,\epsilon}$ relaxation operator does not hold.

▶ **Proposition 13.** *Let $\mu_1 : \mathcal{T}_1 \to \mathcal{Y}$ and $\mu_2 : \mathcal{T}_2 \to \mathcal{Y}$ be two $\mathcal{Y}$-valued timed traces. If $\mu_1 \not\sqsubseteq_{\tau,\epsilon} \mu_2$, then there is a formula $\phi \in \mathsf{MTL}^+$ such that ϕ distinguishes μ_1 from μ_2 modulo relaxation $\mathsf{rlx}^{\sqsubseteq}_{\tau,\epsilon}$, that is $\mu_1 \models \phi \wedge \mu_2 \not\models \mathsf{rlx}^{\sqsubseteq}_{\tau,\epsilon}(\phi)$*

Proof. Suppose that there is some $t_1 \in \mathcal{T}_1$ for which there is no $t_2 \in \mathcal{T}_2$ such that $|t_2 - t_1| \le \tau$ and $|\mu_2(t_2) - \mu_1(t_1)| \le \epsilon$. Consider an MTL formula $\phi = \Diamond_{[t_1,t_1]} p$, where $\mathcal{O}(p) = \{\mu_1(t_1)\}$. Obviously, we have $\mu_1 \models \phi$, however, the relaxed version of the formula $\mathsf{rlx}^{\sqsubseteq}_{\tau,\epsilon}(\phi) = \Diamond_{[t_1,t_1]}\Diamond_{[-\tau,\tau]} p^+(\epsilon)$ cannot be satisfied by μ_2. ◀

4.2.2 Characterisation of hybrid systems

4.2.2.1 Finitely branching systems

Propositions 12 and 13 provide the characterisation of relation $\sqsubseteq_{\tau,\epsilon}$ by MTL^+ through the relaxation $\mathsf{rlx}^{\sqsubseteq}_{\tau,\epsilon}$ on individual traces. Based on those results, for hybrid systems that are finitely branching (i.e. have bounded non-determinism, see definition 3), the characterisation result for hybrid refinement can be obtained in a straightforward manner.

▶ **Theorem 14.** *The logic MTL^+, together with the relaxation operator $\mathsf{rlx}^{\sqsubseteq}_{\tau,\epsilon}$, characterise the conformance relation $\sqsubseteq_{\tau,\epsilon}$ on finitely branching hybrid systems. That is, for arbitrary finitely branching hybrid systems H and H', the following statements hold:*

$$H \sqsubseteq_{\tau,\epsilon} H' \iff (\forall \phi \in \mathsf{MTL}^+.\ H \models_\exists \phi \implies H' \models_\exists \mathsf{rlx}^{\sqsubseteq}_{\tau,\epsilon}(\phi))$$

Proof.

- (preservation): Take any two hybrid systems H_1, H_2 such that $H_1 \sqsubseteq_{\tau,\epsilon} H_2$. Take any $c \in \mathcal{C}, i \in \mathcal{I}$. Suppose w.l.o.g. that $H_1(c,i) \models_\exists \phi$; we need to show that $H_2(c,i) \models_\exists \mathsf{rlx}^{\sqsubseteq}_{\tau,\epsilon}(\phi)$. From $H_1(c,i) \models_\exists \phi$ we know that there is a $\mu_1 \in H_1(c,i)$ such that $\mu_1 \models \phi$. Moreover, since $H_1 \sqsubseteq_{\tau,\epsilon} H_2$, there is some $\mu_2 \in H_2(c,i)$ such that $\mu_1 \sqsubseteq_{\tau,\epsilon} \mu_2$. From Proposition 12 we thus obtain $\mu_2 \models \mathsf{rlx}^{\sqsubseteq}_{\tau,\epsilon}(\phi)$, and hence $H_2(c,i) \models_\exists \mathsf{rlx}^{\sqsubseteq}_{\tau,\epsilon}(\phi)$.
- (reflection/distinguishing formula): Suppose that $H_1 \not\sqsubseteq_{\tau,\epsilon} H_2$. Then for certain $c \in \mathcal{C}, i \in \mathcal{I}$ there is some $\mu_1 \in H_1(c,i)$ such that for all $\mu_2^j \in H_2(c,i)$ we have $\mu_1 \not\sqsubseteq_{\tau,\epsilon} \mu_2^j$. From Proposition 13 we know that for each such $\mu_2^j \in H_2(c,i)$ there is a distinguishing formula ϕ_j such that $\mu_1 \models \phi_j$ and $\mu_2^j \not\models \mathsf{rlx}^{\sqsubseteq}_{\tau,\epsilon}(\phi_j)$. Consider a formula $\Phi = \bigwedge_{j:\mu_2^j \in H_2(c,i)} \phi_j$. Since $H_2(c,i)$ is a finite set, Φ is a well-formed MTL^+formula. We now have $H_1(c,i) \models_\exists \Phi$, but since obviously for any j, $\mu_2^j \not\models \mathsf{rlx}^{\sqsubseteq}_{\tau,\epsilon}(\Phi)$, we also have $H_2(c,i) \not\models_\exists \mathsf{rlx}^{\sqsubseteq}_{\tau,\epsilon}(\Phi)$. Hence Φ distinguishes $H_1(c,i)$ from $H_2(c,i)$. ◀

4.2.2.2 Systems with unbounded non-determinism

In order to provide characterisation for hybrid refinement on systems with infinite branching, one needs to endow the logic MTL^+ with infinite conjunctions and disjunction; the syntax of such logic, denoted with MTL^+_∞, is given below (*Ind* ranges over arbitrary sets of indices).

$$\phi ::= \mathsf{T} \mid \mathsf{F} \mid p \mid \neg p \mid \bigwedge_{i \in Ind} \phi_i \mid \bigvee_{i \in Ind} \phi_i \mid \phi\, \mathcal{U}_I\, \phi \mid \phi\, \mathcal{R}_I\, \phi$$

▶ **Theorem 15.** *The logic MTL^+_∞, together with the relaxation operator $\mathsf{rlx}^{\sqsubseteq}_{\tau,\epsilon}$, characterise the conformance relation $\sqsubseteq_{\tau,\epsilon}$ on arbitrary hybrid systems.*

Proof. The proof is nearly the same as the one of Theorem 14, except that while proving the reflection property, the set of distinguishing formulae for individual traces may be infinite. However, a disjunction over such a set is now a well-formed MTL^+_∞ formula, hence the construction is valid. ◀

4.3 Characterisation of hybrid conformance

4.3.1 Relaxation operator $\mathsf{rlx}^{\sim}_{\tau,\epsilon}$

While the relaxation operator $\mathsf{rlx}^{\sqsubseteq}_{\tau,\epsilon}$ introduced in the previous section allows one to preserve – up to the relevant (τ,ϵ)-window – properties of (signal values at) individual timepoints, it falls short of preserving properties of entire intervals. Therefore, in order to characterise the standard, symmetric notion of (τ,ϵ)-closeness, or hybrid conformance, one needs a finer notion of relaxation.

In what follows, we shall use the following notation: for an interval I, by $I_{<a,b>}$ we denote the modified interval: $I_{<a,b>} := \{x \in \mathbb{R} \mid \exists x_a, x_b \in I : x_a + a \leq x \land x \leq x_b + b\}$.

Below, we define a relaxation operator $\mathsf{rlx}^{\sim}_{\tau,\epsilon}$ where:

- for propositions not in the scope of a temporal operator, the relaxation is done similarly as in the $\mathsf{rlx}^{\sqsubseteq}_{\tau,\epsilon}$ operator

- for temporal operators, the interval endpoints are modified (i.e. "shrinked" to relax the temporal obligations accordingly)
- for propositions guarded by a temporal operator, only ϵ-relaxation of a signal value is perfomed (the relaxation of timeline has already been handled through interval relaxation)

▶ **Definition 16.** *Let $\tau, \epsilon \geq 0$. The relaxation operator $\mathsf{rlx}^{\sim}_{\tau,\epsilon} : \mathsf{MTL}^{+} \to \mathsf{MTL}^{+}_{ext}$ is defined as follows:*

$$\begin{array}{lcl}
\mathsf{rlx}^{\sim}_{\tau,\epsilon}(\mathsf{T}) = \mathsf{T} & , & \mathsf{rlx}^{\sim}_{\tau,\epsilon}(\mathsf{F}) = \mathsf{F} \\
\mathsf{rlx}^{\sim}_{\tau,\epsilon}(p) = \Diamond_{[-\tau,\tau]} p^{+}(\epsilon) & , & \mathsf{rlx}^{\sim}_{\tau,\epsilon}(\neg p) = \Diamond_{[-\tau,\tau]} p^{-}(\epsilon) \\
\mathsf{rlx}^{\sim}_{\tau,\epsilon}(\phi_1 \wedge \phi_2) & = & \mathsf{rlx}^{\sim}_{\tau,\epsilon}(\phi_1) \wedge \mathsf{rlx}^{\sim}_{\tau,\epsilon}(\phi_2) \\
\mathsf{rlx}^{\sim}_{\tau,\epsilon}(\phi_1 \vee \phi_2) & = & \mathsf{rlx}^{\sim}_{\tau,\epsilon}(\phi_1) \vee \mathsf{rlx}^{\sim}_{\tau,\epsilon}(\phi_2) \\
\mathsf{rlx}^{\sim}_{\tau,\epsilon}(\phi\, \mathcal{U}_I\, \psi) & = & \begin{cases} \Diamond_{[\tau,\tau]} \left(\mathsf{t\text{-}rlx}^{\sim}_{\tau,\epsilon}(\phi)\, \mathcal{U}_{I_{<0,-2\tau>}}\, (\Diamond_{[0,2\tau]} \mathsf{t\text{-}rlx}^{\sim}_{\tau,\epsilon}(\psi))\right) & if\ \ I_{<0,-2\tau>} \neq \emptyset \\ \Diamond_{I_{<-\tau,\tau>}} \mathsf{t\text{-}rlx}^{\sim}_{\tau,\epsilon}(\psi) & if\ \ I_{<0,-2\tau>} = \emptyset \end{cases} \\
\mathsf{rlx}^{\sim}_{\tau,\epsilon}(\phi\, \mathcal{R}_I\, \psi) & = & (\Diamond_{[-\tau,\tau]} \mathsf{t\text{-}rlx}^{\sim}_{\tau,\epsilon}(\phi))\, \mathcal{R}_{I_{<\tau,-\tau>}}\, \mathsf{t\text{-}rlx}^{\sim}_{\tau,\epsilon}(\psi)
\end{array}$$

where the auxilliary relaxation $\mathsf{t\text{-}rlx}^{\sim}_{\tau,\epsilon}$ for subformulae guarded by a temporal operator is defined as follows:

$$\begin{array}{lcl}
\mathsf{t\text{-}rlx}^{\sim}_{\tau,\epsilon}(\mathsf{T}) = \mathsf{T} & , & \mathsf{t\text{-}rlx}^{\sim}_{\tau,\epsilon}(\mathsf{F}) = \mathsf{F} \\
\mathsf{t\text{-}rlx}^{\sim}_{\tau,\epsilon}(p) = p^{+}(\epsilon) & , & \mathsf{t\text{-}rlx}^{\sim}_{\tau,\epsilon}(\neg p) = p^{-}(\epsilon) \\
\mathsf{t\text{-}rlx}^{\sim}_{\tau,\epsilon}(\phi_1 \wedge \phi_2) & = & \mathsf{t\text{-}rlx}^{\sim}_{\tau,\epsilon}(\phi_1) \wedge \mathsf{t\text{-}rlx}^{\sim}_{\tau,\epsilon}(\phi_2) \\
\mathsf{t\text{-}rlx}^{\sim}_{\tau,\epsilon}(\phi_1 \vee \phi_2) & = & \mathsf{t\text{-}rlx}^{\sim}_{\tau,\epsilon}(\phi_1) \vee \mathsf{t\text{-}rlx}^{\sim}_{\tau,\epsilon}(\phi_2) \\
\mathsf{t\text{-}rlx}^{\sim}_{\tau,\epsilon}(\phi\, \mathcal{U}_I\, \psi) & = & \begin{cases} \Diamond_{[\tau,\tau]} \left(\mathsf{t\text{-}rlx}^{\sim}_{\tau,\epsilon}(\phi)\, \mathcal{U}_{I_{<0,-2\tau>}}\, (\Diamond_{[0,2\tau]} \mathsf{t\text{-}rlx}^{\sim}_{\tau,\epsilon}(\psi))\right) & if\ \ I_{<0,-2\tau>} \neq \emptyset \\ \Diamond_{I_{<-\tau,\tau>}} \mathsf{t\text{-}rlx}^{\sim}_{\tau,\epsilon}(\psi) & if\ \ I_{<0,-2\tau>} = \emptyset \end{cases} \\
\mathsf{t\text{-}rlx}^{\sim}_{\tau,\epsilon}(\phi\, \mathcal{R}_I\, \psi) & = & (\Diamond_{[-\tau,\tau]} \mathsf{t\text{-}rlx}^{\sim}_{\tau,\epsilon}(\phi))\, \mathcal{R}_{I_{<\tau,-\tau>}}\, \mathsf{t\text{-}rlx}^{\sim}_{\tau,\epsilon}(\psi)
\end{array}$$

4.3.2 Characterisation of traces

4.3.2.1 Preservation

Before stating the main preservation property, we prove the key lemma which lists certain properties of the auxilliary relaxation operator $\mathsf{t\text{-}rlx}^{\sim}_{\tau,\epsilon}$.

▶ **Lemma 17.** *Suppose $\mu_1 \sim_{\tau,\epsilon} \mu_2$. For any $\phi \in \mathsf{MTL}^{+}$ we have:*

1. $\mu_1, t \models \phi \implies \exists t' \in [t-\tau, t+\tau].\ \mu_2, t' \models \mathsf{t\text{-}rlx}^{\sim}_{\tau,\epsilon}(\phi)$
2. $(\forall t \in I.\, \mu_1, t \models \phi) \implies (\forall t \in I_{<\tau,-\tau>}.\ \mu_2, t \models \mathsf{t\text{-}rlx}^{\sim}_{\tau,\epsilon}(\phi))$
3. *if in addition ϕ is of the form $\chi\, \mathcal{U}_I\, \psi$ or $\psi\, \mathcal{R}_I\, \chi$, then $\mu_1, t \models \phi \implies \mu_2, t \models \mathsf{t\text{-}rlx}^{\sim}_{\tau,\epsilon}(\phi)$*

Proof. We proceed by structural induction on ϕ; for technical reasons, it is convenient to prove all the properties simultaneously. We focus on three key cases: atomic propositions, as well as the until and release operators.

- $\phi = p$:
  1. Suppose $\mu_1, t \models p$; from the semantics of MTL^{+} this means that $\mu_1(t) \in \mathcal{O}(p)$. Since $\mu_1 \sim_{\tau,\epsilon} \mu_2$, there is some $t' \in [t-\tau, t+\tau]$ such that $d_{\mathcal{Y}}(\mu_1(t), \mu_2(t)') \leq \epsilon$. From this and $\mu_1(t) \in \mathcal{O}(p)$ we obtain $\mu_2(t') \in E(\mathcal{O}(p), \epsilon)$, and hence $\mu_2, t' \models p^{+}(\epsilon) = \mathsf{t\text{-}rlx}^{\sim}_{\tau,\epsilon}(p)$.
  2. Suppose that for all $t \in I$ we have $\mu_1, t \models p$, that is, for all $t \in I$ $\mu_1(t) \in \mathcal{O}(p)$. Take any $t_2 \in I_{<\tau,-\tau>}$. Observe that the "matching" timepoint for μ_2 and t_2 in μ_1 must be in the interval I, i.e. there is some $t_1 \in I$ such that $d_{\mathcal{Y}}(\mu_1(t_1), \mu_2(t_2)) \leq \epsilon$. Since $t_1 \in I$, we have $\mu_1(t_1) \in \mathcal{O}(p)$, and hence $\mu_2(t_2) \in E(\mathcal{O}(p, \epsilon))$, from which $\mu_2, t_2 \models p^{+}(\epsilon) = \mathsf{t\text{-}rlx}^{\sim}_{\tau,\epsilon}(p)$ follows.

- $\phi = \chi \mathcal{U}_I \psi$: we only need to prove the third statement, as it is stronger than the first two. Moreover, we consider only the more involved case when $I_{<0,-2\tau>} \neq \emptyset$.
  Suppose $\mu_1, t \models \chi \mathcal{U}_I \psi$. Then there is some $t_\psi \in t + I$ such that $\mu_1, t_\psi \models \psi$ (note that since $I_{<0,-2\tau>} \neq \emptyset$, we have $t_\psi - t \geq 2\tau$). From $\mu_1 \sim_{\tau,\epsilon} \mu_2$ and applying the inductive hypothesis on statement 1 of Lemma 17 there is some $t'_\psi \in [t_\psi - \tau, t_\psi + \tau]$ such that $\mu_2, t'_\psi \models \mathsf{t\text{-}rlx}^{\sim}_{\tau,\epsilon}(\psi)$. This in particular implies that
  (*) $\mu_2, t_\psi - \tau \models \Diamond_{[0,2\tau]}\mathsf{t\text{-}rlx}^{\sim}_{\tau,\epsilon}(\psi)$.
  From $\mu_1, t \models \chi \mathcal{U}_I \psi$ it further follows that for all $t' \in [t, t_\psi)$ we have $\mu_1, t' \models \chi$. From applying the inductive hypothesis on statement 2 of Lemma 17 we therefore have
  (**) for all $t' \in [t + \tau, t_\psi - \tau)$ we have $\mu_2, t' \models \mathsf{t\text{-}rlx}^{\sim}_{\tau,\epsilon}(\chi)$.
  That $\mu_2, t \models \Diamond_{[\tau,\tau]} \left(\mathsf{t\text{-}rlx}^{\sim}_{\tau,\epsilon}(\chi) \mathcal{U}_{I_{<0,-2\tau>}} (\Diamond_{[0,2\tau]}\mathsf{t\text{-}rlx}^{\sim}_{\tau,\epsilon}(\psi))\right) = \mathsf{t\text{-}rlx}^{\sim}_{\tau,\epsilon}(\chi \mathcal{U}_I \psi)$ now follows immediately from (*) and (**).
- $\phi = \psi \mathcal{R}_I \chi$: similarly as above, we only prove the third statement. Note that whenever the interval I is strictly shorter than 2τ, we have $I_{<0,-2\tau>} = \emptyset$, and the relaxation yields a formula equivalent to T.
  Take any $t'_{\neg\chi} \in t + I_{<\tau,-\tau>}$ such that $\mu_2, t'_{\neg\chi} \not\models \mathsf{t\text{-}rlx}^{\sim}_{\tau,\epsilon}(\chi)$. Consider the interval $I \cap [t, t'_{\neg\chi} + \tau]$. There must be some $t_{\neg\chi} \in [t'_{\neg\chi} - \tau, t'_{\neg\chi} + \tau] \subseteq t + I$ such that $\mu_1, t_{\neg\chi} \not\models \chi$. Indeed, were it not the case, then from the inductive hypothesis (statement 2), we would have that for all $t' \in [t'_{\neg\chi}, t'_{\neg\chi}]$, $t' \models \mathsf{t\text{-}rlx}^{\sim}_{\tau,\epsilon}(\chi)$, contradicting $\mu_2, t'_{\neg\chi} \not\models \mathsf{t\text{-}rlx}^{\sim}_{\tau,\epsilon}(\chi)$.
  From $\mu_1, t \models \psi \mathcal{R}_I \chi$ and $\mu_1, t_{\neg\chi} \not\models \chi$, one obtains existence of some $t_\psi \in [t, t_{\neg\chi})$ such that $\mu_1, t_\psi \models \psi$. From the inductive hypothesis (1) we know that $\mu_2, t_\psi \models \Diamond_{[-\tau,\tau]}\mathsf{t\text{-}rlx}^{\sim}_{\tau,\epsilon}(\psi)$.
  We have thus shown that $\mu_2, t \models (\Diamond_{[-\tau,\tau]}\mathsf{t\text{-}rlx}^{\sim}_{\tau,\epsilon}(\psi)) \mathcal{R}_{I_{<\tau,-\tau>}} \mathsf{t\text{-}rlx}^{\sim}_{\tau,\epsilon}(\chi) = \mathsf{t\text{-}rlx}^{\sim}_{\tau,\epsilon}(\psi \mathcal{R}_I \chi)$ ◀

The preservation property is given in the proposition below.

▶ **Proposition 18.** $\mu_1 \sim_{\tau,\epsilon} \mu_2 \implies \forall \phi, t.\ \mu_1, t \models \phi \implies \mu_2, t \models \mathit{rlx}^{\sim}_{\tau,\epsilon}(\phi)$

Proof. Formally, the proof proceeds by structural induction. However, the key cases of temporal operators are now immediate corollaries of Lemma 17 (point 3); while for the remaining cases including base the proof is very straightforward. ◀

4.3.2.2 Reflection

We proceed to show that for non-conforming traces, one can always find a distinguishing formula, regardless of the "direction" in which the conformance fails. Since $\sim_{\tau,\epsilon}$ is symmetric, this is equivalent to the statement that if $\mu_1 \not\sim_{\tau,\epsilon} \mu_2$, then one can find both a formula distinguishing μ_1 from μ_2, and also one that distinguishes μ_2 from μ_1.

▶ **Proposition 19.** $\mu_1 \not\sim_{\tau,\epsilon} \mu_2 \implies \exists \phi.\ \mu_1 \models \phi \wedge \mu_2 \not\models \mathit{rlx}^{\sim}_{\tau,\epsilon}(\phi)$

Proof. Suppose $\mu_1 \not\sim_{\tau,\epsilon} \mu_2$; we show that there is always a formula that distinguishes μ_1 from μ_2. We distinguish two cases:

- there is some $t_1 \in \mathcal{T}_1$ such that the value $\mu_1(t_1)$ cannot be matched within the (τ, ϵ)-window by μ_2, that is:

  $$(*) \ \ \forall t' \in \mathcal{T}_2.\ |t' - t_1| \leq \tau \implies d_{\mathcal{Y}}(\mu_2(t'), \mu_1(t_1)) > \epsilon$$

  We use a similar construction as for the relaxation $\mathsf{rlx}^{\sqsubseteq}_{\tau,\epsilon}$, by defining

  $$\Phi_{DIST} := \Diamond_{[t_1,t_1]} p$$

  where $\mathcal{O}(p) = \{\mu_1(t_1)\}$. Then $\mathsf{rlx}^{\sim}_{\tau,\epsilon}(\Phi_{DIST}) = \Diamond_{[t_1-\tau,t_1+\tau]}\, p^+(\epsilon)$. We have $\mu_1 \models \Phi_{DIST}$, but from (*) we clearly have $\mu_2 \not\models \mathsf{rlx}^{\sim}_{\tau,\epsilon}(\Phi_{DIST})$.

- there is some $t_2 \in \mathcal{T}_2$ that cannot be matched by μ_1, that is: that is:

  $$\forall t' \in \mathcal{T}_1.\ |t' - t_2| \leq \tau \implies d_{\mathcal{Y}}(\mu_1(t'), \mu_2(t_2)) > \epsilon$$

  we define

  $$\Phi_{DIST} := \Box_{[t_2-\tau, t_2+\tau]} p$$

  where $\mathcal{O}(p) = \{y \in \mathcal{Y} \mid d_{\mathcal{Y}}(y, \mu_2(t_2)) > \epsilon\}$. Note that $p^+(\epsilon) = \mathcal{Y} \setminus \{\mu_2(t_2)\}$ (at this point using our definition of expansion operator rather than the one from [1] proves essential). We have $\mu_1 \models \Phi_{DIST}$, but on the other hand: $\mathsf{rlx}^{\sim}_{\tau,\epsilon}(\Phi_{DIST}) = (\Diamond_{[-\tau,\tau]} \mathsf{F})\, \mathcal{R}_{[t_2,t_2]}\, p^+(\epsilon) \equiv \Box_{[t_2,t_2]} p^+(\epsilon)$, and since $\mu_2(t_2) \notin \mathcal{Y} \setminus \{\mu_2(t_2)\} = p^+(\epsilon)$, we have $\mu_2 \not\models \mathsf{rlx}^{\sim}_{\tau,\epsilon}(\Phi_{DIST})$. ◀

4.3.3 Characterisation of hybrid systems

Characterisation results for hybrid conformance and their proofs share many similarities with those for hybrid refinement. One fine point worth noting is the proof of reflection property: when, similarly as in the proof of Theorem 14, we arrive at the case when $\mu_1 \not\sim_{\tau,\epsilon} \mu_2^j$, we know from Proposition 19 that for all j there is a formula that distinguishes μ_1 *from* μ_2^j, regardless of the direction in which the (τ, ϵ)-matching fails . We therefore have a family of formulae distinguishing μ_1 from μ_2^j for each j, and hence can construct a distinguishing formula by taking their conjunction.

In addition, since hybrid conformance is based on a symmetric relation on individual traces, the characterisation result holds for the standard (universal) interpretation of satisfaction relation as well.

▶ **Theorem 20.** *The logic MTL^+ [resp. MTL^+_∞], together with the relaxation operator $\mathsf{rlx}^{\sim}_{\tau,\epsilon}$, characterise the conformance relation $\sqsubseteq_{\tau,\epsilon}$ on finitely branching [resp. arbitrary] hybrid systems. That is, for finitely branching [resp. arbitrary] hybrid systems H and H', the following statements hold:*

$$H \sim_{\tau,\epsilon} H' \iff (\forall \phi \in \mathsf{MTL}^+\, [\mathsf{MTL}^+_\infty].\ H \models_\exists \phi \implies H' \models_\exists \mathsf{rlx}^{\sqsubseteq}_{\tau,\epsilon}(\phi))$$

Moreover, the characterisation result holds for the universal interpretation of satisfaction relation as well, that is:

$$H \sim_{\tau,\epsilon} H' \iff (\forall \phi \in \mathsf{MTL}^+\, [\mathsf{MTL}^+_\infty].\ H' \models \phi \implies H \models \mathsf{rlx}^{\sqsubseteq}_{\tau,\epsilon}(\phi))$$

5 Comparison with an existing relaxation

In this section, we discuss the existing relaxation operator for MTL from the literature due to Abbas, Mittelmann, and Fainekos [1], which is known to preserve MTL formulae for discrete samplings (timed-state sequences). We show that their relaxation cannot distinguish between traces not related by hybrid conformance, and hence is too lax for the purpose of logical characterisation for either hybrid conformance, or refinement.

5.1 AMF-Relaxation

We recall the relaxation operator from [1], which we call AMF-relaxation (for Abbas, Mittelmann, and Fainekos). Originally the definition was given on the super-dense time domain (i.e., a time domain that allows for specifying the ordering of simultaneous events).

Since the "super-denseness" of the time domain does not have any influence on our study, we simplify the time domain to a dense time domain (such as non-negative real numbers). We also adapt the presentation to the generalised timed traces framework.

▶ **Definition 21.** *Given $\tau, \epsilon \geq 0$, the relaxation operator $[]^{AMF}_{\tau,\epsilon} : \mathsf{MTL}^+ \to \mathsf{MTL}^+_{ext}$ is defined as follows:*

$$\begin{array}{lcl}
[\mathsf{T}]^{AMF}_{\tau,\epsilon} = \mathsf{T} & , & [\mathsf{F}]^{AMF}_{\tau,\epsilon} = \mathsf{F} \\
[p]^{AMF}_{\tau,\epsilon} = p^+(\epsilon) & , & [\neg p]^{AMF}_{\tau,\epsilon} = p^-(\epsilon) \\
[\phi_1 \wedge \phi_2]^{AMF}_{\tau,\epsilon} & = & [\phi_1]^{AMF}_{\tau,\epsilon} \wedge [\phi_2]^{AMF}_{\tau,\epsilon} \\
[\phi_1 \vee \phi_2]^{AMF}_{\tau,\epsilon} & = & [\phi_1]^{AMF}_{\tau,\epsilon} \vee [\phi_2]^{AMF}_{\tau,\epsilon} \\
[\phi\, \mathcal{U}_I\, \psi]^{AMF}_{\tau,\epsilon} & = & (\Diamond_{(-2\tau,0]} [\phi]^{AMF}_{\tau,\epsilon})\, \mathcal{U}_{I_{\ll -2\tau, 2\tau \gg}}\, (\Diamond_{[0,2\tau)} [\psi]^{AMF}_{\tau,\epsilon}) \\
[\phi\, \mathcal{R}_I\, \psi]^{AMF}_{\tau,\epsilon} & = & (\Diamond_{(-2\tau,0]} [\phi]^{AMF}_{\tau,\epsilon})\, \mathcal{R}_{I_{\ll 2\tau, -2\tau \gg}}\, (\Diamond_{[0,2\tau)} [\psi]^{AMF}_{\tau,\epsilon}),
\end{array}$$

where $I_{\ll a,b \gg}$ is the relaxation of the bounds of interval I with constants a and b, formally defined as follows. For $a, b \in \mathbb{R}$, let $\mathcal{T}(a,b) := \{[a,b], (a,b], [a,b), (a,b)\}$; then for any interval $I \in \mathcal{T}(a,b)$, $I_{\ll c,d \gg} := (a+c, b+d)$.

Note that the interval relaxation $I_{\ll a,b \gg}$ differs from $I_{<a,b>}$ in that the former always yields an open interval, while the latter yields an interval of the same kind as I. For instance $[4,7]_{\ll -1,1 \gg} = (3,8)$, whereas $[4,7]_{<-1,1>} = [3,8]$.

It follows from Definition 21 that the relaxation operator $[]^{\mathrm{AMF}}_{\tau,\epsilon}$ applied to until or release formulae annotated with any interval from $\mathcal{T}(a,b)$ produces the same formulae:

▶ **Observation 22.** *For any $I \in \mathcal{T}(a,b)$, we have:*

$$\begin{array}{lcl}
[\phi\, \mathcal{U}_I\, \psi]^{AMF}_{\tau,\epsilon} & = & (\Diamond_{(-2\tau,0]} [\phi]^{AMF}_{\tau,\epsilon})\, \mathcal{U}_{(a-2\tau, b+2\tau)}\, (\Diamond_{[0,2\tau)} [\psi]^{AMF}_{\tau,\epsilon}) \\
[\phi\, \mathcal{R}_I\, \psi]^{AMF}_{\tau,\epsilon} & = & (\Diamond_{(-2\tau,0]} [\phi]^{AMF}_{\tau,\epsilon})\, \mathcal{R}_{(a+2\tau, b-2\tau)}\, (\Diamond_{[0,2\tau)} [\psi]^{AMF}_{\tau,\epsilon})
\end{array}$$

The following preservation result can be found in [1].

▶ **Theorem 23.** *Let $\phi \in \mathsf{MTL}^+$. Let $\mu_1 : \mathcal{T}_1 \to \mathcal{Y}$ and $\mu_2 : \mathcal{T}_2 \to \mathcal{Y}$ be two discrete GTTs, i.e. $\mathcal{T}_1, \mathcal{T}_2 \subseteq \mathcal{P}_{FIN}(\mathbb{R}_{\geq 0})$. If $\mu_1 \sim_{\tau,\epsilon} \mu_2$, then for any $t_1 \in \mathcal{T}_1$ if $(\mu_1, t_1) \models \phi$, then for all $t_2 \in \mathcal{T}_2$ such that $|t_2 - t_1| \leq \tau$ and $|\mu_2(t_2) - \mu_1(t_1)| \leq \epsilon$, we have $\mu_1, t_1 \models \phi \implies \mu_2, t_2 \models [\phi]^{AMF}_{\tau,\epsilon}$.*

Observe that the above preservation property is very strong: it holds for *any* sampling point in the conforming trace that matches the given point within the (τ, ϵ)-"window". This kind of result comes at a price of having to employ a relaxation operator which yields considerably weaker formulae, which explains the significant relaxation of intervals in $[]^{\mathrm{AMF}}_{\tau,\epsilon}$.

5.2 Laxness of AMF-Relaxation

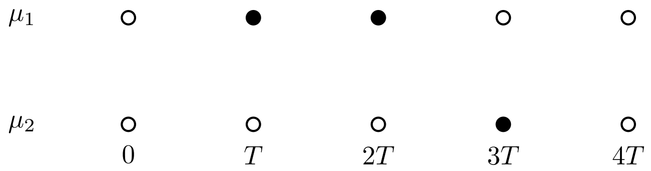
In this section, we prove that the notion of AMF-relaxation is too lax for the purpose of logical characterisation of hybrid conformance, i.e. there is a class of non-conforming implementations which preserve AMF-relaxations of all MTL properties satisfied by their specifications.

Throughout this section, we assume a simple setting where values range over Booleans, i.e. $\mathcal{Y} = \mathbb{B} = \{\mathbf{true}, \mathbf{false}\}$. The associated metric on $\mathcal{P}(\mathbb{B})$ is defined as $d(b_1, b_2) = 0$ if $b_1 = b_2$, and ∞ otherwise.

Recall that we refer to generalised timed traces with a finite time domain as timed state sequences, or TSSs.

We first explain the gist of our proof by showing one instance of the above-mentioned family of non-conforming counter-examples.

▶ **Example 24.** Fix $\tau > 0$ and let T be a value very slightly smaller than τ, i.e. $T = \tau - \delta$, where $\delta \ll \tau$. Consider the discretised GTTs presented in Example 2, which we recall here for the sake of convenience; μ_1 holds value **true** only at T and $2T$ and μ_2 holds value **true** at $3T$ and **false**, otherwise. The two TSSs can be depicted as follows (white/black dots represent states that have value, respectively, **true** / **false**):



μ_1 and μ_2 are not $(\tau, 0)$-close, not even $(t, 0)$-close for any $t < 2T$. To observe this note that for instance $\mu_1(T)$ cannot be matched by μ_2 within $(-T, 3T)$ since no state in μ_2 has value **false** in this interval. On the other hand, as we show next, TSSs μ_2 satisfies the AMF-relaxation of all MTL formulae satisfied by μ_1 (relaxed by parameters $(\tau, 0)$ and vice versa. Intuitively, this is because the intervals in the until and release formulae are respectively expanded and compressed by 2τ, allowing for shifts by 2τ in the states of TSS without affecting the satisfaction of formulae.

In the remainder of this section, we generalise this example and prove this fact for a broader, infinite class of pairs of TSSs which are not $(t, 0)$-equivalent for any $t < 2\tau$.

▶ **Definition 25.** *For a pair of TSSs $\mu_A : \mathcal{T}_A \to \mathbb{B}$ and $\mu_B : \mathcal{T}_B \to \mathbb{B}$, we say that μ_B is* stretched to the right of μ_A by less than t, *if there is some $K \in \mathbb{N}$ and functions $\textsc{chunk}_A : \mathcal{T}_A \to \{1, \ldots, K\}$ and $\textsc{chunk}_B : \mathcal{T}_B \to \{1, \ldots, K\}$ such that the following hold:*

- *$\textsc{chunk}_A$ and $\textsc{chunk}_B$ are surjective and non-decreasing*
- *all states that map to the same chunk number have the same value, i.e. for all $k \in \{1, \ldots, K\}$ and for all $t_A \in \mathcal{T}_A$, $t_B \in \mathcal{T}_A$ such that $\textsc{chunk}_A(t_A) = \textsc{chunk}_B(t_B) = k$, we have $\mu_A(t_A) = \mu_B(t_B)$*
- *for any $t_A \in \mathcal{T}_A$, there is some $t_B \in \mathcal{T}_B$ such that*

  $$(*) \quad 0 \le t_B - t_A < t \quad \wedge \quad \textsc{chunk}_A(t_A) = \textsc{chunk}_B(t_B)$$

  and conversely, for any $t_B \in \mathcal{T}_B$ there is some $t_A \in \mathcal{T}_A$ such that (*) holds. We shall call a pair $(\mu_A, t_A), (\mu_B, t_B)$ satisfying (*) a pair of ***t-corresponding states****.*

Note that in the last condition, the inequality in (*) involves the actual difference between t_B and t_A, not its absolute value – we allow μ_B to be shifted only to the right as compared to μ_A. The following example illustrates this definition.

▶ **Example 26.** Consider the TSSs in Example 24; the TSS μ_2 is stretched to the right of μ_1 by less than 2τ, as witnessed by the following functions $\textsc{chunk}_1$ and $\textsc{chunk}_2$:

$\textsc{chunk}_1(0) = 1$	$\textsc{chunk}_2(t) = 1$ for $t \in \{0, T, 2T\}$
$\textsc{chunk}_1(t) = 2$ for $t \in \{T, 2T\}$	$\textsc{chunk}_2(3T) = 2$
$\textsc{chunk}_1(t) = 3$ for $t \in \{3T, 4T\}$	$\textsc{chunk}_2(4T) = 3$

▶ **Example 27.** Considering Example 24 and propositions $p_{\mathbf{t}}$ and $p_{\mathbf{f}}$ such that $\mathcal{O}(p_{\mathbf{t}}) = \{\mathbf{true}\}$ and $\mathcal{O}(p_{\mathbf{f}}) = \{\mathbf{false}\}$; we have $(\mu_2, 0) \models p_{\mathbf{t}}\, \mathcal{U}_{[3T,3T]}\, p_{\mathbf{f}}$, and the 2τ-corresponding state $(\mu_1, 0)$ satisfies the relaxed formula $[p_{\mathbf{t}}\, \mathcal{U}_{[3T,3T]}\, p_{\mathbf{f}}]^{\mathrm{AMF}}_{\tau,0}$. The latter statement can be deduced from that $(\mu_1, 0)$ satisfies $p_{\mathbf{t}}\, \mathcal{U}_{(3T-2\tau,3T+2\tau)}\, p_{\mathbf{f}}$, a simpler formula that logically entails $[p_{\mathbf{t}}\, \mathcal{U}_{[3T,3T]}\, p_{\mathbf{f}}]^{\mathrm{AMF}}_{\tau,0}$.

The key proposition below states that for 2τ-corresponding states, the satisfaction of all formulae in MTL^+ is preserved modulo relaxation $[]^{\text{AMF}}_{\tau,0}$.

▶ **Proposition 28.** *Suppose μ_B is stretched to the right of μ_A by less than 2τ. Then for any $t_A \in \mathcal{T}_A$, and any $t_B \in \mathcal{T}_B$ satisfying*

$$(*) \quad 0 \leq t_B - t_A < 2\tau \quad \wedge \quad \mathit{CHUNK}_A(t_A) = \mathit{CHUNK}_B(t_B)$$

we have, for all formulae $\phi \in \mathsf{MTL}^+$: $(\mu_A, t_A) \models \phi \implies (\mu_B, t_B) \models [\phi]^{AMF}_{\tau,0}$, and $(\mu_B, t_B) \models \phi \implies (\mu_A, t_A) \models [\phi]^{AMF}_{\tau,0}$.

Proof. The proof by structural induction on ϕ is rather tedious and technical, and omitted in this version of the paper. ◀

6 Conclusions and Future Work

In this paper, we have studied the notion of hybrid conformance from the literature, as well its associated preorder, called hybrid refinement. We have presented a logical characterisation of both relations in Metric Temporal Logic. Since the notions of refinement and conformance allow for some deviations (in time and value), the characterisation is expressed in terms of a relaxation of the set of formulae satisfied by a system. The relaxation operators corresponding to the two relations differ considerably – while for hybrid refinement it suffices to perform relaxation on the level of propositions only, characterising hybrid conformance requires relaxing bounds of intervals in temporal operators. We note that with hybrid conformance we obtain stronger characterisation result; it holds in particular under both existential and universal interpretation of the satisfaction relation.

We have also showed that the existing relaxation scheme proposed by Abbas, Fainekos, and Mittelmann is too lax to serve for a characterisation, i.e., there is a class of non-conforming systems that do satisfy all relaxations of the specification properties. Hence, we proposed a tighter notion of relaxation and showed that it is the appropriate notion to provide a characterisation of hybrid conformance.

Our preservation and characterisation results for hybrid refinement are formulated using the existential interpretation of the satisfaction relation, while our results for hybrid conformance hold both for the existential- and universal interpretation of the satisfaction relation. This is inherent to our notion of hybrid refinement and cannot be remedied in any straightforward manner, as far as we could investigate. We envisage that there could be other definitions of hybrid refinement that are well-behaved in this respect and we would like to study and propose such notions in the future.

As another line of future research, we would also like to investigate the possibility of characterising Skorokhod conformance with Freeze Temporal Logic and the notion of relaxation provided by Deshmukh, Majumdar, and Prabhu [16]. Coming up with the notion of characteristic formulae is another avenue for our future research, which leads to a new technique for checking hybrid conformance.

References

1 Houssam Abbas, Hans D. Mittelmann, and Georgios E. Fainekos. Formal property verification in a conformance testing framework. *Proceedings of MEMOCODE 2014*, pages 155–164, 2014. doi:10.1109/MEMCOD.2014.6961854.

2 Houssam Y. Abbas. *Test-Based Falsification and Conformance Testing for Cyber-Physical Systems*. PhD thesis, Arizona State University, 2015. URL: http://hdl.handle.net/2286/R.A.150686.

3 Samson Abramsky. Observation equivalence as a testing equivalence. *Theor. Comput. Sci.*, 53:225–241, 1987.

4 Luca Aceto, Anna Ingolfsdottir, Kim Guldstrand Larsen, and Jiri Srba. *Reactive Systems: Modelling, Specification and Verification*. Cambridge University Press, 2007. doi:10.1017/CBO9780511814105.

5 Rajeev Alur, Tomás Feder, and Thomas A. Henzinger. The benefits of relaxing punctuality. *J. ACM*, 43(1):116–146, 1996. doi:10.1145/227595.227602.

6 Rajeev Alur and Thomas A. Henzinger. Real-time logics: Complexity and expressiveness. *Information and Computation*, 104(1):35–77, 1993. doi:10.1006/inco.1993.1025.

7 Bard Bloom, Wan Fokkink, and Rob J. van Glabbeek. Precongruence formats for decorated trace preorders. In *15th Annual IEEE Symposium on Logic in Computer Science, Santa Barbara, California, USA, June 26-29, 2000*, pages 107–118. IEEE Computer Society, 2000. doi:10.1109/LICS.2000.855760.

8 Manfred Broy, Bengt Jonsson, Joost-Pieter Katoen, Martin Leucker, and Alexander Pretschner. *Model-Based Testing of Reactive Systems*, volume 3472 of *Lecture Notes in Computer Science*. Springer, 2005.

9 Valentina Castiglioni, Konstantinos Chatzikokolakis, and Catuscia Palamidessi. A logical characterization of differential privacy. *Sci. Comput. Program.*, 188:102388, 2020. doi:10.1016/j.scico.2019.102388.

10 Rance Cleaveland and Steve Sims. The NCSU concurrency workbench. In *Proceedings of the 8th International Conference on Computer Aided Verification (CAV '96)*, volume 1102 of *Lecture Notes in Computer Science*, pages 394–397. Springer, 1996. doi:10.1007/3-540-61474-5_87.

11 Thao Dang. Model-based testing of hybrid systems. *Monograph in Model-Based Testing for Embedded Systems, CRC Press*, 2010.

12 Luca de Alfaro, Marco Faella, and Mariëlle Stoelinga. Linear and branching system metrics. *IEEE Trans. Software Eng.*, 35(2):258–273, 2009.

13 Rocco De Nicola and Matthew Hennessy. Testing equivalences for processes. *Theor. Comput. Sci.*, 34:83–133, 1984. doi:10.1016/0304-3975(84)90113-0.

14 Josee Desharnais, Vineet Gupta, Radha Jagadeesan, and Prakash Panangaden. Metrics for labelled markov processes. *Theor. Comput. Sci.*, 318(3):323–354, 2004. doi:10.1016/j.tcs.2003.09.013.

15 Jyotirmoy V. Deshmukh, Alexandre Donzé, Shromona Ghosh, Xiaoqing Jin, Garvit Juniwal, and Sanjit A. Seshia. Robust online monitoring of signal temporal logic. *Formal Methods in System Design*, 51(1):5–30, 2017. doi:10.1007/s10703-017-0286-7.

16 Jyotirmoy V. Deshmukh, Rupak Majumdar, and Vinayak S. Prabhu. Quantifying conformance using the Skorokhod metric. *Formal Methods in System Design*, 50(2-3):168–206, 2017. doi:10.1007/s10703-016-0261-8.

17 Daniel Gburek and Christel Baier. Bisimulations, logics, and trace distributions for stochastic systems with rewards. In *Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control (HSCC 2018)*, pages 31–40. ACM, 2018.

18 Shromona Ghosh, Dorsa Sadigh, Pierluigi Nuzzo, Vasumathi Raman, Alexandre Donzé, Alberto L. Sangiovanni-Vincentelli, S. Shankar Sastry, and Sanjit A. Seshia. Diagnosis and repair for synthesis from signal temporal logic specifications. In *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control, HSCC 2016, Vienna, Austria, April 12-14, 2016*, pages 31–40. ACM, 2016. doi:10.1145/2883817.2883847.

19 Antoine Girard, A. Agung Julius, and George J. Pappas. Approximate simulation relations for hybrid systems. *Discrete Event Dynamic Systems*, 18(2):163–179, 2008. doi:10.1007/s10626-007-0029-9.

20 Antoine Girard and George J. Pappas. Approximation metrics for discrete and continuous systems. *IEEE Trans. Automat. Contr.*, 52(5):782–798, 2007. doi:10.1109/TAC.2007.895849.

21 Mathew Hennessy and Robin Milner. Algebraic laws for nondeterminism and concurrency. *J. ACM*, 32(1):137–161, 1985. doi:10.1145/2455.2460.

22 Narges Khakpour and Mohammad Reza Mousavi. Notions of conformance testing for cyber-physical systems: Overview and roadmap (invited paper). In *Proc. of the 26th International Conference on Concurrency Theory, CONCUR 2015*, volume 42 of *LIPIcs*, pages 18–40. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

23 Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990. doi:10.1007/BF01995674.

24 Oded Maler and Dejan Nickovic. Monitoring temporal properties of continuous signals. In *Proceedings of the Joint International Conferences on Formal Modelling and Analysis of Timed Systems and Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant (FORMATS and FTRTFT) 2004*, volume 3253 of *Lecture Notes in Computer Science*, pages 152–166. Springer, 2004. doi:10.1007/978-3-540-30206-3_12.

25 Amir Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FOCS 1977)*, pages 46–57. IEEE Computer Society, 1977. doi:10.1109/SFCS.1977.32.

26 Pavithra Prabhakar, Vladimeros Vladimerou, Mahesh Viswanathan, and Geir E. Dullerud. Verifying tolerant systems using polynomial approximations. In Theodore P. Baker, editor, *Proceedings of the 30th IEEE Real-Time Systems Symposium, RTSS 2009, Washington, DC, USA, 1-4 December 2009*, pages 181–190. IEEE Computer Society, 2009. doi:10.1109/RTSS.2009.28.

27 Sriram Sankaranarayanan, Suhas Akshar Kumar, Faye Cameron, B. Wayne Bequette, Georgios E. Fainekos, and David M. Maahs. Model-based falsification of an artificial pancreas control system. *SIGBED Review*, 14(2):24–33, 2017. doi:10.1145/3076125.3076128.

28 Jan Tretmans. Model based testing with labelled transition systems. In *Formal Methods and Testing, An Outcome of the FORTEST Network, Revised Selected Papers*, volume 4949 of *Lecture Notes in Computer Science*, pages 1–38. Springer, 2008.

29 Cumhur Erkan Tuncali, Bardh Hoxha, Guohui Ding, Georgios E. Fainekos, and Sriram Sankaranarayanan. Experience report: Application of falsification methods on the UxAS system. In *Proceedings of the 10th International NASA Formal Methods Symposium (NFM 2018)*, volume 10811 of *Lecture Notes in Computer Science*, pages 452–459. Springer, 2018. doi:10.1007/978-3-319-77935-5_30.

30 Rob J. van Glabbeek. The linear time-branching time spectrum (extended abstract). In Jos C. M. Baeten and Jan Willem Klop, editors, *CONCUR '90, Theories of Concurrency: Unification and Extension, Amsterdam, The Netherlands, August 27-30, 1990, Proceedings*, volume 458 of *Lecture Notes in Computer Science*, pages 278–297. Springer, 1990. doi:10.1007/BFb0039066.

31 Mihalis Yannakakis and David Lee. Testing of finite state systems. In *Proceedings of Computer Science Logic (CSL 1999)*, volume 1584 of *Lecture Notes in Computer Science*, pages 29–44. Springer Berlin Heidelberg, 1999.

Hrushovski's Encoding and ω-Categorical CSP Monsters

Pierre Gillibert
Institut für Diskrete Mathematik und Geometrie, Technische Universität Wien, Austria

Julius Jonušas
Institut für Diskrete Mathematik und Geometrie, Technische Universität Wien, Austria

Michael Kompatscher
Department of Algebra, Faculty of Mathematics and Physics, Charles University,
Prague, Czech Republic

Antoine Mottet
Department of Algebra, Faculty of Mathematics and Physics, Charles University,
Prague, Czech Republic
http://www.karlin.mff.cuni.cz/~mottet/

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

Abstract

We produce a class of ω-categorical structures with finite signature by applying a model-theoretic construction – a refinement of an encoding due to Hrushosvki – to ω-categorical structures in a possibly infinite signature. We show that the encoded structures retain desirable algebraic properties of the original structures, but that the constraint satisfaction problems (CSPs) associated with these structures can be badly behaved in terms of computational complexity. This method allows us to systematically generate ω-categorical templates whose CSPs are complete for a variety of complexity classes of arbitrarily high complexity, and ω-categorical templates that show that membership in any given complexity class cannot be expressed by a set of identities on the polymorphisms. It moreover enables us to prove that recent results about the relevance of topology on polymorphism clones of ω-categorical structures also apply for CSP templates, i.e., structures in a finite language. Finally, we obtain a concrete algebraic criterion which could constitute a description of the delineation between tractability and NP-hardness in the dichotomy conjecture for first-order reducts of finitely bounded homogeneous structures.

2012 ACM Subject Classification Mathematics of computing → Combinatoric problems

Keywords and phrases Constraint satisfaction problem, complexity, polymorphism, pointwise convergence topology, height 1 identity, ω-categoricity, orbit growth

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.131

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

Related Version A long version of this abstract is available on arXiv [22], https://arxiv.org/abs/2002.07054.

Funding *Pierre Gillibert*: received funding from the Austrian Science Fund (FWF) through projects No P27600 and P32337.
Julius Jonušas: received funding from the Austrian Science Fund (FWF) through Lise Meitner grant No M 2555.
Michael Kompatscher: supported by the grants PRIMUS/SCI/12 and UNCE/SCI/022 of Charles University Research Centre programs, as well as grant No 18-20123S of the Czech Science Foundation.
Antoine Mottet: received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant Agreement No 771005, CoCoSym).
Michael Pinsker: received funding from the Austrian Science Fund (FWF) through projects No P27600 and P32337 and from the Czech Science Foundation (grant No 18-20123S).

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 131; pp. 131:1–131:17
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

1.1 Constraint Satisfaction Problems

The *Constraint Satisfaction Problem*, or CSP for short, over a relational structure $\mathbb{A}$ is the computational problem of deciding whether a given finite relational structure $\mathbb{B}$ in the signature of $\mathbb{A}$ can be homomorphically mapped into $\mathbb{A}$. The structure $\mathbb{A}$ is known as the *template* or *constraint language* of the CSP, and the CSP of the particular structure $\mathbb{A}$ is denoted by $\mathrm{CSP}(\mathbb{A})$. A host of interesting computational problems can be modelled using CSPs by choosing an appropriate template. For example, if $\mathbb{A}$ is the structure with domain $\{0,1\}$ and all binary relations on the set $\{0,1\}$, then $\mathrm{CSP}(\mathbb{A})$ is precisely the 2-SAT problem, and if $\mathbb{A}$ is the complete graph on three vertices, then $\mathrm{CSP}(\mathbb{A})$ is the 3-colouring problem. Note that the template $\mathbb{A}$ which defines the problem can also be infinite – only the input structure $\mathbb{B}$ is required to be finite in order to obtain a computational problem. Many well-known computational problems can be modelled, and can in fact only be modelled, using an infinite template. One example is the CSP of the order of the rational numbers $(\mathbb{Q};<)$, which is equivalent to the problem of deciding whether a given finite directed graph is acyclic. The size of the signature of the template $\mathbb{A}$, or in other words the number of its relations, is however generally required to be finite: otherwise, the encoding of its relational symbols might influence the computational complexity of $\mathrm{CSP}(\mathbb{A})$, so that this complexity is not well-defined as per the structure $\mathbb{A}$ itself. To emphasize the importance of this requirement, we shall henceforth only call relational structures in a finite signature *CSP templates*.

The general aim in the study of CSPs is to understand the structural reasons for the hardness or the tractability of such problems. This has been successfully achieved for CSPs of structures over a finite domain. As it turns out, every finite template either has, in a certain precise sense, as little symmetry as the 3-colouring problem above, in which case its CSP is NP-complete; or it has more symmetry and its CSP is polynomial-time solvable, just like the 2-SAT problem. This dichotomy result was conjectured by Feder and Vardi [19, 20], and proved, almost 25 years later, independently by Bulatov [18] and Zhuk [28].

1.2 A dichotomy conjecture and local identities

The algebraic approach behind these proofs does not require the template to be finite, but also works under the assumption of ω-categoricity. And although every computational decision problem is polynomial-time Turing-equivalent to the CSP of some infinite template [9], for a large and natural class of ω-categorical templates, which considerably expands the class of finite templates, a similar conjecture as for finite-domain CSPs has been formulated.

▶ **Conjecture 1** (see [3, 5, 17]). *Let $\mathbb{A}$ be a CSP template which is a first-order reduct of a countable finitely bounded homogeneous structure. Then one of the following holds.*

- *$\mathbb{A}$ satisfies some non-trivial set of h1 identities locally, i.e., on every finite subset of its domain, and $\mathrm{CSP}(\mathbb{A})$ is in P.*
- *There exists a finite subset of its domain on which $\mathbb{A}$ satisfies no non-trivial set of h1 identities, and $\mathrm{CSP}(\mathbb{A})$ is NP-complete.*

The conjectured P/NP-complete dichotomy has been demonstrated for numerous subclasses: for example for all CSPs in the class MMSNP [11], as well as for the CSPs of the first-order reducts of $(\mathbb{Q};<)$ [10], of any countable homogeneous graph [12] (including the random graph [15]), and of the random poset [24].

It is thus the *local h1 identities* which are believed to be the "right" measure of symmetry of a template $\mathbb{A}$ – according to the conjecture, they determine tractability or hardness of its CSP. The distinction between local and global h1 identities is, of course, void in the case of a finite template $\mathbb{A}$, and the above-stated dichotomy is true by the theorem of Bulatov and Zhuk and results from [5]. One of the main challenges towards proving Conjecture 1 is to determine whether this distinction could be void as well for structures within its range. The satisfaction of non-trivial h1 identities in $\mathbb{A}$ is characterised by the non-existence of particular maps called *minion homomorphisms* from the *polymorphism clone* $\mathrm{Pol}(\mathbb{A})$ of $\mathbb{A}$ to the *projection clone* $\mathscr{P}$. The mentioned distinction between local and global h1 identities is then mirrored by the distinction between those maps that are *uniformly continuous* and those that are not. Hence, the question whether non-trivial local h1 identities imply non-trivial global h1 identities in a relational structure $\mathbb{A}$ raises the following question.

▶ **Question 2.** *Does the existence of a minion homomorphism from* $\mathrm{Pol}(\mathbb{A})$ *to* $\mathscr{P}$ *imply the existence of a uniformly continuous minion homomorphism from* $\mathrm{Pol}(\mathbb{A})$ *to* $\mathscr{P}$*?*

2 Results

Among all finite structures, it is known that CSP templates (i.e., those structures with finite signature) have considerably better algebraic properties than other structures [2, 1]. We refine a model-theoretic trick due to Hrushovski [23] to encode ω-categorical structures with an infinite signature into ω-categorical CSP templates whilst preserving certain properties of the original, showing that a similar phenomenon does not seem to appear within the class of ω-categorical structures. Using this method, we produce ω-categorical CSP templates with various “untame” properties of both algebraic and complexity-theoretic nature.

2.1 Local versus global identities

Recently, in [13, 14], an example of an ω-categorical structure answering Question 2 in the negative was given; however, this structure had an infinite language and therefore this result had *a priori* no consequence for the study of CSPs. Using our encoding, we provide a negative answer within the realm of CSP templates.

▶ **Theorem 3.** *There is an ω-categorical CSP template $\mathbb{U}$ with slow orbit growth such that there exists a minion homomorphism from* $\mathrm{Pol}(\mathbb{U})$ *to* $\mathscr{P}$*, but no uniformly continuous one.*

We also encode a counterexample from [17] for *clone homomorphisms*, which are mappings preserving arbitrary (not only h1) identities, into a finite language. Clone homomorphisms appear in the original (and equivalent [3, 4]) formulation of Conjecture 1 from [17] (also see [6, 7]).

▶ **Theorem 4.** *There exists an ω-categorical CSP template $\mathbb{U}$ with a clone homomorphism from* $\mathrm{Pol}(\mathbb{U})$ *to* $\mathscr{P}$ *that is not uniformly continuous.*

2.2 Dissected weak near-unanimity identities

Our proof of the fact that the template $\mathbb{U}$ from Theorem 3 satisfies non-trivial h1 identities locally is constructive: we exhibit a concrete set of such identities which we call *dissected weak near-unanimity*. Moreover, we obtain quite general conditions on the symmetry of a structure which force our identities to be satisfied locally. It follows that the original infinite-language structure from [13, 14] satisfies them; this contrasts the indirect proof in [13, 14] which does not provide any concrete set of h1 identities.

▶ **Theorem 5.** *Let $\mathbb{U}$ be a homogeneous structure. Let F be a finite subset of its domain, and let $k \geq 1$. Suppose that:*

(i) *Only relations of arity smaller than k hold for tuples of elements in F.*

(ii) *There is an embedding from $\mathbb{U}^2$ into $\mathbb{U}$.*

Then $\mathbb{U}$ satisfies (n,k) dissected weak near-unanimity identities on F for all $n > k$.

Dissected weak near-unanimity identities can be viewed as a generalization of *weak near-unanimity* identities. It follows from [26] and [5] that if $\mathbb{U}$ is a finite relational structure satisfying non-trivial h1 identities, then $\mathbb{U}$ satisfies weak near-unanimity identities. The satisfaction of dissected weak near-unanimity identities has been proven for a large number of structures within the range of Conjecture 1 in [4, 3]; since they now reappeared in the rather different context of Theorem 3, the following question emerges naturally.

▶ **Question 6.** *Let $\mathbb{U}$ be an ω-categorical structure with slow orbit growth which satisfies non-trivial h1 identities locally. Does $\mathbb{U}$ satisfy dissected weak near-unanimity identities locally?*

2.3 ω-categorical CSP monsters

The complexity of CSP($\mathbb{A}$) is, for every ω-categorical CSP template $\mathbb{A}$, determined by Pol($\mathbb{A}$) viewed as a topological clone: if there exists a topological clone isomorphism Pol($\mathbb{A}$) $\to$ Pol($\mathbb{B}$) and $\mathbb{A}$ and $\mathbb{B}$ are ω-categorical, then CSP($\mathbb{A}$) and CSP($\mathbb{B}$) are equivalent under log-space reductions [16]. In other words, the local (not necessarily h1) identities satisfied in Pol($\mathbb{A}$) encode the complexity of CSP($\mathbb{A}$). Conjecture 1 even postulates that for every template $\mathbb{A}$ within its scope, membership of CSP($\mathbb{A}$) in P only depends on the local h1 identities of $\mathbb{A}$. The latter is equivalent to the statement that polynomial-time tractability is characterised by the global satisfaction of the single identity $\alpha s(x,y,x,z,y,z) = \beta s(y,x,z,x,z,y)$ [4, 7].

Using our encoding, we prove that global identities do not characterise membership in P – or, in fact, in any other non-trivial class of languages containing AC^0 – for the class of homogeneous CSP templates.

▶ **Theorem 7.** *Let $\mathcal{C}$ be any class of languages that contains AC^0 and that does not intersect every Turing degree. Then there is no countable set Θ of identities such that for all homogeneous CSP templates membership in $\mathcal{C}$ is equivalent to the satisfaction of Θ.*

The proof of Theorem 7 relies on encoding arbitrary languages as CSPs of homogeneous templates. These templates are obtained by applying our encoding to structures which have only empty relations, but a complicated infinite signature. On the way, we obtain a new proof of a result by Bodirsky and Grohe [9].

▶ **Theorem 8.** *Let $\mathcal{C}$ be a complexity class such that there exist* CONP$^{\mathcal{C}}$*-complete problems. Then there exists a homogeneous CSP template that satisfies non-trivial h1 identities and whose CSP is* CONP$^{\mathcal{C}}$*-complete. Moreover, if P $\neq$* CONP*, then there exists a CSP template with these algebraic properties whose CSP has* CONP*-intermediate complexity.*

In particular, Theorem 8 gives complete problems for classes such as Π_n^{P} for every $n \geq 1$, PSPACE, EXPTIME, or even every fast-growing time complexity class $\mathbf{F}_\alpha$ where $\alpha \geq 2$ is an ordinal (such as the classes TOWER or ACKERMANN, see [27]).

3 Outline

The most important definitions, in particular those of notions which appear in the introduction and the results, are given in Section 4. A complete exposition of all notions, as well as most proofs are left to the appendix due to space restrictions; we also refer to the long version of this extended abstract which is available on arXiv [22]. Our variant of Hrushovski's encoding and its most important properties are presented in Section 5. Dissected weak near-unanimity identities, as well as the proof of Theorem 3, are the contents of Section 6. In Section 7, we sketch the proofs of Theorem 7 and of the first statement of Theorem 8. Due to space restrictions, we cannot address the proofs of Theorems 4 and of the second statement of Theorem 8 at all – they can be found in the long version.

4 Preliminaries

If $\mathbb{A}$ is a relational structure in a finite signature, called a *CSP template*, then $\mathrm{CSP}(\mathbb{A})$ is the set of all finite structures $\mathbb{B}$ in the same signature with the property that there exists a homomorphism from $\mathbb{B}$ into $\mathbb{A}$. This set can be viewed as a computational problem where we are given a finite structure $\mathbb{B}$ in that signature, and we have to decide whether $\mathbb{B} \in \mathrm{CSP}(\mathbb{A})$.

We will tacitly assume that all relational structures, as well as their signatures, are at most countably infinite.

4.1 Polymorphisms, identities, and clone and minion homomorphisms

A *polymorphism* of a relational structure $\mathbb{A}$ is a homomorphism from some finite power $\mathbb{A}^n$ of the structure into $\mathbb{A}$. The set of all polymorphisms of $\mathbb{A}$ is called the *polymorphism clone* of $\mathbb{A}$ and is denoted by $\mathrm{Pol}(\mathbb{A})$.

An *identity* is a formal expression $s(x_1, \dots, x_n) = t(y_1, \dots, y_m)$ where s and t are abstract terms of function symbols, and $x_1, \dots, x_n, y_1, \dots, y_m$ are the variables that appear in these terms. The identity is of *height 1*, and called *h1 identity*, if the terms s and t contain precisely one function symbol, i.e., no nesting of function symbols is allowed, and no term may be just a variable.

A set Θ of identities is *satisfied* in $\mathbb{A}$ if the function symbols of Θ can be assigned functions in $\mathrm{Pol}(\mathbb{A})$ in such a way that all identities of Θ become true for all possible values of their variables in $\mathbb{A}$. If F is a finite subset of the domain of $\mathbb{A}$, then Θ *is satisfied locally on* F if the above situation holds where only values within F are considered for the variables.

A set of identities is called *trivial* if it is satisfied in the *projection clone* $\mathscr{P}$ consisting of the projection operations on the set $\{0, 1\}$. Otherwise, the set is called *non-trivial*. We say that $\mathbb{A}$ satisfies non-trivial identities *locally* if on every finite subset of its domain it locally satisfies some non-trivial set of identities. We shall use similar terminology for h1 identities.

A map $\xi\colon \mathrm{Pol}(\mathbb{A}) \to \mathrm{Pol}(\mathbb{B})$ is called a *clone homomorphism* if it preserves arities, maps the i-th n-ary projection in $\mathrm{Pol}(\mathbb{A})$ to the i-th n-ary projection in $\mathrm{Pol}(\mathbb{B})$ for all $1 \leq i \leq n$, and satisfies $\xi(f \circ (g_1, \dots, g_n)) = \xi(f) \circ (\xi(g_1), \dots, \xi(g_n))$ for all $n, m \geq 1$, all n-ary $f \in \mathrm{Pol}(\mathbb{A})$, and all m-ary $g_1, \dots, g_n \in \mathrm{Pol}(\mathbb{A})$. This is the case if and only if the map ξ *preserves identities*, i.e., whenever some functions in $\mathrm{Pol}(\mathbb{A})$ witness the satisfaction of some identity in $\mathrm{Pol}(\mathbb{A})$, then their images under ξ witness the satisfaction of the same identity in $\mathrm{Pol}(\mathbb{B})$.

A map $\xi\colon \mathrm{Pol}(\mathbb{A}) \to \mathrm{Pol}(\mathbb{B})$ is called a *minion homomorphism* if it preserves arities and composition with projections; the latter meaning that for all $n, m \geq 1$, all n-ary $f \in \mathrm{Pol}(\mathbb{A})$, and all m-ary projections $p_1, \dots, p_n \in \mathrm{Pol}(\mathbb{A})$, we have $\xi(f \circ (p_1, \dots, p_n)) = \xi(f) \circ (p'_1, \dots, p'_n)$, where p'_i is the m-ary projection in $\mathrm{Pol}(\mathbb{B})$ onto the same variable as p_i, for all $1 \leq i \leq n$. This is the case if and only if the map ξ preserves h1 identities in the sense above.

The existence of clone and minion homomorphisms $\mathrm{Pol}(\mathbb{A}) \to \mathscr{P}$ is connected to the satisfaction of non-trivial identities in a relational structure $\mathbb{A}$. Namely, there exists a clone homomorphism $\mathrm{Pol}(\mathbb{A}) \to \mathscr{P}$ if and only if every set of identities satisfied in $\mathbb{A}$ is trivial; and there exists a minion homomorphism $\mathrm{Pol}(\mathbb{A}) \to \mathscr{P}$ if and only if every set of h1 identities satisfied in $\mathbb{A}$ is trivial.

Similarly, the *local satisfaction* of identities and h1 identities can be characterised via *uniformly continuous* clone and minion homomorphisms, respectively [16, 21, 5]. However, the reader will not need any knowledge of topology and can only keep in mind that the topology reflects the local/global distinction.

4.2 Homogeneity, boundedness, reducts, ω-categoricity, orbit growth

Let $\mathcal{C}$ be a class of finite structures in a common relational signature which is closed under isomorphisms. We define the following properties the class $\mathcal{C}$ might have.

Hereditary property (HP): if $\mathbb{A} \in \mathcal{C}$ and if $\mathbb{B}$ is a substructure of $\mathbb{A}$, then $\mathbb{B} \in \mathcal{C}$;

Amalgamation property (AP): if $\mathbb{A}, \mathbb{B}, \mathbb{C} \in \mathcal{C}$ and if $f_1 \colon \mathbb{A} \to \mathbb{B}$ and $f_2 \colon \mathbb{A} \to \mathbb{C}$ are embeddings, then there exist $\mathbb{D} \in \mathcal{C}$ and embeddings $g_1 \colon \mathbb{B} \to \mathbb{D}$ and $g_2 \colon \mathbb{C} \to \mathbb{D}$ such that $g_1 \circ f_1 = g_2 \circ f_2$;

Strong amalgamation property (SAP): $\mathcal{C}$ satisfies AP and in addition g_1 and g_2 can be chosen to have disjoint ranges, except for the common values enforced by above equation.

A relational structure $\mathbb{C}$ is *homogeneous* if every isomorphism between finite induced substructures extends to an automorphism of the entire structure $\mathbb{C}$. In that case, $\mathbb{C}$ is uniquely determined, up to isomorphism, by its *age*, i.e., the class of its finite induced substructures up to isomorphism. This is a consequence of the following theorem.

▶ **Theorem 9** (Fraïssé's Theorem, see [23]). *Let σ be a relational signature and let $\mathcal{C}$ be a class of finite σ-structures which is closed under isomorphisms and satisfies HP and AP. Then there exists a σ-structure $\mathbb{A}$ such that $\mathbb{A}$ is countable, homogeneous, and the age of $\mathbb{A}$ equals $\mathcal{C}$. Furthermore $\mathbb{A}$ is unique up to isomorphism.*

The structure $\mathbb{A}$ above is called the *Fraïssé limit* of $\mathcal{C}$, and the class $\mathcal{C}$ a *Fraïssé class.*

A class $\mathcal{C}$ of finite structures in the same finite signature is *finitely bounded* if it is given by a finite set $\mathcal{F}$ of forbidden finite substructures, i.e., $\mathcal{C}$ consists precisely of those finite structures in its signature which do not embed any member of $\mathcal{F}$. A class $\mathcal{C}$ of finite structures in the same signature is *homomorphically bounded* by a (possibly infinite) set $\mathcal{F}$ of finite structures if it is defined by forbidding the structures in $\mathcal{F}$ homomorphically, i.e., $\mathcal{C}$ consists precisely of those finite structures in its signature which do not contain a homomorphic image of any member of $\mathcal{F}$ as a substructure. A structure $\mathbb{A}$ is finitely bounded (homomorphically bounded) if its age is.

A *first-order reduct* of a relational structure $\mathbb{C}$ is a relational structure $\mathbb{A}$ on the same domain which is first-order definable without parameters in $\mathbb{C}$. Every first-order reduct $\mathbb{A}$ of a finitely bounded homogeneous structure is *ω-categorical*, i.e., the automorphism group $\mathrm{Aut}(\mathbb{A})$ has finitely many orbits in its componentwise action on A^n, for all finite $n \geq 1$. In fact, if $\mathbb{A}$ is such a first-order reduct, then the number of orbits in the action of $\mathrm{Aut}(\mathbb{A})$ on A^n grows exponentially in n; in general, we say that ω-categorical structures where this number grows less than double exponentially in n have *slow orbit growth.*

For a relational structure $\mathbb{A}$ in signature $\sigma = (R_i)_{i \in I}$, and $J \subseteq I$, we call the structure $(A; (R_i^{\mathbb{A}})_{i \in J})$ in signature $\rho := (R_i)_{i \in J}$ the *ρ-reduct* of $\mathbb{A}$; conversely $\mathbb{A}$ is called an *expansion* of any of its reducts, and a *first-order expansion* of a reduct if all of its relations have a first-order definition in the reduct. We say that a structure is *homogenizable* if it has a homogeneous first-order expansion. All ω-categorical structures are homogenizable. A homogenizable structure $\mathbb{A}$ has *no algebraicity* if the age of any, or equivalently some, homogeneous first-order expansion of $\mathbb{A}$ has SAP.

A formula is *primitive positive*, in short *pp*, if it contains only existential quantifiers, conjunctions, equalities, and relational symbols. If $\mathbb{A}$ is a relational structure, then a relation is *pp-definable* in $\mathbb{A}$ if it can be defined by a pp-formula in $\mathbb{A}$. A structure $\mathbb{A}$ *pp-interprets* $\mathbb{B}$ if a structure isomorphic to $\mathbb{B}$ can be constructed from $\mathbb{A}$ by pp-defining a subset S of some finite power of its domain, then pp-defining an equivalence relation $\sim$ on S, and then pp-defining relations on the equivalence classes of $\sim$.

5 The encoding

We present the encoding of an arbitrary homogenizable structure with no algebraicity into a CSP template, which will be the basis of our results. The construction is originally due to Hrushovski [23, Section 7.4]; it was designed to capture properties of the first-order theory and consequently the automorphism group of the original structure. We refine his construction in order to also compare the polymorphism clones of the original structure and its encoded counterpart, and to control the complexity of the CSPs of the produced templates.

5.1 Encoding and Decoding

Let Σ be a finite alphabet, and let $\Sigma^{\geq 2}$ denote the set of all finite words over Σ of length at least two. We are going to encode structures with a signature of the form $\rho = (R_w)_{w \in W}$, where $W \subseteq \Sigma^{\geq 2}$ and where the arity of each symbol R_w equals the length $|w|$ of the word w. For the rest of this section we fix Σ and ρ. Our goal is to encode any homogenizable ρ-structure $\mathbb{A}$ with no algebraicity into a structure $\mathbf{E}\,\mathbb{A}$ (where $\mathbf{E}$ stands for *E. Hrushovski*) in a finite signature θ which is disjoint from ρ and only depends on Σ.

Note that by renaming its signature, and possibly artificially inflating the arity of its relations (by adding dummy variables), any arbitrary structure with countably many relations can be given a signature of the above form without changing, for example, its polymorphism clone. However, the encoding will depend on these modifications, and their effect on the algebraic and combinatorial properties of the encoding is beyond the scope of this article. The original encoding [23, Section 7.4] roughly corresponds to the case where $|\Sigma| = 1$, and our generalization allows us to avoid such modifications for the structures we wish to encode, making in particular our complexity-theoretic results possible.

▶ **Definition 10.** *Let θ denote the signature $\{P, \iota, \tau, S\} \cup \{H_s \mid s \in \Sigma\}$, where P, ι, τ are unary relation symbols, H_s is a binary relation symbol for each $s \in \Sigma$, and S is a 4-ary relation symbol. For every signature σ disjoint from θ, define σ^+ to be the union $\sigma \cup \theta$.*

The encoding of a ρ-structure $\mathbb{A}$ will roughly be obtained as follows: first, one takes a homogeneous first-order expansion $\mathbb{B}$ in some signature σ; from its age K, one defines a class K^+ of finite structures in signature σ^+; and the encoding is the θ-reduct of the Fraïssé limit of K^+. In order to define the class K^+, we need the following definitions.

▶ **Definition 11.** *Let σ be a signature disjoint from θ, let $\mathbb{A}$ be a σ^+-structure, and let $w \in \Sigma^{\geq 2}$. A tuple $(a_1, \dots, a_{|w|}, c_1, \dots, c_{|w|})$ of elements of $\mathbb{A}$ is a* valid w-code *in $\mathbb{A}$ if the following hold:*

(a) $a_1, \dots, a_{|w|} \in P^{\mathbb{A}}$;

(b) $H^{\mathbb{A}}_{w_i}(c_i, c_j)$ *for all* $1 \leq i, j \leq |w|$ *such that* $j \equiv i + 1 \pmod{|w|}$;

(c) $\iota^{\mathbb{A}}(c_1)$ *and* $\tau^{\mathbb{A}}(c_{|w|})$;

(d) $S^{\mathbb{A}}(a_i, a_j, c_i, c_j)$ *for all* $1 \leq i, j \leq |w|$ *with* $i \neq j$.

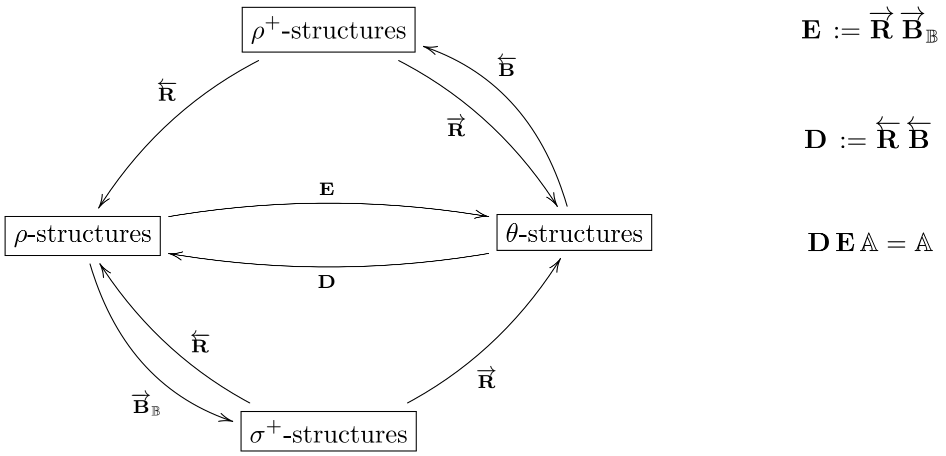

ρ^+-structures
$\overleftarrow{\mathbf{R}}$
$\overleftarrow{\mathbf{B}}$
$\overrightarrow{\mathbf{R}}$
ρ-structures
$\mathbf{E}$
θ-structures
$\mathbf{D}$
$\overleftarrow{\mathbf{R}}$
$\overrightarrow{\mathbf{R}}$
$\overrightarrow{\mathbf{B}}_{\mathbb{B}}$
σ^+-structures
$\mathbf{E} := \overrightarrow{\mathbf{R}}\,\overrightarrow{\mathbf{B}}_{\mathbb{B}}$
$\mathbf{D} := \overleftarrow{\mathbf{R}}\,\overleftarrow{\mathbf{B}}$
$\mathbf{D}\,\mathbf{E}\,\mathbb{A} = \mathbb{A}$

Figure 1 Sources and destinations of the operators related to the encoding and decoding.

▶ **Definition 12.** *Let σ be a signature disjoint from θ, and let $\mathbb{A}$ be a σ^+-structure. Then $\mathbb{A}$ is called* separated *if*

(i) *$H_s^{\mathbb{A}}$ only relates pairs within $A \setminus P^{\mathbb{A}}$ for all $s \in \Sigma$;*
(ii) *$\iota^{\mathbb{A}}, \tau^{\mathbb{A}}$ are contained in $A \setminus P^{\mathbb{A}}$;*
(iii) *If $(a, b, c, d) \in S^{\mathbb{A}}$, then $c, d \in A \setminus P^{\mathbb{A}}$ and $c \neq d$.*

It follows from (iii) above that in a separated structure a valid w-code can only exist if $|w| \geq 2$; this is the reason for the exclusion of unary relation symbols from ρ.

▶ **Definition 13.** *Let $\mathbb{A}$ be a ρ-structure and let $\mathbb{B}$ be a homogeneous first-order expansion of $\mathbb{A}$ with signature σ and age K. Define K^+ to be the class of all finite σ^+-structures $\mathbb{C}$ with the following properties:*

(1) *The σ-reduct of the restriction of $\mathbb{C}$ to $P^{\mathbb{C}}$ is an element of K;*
(2) *$\mathbb{C}$ is separated and for every $R \in \sigma$ the relation $R^{\mathbb{C}}$ only relates tuples which lie entirely within $P^{\mathbb{C}}$;*
(3) *If $R_w \in \rho$ and $(a_1, \dots, a_{|w|}, c_1, \dots, c_{|w|})$ is a valid w-code in $\mathbb{C}$, then $(a_1, \dots, a_{|w|}) \in R_w^{\mathbb{C}}$.*

▶ **Lemma 14.** *Let $\mathbb{A}$ be a ρ-structure and let $\mathbb{B}$ be a homogeneous first-order expansion of $\mathbb{A}$ with age K. If K has the HP and the SAP, then K^+ has the HP and the SAP as well.*

By Lemma 14, if $\mathbb{A}$ has no algebraicity[1], and $\mathbb{B}$ is a homogeneous first-order expansion of $\mathbb{A}$ with age K, then K^+ has a Fraïssé limit, allowing us to define our encoding as follows.

▶ **Definition 15.** *Let $\mathbb{A}$ be a ρ-structure with no algebraicity and let $\mathbb{B}$ be a homogeneous first-order expansion of $\mathbb{A}$ with age K. We define $\overrightarrow{\mathbf{B}}_{\mathbb{B}}\,\mathbb{A}$, the* encoding blow up *of $\mathbb{A}$, to be the Fraïssé limit of K^+. Moreover, we define $\overrightarrow{\mathbf{R}}\,\mathbb{C}$ to be the θ-reduct of any structure $\mathbb{C}$ with signature containing θ. The* Hrushovski-encoding *$\mathbf{E}\,\mathbb{A}$ is defined by $\mathbf{E}\,\mathbb{A} := \overrightarrow{\mathbf{R}}\,\overrightarrow{\mathbf{B}}_{\mathbb{B}}\,\mathbb{A}$.*

▶ Remark 16. All ω-categorical structures have a homogeneous first-order expansion. This expansion is not unique, but the encoding $\mathbf{E}\,\mathbb{A}$ of $\mathbb{A}$ does not depend on it.

[1] Contrary to a claim in [23], AP of K is not a sufficient assumption for AP of K^+ in Lemma 14.

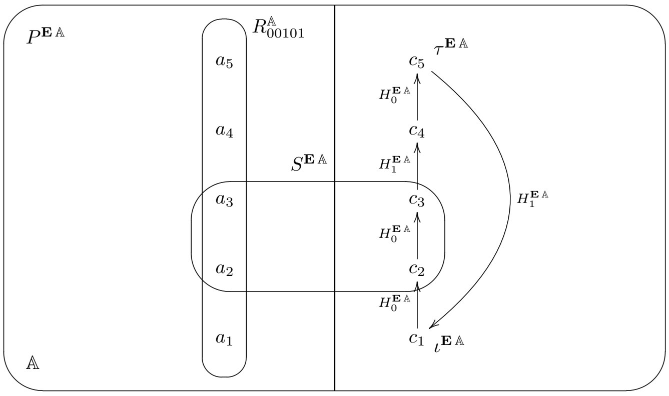

$P^{\mathbf{E}\,\mathbb{A}}$
$R^{\mathbb{A}}_{00101}$
$\tau^{\mathbf{E}\,\mathbb{A}}$
a_5
c_5
$H_0^{\mathbf{E}\,\mathbb{A}}$
a_4
c_4
$S^{\mathbf{E}\,\mathbb{A}}$
$H_1^{\mathbf{E}\,\mathbb{A}}$
a_3
c_3
$H_1^{\mathbf{E}\,\mathbb{A}}$
$H_0^{\mathbf{E}\,\mathbb{A}}$
a_2
c_2
$H_0^{\mathbf{E}\,\mathbb{A}}$
a_1
c_1
$\iota^{\mathbf{E}\,\mathbb{A}}$
$\mathbb{A}$

Figure 2 The encoding $\mathbf{E}\,\mathbb{A}$ of a structure $\mathbb{A}$.

It might be of help to the reader if we note that the operators used in the encoding of a structure, i.e., $\overrightarrow{\mathbf{B}}_{\mathbb{B}}$ and $\overrightarrow{\mathbf{R}}$, bear arrows from left to right; the operators used in the decoding of a structure, defined below, bear arrows in the opposite direction. Table 1 contains an informal summary of all operators, and Figure 1 describes on which classes of structures they operate.

Table 1 The meaning of the operators.

Operator	Name	Description
$\overrightarrow{\mathbf{B}}_{\mathbb{B}}$	encoding blow up	The first step in an encoding, extends the domain and defines relations for the signature θ via a homogeneous expansion $\mathbb{B}$ of the input.
$\overrightarrow{\mathbf{R}}$	θ-reduct	Returns the θ-reduct of a structure.
$\mathbf{E}$	encoding	Combines $\overrightarrow{\mathbf{B}}_{\mathbb{B}}$ and $\overrightarrow{\mathbf{R}}$ to obtain a θ-structure from a ρ-structure.
$\overleftarrow{\mathbf{B}}$	decoding blow up	The first step in decoding a θ-structure, it converts valid codes into corresponding relations in ρ.
$\overleftarrow{\mathbf{R}}$	relativised reduct	Restricts a structure to the set named by P and forgets relations not in ρ.
$\mathbf{D}$	decoding	Combines $\overleftarrow{\mathbf{R}}$ and $\overleftarrow{\mathbf{B}}$ to obtain the ρ-structure $\mathbb{A}$ from the encoded θ-structure $\mathbf{E}\,\mathbb{A}$.
$\mathbf{C}$	canonical code	Defines in a canonical way a finite θ-structure from a finite ρ-structure in which every relation which holds in the input is witnessed by a valid code.

Like the encoding of a structure, the decoding of a structure is a composition of two steps; first a *decoding blow up*, and then a *relativised reduct.*

▶ **Definition 17.** *Let $\mathbb{C}$ be a θ-structure. Then the* decoding blow up $\overleftarrow{\mathbf{B}}\,\mathbb{C}$ *of $\mathbb{C}$ is the expansion of $\mathbb{C}$ in signature ρ^+, where for any symbol $R_w \in \rho$ the relation $R_w^{\overleftarrow{\mathbf{B}}\,\mathbb{C}}$ is defined to consist of those tuples $(a_1, \ldots, a_{|w|})$ for which there exist $c_1, \ldots, c_{|w|}$ such that the tuple $(a_1, \ldots, a_{|w|}, c_1, \ldots, c_{|w|})$ is a valid w-code in $\mathbb{C}$.*

For a structure $\mathbb{D}$ in a signature containing ρ^+, the relativised reduct $\overleftarrow{\mathbf{R}}\,\mathbb{D}$ *of $\mathbb{D}$ is defined to be the ρ-reduct of $\mathbb{D}$ restricted to $P^{\mathbb{D}}$.*

Finally, we set $\mathbf{D}\,\mathbb{C} := \overleftarrow{\mathbf{R}}\,\overleftarrow{\mathbf{B}}\,\mathbb{C}$, the decoding of $\mathbb{C}$, *for any θ-structure $\mathbb{C}$.*

The following proposition states that the operator $\mathbf{D}$ indeed decodes $\mathbf{E}\,\mathbb{A}$. It also allows us to identify $\mathbb{A}$ and $\mathbf{D}\,\mathbf{E}\,\mathbb{A}$, an assumption we shall thenceforth make.

▶ **Proposition 18.** *Let $\mathbb{A}$ be a homogenizable ρ-structure with no algebraicity. Then $\mathbb{A}$ and $\mathbf{D}\,\mathbf{E}\,\mathbb{A}$ are isomorphic. Moreover, for any θ-structure $\mathbb{D}$, the structure $\mathbf{D}\,\mathbb{D}$ has a pp-interpretation in $\mathbb{D}$.*

5.2 The relationship between $\mathbb{A}$ and $\mathbf{E}\,\mathbb{A}$

We derive the following main properties of the encoding $\mathbf{E}\,\mathbb{A}$ of a ρ-structure $\mathbb{A}$:

- $\mathbf{E}\,\mathbb{A}$ is ω-categorical if and only if $\mathbb{A}$ is, and has slow orbit growth if and only if $\mathbb{A}$ does (Proposition 19);
- There exists a uniformly continuous clone homomorphism ξ from $\mathrm{Pol}(\mathbf{E}\,\mathbb{A})$ into $\mathrm{Pol}(\mathbb{A})$; conversely, if $\mathbb{A}$ is ω-categorical, then injective polymorphisms of $\mathbb{A}$ essentially extend to polymorphisms of $\mathbf{E}\,\mathbb{A}$ (Proposition 20).

We start by investigating the relationship of the orbits of $\mathrm{Aut}(\mathbb{A})$ with those of $\mathrm{Aut}(\mathbf{E}\,\mathbb{A})$, showing that ω-categoricity and slow orbit growth are preserved by the encoding.

▶ **Proposition 19.** *Let $\mathbb{A}$ be a homogenizable ρ-structure with no algebraicity.*

1. *$\mathbb{A}$ is ω-categorical if and only if $\mathbf{E}\,\mathbb{A}$ is.*
2. *Let $\mathbb{A}$ be ω-categorical. For $n \geq 1$, write $f(n)$ and $g(n)$ for the number of orbits of n-tuples under the action of $\mathrm{Aut}(\mathbb{A})$ and $\mathrm{Aut}(\mathbf{E}\,\mathbb{A})$, respectively. Then $f(n) \leq g(n) \leq 2^{6|\Sigma|n^4} f(n)$ for all $n \geq 1$. In particular, $\mathbb{A}$ has slow orbit growth if and only if $\mathbf{E}\,\mathbb{A}$ does.*

We now turn to the polymorphism clones of $\mathbb{A}$ and $\mathbf{E}\,\mathbb{A}$. An immediate consequence of Proposition 18 is that polymorphisms of $\mathbf{E}\,\mathbb{A}$ can be restricted to polymorphisms of $\mathbb{A}$. Conversely, one can prove that assuming ω-categoricity of $\mathbb{A}$, for every injective $f \in \mathrm{Pol}(\mathbb{A})$ there exists an embedding u of $\mathbb{A}$ such that uf can be extended to a polymorphism of $\mathbf{E}\,\mathbb{A}$.

▶ **Proposition 20.** *Let $\mathbb{A}$ be a structure with no algebraicity, and let $\mathbb{B}$ be a homogeneous first-order expansion of $\mathbb{A}$. Then the following hold:*

(1) *For every $f \in \mathrm{Pol}(\mathbf{E}\,\mathbb{A})$, the restriction $f|_{P^{\mathbf{E}\,\mathbb{A}}}$ of f to $P^{\mathbf{E}\,\mathbb{A}}$ is a polymorphism of $\mathbb{A}$. The map $f \mapsto f|_{P^{\mathbf{E}\,\mathbb{A}}}$ is a uniformly continuous clone homomorphism $\mathrm{Pol}(\mathbf{E}\,\mathbb{A}) \to \mathrm{Pol}(\mathbb{A})$.*

(2) *If $\mathbb{A}$ is ω-categorical, then for every injective $f \in \mathrm{Pol}(\mathbb{A})$ there exists an embedding $u\colon \overrightarrow{\mathbf{B}}_{\mathbb{B}}\,\mathbb{A} \to \overrightarrow{\mathbf{B}}_{\mathbb{B}}\,\mathbb{A}$ such that uf extends to a polymorphism of $\mathbf{E}\,\mathbb{A}$.*

(3) *If $\mathbb{A}$ is ω-categorical, then for all $k \geq 1$, $\mathbb{B}^k$ embeds into $\mathbb{B}$ if and only if $(\overrightarrow{\mathbf{B}}_{\mathbb{B}}\,\mathbb{A})^k$ embeds into $\overrightarrow{\mathbf{B}}_{\mathbb{B}}\,\mathbb{A}$.*

Propositions 19 and 20 are the fundamental results upon which the following sections rely. They allow us to relate $\mathrm{Pol}(\mathbf{E}\,\mathbb{A})$ and $\mathrm{Pol}(\mathbb{A})$ and to transfer the exotic behaviour of the latter (for a well-chosen $\mathbb{A}$ with infinite signature) into the former.

5.3 Homomorphisms and the encoding

We now examine the relationship between the finite structures that homomorphically map into a structure $\mathbb{A}$ with those that homomorphically map into its encoding $\mathbf{E}\,\mathbb{A}$ (which is precisely $\mathrm{CSP}(\mathbf{E}\,\mathbb{A})$). This will be particularly relevant in Section 7 where we investigate the complexity of CSPs of structures encoded with our encoding.

▶ **Proposition 21.** *There exists a log-space computable function $\mathbb{C} \mapsto \mathbf{C}\,\mathbb{C}$ from the set of finite ρ-structures to the set of finite θ-structures satisfying the following properties:*

- *For every finite ρ-structure $\mathbb{C}$, we have $\mathbf{D}\,\mathbf{C}\,\mathbb{C} = \mathbb{C}$.*
- *If $\mathbb{C}$ is a finite ρ-structure, and $\mathbb{D}$ is a θ-structure, then there exists a homomorphism from $\mathbb{C}$ to $\mathbf{D}\,\mathbb{D}$ if and only if there exists a homomorphism from $\mathbf{C}\,\mathbb{C}$ to $\mathbb{D}$.*

The properties from Proposition 21 are enough to give a concrete description of $\mathrm{CSP}(\mathbf{E}\,\mathbb{A})$ when $\mathbb{A}$ is homomorphically bounded.

▶ **Proposition 22.** *Let $\mathbb{A}$ be a homogenizable ρ-structure with no algebraicity which is homomorphically bounded by a set $\mathcal{G}$ of finite ρ-structures. Let $\mathbb{X}$ be a θ-structure. Then the following are equivalent.*

(1) *There exists an embedding of $\mathbb{X}$ into $\mathbf{E}\,\mathbb{A}$;*
(2) *There exists a homomorphism from $\mathbb{X}$ to $\mathbf{E}\,\mathbb{A}$;*
(3) *$\mathbb{X}$ is separated and for all $\mathbb{G} \in \mathcal{G}$ there exists no homomorphism from $\mathbf{C}\,\mathbb{G}$ to $\mathbb{X}$.*

Note that being separated can be characterised by not containing the homomorphic image of any element of a finite set $\mathcal{S}$ of finite θ-structures. As an immediate consequence of Proposition 22 we therefore obtain the following corollary.

▶ **Corollary 23.** *Let $\mathbb{A}$ be a homogenizable ρ-structure with no algebraicity which is homomorphically bounded by a set $\mathcal{G}$ of finite ρ-structures. Then $\mathbf{E}\,\mathbb{A}$ is homomorphically bounded by $\{\mathbf{C}\,\mathbb{G} \mid \mathbb{G} \in \mathcal{G}\} \cup \mathcal{S}$.*

6 Height 1 identities: local without global

Recall that in [13] a negative answer to Question 2 of the introduction was established by an infinite-language example which is ω-categorical and has slow orbit growth. We are now going to prove that the encoding of that structure, or in fact, of a simplification $\mathbb{S}$ thereof, also provides an example. Since $\mathbf{E}\,\mathbb{S}$ is a CSP template, and since both ω-categoricity and slow orbit growth are preserved by the encoding, $\mathbf{E}\,\mathbb{S}$ is a witness for the truth of Theorem 3. While the non-satisfaction of non-trivial global h1 identities lifts from $\mathbb{S}$ to $\mathbf{E}\,\mathbb{S}$ by virtue of Proposition 20 (1), we do not know in general when this is the case for the local satisfaction of non-trivial h1 identities. Our proof thus relies on specific structural properties of $\mathbb{S}$; we show that both $\mathbb{S}$ and $\mathbf{E}\,\mathbb{S}$ locally satisfy *dissected weak near-unanimity identities.*

6.1 Dissected weak near-unanimity identities

▶ **Definition 24.** *Let $n > k > 1$, let $g_1, \ldots, g_n$ be binary function symbols, and for every injective function $\psi\colon \{1, \ldots, k\} \to \{1, \ldots, n\}$ let f_ψ be a k-ary function symbol. Then the set of (n, k)* dissected weak near-unanimity identities *consists of the identities*

$$f_\psi(x, \ldots, x, \overset{\overset{i}{\downarrow}}{y}, x, \ldots, x) = g_{\psi(i)}(x, y)$$

for all injective functions $\psi\colon \{1, \ldots, k\} \to \{1, \ldots, n\}$ and $i \in \{1, \ldots, k\}$.

Note that any polymorphism clone which satisfies identities of the form

$$f(y, x, \ldots, x) = \cdots = f(x, \ldots, x, y),$$

called *k-ary weak near-unanimity identities* when f is k-ary for some $k \geq 3$, must also satisfy the (n, k) dissected weak near-unanimity identities for all $n > k$. This can be seen by setting $f_\psi = f$ for every ψ. Moreover, there exist polymorphism clones which satisfy dissected weak near-unanimity identities, but do not satisfy any weak near-unanimity identities: one example is the polymorphism clone consisting of all injective functions (up to dummy variables) on a countable set, see [4]. Hence, we can regard dissected weak near-unanimity identities as a strict weakening of the weak near-unanimity identities.

Further note that, for all parameters $m \geq n > k > 1$, the (n,k) dissected weak near-unanimity identities form a subset of the (m,k) dissected weak near-unanimity identities. Thus for every fixed $k > 1$ the family of (n,k) dissected weak near-unanimity identities form an infinite chain of h1 identities of increasing strength. In the special case $k = 2$, the satisfaction of any of the $(n,2)$ dissected weak near-unanimity identities is equivalent to the existence of a binary commutative polymorphism (as they imply $g_1(x,y) = g_2(y,x) = g_3(x,y) = g_1(y,x)$).

▶ **Lemma 25.** *For all $n > k > 1$ the (n,k) dissected weak near-unanimity identities are non-trivial.*

Proof. Assume to the contrary that there exist projections $g_1, \ldots, g_n \in \mathscr{P}$ and $f_\psi \in \mathscr{P}$ for every injection $\psi\colon \{1,\ldots,k\} \to \{1,\ldots,n\}$ that satisfy the (n,k) dissected weak near-unanimity identities. First, suppose that there are two distinct $1 \leq i,j \leq k$ such that g_i, g_j are both the projection onto the second coordinate. Then let ψ be an injective function with $\psi(1) = i, \psi(2) = j$. It follows from the identities that $f_\psi(y,x,\ldots,x) = f_\psi(x,y,\ldots,x) = y$ holds for all values of the variables, which contradicts f_ψ being a projection. Therefore at most one operation g_i equals the projection to its second coordinate. Since $n > k$, there is an injective function $\psi\colon \{1,\ldots,k\} \to \{1,\ldots,n\}$ such that $g_{\psi(i)}$ is the first projection for all $i \in \{1,\ldots,k\}$. Then f_ψ satisfies the weak near-unanimity identities, which again contradicts f_ψ being a projection. ◀

▶ **Lemma 26.** *Let $\mathbb{U}$ be a relational structure and let $n \geq 2$. Then there exists an embedding from $\mathbb{U}^2$ into $\mathbb{U}$ if and only if there exists an embedding from $\mathbb{U}^n$ into $\mathbb{U}$.*

Proof. If there is an embedding $f\colon \mathbb{U}^n \to \mathbb{U}$ for some $n \geq 2$, then $g\colon \mathbb{U}^2 \to \mathbb{U}$, defined by $g(x,y) := f(x,y,\ldots,y)$, is also an embedding. On the other hand, if for some $n \geq 2$ there exist embeddings $g\colon \mathbb{U}^2 \to \mathbb{U}$ and $h\colon \mathbb{U}^n \to \mathbb{U}$, then the composition $f(x_1,\ldots,x_{n+1}) := g(h(x_1,\ldots,x_n),x_{n+1})$ is an embedding from $\mathbb{U}^{n+1}$ into $\mathbb{U}$. Hence by induction the existence of an embedding from $\mathbb{U}^2$ into $\mathbb{U}$ implies the existence of an embedding from $\mathbb{U}^n$ into $\mathbb{U}$ for all $n \geq 2$. ◀

Proof of Theorem 5. For all $l \geq 2$, define $X_l \subseteq F^l$ by

$$X_l := \bigcup_{a,b \in F} \{(a,\ldots,a,b),(a,\ldots,a,b,a),\ldots,(b,a,\ldots,a)\},$$

and let $\mathbb{X}_l$ be the substructure which X_l induces in $\mathbb{U}^l$.

The first step of our proof is to show that if $n \geq k$, then there exists an embedding $h\colon \mathbb{X}_k \to \mathbb{X}_n$ such that $\mathbf{x}$ is an initial segment of $h(\mathbf{x})$ for all $\mathbf{x} \in X_k$. Let us first assume that $k \geq 3$. For every tuple $\mathbf{x} \in \mathbb{X}_k$ we denote the unique element of F which occurs more than once among its entries by $s(\mathbf{x})$. Define $h\colon X_k \to X_n$ to be the map that extends the tuple $\mathbf{x}$ by $n-k$ many entries with value $s(\mathbf{x})$. In order to prove that h is an embedding let $\mathbf{x}_1,\ldots,\mathbf{x}_m \in X_k$ be such that $R^{\mathbb{U}^k}(\mathbf{x}_1,\ldots,\mathbf{x}_m)$ holds for some m-ary relation symbol R in the signature of $\mathbb{U}$. By assumption (i) we have $m < k$. Thus there exists $1 \leq j \leq k$ such that the projection of each $\mathbf{x}_i$ to its j-th coordinate equals $s(\mathbf{x}_i)$. Therefore $(s(\mathbf{x}_1),\ldots,s(\mathbf{x}_m)) \in R^{\mathbb{U}}$, and hence h is a homomorphism. Also its inverse – the projection of n-tuples to the first k-coordinates – is a homomorphism, and thus h is an embedding. Now assume the remaining case where $k = 2$. Define a map $h\colon X_2 \to X_n$ by $(x_1, x_2) \mapsto (x_1, x_2, \dots, x_2)$. To check that that h is an embedding, by assumption (i), we only need to check that h is an embedding with respect to unary relations, which however follows from its definition.

Observe that h was defined in such a way that, for each index $1 \le i \le k$, the i-th projection of $h(\mathbf{x})$ is equal to x_i. By permuting the coordinates of its image in a suitable manner, we can obtain embeddings $h_\psi\colon \mathbb{X}_k \to \mathbb{X}_n$ for every injection $\psi\colon \{1, \dots, k\} \to \{1, \dots, n\}$ such that the $\psi(i)$-th projection of $h_\psi(\mathbf{x})$ is equal to x_i for all $1 \le i \le k$.

In order to construct the operations f_ψ on F, let $f\colon \mathbb{U}^k \to \mathbb{U}$ and $g\colon \mathbb{U}^n \to \mathbb{U}$ be embeddings, which exist by (ii) and Lemma 26. For every injection $\psi : \{1, \dots, k\} \to \{1, \dots, n\}$ define the map $u_\psi\colon f(\mathbb{X}_k) \to g(\mathbb{X}_n)$ by

$$u_\psi(f(a, \dots, a, \underset{i^{\text{th}}}{b}, a, \dots, a)) := g(a, \dots, a, \underset{\psi(i)^{\text{th}}}{b}, a, \dots, a). \tag{1}$$

Then u_ψ is equal to $g \circ h_\psi \circ f^{-1}$. Since h_ψ is an embedding, $u_\psi\colon f(\mathbb{X}_k) \to u_\psi(f(\mathbb{X}_k))$ is an isomorphism between finite substructures of $\mathbb{U}$. By the homogeneity of $\mathbb{U}$, it can be extended to an automorphism v_ψ of $\mathbb{U}$. Set $f_\psi := v_\psi \circ f$ and, for all $1 \le i \le n$, define $g_i(x, y) := g(x, \dots, x, y, x, \dots, x)$, where the only y appears at the i-th coordinate of g. It then follows from (1) that these polymorphisms satisfy the (n, k) dissected weak near-unanimity identities on F, concluding the proof. ◀

6.2 Revisiting the infinite-language counterexample

We now investigate the simplification $\mathbb{S}$ of the infinite-language structure from [13]; $\mathbb{S}$ satisfies the same local and global h1 identities as the structure in [13]. Namely, we consider the superposition as in [13, Construction 6.4], but directly of the CSS structures in the proof of [13, Lemma 6.3] rather than of their model-complete cores; we are able to do this due to our constructive, rather than indirect, proof of the local satisfaction of h1 identities. The structure $\mathbb{S}$ has the following properties.

▶ **Proposition 27** (Consequence of the results from [13]). *There exist ω-categorical structures $\mathbb{S}$ and $\mathbb{H}$ with slow orbit growth and without algebraicity, as well as a strictly increasing function $\alpha\colon \mathbb{N} \to \mathbb{N}$ such that the following hold:*

(1) *$\mathbb{H}$ is a homogeneous expansion of $\mathbb{S}$ by pp-definable relations.*

(2) *Every relation of $\mathbb{H}$ has arity $k \cdot \alpha(n)$ for some $k, n \ge 1$, and for every $n \ge 1$ there exist only finitely many relations of arity of the form $k \cdot \alpha(n)$. Moreover, if $(a_1, \dots, a_{k\cdot\alpha(n)}) \in R$ for some relation R of $\mathbb{H}$, then $\{a_1, \dots, a_{k\cdot\alpha(n)}\}$ has size at least $\alpha(n)$.*

(3) *$\mathbb{H}$ is homomorphically bounded.*

(4) *There exists a minion homomorphism from $\mathrm{Pol}(\mathbb{S})$ to $\mathscr{P}$.*

Proof of Theorem 3. Let $\mathbb{S}, \mathbb{H}$ be as in Proposition 27. We can assume that $\mathbb{S}$ has signature ρ as in Section 5 since this change does not affect the properties claimed in Proposition 27; see the remark at the beginning of Section 5.1. Since $\mathbb{S}$ has no algebraicity, it has an ω-categorical finite-language encoding $\mathbb{U} = \mathbf{E}\,\mathbb{S}$. Moreover, since $\mathbb{S}$ has slow orbit growth, so does $\mathbb{U}$ by Proposition 19. There is a minion homomorphism $\mathrm{Pol}(\mathbb{U}) \to \mathrm{Pol}(\mathbb{S})$ by Proposition 20 (1) and a minion homomorphism $\mathrm{Pol}(\mathbb{S}) \to \mathscr{P}$ by Proposition 27, thus we obtain a minion homomorphism $\mathrm{Pol}(\mathbb{U}) \to \mathscr{P}$ by composition.

It remains to prove that there is no uniformly continuous minion homomorphism from $\mathrm{Pol}(\mathbb{U})$ to $\mathscr{P}$. We do so by showing that for every finite subset F of the domain, there exists $k > 1$ such that $\mathrm{Pol}(\mathbb{U})$ satisfies the (n, k) dissected weak near-unanimity identities on F for all

$n > k$. Note that $\mathbb{U} = \mathbf{E}\,\mathbb{S}$ is a reduct of the blowup $\overrightarrow{\mathbf{B}}_{\mathbb{H}}\,\mathbb{S}$, and hence $\mathrm{Pol}(\overrightarrow{\mathbf{B}}_{\mathbb{H}}\,\mathbb{S}) \subseteq \mathrm{Pol}(\mathbf{E}\,\mathbb{S})$. It is therefore sufficient to prove that there exists some $k > 1$ for which $\overrightarrow{\mathbf{B}}_{\mathbb{H}}\,\mathbb{S}$ satisfies the (n,k) dissected weak near-unanimity identities on F for all $n > k$.

In order to prove this statement, we verify that conditions (i) and (ii) of Theorem 5 hold for $\overrightarrow{\mathbf{B}}_{\mathbb{H}}\,\mathbb{S}$, F, and a suitable $k > 1$. Since $\mathbb{H}$ is homomorphically bounded, it is well-known that $\mathbb{H}^2$ embeds into $\mathbb{H}$. By Proposition 20 (3), there exists an embedding of $(\overrightarrow{\mathbf{B}}_{\mathbb{H}}\,\mathbb{S})^2$ into $\overrightarrow{\mathbf{B}}_{\mathbb{H}}\,\mathbb{S}$, and thus condition (ii) holds.

It remains to check (i) which states that there exists an upper bound on the arity of tuples in F that satisfy some relation from $\overrightarrow{\mathbf{B}}_{\mathbb{H}}\,\mathbb{S}$. Denote the signature of $\mathbb{H}$ by σ. Suppose that $R^{\overrightarrow{\mathbf{B}}_{\mathbb{H}}\,\mathbb{S}}$ contains a tuple entirely within F for some $R \in \sigma^+$, the language of $\overrightarrow{\mathbf{B}}_{\mathbb{H}}\,\mathbb{S}$. Since $\sigma^+ = \sigma \cup \theta$, and all relations in θ have arity at most 4, we may assume that $R \in \sigma$. Then any tuple in $R^{\overrightarrow{\mathbf{B}}_{\mathbb{H}}\,\mathbb{S}}$ must lie entirely within $P^{\overrightarrow{\mathbf{B}}_{\mathbb{H}}\,\mathbb{S}}$, and essentially the proof of Proposition 18 shows that this implies that the tuple is an element of $R^{\mathbb{H}}$.

By Proposition 27 (2), R has arity $k \cdot \alpha(n)$ for some $n, k \geq 1$ and at least $\alpha(n)$ many of the values of any tuple in $R^{\mathbb{H}}$ are distinct. Therefore, $\alpha(n)$ must be smaller than $|F|$. Since α is a strictly increasing function, it follows that only finitely many relations of $R^{\overrightarrow{\mathbf{B}}_{\mathbb{H}}\,\mathbb{S}}$ have tuples that lie entirely in F. Let $k > 1$ be a strict upper bound on the arity of those relations. For this choice of k we have that (i) of Theorem 5 holds, and thus $R^{\overrightarrow{\mathbf{B}}_{\mathbb{H}}\,\mathbb{S}}$ satisfies the (n,k) dissected weak near-unanimity identities on F for all $n > k$. ◀

▶ Remark 28. It follows that the original structure $\mathbb{S}$ satisfies dissected weak near-unanimity identities locally as well, since by Proposition 20 (1), there is a uniformly continuous minion homomorphism from $\mathrm{Pol}(\mathbf{E}\,\mathbb{S})$ to $\mathrm{Pol}(\mathbb{S})$. This result is new and no other explicit description of non-trivial local h1 identities of $\mathbb{S}$ was given in [13].

7 Identities and CSPs with Homogeneous Templates

7.1 Encoding arbitrary languages as CSPs

Let Σ be a finite alphabet and $W \subseteq \Sigma^{\geq 2}$. Let ρ_W be the signature consisting of one $|w|$-ary relation symbol R_w for every word $w \in W$. The *trivial structure* $\mathbb{T}_W$ is the countable ρ_W-structure whose relations are all empty. For every word $w \in W$, the *w-edge structure* $\mathbb{F}_w$ is the ρ_W-structure on the set $F_w = \{1, \ldots, |w|\}$ whose only non-empty relation is $R_w^{\mathbb{F}_w} = \{(1, \ldots, |w|)\}$.

The trivial structure $\mathbb{T}_W$ is homomorphically bounded by the set of all edge-structures $\mathbb{F}_w$ with $w \in W$. Moreover, $\mathbb{T}_W$ has no algebraicity. It is not hard to see that $\mathbf{E}\,\mathbb{T}_W$ is homogeneous. Applying Theorem 5 and a compactness argument, one can see that $\mathbf{E}\,\mathbb{T}_W$ satisfies non-trivial h1 identities.

Since $\mathbb{T}_W$ is homomorphically bounded, Corollary 23 can be used to give an explicit description of $\mathrm{CSP}(\mathbf{E}\,\mathbb{T}_W)$, which we use to prove the results of this section. We employ the notion of coNP-many-one reduction first defined in [8]: a language K coNP-many-one reduces to L if there is a non-deterministic polynomial-time Turing machine M such that for all words w, we have $w \in K$ if and only if each computational path of M, on input w, produces a word in L. Note that if K has a coNP-many-one reduction to L, then in particular K is in coNP^L.

▶ **Proposition 29.** *Let $L \subseteq \Sigma^{\geq 2}$ and $W := \Sigma^{\geq 2} \setminus L$ both be nonempty. Then L has a log-space many-one reduction to $\mathrm{CSP}(\mathbf{E}\,\mathbb{T}_W)$, and $\mathrm{CSP}(\mathbf{E}\,\mathbb{T}_W)$ coNP-many-one reduces to L.*

Proof. The function $w \mapsto \mathbf{C}\,\mathbb{F}_w$, for every $w \in \Sigma^{\geq 2}$, is computable in logarithmic space with respect to $|w|$ by Proposition 21. Also note that there is a homomorphism $\mathbf{C}\,\mathbb{F}_u \to \mathbf{C}\,\mathbb{F}_w$ if and only if $w = u$. Moreover, it follows from Proposition 22 that there is a homomorphism $\mathbf{C}\,\mathbb{F}_w \to \mathbf{E}\,\mathbb{T}_W$ if and only if $w \in L$. Thus L has a log-space many-one reduction to $\mathrm{CSP}(\mathbb{T}_W)$.

For the other reduction, let $\mathbb{X}$ be a finite θ-structure, an instance of $\mathrm{CSP}(\mathbf{E}\,\mathbb{T}_W)$. If there is no homomorphism $\mathbb{X} \to \mathbf{E}\,\mathbb{T}_W$, by Proposition 22, either $\mathbb{X}$ is not separated (which can be checked in polynomial time), or there is a word $w \in W$ not longer than the size of the domain of $\mathbb{X}$ and a homomorphism $f\colon\, \mathbf{C}\,\mathbb{F}_w \to \mathbb{X}$. The reduction does the following: if $\mathbb{X}$ is not separated, we map it to a fixed element of W. Otherwise, we guess a word w not longer than the size of the domain of $\mathbb{X}$ and a function $f\colon\, \mathbf{C}\,\mathbb{F}_w \to \mathbb{X}$. If this function is not a homomorphism, we map $\mathbb{X}$ to a fixed word of L. If f is a homomorphism, we map $\mathbb{X}$ to w. Thus, if $\mathbb{X} \in \mathrm{CSP}(\mathbf{E}\,\mathbb{T}_W)$ then all runs of the reduction output a word of L. Moreover, if $\mathbb{X} \notin \mathrm{CSP}(\mathbf{E}\,\mathbb{T}_W)$, then at least one run outputs word in W. ◀

We can now prove the first statement of Theorem 8.

Proof of the first statement of Theorem 8. Let $L \subseteq \Sigma^{\geq 2}$ be a $\mathrm{coNP}^{\mathcal{C}}$-complete language, and let W be its complement. Then L reduces to $\mathrm{CSP}(\mathbf{E}\,\mathbb{T}_W)$ by Proposition 29, so $\mathrm{CSP}(\mathbf{E}\,\mathbb{T}_W)$ is $\mathrm{coNP}^{\mathcal{C}}$-hard. Moreover, it follows from Proposition 29 and the fact that $\mathrm{coNP}^{\mathcal{C}}$ is closed under coNP-many-one reductions that $\mathrm{CSP}(\mathbf{E}\,\mathbb{T}_W)$ belongs to $\mathrm{coNP}^{\mathcal{C}}$. ◀

The second statement of Theorem 8 follows from the following proposition whose proof is inspired by Ladner's proof on the existence of NP-intermediate problems [25].

▶ **Proposition 30.** *Let $L \subseteq \{0,1\}^{\geq 2}$ be a language in coNP $\setminus$ P. Then there is a unary language $I \subseteq \{0\}^{\geq 2}$ such that* $\mathrm{CSP}(\mathbf{E}\,\mathbb{T}_I)$ *is also in coNP $\setminus$ P, but L is not polynomial-time reducible to* $\mathrm{CSP}(\mathbf{E}\,\mathbb{T}_I)$.

Finally, we are ready to prove Theorem 7. In the following, let $\mathcal{L}$ be the extension of existential second-order logic allowing countably many second-order quantifiers, followed by a countable conjunction of first-order formulas. It can be seen that the upward direction of Łoś's theorem and the downward Löwenheim-Skolem theorem hold for this logic.

Proof of Theorem 7. We prove the following: there is no countable set Θ of θ-formulas in $\mathcal{L}$ such that the equivalence $\mathbb{A} \models \Theta \Leftrightarrow \mathrm{CSP}(\mathbb{A}) \in \mathcal{C}$ holds for all homogeneous θ-structures $\mathbb{A}$. This proves the theorem, as the satisfaction of a countable set of identities by polymorphisms can be expressed in $\mathcal{L}$.

Assume that such a Θ exists. Let L be a language over Σ whose Turing-degree is not intersected by $\mathcal{C}$, and let $W = \Sigma^{\geq 2} \setminus L$. For every $n \in \mathbb{N}$, let $W \cap \Sigma^{\leq n}$ be the set of words of length at most n in W. Corollary 23 implies that $\mathrm{CSP}(\mathbf{E}\,\mathbb{T}_{W\cap\Sigma^{\leq n}})$ is finitely bounded, hence $\mathrm{CSP}(\mathbf{E}\,\mathbb{T}_{W\cap\Sigma^{\leq n}})$ is in AC^0. Therefore, since $\mathbf{E}\,\mathbb{T}_{W\cap\Sigma^{\leq n}}$ is homogeneous, our assumption implies that $\mathbf{E}\,\mathbb{T}_{W\cap\Sigma^{\leq n}} \models \Theta$. Let $\mathcal{U}$ be a non-principal ultrafilter on $\mathbb{N}$, and let $\mathbb{X}$ be the ultraproduct $\big(\prod_{n\in\mathbb{N}} \mathbf{E}\,\mathbb{T}_{W\cap\Sigma^{\leq n}}\big)/\mathcal{U}$. Then $\mathbb{X} \models \Theta$ by Łoś's theorem and $\mathbb{X}$ is homogeneous, as all the factors in the ultraproduct are homogeneous. By the Löwenheim-Skolem theorem, $\mathbb{X}$ has a countable elementary substructure $\mathbb{Y}$ that also satisfies Θ. Note that $\mathbb{Y}$ is homogeneous and has the same age as $\mathbb{X}$, as it is an elementary substructure of $\mathbb{X}$.

Finally, we claim that $\mathbb{X}$ and $\mathbf{E}\,\mathbb{T}_W$ have the same age. Every finite substructure of $\mathbf{E}\,\mathbb{T}_W$ embeds into $\mathbf{E}\,\mathbb{T}_{W\cap\Sigma^{\leq n}}$ for all $n \in \mathbb{N}$, by Corollary 23, and therefore into their ultraproduct, which is $\mathbb{X}$. Conversely, assume that a finite structure $\mathbb{C}$ embeds into $\mathbb{X}$. This precisely means that $I := \{n \in \mathbb{N} \mid \mathbb{C} \text{ embeds into } \mathbf{E}\,\mathbb{T}_{W\cap\Sigma^{\leq n}}\}$ is in $\mathcal{U}$. Moreover, since $\mathcal{U}$ is not principal, I is infinite. Let $w \in W$. Since I is infinite there is an $n \geq |w|$ such that $\mathbb{C}$ embeds into

$\mathbf{E}\,\mathbb{T}_{W\cap\Sigma^{\leq n}}$. Since $w \in W \cap \Sigma^{\leq n}$, Corollary 23 gives that $\mathbf{C}\,\mathbb{F}_w$ does not homomorphically map to $\mathbb{C}$, and that $\mathbb{C}$ is separated. Since this holds for all $w \in W$, it follows that $\mathbb{C}$ embeds into $\mathbf{E}\,\mathbb{T}_W$.

By Theorem 9, the two structures $\mathbb{Y}$ and $\mathbf{E}\,\mathbb{T}_W$ are isomorphic. By Proposition 29, L and CSP($\mathbf{E}\,\mathbb{T}_W$) have the same Turing-degree, therefore CSP($\mathbf{E}\,\mathbb{T}_W$) is not in $\mathcal{C}$, a contradiction. ◀

References

1 Libor Barto. Finitely related algebras in congruence distributive varieties have near unanimity terms. *Canadian Journal of Mathematics*, 65(1):3–21, 2013.

2 Libor Barto. Finitely related algebras in congruence modular varieties have few subpowers. *Journal of the European Mathematical Society*, 20(6):1439–1471, 2018.

3 Libor Barto, Michael Kompatscher, Miroslav Olšák, Trung Van Pham, and Michael Pinsker. The equivalence of two dichotomy conjectures for infinite domain constraint satisfaction problems. In *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–12, 2017.

4 Libor Barto, Michael Kompatscher, Miroslav Olšák, Trung Van Pham, and Michael Pinsker. Equations in oligomorphic clones and the constraint satisfaction problem for ω-categorical structures. *Journal of Mathematical Logic*, 19(2):#1950010, 2019.

5 Libor Barto, Jakub Opršal, and Michael Pinsker. The wonderland of reflections. *Israel Journal of Mathematics*, 223(1):363–398, 2018.

6 Libor Barto and Michael Pinsker. The algebraic dichotomy conjecture for infinite domain constraint satisfaction problems. In *Proceedings of the 31th Annual IEEE Symposium on Logic in Computer Science – LICS'16*, pages 615–622, 2016.

7 Libor Barto and Michael Pinsker. Topology is irrelevant. *SIAM Journal on Computing*, 49(2):365–393, 2020.

8 Richard Beigel, Richard Chang, and Mitsunori Ogiwara. A relationship between difference hierarchies and relativized polynomial hierarchies. *Mathematical Systems Theory*, 26(3):293–310, 1993.

9 Manuel Bodirsky and Martin Grohe. Non-dichotomies in constraint satisfaction complexity. In *Proceedings of the International Colloquium on Automata, Languages and Programming (ICALP)*, Lecture Notes in Computer Science, pages 184–196. Springer Verlag, 2008.

10 Manuel Bodirsky and Jan Kára. The complexity of temporal constraint satisfaction problems. *Journal of the ACM*, 57(2):1–41, 2009. A conference version appeared in the Proceedings of the Symposium on Theory of Computing (STOC) 2008.

11 Manuel Bodirsky, Florent Madelaine, and Antoine Mottet. A universal-algebraic proof of the complexity dichotomy for monotone monadic snp. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '18, pages 105–114, 2018.

12 Manuel Bodirsky, Barnaby Martin, Michael Pinsker, and András Pongrácz. Constraint satisfaction problems for reducts of homogeneous graphs. *SIAM Journal on Computing*, 48(4):1224–1264, 2019. A conference version appeared in the Proceedings of the 43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, pages 119:1–119:14.

13 Manuel Bodirsky, Antoine Mottet, Miroslav Olšák, Jakub Opršal, Michael Pinsker, and Ross Willard. Topology is relevant (in the infinite-domain dichotomy conjecture for constraint satisfaction problems). In *Proceedings of the Symposium on Logic in Computer Science – LICS'19*, 2019.

14 Manuel Bodirsky, Antoine Mottet, Miroslav Olšák, Jakub Opršal, Michael Pinsker, and Ross Willard. ω-categorical structures avoiding height 1 identities. *Transactions of the American Mathematical Society*, 2020. Accepted for publication.

15 Manuel Bodirsky and Michael Pinsker. Schaefer's theorem for graphs. *Journal of the ACM*, 62(3):52 pages (article number 19), 2015. A conference version appeared in the Proceedings of STOC 2011, pages 655–664.

16 Manuel Bodirsky and Michael Pinsker. Topological Birkhoff. *Transactions of the American Mathematical Society*, 367:2527–2549, 2015.

17 Manuel Bodirsky, Michael Pinsker, and András Pongrácz. Projective clone homomorphisms. *Journal of Symbolic Logic*, 2019. To appear. Preprint `arXiv:1409.4601`. `doi:10.1017/jsl.2019.23`.

18 Andrei A. Bulatov. A dichotomy theorem for nonuniform CSPs. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017*, pages 319–330, 2017.

19 Tomás Feder and Moshe Y. Vardi. Monotone monadic SNP and constraint satisfaction. In *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 612–622, 1993.

20 Tomás Feder and Moshe Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: a study through Datalog and group theory. *SIAM Journal on Computing*, 28:57–104, 1999.

21 Mai Gehrke and Michael Pinsker. Uniform Birkhoff. *Journal of Pure and Applied Algebra*, 222(5):1242–1250, 2018.

22 Pierre Gillibert, Yannis Jonušas, Michael Kompatscher, Antoine Mottet, and Michael Pinsker. When symmetries are not enough: a hierarchy of hard constraint satisfaction problems. Preprint, 2020. `arXiv:2002.07054`.

23 Wilfrid Hodges. *Model theory*. Cambridge University Press, 1993.

24 Michael Kompatscher and Trung Van Pham. A complexity dichotomy for poset constraint satisfaction. *Journal of Applied Logics*, 5(8):1663–1695, 2018.

25 Richard E. Ladner. On the structure of polynomial time reducibility. *Journal of the ACM*, 22(1):155–171, 1975.

26 Miklós Maróti and Ralph McKenzie. Existence theorems for weakly symmetric operations. *Algebra Universalis*, 59(3-4):463–489, 2008.

27 Sylvain Schmitz. Complexity hierarchies beyond elementary. *ACM Transactions on Computation Theory (TOCT)*, 8(1):3:1–3:36, 2016. `doi:10.1145/2858784`.

28 Dmitriy N. Zhuk. A proof of CSP dichotomy conjecture. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017*, pages 331–342, 2017.

Descriptive Complexity on Non-Polish Spaces II

Mathieu Hoyrup
Université de Lorraine, CNRS, Inria, LORIA, Nancy, France
mathieu.hoyrup@inria.fr

Abstract

This article is a study of descriptive complexity of subsets of represented spaces. Two competing measures of descriptive complexity are available. The first one is topological and measures how complex it is to obtain a set from open sets using boolean operations. The second one measures how complex it is to test membership in the set, and we call it symbolic complexity because it measures the complexity of the symbolic representation of the set. While topological and symbolic complexity are equivalent on countably-based spaces, they differ on more general spaces. Our investigation is aimed at explaining this difference and highly suggests that it is related to the well-known mismatch between topological and sequential aspects of topological spaces.

2012 ACM Subject Classification Theory of computation → Turing machines; Mathematics of computing → Point-set topology

Keywords and phrases Represented space, Computable analysis, Descriptive set theory, Scott topology

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.132

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

Related Version A full version of the paper is available at `https://hal.inria.fr/hal-02483114`.

Acknowledgements I want to thank the anonymous referees for their useful comments, and for suggesting the generalization of Corollary 2.4 to quasi-zero-dimensional spaces.

1 Introduction

This article fits in the line of research extending descriptive set theory, mainly developed on Polish spaces, to other classes of topological spaces relevant to theoretical computer science, such as domains [21], quasi-Polish spaces [2], and represented spaces [13, 4, 1]. We pursue our investigation of descriptive set theory on represented spaces, started in [1].

Theoretical computer science, logic and descriptive set theory closely interact, providing different ways of describing properties, by programs, formulas or boolean operation from basic properties, all intimately related. For instance, a property of real numbers that is decidable in the limit must belong to the class $\undertilde{\mathbf{\Delta}}^0_2$, and every $\undertilde{\mathbf{\Delta}}^0_2$-property is decidable in the limit relative to some oracle.

This correspondence works very well on Polish spaces and more generally countably-based topological spaces. However, little is known for other topological spaces whose points can be represented and processed by a program, and it has been shown in [1] that the correspondence fails, even on natural spaces such as the space of polynomials with real coefficients: there is a property which can be decided with 2 mind-changes, but which is not a difference of two open sets, and is in no level below $\undertilde{\mathbf{\Delta}}^0_2$.

We introduce *symbolic* descriptive complexity, which captures the algorithmic complexity of a set, and compare it to topological descriptive complexity. Our general goal is to understand when and why these two measures of complexity differ, and what topological properties of the underlying space cause this disagreement. Our results suggest that the mismatch between the two measures of complexity reflects the discordance between the sequential and the topological aspects of the space, so that symbolic complexity may be

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 132; pp. 132:1–132:17
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

interpreted as a measure of *sequential* complexity rather than topological complexity, in the same way as many topological notions have a sequential counterpart (sequential continuity, sequential compactness, sequential closure, etc.).

More precisely, we show that among Hausdorff spaces, the spaces that are not Fréchet-Urysohn exhibit a disagreement between symbolic and topological complexity at the lowest level above the open sets, namely the differences of open sets. This result extends a similar result obtained in [1] for the subclass of coPolish spaces.

We focus on the space of open sets of a Polish space, and relate the disagreement between symbolic and topological complexity to the compactness properties of the Polish space, by dividing Polish spaces into 4 classes, ranging from the locally compact to the non σ-compact spaces, and giving a detailed analysis of descriptive complexity of sets in each case.

Along the way, we develop several tools and techniques that are needed to prove our results and are interesting on their own right. In particular we argue that the classical notion of hardness, which makes sense on countably-based spaces, is too restrictive on other spaces and we solve the problem by introducing the weaker notion of hard* set.

We finally observe that the discordance between topological and sequential aspects is already at the core of the theory of admissibly represented topological spaces. These spaces, also characterized as the T_0 quotients of countably-based spaces, are all sequential and form a subclass of topological spaces which behave particularly well from a categorical perspective: for instance, contrary to general topological spaces, they form a cartesian closed category. More concretely, in this category, the space constructions such as product space or subspaces do not coincide with the ones in the category of topological spaces, but with their sequentializations. Our separation results between symbolic and topological complexity heavily rely on the disagreement between sequential and topological space constructions.

1.1 Summary of the main results

We give a quick overview of the main results, stated informally.

In a represented space $\mathbf{X} = (X, \delta_{\mathbf{X}})$, we introduce the *symbolic* complexity of a set $A \subseteq X$. If Γ is a descriptive complexity class, such as $\undertilde{\mathbf{\Sigma}}^0_n$ or $\undertilde{\mathbf{D}}_n$ (difference of n open sets), then we define the corresponding symbolic complexity class $[\Gamma]$ as follows:

$$A \in [\Gamma](\mathbf{X}) \iff \delta_{\mathbf{X}}^{-1}(A) \in \Gamma(\mathrm{dom}(\delta_{\mathbf{X}})).$$

In a topological space with an admissible representation, one usually has $\Gamma(\mathbf{X}) \subseteq [\Gamma](\mathbf{X})$ and our goal is to understand when and why the other inclusion does not hold, i.e. when and why the topological and symbolic measures of complexity differ. It is know from [2] that they coincide when $\mathbf{X}$ is a countably-based space.

We first observe that the classical notion of hardness, which is very useful to identify the complexity of a set, is closely related to symbolic rather than topological complexity. We introduce a weaker version, called hard* set and prove:

▶ **Theorem** (Theorem 3.2)**.** *For a Borel subset A of an analytic space $\mathbf{X}$,*

$$\begin{aligned} A \textit{ is } \Gamma\textit{-hard} &\iff A \notin [\check{\Gamma}](\mathbf{X}), \\ A \textit{ is } \Gamma\textit{-hard*} &\iff A \notin \check{\Gamma}(\mathbf{X}). \end{aligned}$$

A topological subspace of a sequential space is not always sequential, so the subspace constructions differ in the categories of topological and sequential spaces. This difference implies a difference between symbolic and topological complexity.

The sequential spaces whose subspaces are sequential are called the Fréchet-Urysohn spaces. The class $\underset{\sim}{\mathbf{D}}_2$ consists of differences of two open sets.

▶ **Theorem** (Theorem 4.1). *If $\mathbf{X}$ is admissibly represented, Hausdorff and not Fréchet-Urysohn, then*

$$[\underset{\sim}{\mathbf{D}}_2](\mathbf{X}) \not\subseteq \underset{\sim}{\mathbf{D}}_2(\mathbf{X}).$$

The assumption that the space is Hausdorff is needed. Indeed, spaces of open sets behave better at low complexity levels.

▶ **Theorem** (Theorem 5.1). *If $\mathbf{X}$ is admissibly represented then*

$$[\underset{\sim}{\mathbf{D}}_n](\mathcal{O}(\mathbf{X})) = \underset{\sim}{\mathbf{D}}_n(\mathcal{O}(\mathbf{X})).$$

However, the proof is not constructive and we show that the corresponding effective classes disagree. The class D_2 consists of differences of two *effective* open sets. Let $\mathcal{N}_1$ be the space of functions $\mathbb{N} \to \mathbb{N}$ having at most 1 non-zero value.

▶ **Theorem** (Theorem 5.3). *One has $[\mathrm{D}_2](\mathcal{O}(\mathcal{N}_1)) \not\subseteq \mathrm{D}_2(\mathcal{O}(\mathcal{N}_1))$.*

Finally, we give a rather detailed study of descriptive complexity on the spaces $\mathcal{O}(\mathbf{X})$ when $\mathbf{X}$ is Polish. More precisely, we connect the relationship between symbolic and topological complexity classes to the compactness properties of $\mathbf{X}$. Some of the proofs heavily rely on the fact that the product topology is not sequential in general, so product space constructions differ in the categories of topological and sequential spaces.

In particular, symbolic and topological complexity differ at higher levels when $\mathbf{X}$ is Polish and not locally compact.

▶ **Theorem** (Theorem 6.5).

- *There exists $A \in [\underset{\sim}{\mathbf{D}}_\omega](\mathcal{O}(\mathcal{N}_1))$ which is $\underset{\sim}{\mathbf{\Delta}}^0_3$-complete*.*
- *There exists $A \in [\underset{\sim}{\mathbf{\Sigma}}^0_k](\mathcal{O}(\mathbb{N} \times \mathcal{N}_1))$ which is $\underset{\sim}{\mathbf{\Sigma}}^0_{k+1}$-complete*, for each $k \geq 2$.*
- *There exists $A \in [\underset{\sim}{\mathbf{\Sigma}}^0_2](\mathcal{O}(\mathcal{N}))$ which is not Borel.*

The paper is organized as follows. In Section 2, after giving the needed background on represented spaces, we introduce symbolic complexity and provide simple tools for its study. In Section 3 we introduce and study the notion of hard* set, used to capture the topological complexity of sets. In Section 4 we prove that Hausdorff spaces that are not Fréchet-Urysohn exhibit a disagreement between symbolic and topological complexity at the lowest level. In Section 5, we study spaces of open sets. In particular, in Section 6 we focus on open subsets of Polish spaces and locate symbolic complexity classes depending on the compactness properties of the Polish space.

We sometimes include the proof in the body of the article, and sometimes only give the intuition. Complete proofs can be found in [9].

2 Symbolic complexity

2.1 Represented spaces

The Baire space is $\mathcal{N} = \mathbb{N}^{\mathbb{N}}$, whose elements are either viewed as functions or infinite sequences. To finite sequence of natural numbers $\sigma \in \mathbb{N}^*$, we associate the cylinder $[\sigma]$ which is the set of elements of $\mathcal{N}$ extending σ. The Baire space is then endowed with the topology generated by the cylinders. Every subset of $\mathcal{N}$ is endowed with the subspace topology.

A **represented space** is a pair $\mathbf{X} = (X, \delta_{\mathbf{X}})$ where X is a set and $\delta_{\mathbf{X}} :\subseteq \mathcal{N} \to X$ is a partial surjective function called a **representation**. If $\delta_{\mathbf{X}}(p) = x$, then p is a **name** of x. If $\mathbf{X}, \mathbf{Y}$ are represented spaces then a function $F :\subseteq \mathcal{N} \to \mathcal{N}$ is a **realizer** of $f : \mathbf{X} \to \mathbf{Y}$ if $f \circ \delta_{\mathbf{X}} = \delta_{\mathbf{Y}} \circ F$. f is **computable** if it has a computable realizer. We write $\mathbf{X} \cong \mathbf{Y}$ if there exists a bijection between $\mathbf{X}$ and $\mathbf{Y}$ which is computable in both directions.

A representation δ of a topological space (X, τ) is **admissible** if τ is the final topology of δ and every partial continuous function $f :\subseteq \mathcal{N} \to X$ has a continuous realizer, which is a continuous function $F :\subseteq \mathcal{N} \to \mathcal{N}$ satisfying $f = \delta \circ F$.

If $\mathbf{X}, \mathbf{Y}$ are admissibly represented spaces, then a function $f : \mathbf{X} \to \mathbf{Y}$ is continuous if and only if it has a continuous realizer. In particular, f is continuous if and only if it is computable relative to some oracle.

The topological spaces having an admissible representation are exactly the T_0-spaces that are quotients of countably-based spaces, and are also called QCB_0-spaces. These spaces form a cartesian closed category, with very natural representations for products and function spaces. They enjoy the following remarkable but overlooked properties, as proved by Schröder [16]: if X is a QCB_0-space, then

- X is sequential, separable and has a countable network, i.e. a countable family of subsets such that every open set is a union of them,
- X is first-countable if and only if X is countably-based,
- X is hereditarily Lindelöf,
- When identifying the space of open sets $\mathcal{O}(X)$ with the function space $\mathbb{S}^X$ where $\mathbb{S}$ is the Sierpinski space, the topology on $\mathcal{O}(X)$ is the Scott topology.

As far as topology is concerned, admissibly represented spaces and QCB_0-spaces are the same. However, computability can only be expressed in terms of representations, so we will refer to admissibly represented spaces rather than QCB_0-spaces.

Countably-based T_0-spaces have a particular representation, called standard representation, which is admissible. Once a countable basis indexed by $\mathbb{N}$ has been chosen, say $(B_i)_{i\in\mathbb{N}}$, a name of x is any sequence $p \in \mathcal{N}$ such that $\{i \in \mathbb{N} : \exists n, p(n) = i + 1\} = \{i \in \mathbb{N} : x \in B_i\}$, so that x is described by enumerating its basic neighbourhoods in any order.

2.2 Symbolic complexity

Let Γ be a descriptive complexity class, i.e. a family $\Gamma = \{\Gamma(X)\}$ where X ranges over topological spaces and $\Gamma(X)$ is a collection of subsets of X. The simplest class is $\boldsymbol{\Sigma}^0_1 = \{\boldsymbol{\Sigma}^0_1(X)\}$ where $\boldsymbol{\Sigma}^0_1(X)$ is the collection of open subsets of X. The class $\boldsymbol{\Sigma}^0_n$ is inductively defined as the class of countable unions of differences of $\boldsymbol{\Sigma}^0_{n-1}$-sets. The class $\mathbf{D}_n$ is inductively defined as follows: $\mathbf{D}_1 = \boldsymbol{\Sigma}^0_1$ and $\mathbf{D}_{n+1}$ consists of sets $U \setminus A$ where $U \in \boldsymbol{\Sigma}^0_1$ and $A \in \mathbf{D}_n$. For any class Γ, the class $\check{\Gamma}$ consists of the complements of sets in Γ.

Let $\mathbf{X} = (X, \delta_{\mathbf{X}})$ be a represented space, which is also a topological space by taking the final topology of $\delta_{\mathbf{X}}$: $U \subseteq \mathbf{X}$ is open iff $\delta_{\mathbf{X}}^{-1}(U)$ is open in $\mathrm{dom}(\delta_{\mathbf{X}})$.

▶ **Definition 2.1.** *Let $\mathbf{X} = (X, \delta_{\mathbf{X}})$ be a represented space. We define the **symbolic** complexity class $[\Gamma](\mathbf{X})$ as follows: for $A \subseteq X$,*

$$A \in [\Gamma](\mathbf{X}) \iff \delta_{\mathbf{X}}^{-1}(A) \in \Gamma(\mathrm{dom}(\delta_{\mathbf{X}})).$$

By definition of the final topology of $\delta_{\mathbf{X}}$, one always has $[\boldsymbol{\Sigma}^0_1](\mathbf{X}) = \boldsymbol{\Sigma}^0_1(\mathbf{X})$. A descriptive complexity class Γ is often closed under continuous preimages, and in that case one has $\Gamma(\mathbf{X}) \subseteq [\Gamma](\mathbf{X})$ because $\delta_{\mathbf{X}}$ is continuous. Moreover, for such classes Γ, it is not hard to see that the symbolic complexity class $[\Gamma]$ does not depend on the choice of the admissible representation, hence $\Gamma(\mathbf{X})$ is intrinsic to X as a topological space. However, effective complexity classes are sensitive to the choice of an admissible representation (but not to the choice of a computably admissible one, which we do not discuss here).

Our general goal is to understand symbolic complexity classes, relate them to topological complexity classes and understand for which $\mathbf{X}$ and which Γ we have $[\Gamma](\mathbf{X}) = \Gamma(\mathbf{X})$.

An important result due to de Brecht [2] is that symbolic and topological complexity coincide when $\mathbf{X}$ is a countably-based T_0-space with its standard representation.

▶ **Theorem 2.2** ([2])**.** *Let $\mathbf{X}$ be a countably-based T_0-space with its standard representation, and let $\alpha, \beta < \omega_1$. One has*

$$[\underset{\sim}{\mathbf{D}}_\alpha(\underset{\sim}{\mathbf{\Sigma}}^0_\beta)](\mathbf{X}) = \underset{\sim}{\mathbf{D}}_\alpha(\underset{\sim}{\mathbf{\Sigma}}^0_\beta)(\mathbf{X}).$$

It was improved in [1] by observing that the equality is uniform and effective, so it also holds for the effective complexity classes $\mathrm{D}_m(\Sigma^0_n)$, $m, n \in \mathbb{N}$.

2.3 Tools

We give a simple way of locating a symbolic complexity class. A **network** in a topological space X is a family $\mathcal{F}$ of subsets of X such that every open set is a union of elements of $\mathcal{F}$ [5]. Every admissibly represented space has a countable network, given by the images of cylinders under the admissible representation.

▶ **Proposition 2.3.** *Let $\mathbf{X}$ be admissibly represented. Assume that $\mathbf{X}$ has a countable network of sets in $\underset{\sim}{\mathbf{\Sigma}}^0_{i+1}(\mathbf{X})$. For all $n \in \mathbb{N}$, one has*

$$[\underset{\sim}{\mathbf{\Sigma}}^0_n](\mathbf{X}) \subseteq \underset{\sim}{\mathbf{\Sigma}}^0_{n+i}(\mathbf{X}).$$

Proof. Let $\mathbf{Y}$ be the topological space with underlying set X and whose topology is generated by the countable network of $\mathbf{X}$. $\mathbf{Y}$ is countably-based and inherits the T_0-property of $\mathbf{X}$, let $\delta_{\mathbf{Y}}$ be its standard representation. Therefore, one has $[\underset{\sim}{\mathbf{\Sigma}}^0_n](\mathbf{Y}) = \underset{\sim}{\mathbf{\Sigma}}^0_n(\mathbf{Y})$ by Theorem 2.2.

By definition of a network, every open subset of $\mathbf{X}$ is an open subset of $\mathbf{Y}$. In other words, $\mathrm{id} : \mathbf{Y} \to \mathbf{X}$ is continuous hence continuously realizable, which implies that $[\underset{\sim}{\mathbf{\Sigma}}^0_n](\mathbf{X}) \subseteq [\underset{\sim}{\mathbf{\Sigma}}^0_n](\mathbf{Y})$. Conversely, every open subset of $\mathbf{Y}$ belongs to $\underset{\sim}{\mathbf{\Sigma}}^0_{i+1}(\mathbf{X})$ which implies, by induction on n, that $\underset{\sim}{\mathbf{\Sigma}}^0_n(\mathbf{Y}) \subseteq \underset{\sim}{\mathbf{\Sigma}}^0_{n+i}(\mathbf{X})$.

Putting everything together, we obtain $[\underset{\sim}{\mathbf{\Sigma}}^0_n](\mathbf{X}) \subseteq [\underset{\sim}{\mathbf{\Sigma}}^0_n](\mathbf{Y}) = \underset{\sim}{\mathbf{\Sigma}}^0_n(\mathbf{Y}) \subseteq \underset{\sim}{\mathbf{\Sigma}}^0_{n+i}(\mathbf{X})$. ◀

We take the following notion from [19]. An admissibly represented space is **quasi-zero-dimensional** if it is the sequentialization of a zero-dimensional space, i.e. if its open sets are the sequentially open sets of a space having a basis of clopen sets. For instance, the spaces $2^{\mathcal{N}}$ and $\mathbb{N}^{\mathcal{N}}$ are not zero-dimensional, as proved by Schröder [18], but they are quasi-zero-dimensional.

▶ **Corollary 2.4.** *Let $\mathbf{X}$ be quasi-zero-dimensional, for instance $\mathbf{X} = 2^{\mathcal{N}}$ or $\mathbb{N}^{\mathcal{N}}$. One has*

$$[\underset{\sim}{\mathbf{\Sigma}}^0_n](\mathbf{X}) \subseteq \underset{\sim}{\mathbf{\Sigma}}^0_{n+1}(\mathbf{X}).$$

Proof. The images of cylinders under the representation are closed subsets of $\mathbf{X}$ (Proposition 7 in [19]). ◀

A common technique to prove a separation result in a space $\mathbf{Y}$ is to prove it in a simpler space $\mathbf{X}$ and then transfer the result to $\mathbf{Y}$ by including $\mathbf{X}$ into $\mathbf{Y}$.

▶ **Proposition 2.5.** *Let Γ, Γ' be complexity classes that are closed under continuous (resp. computable) preimages.*

Let $\mathbf{X}$ be a continuous (resp. computable) retract of $\mathbf{Y}$. If $[\Gamma](\mathbf{Y}) \subseteq \Gamma'(\mathbf{Y})$, then $[\Gamma](\mathbf{X}) \subseteq \Gamma'(\mathbf{X})$.

Proof. Let $r : \mathbf{Y} \to \mathbf{X}$ and $s : \mathbf{X} \to \mathbf{Y}$ be continuous (resp. computable) functions such that $r \circ s = \mathrm{id}_{\mathbf{X}}$. Let $A \in [\Gamma](\mathbf{X})$ and $B = r^{-1}(A)$. As r is continuous hence continuously realizable (resp. computable), one has $B \in [\Gamma](\mathbf{Y})$ so $B \in \Gamma(\mathbf{Y})$. As s is continuous (resp. computable) and $A = s^{-1}(B)$, we conclude that $A \in \Gamma'(\mathbf{X})$. ◀

If $\mathbf{Y} = (Y, \delta_{\mathbf{Y}})$ is a represented space and $X \subseteq Y$, then $\mathbf{X} := (X, \delta_{\mathbf{X}})$ is a represented space by taking $\delta_{\mathbf{X}}$ as the restriction of $\delta_{\mathbf{Y}}$ to $\delta_{\mathbf{Y}}^{-1}(X)$. Observe that as a topological space, $\mathbf{X}$ is not always a topological subspace of $\mathbf{Y}$, but the sequentialization of the topological subspace [17].

▶ **Proposition 2.6.** *Let Γ be closed under finite intersections and continuous (resp. computable) preimages, and Γ' be closed under continuous (resp. computable) preimages. Let $X \in \Gamma(\mathbf{Y})$. If $[\Gamma](\mathbf{Y}) \subseteq \Gamma'(\mathbf{Y})$, then $[\Gamma](\mathbf{X}) \subseteq \Gamma'(\mathbf{X})$.*

Proof. The representation $\delta_{\mathbf{X}}$ of $\mathbf{X}$ is the restriction of $\delta_{\mathbf{Y}}$ to $\delta_{\mathbf{Y}}^{-1}(X)$.

Let $A \in [\Gamma](\mathbf{X})$. One has $A \in [\Gamma](\mathbf{Y})$. Indeed, $\delta_{\mathbf{Y}}^{-1}(A) = \delta_{\mathbf{X}}^{-1}(A) = S \cap \mathrm{dom}(\delta_{\mathbf{X}})$ for some $S \in \Gamma(\mathcal{N})$, and $\mathrm{dom}(\delta_{\mathbf{X}}) = \delta_{\mathbf{Y}}^{-1}(X) = T \cap \mathrm{dom}(\delta_{\mathbf{Y}})$ for some $T \in \Gamma(\mathcal{N})$. By assumption, $U := S \cap T \in \Gamma(\mathcal{N})$ so $\delta_{\mathbf{Y}}^{-1}(A) = U \cap \mathrm{dom}(\delta_{\mathbf{Y}})$ and $A \in [\Gamma](\mathbf{Y})$.

Threfore, $A \in \Gamma'(\mathbf{Y})$ so by continuity (resp. computability) of the identity from $\mathbf{X}$ to $\mathbf{Y}$, we conclude that $A \in \Gamma'(\mathbf{X})$. ◀

3 Hardness

An important tool to pinpoint the descriptive complexity of a set is provided by the notions of hardness and completeness. If Γ is a descriptive complexity class, then in any topological space X, one can define a set $A \subseteq X$ to be **Γ-hard** if for each $C \in \Gamma(\mathcal{N})$, there is a continuous reduction from C to A, i.e. a continuous function $f : \mathcal{N} \to X$ such that $C = f^{-1}(A)$. Note that the reduction always starts from $\mathcal{N}$. It contrasts with the generalizations of Wadge reducibility between subsets of a topological or represented spaces investigated in [14, 15].

As is well known in descriptive set theory on Polish (and even quasi-Polish) spaces, the hardness of a set is closely related to its complexity: Wadge's Lemma implies that for any class $\Gamma \neq \check{\Gamma}$ of Borel sets and any Borel subset A of a Polish space X,

$$A \text{ is } \Gamma\text{-hard} \iff A \notin \check{\Gamma}(X).$$

However, outside countably-based spaces it turns out that the hardness of a set is related to its symbolic rather than topological complexity, which usually differ as we will see shortly.

Therefore, we need another notion of hardness which reflects the topological complexity of a set.

▶ **Definition 3.1.** *Let (X, τ) be a topological space and Γ a descriptive complexity class. We say that $A \subseteq X$ is Γ-**hard*** if for every countably-based topology $\tau' \subseteq \tau$, A is Γ-hard in (X, τ'). A set is Γ-**complete*** if it belongs to $\Gamma(X)$ and is Γ-hard*.*

Note that when (X, τ) is countably-based, these notions coincide with the standard notions of hardness and completeness.

Now we can state the main result of this section, making clear that hardness is related to symbolic complexity, while hardness* is related to topological complexity. Say that a topological space is analytic if it is a continuous image of $\mathcal{N}$.

▶ **Theorem 3.2.** *Let $\Gamma = \mathbf{D}_\alpha(\boldsymbol{\Sigma}^0_\beta)$ where α, β are countable ordinals. For an analytic admissibly represented space X and $A \subseteq X$ Borel,*

$$\begin{aligned} A \text{ is } \Gamma\text{-hard} &\iff A \notin [\check{\Gamma}](X), \\ A \text{ is } \Gamma\text{-hard*} &\iff A \notin \check{\Gamma}(X). \end{aligned}$$

For $\beta = 1$, the assumptions that the space is analytic and that A is Borel can be dropped. The proof assumes $\boldsymbol{\Sigma}^1_1$-determinacy.

3.1 Hausdorff-Kuratowski Theorem

On Polish and even quasi-Polish spaces, there is no $\boldsymbol{\Delta}^0_n$-complete set because of the Hausdorff-Kuratowski Theorem. Other spaces may admit $\boldsymbol{\Delta}^0_n$-complete* sets, and we show that this possibility is again tightly related to the validity of the Hausdorff-Kuratowski Theorem for $\boldsymbol{\Delta}^0_n$-sets.

▶ **Definition 3.3.** *A topological space X has the **Hausdorff-Kuratowski property at level $\boldsymbol{\Delta}^0_n$** if*

$$\boldsymbol{\Delta}^0_n(X) = \bigcup_{\alpha < \omega_1} \mathbf{D}_\alpha(\boldsymbol{\Sigma}^0_{n-1})(X).$$

▶ **Theorem 3.4.** *Let X be an analytic topological space.*

For each $n \geq 2$, X has the Hausdorff-Kuratowski property at level $\boldsymbol{\Delta}^0_n$ if and only if X has no $\boldsymbol{\Delta}^0_n$-complete set.*

For $n = 2$, the analyticity assumption can be droppped.

Proof. If the HK property is satisfied, then there is no $\boldsymbol{\Delta}^0_n$-complete* set. Indeed, such a set A would be in $\mathbf{D}_\alpha(\boldsymbol{\Sigma}^0_{n-1})$ for some $\alpha < \omega_1$ and some countably-based topology, and $\boldsymbol{\Delta}^0_n$-hard for that topology, which would imply that $\boldsymbol{\Delta}^0_n(\mathcal{N}) \subseteq \mathbf{D}_\alpha(\boldsymbol{\Sigma}^0_{n-1})(\mathcal{N})$, which is known to be false (the difference hierarchies do not collapse on $\mathcal{N}$).

Conversely, if the HK property does not hold, then there exists $A \in \boldsymbol{\Delta}^0_n(X)$ such that $A \notin \mathbf{D}_\alpha(\boldsymbol{\Sigma}^0_{n-1})$ for any $\alpha < \omega_1$. If X is analytic or $n = 2$, then A is $\check{\mathbf{D}}_\alpha(\boldsymbol{\Sigma}^0_{n-1})$-hard* for each $\alpha < \omega_1$ by Theorem 3.2. As a result, A is $\boldsymbol{\Delta}^0_n$-hard*, hence $\boldsymbol{\Delta}^0_n$-complete*. ◀

We now give a criterion for the validity of the Hausdorff-Kuratowski property at a given level.

▶ **Theorem 3.5.** *Let (X, τ) be a topological space. If there exists a Polish topology τ' such that $\tau \subseteq \tau' \subseteq \boldsymbol{\Sigma}^0_n(\tau)$, then (X, τ) has the Hausdorff-Kuratowski property for all levels $\boldsymbol{\Delta}^0_k$ with $k \geq n + 1$.*

Proof. The proof follows the line of the argument in [10], reducing the case of $\boldsymbol{\Delta}^0_n$ to $\boldsymbol{\Delta}^0_2$ by enriching the topology. However, some care is needed because we have to deal with two topologies.

▷ Claim 3.6. For any $k \leq n$ and any countable family $\mathcal{F} \subseteq \boldsymbol{\Sigma}^0_k(X, \tau)$, there exists a Polish topology $\tau'' \subseteq \boldsymbol{\Sigma}^0_n(X, \tau)$ containing $\mathcal{F}$.

Proof of the Claim. We prove it by induction on k. For $k = 1$, the result is immediate by taking $\tau'' = \tau'$, as $\mathcal{F}$ is already contained in τ'. Assume the result for $k < n$ and let $\mathcal{F} \subseteq \boldsymbol{\Sigma}^0_{k+1}(X, \tau)$. There exists a countable family $\mathcal{G} \subseteq \boldsymbol{\Sigma}^0_k(X, \tau)$ such that each element of $\mathcal{F}$ is a countable union of differences of elements of $\mathcal{G}$. By induction, there is a Polish topology $\tau'' \subseteq \boldsymbol{\Sigma}^0_n(X, \tau)$ containing $\mathcal{G}$. Let τ''' be generated by τ'' and the complements of the elements of $\mathcal{G}$. As the latter sets are closed in τ'' which is Polish, τ''' is Polish. Moreover, those sets belong to $\boldsymbol{\Pi}^0_k(X, \tau) \subseteq \boldsymbol{\Sigma}^0_n(X, \tau)$, so $\tau''' \subseteq \boldsymbol{\Sigma}^0_n(X, \tau)$. Finally, each element of $\mathcal{F}$ is open in τ''', and the claim is proved. ◁

We now prove the theorem. Let $A \in \boldsymbol{\Delta}^0_{n+1}(X, \tau)$. There exists a countable family $\mathcal{F} \subseteq \boldsymbol{\Sigma}^0_n(X, \tau)$ such that A and its complement are countable unions of differences of elements of $\mathcal{F}$. Applying the claim, there exists a Polish topology $\tau'' \subseteq \boldsymbol{\Sigma}^0_n(X, \tau)$ containing $\mathcal{F}$. Therefore, $A \in \boldsymbol{\Delta}^0_2(X, \tau'')$ so applying the Hausdorff-Kuratowski theorem for Polish spaces, one has $A \in \mathbf{D}_\alpha(X, \tau'')$ for some $\alpha < \omega_1$. We conclude by observing that $\tau'' \subseteq \boldsymbol{\Sigma}^0_n(X, \tau)$. ◀

We give two simple applications of this result.

The space $\mathbb{R}[X]$ of polynomials with real coefficients is an example of a coPolish space which is not countably-based [3]. A polynomial is represented by giving an upper bound on its degree as well as standard names of its coefficients. On $\mathbb{R}[X]$, hence on $\mathbb{R}[X]^\mathbb{N}$, there is a set in $[\mathrm{D}_\omega]$ which is $\boldsymbol{\Delta}^0_2$-complete* (Theorem 5.8 in [1]). Theorem 3.5 implies that there is no $\boldsymbol{\Delta}^0_k$-complete* set for $k \geq 3$.

▶ **Corollary 3.7.** *The space $\mathbb{R}[X]^\mathbb{N}$ has the Hausdorff-Kuratowski property for all levels $\boldsymbol{\Delta}^0_k$ with $k \geq 3$, therefore that space has no $\boldsymbol{\Delta}^0_k$-complete* set for $k \geq 3$.*

Proof. For each $n, d \in \mathbb{N}$, the set $C_{n,d} := \{(P_i)_{i\in\mathbb{N}} : \deg(P_i) \leq d\}$ is closed. Enriching the topology on $\mathbb{R}[X]^\mathbb{N}$ with these sets results in a Polish topology contained in $\boldsymbol{\Sigma}^0_2(\mathbb{R}[X]^\mathbb{N})$ (the space becomes homeomorphic to $\mathbb{R}^\mathbb{N}$). ◀

We will see later that on $\mathcal{O}(\mathcal{N}_1)$, hence on $\mathcal{O}(\mathbb{N} \times \mathcal{N}_1)$, there is a set in $[\mathrm{D}_\omega]$ which is $\boldsymbol{\Delta}^0_3$-complete* (Theorem 6.5). Theorem 3.5 implies that there is no $\boldsymbol{\Delta}^0_k$-complete* set for $k \geq 4$.

▶ **Corollary 3.8.** *The space $\mathcal{O}(\mathbb{N}\times\mathcal{N}_1)$ has the Hausdorff-Kuratowski property for all levels $\boldsymbol{\Delta}^0_k$ with $k \geq 4$, therefore that space has no $\boldsymbol{\Delta}^0_k$-complete* set for $k \geq 4$.*

Proof. We add the following sets to the topology: for each $(n, f) \in \mathbb{N} \times \mathcal{N}_1$, the closed set $C_{n,f} := \{U : (n, f) \notin U\}$; for each $(n, p) \in \mathbb{N}^2$, the $\boldsymbol{\Pi}^0_2$-set $P_{n,p} := \{U : \{n\} \times [0^p] \subseteq U\}$, where $[0^p]$ is the set of functions $f \in \mathcal{N}_1$ such that $f(i) = 0$ for all $i < p$. We now show that the resulting topological space is homeomorphic to a closed subset of the Baire space, which implies that it is Polish.

We encode $U \in \mathcal{O}(\mathbb{N} \times \mathcal{N}_1)$ into two sequences g_n, h_n of elements of $\mathcal{N}$, which we can encode into a single element of $\mathcal{N}$. We use a one-to-one enumeration $(f_i)_{i\in\mathbb{N}}$ of the elements of $\mathcal{N}_1$, where f_0 is the null function.

Given U, we define its code $(g_n, h_n)_{n\in\mathbb{N}}$ as follows:

- $g_n(i) = 1$ if $(n, f_i) \in U$ and $g_n(i) = 0$ if $(n, f_i) \notin U$.
- $h_n(0) = 0$ if $(n, f_0) \notin U$, and $h_n(0) = p + 1$ if $(n, f_0) \in U$ and p is minimal such that $\{n\} \times [0^p] \subseteq U$.

It is not hard to see that the function sending U to $(g_n, h_n)_{n\in\mathbb{N}}$ is one-to-one and continuous for the enriched topology, as well as its inverse, and that the subset of $\mathcal{N}$ consisting of the codes of elements of $\mathcal{O}(\mathbb{N} \times \mathcal{N}_1)$ is closed. As a result, the enriched topological space is Polish.

Moreover, the enriched topology is contained in $\boldsymbol{\Sigma}^0_3(\mathcal{O}(\mathbb{N} \times \mathcal{N}_1))$. ◀

4 Fréchet-Urysohn property

In [1] we have given a characterization of the coPolish spaces on which the symbolic complexity differs from the topological complexity at the level $\underset{\sim}{\mathbf{D}}_2$: they are exactly the spaces that are not Fréchet-Urysohn.

We can extend part of the argument from coPolish spaces to Hausdorff admissibly represented spaces. We will see later (Theorem 5.1) that the assumption that the space is Hausdorff cannot be dropped.

▶ **Theorem 4.1.** *Let* $\mathbf{X}$ *be admissibly represented and Hausdorff. If* $\mathbf{X}$ *is not Fréchet-Urysohn, then*

$$[\underset{\sim}{\mathbf{D}}_2](\mathbf{X}) \not\subseteq \underset{\sim}{\mathbf{D}}_2(\mathbf{X}).$$

We use the Arens' space $\mathbf{S_2}$, which is the inductive limit of

$$X_N = \{0\} \cup \left\{\frac{1}{n} : n \in \mathbb{N}\right\} \cup \left\{\frac{1}{n} + \frac{1}{k} X^n : n \leq N, k \in \mathbb{N}\right\}.$$

As in [1], one has $[\underset{\sim}{\mathbf{D}}_2](\mathbf{S_2}) \not\subseteq \underset{\sim}{\mathbf{D}}_2(\mathbf{S_2})$ and a witness is the set $A = \{0\} \cup \{\frac{1}{n} + \frac{1}{k}X^n : n, k \in \mathbb{N}\}$. Therefore, Theorem 4.1 is an immediate corollary of the next result together with Proposition 2.6.

▶ **Proposition 4.2.** *Let* $\mathbf{X}$ *be admissibly represented and Hausdorff.* $\mathbf{X}$ *is not Fréchet-Urysohn if and only if* $\mathbf{X}$ *contains a closed copy of* $\mathbf{S_2}$*.*

▶ Remark 4.3 (Historical remark about Proposition 4.2). Franklin [7] proved that when X is a Hausdorff sequential space, X is Fréchet-Urysohn if and only if it does not contain a set which, endowed with the sequentialization of the subspace topology, is homeomorphic to $\mathbf{S_2}$ (Proposition 7.3 in [7]). It implies that if $\mathbf{X}$ is a Hausdorff admissibly represented space, then $\mathbf{X}$ is not Fréchet-Urysohn if and only if $\mathbf{X}$ does not contain $\mathbf{S_2}$ as a represented subspace.

In [22] and [11] it is proved that when X is a Hausdorff sequential space having a point-countable k-network, X is not Fréchet-Urysohn if and only if it does not contain a *closed* set homeomorphic to $\mathbf{S_2}$ (Theorem 2.12 in [11]). Observe that the subspace topology on a closed subset of a sequential space is always sequential, so there is no need to take the sequentialization of the subspace topology as in Franklin's result. This result implies ours, because admissibly represented spaces are sequential and the images of cylinders under the representation give a countable k-network. However we provide a proof in our setting for self-containedness.

The result was also recently proved in [3] for the subclass of coPolish spaces (Proposition 66 in [3], where $\mathbf{S_2}$ is called $\mathbf{S_{min}}$).

5 Spaces of open sets

As already mentioned, the assumption that the space is Hausdorff is important in Theorem 4.1, because some spaces admitting a rich poset structure behave more smoothly. This phenomenon has been already exploited in many ways in the realm of domain theory. We show that even in the absence of a countable basis, some positive results are still valid.

▶ **Theorem 5.1.** *Let $\mathbf{X}$ be admissibly represented. For every $n \geq 2$,*

$$[\underset{\sim}{\mathbf{D}}_n](\mathcal{O}(\mathbf{X})) = \underset{\sim}{\mathbf{D}}_n(\mathcal{O}(\mathbf{X})).$$

We will see that this equality cannot be extended to level ω: if $\mathbf{X}$ is Polish and not locally compact, then $[\underset{\sim}{\mathbf{D}}_\omega](\mathcal{O}(\mathbf{X}))$ contains a $\underset{\sim}{\boldsymbol{\Delta}}^0_3$-complete* set (Theorem 6.5).

Informal proof. The strategy is inspired from the one developed in [8] and [20] where a similar result is proved in the context of numbered sets and algebraic domains. We show that the result can be proved in the context of represented spaces, and without assuming a countable basis.

We first isolate a property of Scott open sets: say that a set $A \subseteq \mathcal{O}(\mathbf{X})$ is *approximable* if for every directed set $\Delta \subseteq \mathcal{O}(\mathbf{X})$ such that $\sup \Delta \in A$, Δ intersects A. A set is Scott open, i.e. in $\underset{\sim}{\boldsymbol{\Sigma}}^0_1(\mathcal{O}(\mathbf{X}))$ if and only if it is upwards closed and approximable.

We prove the following generalization. A set A is in $\underset{\sim}{\mathbf{D}}_n(\mathcal{O}(\mathbf{X}))$ if and only if both A and A^c are approximable and has no $n+1$-chain, i.e. no sequence $U_0 \subseteq U_1 \subseteq \ldots \subseteq U_n$ such that $U_i \in A$ iff i is even (it indeed generalizes the case of open sets for $n = 1$).

The last step is to prove that:

- A subset of $\mathcal{O}(\mathbf{X})$ that is not approximable is necessarily $\underset{\sim}{\boldsymbol{\Pi}}^0_2$-hard,
- A subset of $\mathcal{O}(\mathbf{X})$ having an $n+1$-chain is necessarily $\check{\underset{\sim}{\mathbf{D}}}_n$-hard.

Therefore, if $A \in [\underset{\sim}{\boldsymbol{\Sigma}}^0_2](\mathcal{O}(\mathbf{X}))$ then A is approximable, and if moreover $A \in [\underset{\sim}{\mathbf{D}}_n](\mathcal{O}(\mathbf{X}))$, then A has no $n+1$-chain.

Putting everything together implies that $[\underset{\sim}{\mathbf{D}}_n](\mathcal{O}(\mathbf{X})) \subseteq \underset{\sim}{\mathbf{D}}_n(\mathcal{O}(\mathbf{X}))$. ◀

A consequence of the preceding development is a characterization of the class $[\underset{\sim}{\boldsymbol{\Delta}}^0_2]$ in certain cases.

▶ **Proposition 5.2.** *Let $\mathbf{X}$ be countably-based. The class $[\underset{\sim}{\boldsymbol{\Delta}}^0_2](\mathcal{O}(\mathbf{X}))$ is precisely the class of approximable and co-approximable sets.*

Proof. We know from the proof of Theorem 5.1 that if $A \in [\underset{\sim}{\boldsymbol{\Delta}}^0_2](\mathcal{O}(\mathbf{X}))$ then both A and A^c are approximable.

Conversely, assume that $A \subseteq \mathcal{O}(\mathbf{X})$ and its complement are approximable. Observe that if U_i is a growing sequence of open sets with union U, then $\mathbf{1}_A(U_i)$ converges to $\mathbf{1}_A(U)$ as $i \to \infty$, as both A and A^c are approximable. Let $(B_i)_{i \in \mathbb{N}}$ be a countable basis of $\mathbf{X}$, closed under finite intersections and unions. Let $E = \{i \in \mathbb{N} : B_i \in A\}$. From a name of an open set $U \in \mathcal{O}(\mathbf{X})$, one can continuously derive a sequence $(i_n)_{n \in \mathbb{N}}$ such that $B_{i_n} \subseteq B_{i_{n+1}}$ and $\bigcup_n B_{i_n} = U$. Therefore, whether $U \in A$ can be tested with finitely mind changes, by testing whether $i_n \in E$. ◀

5.1 Effectiveness

The proof of Theorem 5.1 is not effective. We show that there is no effective argument by proving that $[\mathrm{D}_2](\mathcal{O}(\mathbf{X})) \nsubseteq \mathrm{D}_2(\mathcal{O}(\mathbf{X}))$ for some particular $\mathbf{X}$.

▶ **Theorem 5.3.** *One has*

$$[\mathrm{D}_2](\mathcal{O}(\mathcal{N}_1)) \nsubseteq \mathrm{D}_2(\mathcal{O}(\mathcal{N}_1)),$$

and a witness can be taken in $\underset{\sim}{\mathbf{D}}_2(\mathcal{O}(\mathcal{N}_1))$.

Such a set is a difference of two open sets, but computationally speaking, its name set is strictly easier to describe than the set itself.

Informal proof. We prove the separation result for the space $\mathbb{N} \times \mathcal{O}(\mathcal{N}_1)$ and observe that this space embeds as a D_2-subset of $\mathcal{O}(\mathcal{N}_1)$, to which the result immediately transfers by Proposition 2.6. The following discussion is rather general, and $\mathcal{O}(\mathcal{N}_1)$ is a possible instance of $\mathbf{X}$.

In [1] we proved that if $\mathbf{X}$ is not countably-based, then for sets in $\underset{\sim}{\mathbf{D}}_2(\mathbf{X})$, one cannot continuously convert $[\underset{\sim}{\mathbf{D}}_2]$-names into $\underset{\sim}{\mathbf{D}}_2$-names. Another way of formulating this result is expressed by the existence of a function $f : \mathbf{Y} \to \underset{\sim}{\mathbf{D}}_2(\mathbf{X})$ for some represented space $\mathbf{Y}$, such that:

- $f : \mathbf{Y} \to [\underset{\sim}{\mathbf{D}}_2](\mathbf{X})$ is continuously realizable,
- $f : \mathbf{Y} \to \underset{\sim}{\mathbf{D}}_2(\mathbf{X})$ is not continuously realizable.

Equivalently, it means that $\{(y, x) : x \in f(y)\}$ belongs to $[\underset{\sim}{\mathbf{D}}_2](\mathbf{Y} \times \mathbf{X})$ but not $\underset{\sim}{\mathbf{D}}_2(\mathbf{Y} \times \mathbf{X})$.

If $f : \mathbf{Y} \to [\underset{\sim}{\mathbf{D}}_2](\mathbf{X})$ is moreover computable and if there is an effective enumeration $(y_i)_{i \in \mathbb{N}}$ of the computable elements of $\mathbf{Y}$, then we can consider the following set:

$$A = \{(i, x) \in \mathbb{N} \times \mathbf{X} : x \in f(y_i)\}.$$

One immediately has $A \in \underset{\sim}{\mathbf{D}}_2(\mathbb{N} \times \mathbf{X})$, $A \in [\mathrm{D}_2](\mathbb{N} \times \mathbf{X})$ and one has to prove that $A \notin \mathrm{D}_2(\mathbb{N} \times \mathbf{X})$, which depends on the details of Y and f.

For $\mathbf{X} = \mathcal{O}(\mathcal{N}_1)$, one can make $\mathbf{Y}$ and $f : \mathbf{Y} \to \underset{\sim}{\mathbf{D}}_2(\mathbf{X})$ very explicit: let $\mathbf{Y} = \overline{\mathbb{N}} \times \overline{\mathbb{N}}$ and

$$\begin{aligned} f(\infty, y) &= \{U \in \mathcal{O}(\mathcal{N}_1) : f_\infty \in U\}, \\ f(n, \infty) &= \emptyset, \\ f(n, p) &= \{U \in \mathcal{O}(\mathcal{N}_1) : f_{n,p} \notin U\}, \end{aligned}$$

where f_∞ is the null function and $f_{n,p} \in \mathcal{N}_1$ is the only function satisfying $f_{n,p}(n) = p + 1$.

One can take the following effective indexing of $\mathbf{Y} = \overline{\mathbb{N}} \times \overline{\mathbb{N}}$: if $\langle .,. \rangle : \mathbb{N}^2 \to \mathbb{N}$ is a computable bijection, and t_i is the halting time of Turing machine number i, then let $y_{\langle i,j \rangle} = (t_i, t_j)$. The set $A \subseteq \mathbb{N} \times \mathcal{O}(\mathcal{N}_1)$ becomes:

$$A = \{(\langle i, j \rangle, U) : f_\infty \in U \text{ and } M_i \text{ does not halt, or } M_i \text{ and } M_j \text{ halt and } f_{t_i,t_j} \notin U\}. \qquad \blacktriangleleft$$

▶ **Corollary 5.4.** *If $\mathcal{N}_1$ embeds as a D_2-subset of $\mathbf{X}$, then*

$$[\mathrm{D}_2](\mathcal{O}(\mathbf{X})) \nsubseteq \mathrm{D}_2(\mathcal{O}(\mathbf{X})).$$

Proof. $\mathcal{O}(\mathcal{N}_1)$ is a computable retract of $\mathcal{O}(\mathbf{X})$, so the separation result (Theorem 5.3) about $\mathcal{O}(\mathcal{N}_1)$ extends to $\mathcal{O}(\mathbf{X})$ by Proposition 2.5. ◀

We consider the so-called sequential fan $\mathbf{S}(\omega) = \{0\} \cup \{\frac{1}{p}X^n : n, p \in \mathbb{N}\} \subseteq \mathbb{R}[X]$. It is Fréchet-Urysohn but has one point with no countable basis of neighborhoods. The space $\mathbb{N} \times \mathbf{S}(\omega)$ has infinitely many points with no countable basis of neighborhoods.

▶ **Theorem 5.5.** *Let $\mathbf{X} = \mathbb{N} \times \mathbf{S}(\omega)$. One has*

$$[\mathrm{D}_2](\mathbf{X}) \nsubseteq \mathrm{D}_2(\mathbf{X}),$$

and it is witnessed by an open set.

Proof. We follow the same scheme as in the preceding proof. Let

$$A = \{(n, P) : \text{if } M_n \text{ halts then } \deg(P) > t_n\}.$$

First, A is open because it is Σ^0_1 relative the halting set.

We show that $A \in [\mathrm{D}_2](\mathbf{X})$. We are given (n, P), with an upper bound d on $\deg(P)$. We run M_n for d steps. If M_n halts before d steps, then we know the value of t_n so we can test whether $\deg(P) > t_n$ (i.e. we start rejecting (n, P) until this test succeeds, in which case we accept (n, P)). If M_n does not halt before d steps, then we accept (n, P) and eventually reject it if M_n halts (indeed, in that case one has $\deg(P) \leq d \leq t_n$).

We show that $A \notin \mathrm{D}_2(\mathbf{X})$. Observe that for each n, $(n, 0)$ belongs to the closure of A: if M_n does not halt then $(n, 0) \in A$; if M_n halts then $(n, 0)$ is the limit when p grows of $(n, \frac{1}{p}X^{t_n+1}) \in A$.

Assume that $A = U \setminus V$ where U, V are open subsets of $\mathbf{X}$ and U is effectively open. For each n, as $(n, 0)$ belongs to the closure of A then one must have $(n, 0) \notin V$. Therefore, $(n, 0) \in A \iff (n, 0) \in U$. However, $(n, 0) \in A$ iff M_n does not halt, so it cannot be equivalent to $(n, 0) \in U$ which is a c.e. condition. We obtain a contradiction, so $A \notin \mathrm{D}_2(\mathbf{X})$. ◀

Proposition 2.6 immediately implies

▶ **Corollary 5.6.** *If $\mathbb{N} \times \mathbf{S}(\omega)$ computably embeds as a D_2-subset of $\mathbf{X}$, then*

$$[\mathrm{D}_2](\mathbf{X}) \nsubseteq \mathrm{D}_2(\mathbf{X})$$

with a witness in $\underset{\sim}{\mathbf{D}}_2(\mathbf{X})$.

One easily checks that the spaces $\mathbb{R}[X]$, $2^{\mathbb{N}^\mathbb{N}}$ and $\mathbb{N}^{\mathbb{N}^\mathbb{N}}$ are instances of this result:

- $\mathbb{N} \times \mathbf{S}(\omega) \subseteq \mathbb{R}[X]$ by identifying (n, P) with $\frac{1}{n} + XP$,
- $\mathbb{N} \times \mathbf{S}(\omega) \subseteq 2^{\mathbb{N}^\mathbb{N}}$ by identifying $(n, 0)$ with $\{f \in \mathcal{N} : f(0) = n\}$ and $(n, \frac{1}{p}X^q)$ with $\{f \in \mathcal{N} : f(0) = n \text{ and } f(q+1) \leq p\}$.

6 Spaces of open subsets of Polish spaces

We now focus on spaces of open subsets of Polish spaces, for which we can establish a rather precise picture of the relationship between symbolic and topological complexity, depending on the compactness properties of the space. We show how the behavior of symbolic complexity classes on $\mathcal{O}(X)$ is closely related to the compactness properties of X.

6.1 The 4 classes

The first observation is that when X is locally compact, for instance $X = \mathbb{R}$, $\mathcal{O}(X)$ is countably-based so it behaves very well in terms of descriptive complexity: symbolic and topological complexity coincide. We split the whole class of Polish spaces into 4 disjoint classes, ranging from the locally compact spaces to the non σ-compact spaces.

Let $X_{\text{nk}} = \{x \in X : x \text{ has no compact neighborhood}\}$, which is a closed subset of X.

▶ **Definition 6.1.** *Let X be a Polish space.*

1. *$X \in$ Class I if $X_{\text{nk}} = \emptyset$, i.e. X is locally compact,*
2. *$X \in$ Class II if $X_{\text{nk}} \neq \emptyset$ is finite,*
3. *$X \in$ Class III if $X_{\text{nk}} \neq \emptyset$ is infinite and X is σ-compact,*
4. *$X \in$ Class IV if X is not σ-compact.*

Observe that the union of Classes I, II, III is the class of σ-compact spaces.

▶ **Example 6.2.** Let us give one example for each class:

1. $\mathbb{R}$ belongs to Class I,
2. $\mathcal{N}_1 = \{f \in \mathcal{N} : f \text{ takes at most one positive value}\}$ belongs to Class II, with one element having no compact neighborhood, namely the zero function f_0,

3. $\mathbb{N}\times\mathcal{N}_1$ belongs to Class III, where the elements with no compact neighborhood are the pairs (n, f_0),
4. $\mathcal{N}$ belongs to Class IV.

Moreover, the three latter spaces are minimal in their respective classes, i.e. embed into every space of their classes.

▶ **Proposition 6.3.** *Let X be Polish.*

- *$X \notin$ Class I $\iff$ X contains a closed copy of $\mathcal{N}_1$,*
- *$X \notin$ Classes I or II $\iff$ X contains a $\underset{\sim}{\mathbf{D}}_2$ copy of $\mathbb{N}\times\mathcal{N}_1$,*
- *$X \notin$ Classes I, II or III $\iff$ X contains a closed copy of $\mathcal{N}$.*

Proof. The backwards implications are easy, because if C is a closed subset, or even a $\underset{\sim}{\mathbf{D}}_2$-subset of $\mathbf{X}$ and $x \in C$ has no compact neighborhood in the subspace C, then x has no compact neighborhood in $\mathbf{X}$.

Assume that $\mathbf{X}$ is not locally compact and let $x_0 \in \mathbf{X}_{\mathrm{nk}}$. We define a double-sequence $x_{i,n}$ by induction on i. Let B_0 be a basic neighborhood of x_0. As $\overline{B_0}$ is not compact, it contains a sequence $x_{0,n}$ with no converging subsequence. In particular, there exists a neighborhood B_1 of x_0 such that $\overline{B_1}$ does not contain any $x_{0,n}$. Again, $\overline{B_1}$ is not compact so it contains a sequence $x_{i,n}$ with no converging subsequence. We continue, making sure that the radius of B_i converges to 0. One easily checks that the set $\{x_0\}\cup\{x_{i,n} : i,n\in\mathbb{N}\}$ is closed and homeomorphic to $\mathcal{N}_1$, by sending x_0 to the zero function, and $x_{i,n}$ to the function f such that $f(i) = n$.

Assume that $\mathbf{X}_{\mathrm{nk}}$ is infinite. It contains a copy D of $\mathbb{N}$ with $D \in \underset{\sim}{\mathbf{D}}_2(\mathbf{X})$. Each point $x \in D$ is contained in a neighrbohood B_x such that $\overline{B_x}\cap\overline{B_y} = \emptyset$ for $x \neq y$. Around each point x of D and inside B_x we can build a closed copy of $\mathcal{N}_1$ as in the previous case. Their union is a copy of $\mathbb{N}\times\mathcal{N}_1$ and belongs to $\underset{\sim}{\mathbf{D}}_2(\mathbf{X})$.

The third statement is a particular case of Hurewicz theorem (Theorem 7.10 in [10]). ◀

6.2 Classification

We now relate the behavior of symbolic complexity on $\mathcal{O}(X)$ to the class of X. We first locate the symbolic complexity classes.

▶ **Theorem 6.4** (Classification – Positive results)**.** *Let X be Polish.*

1. *If $X \in$ Class I, then $[\underset{\sim}{\mathbf{\Sigma}}^0_k](\mathcal{O}(X)) = \underset{\sim}{\mathbf{\Sigma}}^0_k(\mathcal{O}(X))$ for all k,*
2. *If $X \in$ Class II, then $[\underset{\sim}{\mathbf{\Sigma}}^0_k](\mathcal{O}(X)) = \underset{\sim}{\mathbf{\Sigma}}^0_k(\mathcal{O}(X))$ for $k \geq 3$,*
3. *If $X \in$ Class III, then $[\underset{\sim}{\mathbf{\Sigma}}^0_k](\mathcal{O}(X)) \subseteq \underset{\sim}{\mathbf{\Sigma}}^0_{k+2}(\mathcal{O}(X))$ for $k \geq 2$.*

Informal proof. If $X \in$ Class I, i.e. if X is locally compact, then $\mathcal{O}(\mathbf{X})$ is countably-based [16], so symbolic and topological complexity coincide there (Theorem 2.2).

If $X \in$ Class II, then up to a finite set, X is countably-based, and we show that this finite set do not affect the complexity of sets for levels $k \geq 3$.

If $X \in$ Class III, then X is σ-compact, so its open sets are σ-compact as well. Therefore, for each open set B, the corresponding set $P_B = \{U \in \mathcal{O}(\mathbf{X}) : B \subseteq U\}$ belongs to $\underset{\sim}{\mathbf{\Pi}}^0_2(\mathcal{O}(X))$. The countable family $\mathcal{B}$ of finite unions basic open subsets of X induces a countable network $(P_B)_{B\in\mathcal{B}}$ of $\mathcal{O}(\mathbf{X})$ made of $\underset{\sim}{\mathbf{\Pi}}^0_2$-sets. Therefore, we can apply Proposition 2.3. ◀

We then identify gaps between symbolic and topological complexity.

▶ **Theorem 6.5** (Classification – Negative results)**.** *Let X be Polish.*

1. *If $X \notin$ Class I, then $[\boldsymbol{D}_\omega](\mathcal{O}(X))$ contains a $\boldsymbol{\Delta}^0_3$-complete* set,*
2. *If $X \notin$ Class II, then $[\boldsymbol{\Sigma}^0_k](\mathcal{O}(X))$ contains a $\boldsymbol{\Sigma}^0_{k+1}$-complete*-set for $k \geq 2$,*
3. *If $X \notin$ Class III, then $[\boldsymbol{\Sigma}^0_2](\mathcal{O}(X))$ contains a non-Borel set.*

We observe that two phenomena are possible. For some spaces, the classes $[\boldsymbol{\Sigma}^0_k]$ and $\boldsymbol{\Sigma}^0_k$ differ for low values of k and then coincide after some rank (if X is in Class II, then they coincide for $k \geq 3$). For other spaces, the classes never coincide (if X is in Class III or IV).

It is open whether $\boldsymbol{\Sigma}^0_k(O(X)) \subseteq \boldsymbol{\Sigma}^0_{k+1}(\mathcal{O}(X))$ when X belongs to Class III. A similar study should be done when X is not Polish.

Informal proof. The difference between symbolic and topological complexity is related to the fact that product spaces usually have two different natural topologies: the product topology (which is the structure obtained as the cartesian product in the category of topological spaces), and its sequentialization (obtained from the cartesian product in the category of QCB$_0$-spaces, or admissibly represented spaces). These two different topologies obviously induce different topological complexity classes, already at the first level $\boldsymbol{\Sigma}^0_1$.

For instance, on $\mathcal{N} \times \mathcal{O}(\mathcal{N})$, the set $\{(f, U) : f \in U\}$ is open but is not Borel for the product topology.

The proof shows how to exploit the difference between the product topology and its sequentialization on the space $\mathcal{N} \times \mathcal{O}(\mathbf{X})$ and turn it into a difference between symbolic and topological complexity on $\mathcal{O}(\mathbf{X})$.

When $\mathbf{X} = \mathbb{N} \times \mathcal{N}_1$, one has $\mathbf{X} \cong \mathbb{N} \times \mathbf{X}$ so $\mathcal{O}(\mathbf{X}) \cong \mathcal{O}(\mathbf{X})^{\mathbb{N}}$. This equality enables one to iterate: if $A_k \in [\Sigma^0_k](\mathcal{O}(\mathbf{X}))$ is $\boldsymbol{\Sigma}^0_{k+1}$-complete*, then the set

$$A_{k+1} = \{(U_i)_{i \in \mathbb{N}} \in \mathcal{O}(\mathbf{X})^{\mathbb{N}} : \exists i, U_i \notin A_k\}$$

belongs to $[\boldsymbol{\Sigma}^0_{k+1}](\mathcal{O}(\mathbf{X})^{\mathbb{N}})$ and it $\boldsymbol{\Sigma}^0_{k+2}$-complete*. ◀

The fact that $\mathbf{X}$ is Polish is essential in the proofs. A particular property of Polish and quasi-Polish spaces that is used is the following.

▶ **Proposition 6.6.** *If $\mathbf{X}$ and $\mathbf{Y}$ are quasi-Polish, then the topologies on the admissiby represented spaces $\mathcal{O}(\mathbf{X})^{\mathbb{N}}$ and $\mathcal{O}(\mathbf{X}) \times \mathcal{O}(\mathbf{Y})$ are the product topologies.*

Proof. As represented spaces, one has $\mathcal{O}(\mathbf{X})^{\mathbb{N}} \cong \mathcal{O}(\mathbb{N} \times \mathbf{X})$ and $\mathcal{O}(\mathbf{X}) \times \mathcal{O}(\mathbf{Y}) \cong \mathcal{O}(\mathbf{X} \sqcup \mathbf{Y})$. The topologies on the admissibly represented spaces $\mathcal{O}(\mathbb{N} \times \mathbf{X})$ and $\mathcal{O}(\mathbf{X} \sqcup \mathbf{Y})$ are the Scott topologies.

On the other hand, for any topological spaces X, Y, it is easy to see that the compact-open topology on $\mathcal{O}(\mathbb{N} \times X)$ and $\mathcal{O}(X \sqcup Y)$ is the product topology on $\mathcal{O}(X)^{\mathbb{N}}$ and $\mathcal{O}(X) \times \mathcal{O}(Y)$ respectively, where $\mathcal{O}(X)$ and $\mathcal{O}(Y)$ are endowed with the compact-open topology.

When $\mathbf{X}$ and $\mathbf{Y}$ are quasi-Polish, so are $\mathbb{N} \times \mathbf{X}$ and $\mathbf{X} \sqcup \mathbf{Y}$, so $\mathbf{X}$, $\mathbf{Y}$, $\mathbb{N} \times \mathbf{X}$ and $\mathbf{X} \sqcup \mathbf{Y}$ are consonant, i.e. the Scott topology and the compact-open topology coincide on their spaces of open sets [4]. As a result the topology on the represented spaces $\mathcal{O}(\mathbf{X})^{\mathbb{N}}$ and $\mathcal{O}(\mathbf{X}) \times \mathcal{O}(\mathbf{Y})$ is the product of the topologies on $\mathcal{O}(\mathbf{X})$ and $\mathcal{O}(\mathbf{Y})$. ◀

Open subsets of the Baire space

We now give the complete proof that the class $[\Sigma^0_2](\mathcal{O}(\mathcal{N}))$ contains a set that is not Borel (Theorem 6.5, item 3.).

Proof. We have seen that two represented spaces $\mathbf{X}$ and $\mathbf{Y}$ naturally induce a third represented space $\mathbf{X} \times \mathbf{Y}$. The topology induced by that representation is not in general the product topology, but its sequentialization.

A simple example is given by $\mathbf{X} = \mathcal{N}$ and $\mathbf{Y} = \mathcal{O}(\mathcal{N})$. The evaluation map $\mathcal{N} \times \mathcal{O}(\mathcal{N}) \to \mathbb{S}$ is continuous (and computable), however it is not continuous w.r.t. the product topology, because $\mathcal{N}$ is not locally compact (see [6] for more details on this topic). In other words the set $\{(f, O) \in \mathcal{N} \times \mathcal{O}(\mathcal{N}) : f \in O\}$ is not open for the product topology (but it is sequentially open, or open for the topology induced by the representation). It is even worse.

▶ **Proposition 6.7.** *$E = \{(f, O) \in \mathcal{N} \times \mathcal{O}(\mathcal{N}) : f \in O\}$ is not Borel for the product topology.*

Proof. We prove that for every Borel set A, there exists a dense G_δ-set $G \subseteq \mathcal{N}$ such that for every $f \in G$, $(f, \mathcal{N} \setminus \{f\}) \in A \iff (f, \mathcal{N}) \in A$. It implies the result as it is obviously false for the set E. To prove it, we show that the class of sets satisfying this condition contains the open sets in the product topology and is closed under taking complements and countable unions, which implies that this class contains the Borel sets.

First, consider a basic open set $A = [u] \times \mathcal{U}_K$ where u is a finite sequence of natural numbers, K is a compact subset of $\mathcal{N}$ and $\mathcal{U}_K = \{O \in \mathcal{O}(\mathcal{N}) : K \subseteq O\}$. Define $G = [u]^c \cup [u] \setminus K$, which is a dense open set. For $f \in [u]^c$, no (f, O) belongs to A. For $f \in [u] \setminus K$, both $(f, \mathcal{N} \setminus \{f\})$ and $(f, \mathcal{N})$ belong to A.

If A satisfies the condition with a dense G_δ-set G, then A^c satisfies the condition with the same G. If A_i satisfy the condition with dense G_δ-sets G_i then $\bigcup_i A_i$ satisfies the condition with $G = \bigcap_i G_i$. ◀

We now use the set E to build a set in $[\Sigma^0_2](\mathcal{O}(\mathcal{N}))$ which is not Borel. We show that $\mathcal{N} \times_{\text{prod}} \mathcal{O}(\mathcal{N})$, which is the topological space endowed with the product topology, is a $[\Sigma^0_2]$-retract of $\mathcal{O}(\mathcal{N})$. We build:

- A continuous function $s : \mathcal{N} \times_{\text{prod}} \mathcal{O}(\mathcal{N}) \to \mathcal{O}(\mathcal{N})$,
- A $[\Sigma^0_2]$-measurable function $r : \mathcal{O}(\mathcal{N}) \to \mathcal{N} \times \mathcal{O}(\mathcal{N})$,
- Such that $r \circ s = \text{id}$.

First, these ingredients enable us to derive the result. Indeed, let E be the set from Proposition 6.7 and $F := r^{-1}(E) \subseteq \mathcal{O}(\mathcal{N})$. As E is open in $\mathcal{N} \times \mathcal{O}(\mathcal{N})$, F is Σ^0_2. However F is not Borel, otherwise $E = s^{-1}(F)$ would be Borel in $\mathcal{N} \times_{\text{prod}} \mathcal{O}(\mathcal{N})$.

Let us now build s and r. We identify $\mathcal{O}(\mathcal{N})$ with $\mathcal{O}(\mathcal{N}) \times \mathcal{O}(\mathcal{N})$ and use the fact that the topology on $\mathcal{O}(\mathcal{N}) \times \mathcal{O}(\mathcal{N})$ coincides with the product topology by Proposition 6.6.

▶ **Lemma 6.8.** *$\mathcal{N}$ is a $[\Sigma^0_2]$-retract of $\mathcal{O}(\mathbb{N})$: there exists $r : \mathcal{O}(\mathbb{N}) \to \mathcal{N}$ which is $[\Sigma^0_2]$-measurable, $s : \mathcal{N} \to \mathcal{O}(\mathbb{N})$ which is computable, such that $r \circ s = \text{id}_{\mathcal{N}}$.*

Proof. Let $\langle .,. \rangle : \mathbb{N}^2 \to \mathbb{N}$ be a computable bijection. Let $r(E) = f_E$ be defined by

$$f_E(i) = \begin{cases} \min\{j \in \mathbb{N} : \langle i, j \rangle \in E\} & \text{if that set is non-empty,} \\ 0 & \text{otherwise.} \end{cases}$$

Let $s(f) = \{\langle i, f(i) \rangle : i \in \mathbb{N}, f(i) \geq 1\}$. One easily checks that r and s satisfy the required conditions. ◀

By Lemma 6.8, $\mathcal{N}$ is a $[\Sigma^0_2]$-retract of $\mathcal{O}(\mathbb{N})$, which is a computable retract of $\mathcal{O}(\mathcal{N})$, so $\mathcal{N}$ is a $[\Sigma^0_2]$-retract of $\mathcal{O}(\mathcal{N})$. It is witnessed by two functions $r_0 : \mathcal{O}(\mathcal{N}) \to \mathcal{N}$ and $s_0 : \mathcal{N} \to \mathcal{O}(\mathcal{N})$ such that $r_0 \circ s_0 = \mathrm{id}_{\mathcal{N}}$.

Let us simply pair s_0 and r_0 with the identity on $\mathcal{O}(\mathcal{N})$: let $s(f, O') = (s_0(f), O')$ and $r(O, O') = (r_0(O), O')$. ◀

In particular, that set is not a countable union of differences of open sets, as it should be on Polish or quasi-Polish spaces. More generally, it is not a countable boolean combination of open sets.

In order to overcome the mismatch between the hierarchy inherited from $\mathcal{N}$ via the representation and the class of Borel sets, one may attempt to change the definition of Borel sets. In [12] the Borel sets are redefined as the smallest class containing the open sets and the saturated compact sets, and closed under countable unions and complements. We observe here that this class is too large in the space $\mathcal{O}(\mathcal{N})$. First, if $U \subseteq \mathcal{N}$ is open then the set $\{V \in \mathcal{O}(\mathcal{N}) : U \subseteq V\}$ is compact and saturated in $\mathcal{O}(\mathcal{N})$. From this it is easy to see that the set built above is Borel in this weaker sense. However this notion of Borel sets is too loose, because compact saturated sets do not usually have a Borel pre-image. For instance, the singleton $\{\mathcal{N}\}$ is compact saturated but its pre-image under the representation is a $\utilde{\mathbf{\Pi}}^1_1$-complete set, hence is not Borel.

References

1 Antonin Callard and Mathieu Hoyrup. Descriptive complexity on non-polish spaces. In Christophe Paul and Markus Bläser, editors, *37th International Symposium on Theoretical Aspects of Computer Science, STACS 2020, March 10-13, 2020, Montpellier, France*, volume 154 of *LIPIcs*, pages 8:1–8:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.STACS.2020.8.

2 Matthew de Brecht. Quasi-Polish spaces. *Ann. Pure Appl. Logic*, 164(3):356–381, 2013. doi:10.1016/j.apal.2012.11.001.

3 Matthew de Brecht, Arno Pauly, and Matthias Schröder. Overt choice. *Computability*, ?, 2019. doi:10.3233/COM-190253.

4 Matthew de Brecht, Matthias Schröder, and Victor Selivanov. Base-complexity classifications of qcb$_0$-spaces. *Computability*, 5(1):75–102, 2016. doi:10.3233/COM-150044.

5 Ryszard Engelking. *General topology. Rev. and compl. ed.*, volume 6. Berlin: Heldermann Verlag, rev. and compl. ed. edition, 1989.

6 Martin Escardó and Reinhold Heckmann. Topologies on spaces of continuous functions. *Topology Proceedings*, 26(2):545–564, 2001-2002.

7 S. Franklin. Spaces in which sequences suffice II. *Fundamenta Mathematicae*, 61(1):51–56, 1967. URL: http://eudml.org/doc/214006.

8 Jacques Grassin. Index sets in Ershov's hierarchy. *The Journal of Symbolic Logic*, 39:97–104, March 1974. doi:10.2307/2272349.

9 Mathieu Hoyrup. Descriptive complexity on non-Polish spaces II, 2020. Preprint. URL: https://hal.inria.fr/hal-02483114.

10 Alexander S. Kechris. *Classical Descriptive Set Theory*. Springer, January 1995.

11 Shou Lin. A note on the Arens' space and sequential fan. *Topology and its Applications*, 81(3):185–196, 1997. doi:10.1016/S0166-8641(97)00031-X.

12 Tommy Norberg and Wim Vervaat. Capacities on non-Hausdorff spaces. *Probability and Lattices*, 110:133–150, 1997.

13 Arno Pauly and Matthew de Brecht. Descriptive set theory in the category of represented spaces. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*, pages 438–449. IEEE Computer Society, 2015. doi:10.1109/LICS.2015.48.

14 Yann Pequignot. A Wadge hierarchy for second countable spaces. *Arch. Math. Log.*, 54(5-6):659–683, 2015. doi:10.1007/s00153-015-0434-y.

15 Luca Motto Ros, Philipp Schlicht, and Victor L. Selivanov. Wadge-like reducibilities on arbitrary quasi-Polish spaces. *Mathematical Structures in Computer Science*, 25(8):1705–1754, 2015. doi:10.1017/S0960129513000339.

16 Matthias Schröder. *Admissible Representations for Continuous Computations.* PhD thesis, FernUniversität Hagen, 2002.

17 Matthias Schröder. Extended admissibility. *Theor. Comput. Sci.*, 284(2):519–538, 2002. doi:10.1016/S0304-3975(01)00109-8.

18 Matthias Schröder. The sequential topology on $\mathbb{N}^{\mathbb{N}^{\mathbb{N}}}$ is not regular. *Mathematical Structures in Computer Science*, 19(5):943–957, 2009. doi:10.1017/S0960129509990065.

19 Matthias Schröder. A note on closed subsets in quasi-zero-dimensional qcb-spaces. *Journal of Universal Computer Science*, 16(18):2711–2732, September 2010.

20 Victor L. Selivanov. Index sets in the hyperarithmetical hierarchy. *Siberian Mathematical Journal*, 25:474–488, 1984. doi:10.1007/BF00968988.

21 Victor L. Selivanov. Difference hierarchy in φ-spaces. *Algebra and Logic*, 43(4):238–248, July 2004. doi:10.1023/B:ALLO.0000035115.44324.5d.

22 Yoshio Tanaka. Theory of k-networks. *Q. and A. in Gen. Top.*, 12:139–164, 1994. URL: https://ci.nii.ac.jp/naid/10010236971/en/.

On Decidability of Time-Bounded Reachability in CTMDPs

Rupak Majumdar
Max-Planck Institute for Software Systems, Kaiserslautern, Germany
rupak@mpi-sws.org

Mahmoud Salamati
Max-Planck Institute for Software Systems, Kaiserslautern, Germany
msalamati@mpi-sws.org

Sadegh Soudjani
Newcastle University, Newcastle upon Tyne, UK
sadegh.soudjani@newcastle.ac.uk

Abstract

We consider the time-bounded reachability problem for continuous-time Markov decision processes. We show that the problem is decidable subject to Schanuel's conjecture. Our decision procedure relies on the structure of optimal policies and the conditional decidability (under Schanuel's conjecture) of the theory of reals extended with exponential and trigonometric functions over bounded domains. We further show that any unconditional decidability result would imply unconditional decidability of the bounded continuous Skolem problem, or equivalently, the problem of checking if an exponential polynomial has a non-tangential zero in a bounded interval. We note that the latter problems are also decidable subject to Schanuel's conjecture but finding unconditional decision procedures remain longstanding open problems.

2012 ACM Subject Classification Theory of computation → Numeric approximation algorithms; Mathematics of computing → Markov processes; Theory of computation → Verification by model checking

Keywords and phrases CTMDP, Time bounded reachability, Continuous Skolem Problem, Schanuel's Conjecture

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.133

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

Funding This research was funded in part by the Deutsche Forschungsgemeinschaft project 389792660-TRR 248 and by the European Research Council under the Grant Agreement 610150 (ERC Synergy Grant ImPACT).

Acknowledgements We thank Joël Ouaknine, James Worrell, and Joost-Pieter Katoen for discussions and pointers.

1 Introduction

Continuous-time Markov decision processes (CTMDPs) are a widely used model for continuous-time systems which exhibit both stochastic and non-deterministic choice. A CTMDP consists of a finite set of states, a finite set of actions, and for each action, a transition rate matrix that determines the rate (in an exponential distribution in continuous time) to go from one state to the next when the action is chosen. A *policy* for a CTMDP maps a timed execution path to state-dependent actions. Given a fixed policy, a CTMDP determines a stochastic process in continuous time, where the rate matrix determines the distribution of switches.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 133; pp. 133:1–133:19
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

A fundamental decision problem for CTMDPs is the *time-bounded reachability problem*, which asks, given a CTMDP $\mathcal{M}$ with a designated "**good**" state, a time bound B, and a rational vector r, whether there exists a policy that controls the Markov decision process such that the probability of reaching the good state from state s within time bound B is at least $r(s)$. The time-bounded reachability problem is at the core of model checking CTMDPs with respect to stochastic temporal logics [5] and has been extensively studied [10, 21, 28, 20, 9].

Existing papers either consider *time-abstract* policies [5, 25, 8, 28, 20] or focus on numerical approximation schemes [10, 21, 3, 13, 9, 26]. However, policies that depend on time are strictly more powerful and the *decision problem* has remained open. For the special case of continuous-time Markov chains (CTMCs), where each state has a unique action, the time-bounded reachability problem is decidable [4]. The proof uses tools from transcendental number theory, specifically, the Lindemann-Weierstrass theorem. One might expect that a similar argument might be used to show decidability for CTMDPs as well.

In this paper, we show *conditional* decidability. Our result uses, like several other conditional results on dynamical systems, Schanuel's conjecture from transcendental number theory (see, e.g., [14]). Our proof has the following ingredients. First, we use the fact that the optimal policy for the time-bounded reachability problem is a timed, *piecewise constant* function with a *finite* number of switches [19, 22, 24]. We show that each switch point of an optimal policy corresponds to a non-tangential zero of an associated linear dynamical system. Second, we use the result of Macintyre and Wilkie [16, 17] that Schanuel's conjecture implies the decidability of the real-closed field together with the exponential, sine, and cosine functions over a bounded domain. The existence of non-tangential zeros of linear dynamical systems can be encoded in this theory. Third, for each natural number $k \in \mathbb{N}$, we write a sentence in this theory whose validity implies there is an optimal strategy with exactly k switch points. By enumerating over k, we find the exact number of switches in an optimal strategy. Finally, we write another sentence in the theory that checks if the reachability probability attained by (an encoding of) the optimal policy is greater than the given bound.

We also study the related decision problem whether there is a *stationary* (i.e., time independent) optimal policy. We show that there is a "direct" conditional decision procedure for this problem based on Schanuel's conjecture and recent results on zeros of exponential polynomials [11], which avoids the result of Macintyre and Wilkie.

At the same time, we show that an *unconditional* decidability result is likely to be very difficult. We show that the bounded continuous-time Skolem problem [7, 11] reduces to checking if there is an optimal stationary policy in the time-bounded CTMDP problem. The bounded continuous Skolem problem is a long-standing open problem about linear dynamical systems [11, 7]; it asks if a linear dynamical system in continuous time has a non-tangential zero in a bounded interval. Our reduction, in essence, demonstrates that CTMDPs can "simulate" any linear dynamical system: a non-tangential zero in the dynamics corresponds to a policy switch point in the simulating CTMDP.

Our result is in the same spirit as several recent results providing conditional decision procedures, based on Schanuel's conjecture, or hardness results, based on variants of the Skolem problem, for problems on probabilistic systems. For example, Daviaud et al. [12] showed conditional decidability of subcases of the containment problem for probabilistic automata subject to the conditional decidability of the theory of real closed fields with the exponential function [18, 27]. For lower bounds, Akshay et al. [2] showed a reduction from the (unbounded, discrete) Skolem problem to reachability on discrete time Markov chains and Piribauer and Baier [23] show that the positivity problem in discrete time can be reduced into several decision problems corresponding to optimization tasks over discrete time MDPs.

In summary, we summarize our contribution as the following theorem.

▶ **Theorem 1.** *(1) The time-bounded reachability problem for CTMDPs is decidable assuming Schanuel's conjecture. (2) Whether the time-bounded reachability problem has a stationary optimal policy is decidable assuming Schanuel's conjecture. (3) The bounded continuous Skolem problem reduces to checking if the time-bounded reachability problem has a stationary optimal policy.*

2 Continuous Time Markov Decision Processes

▶ **Definition 2.** *A continuous-time Markov decision process (CTMDP) is a tuple $\mathcal{M} = (S, \mathcal{D}, \mathbf{Q})$ where*

- *$S = \{1, 2, \ldots, \mathfrak{n}\}$ is a finite set of states for some $\mathfrak{n} > 0$;*
- *a set $\mathcal{D} = \prod_{s=1}^{\mathfrak{n}} \mathcal{D}_s$ of* decision vectors*, where $\mathcal{D}_s$ is a finite set of* actions *that can be taken in state $s \in S$;*
- *$\mathbf{Q}$ is a $\mathcal{D}$-indexed family of $\mathfrak{n} \times \mathfrak{n}$* generator matrices*; we write $\mathbf{Q}^{\mathbf{d}}$ for the generator matrix corresponding to the decision vector $\mathbf{d} \in \mathcal{D}$. The entry $\mathbf{Q}^{\mathbf{d}}(s, s') \geq 0$ for $s' \neq s$ gives the* rate of transition *from state s to state s' under action $\mathbf{d}(s)$, and $\mathbf{Q}^{\mathbf{d}}(s, s')$ is independent of elements of $\mathbf{d}$ except $\mathbf{d}(s)$. The entry $\mathbf{Q}^{\mathbf{d}}(s, s) = -\sum_{s' \neq s} \mathbf{Q}^{\mathbf{d}}(s, s')$.*

A CTMDP $\mathcal{M} = (S, \mathcal{D}, \mathbf{Q})$ with $|\mathcal{D}| = 1$, i.e., when only a unique action can be taken in each state, is called a *continuous-time Markov chain* (CTMC) and is simply denoted by the tuple $(S, \mathbf{Q})$, and with abuse of notation, we also write $\mathbf{Q}$ for the unique generator matrix. The CTMDP $\mathcal{M}$ reduces to a CTMC whenever a decision vector $\mathbf{d}$ is fixed for all time on the CTMDP.

Intuitively, $\mathbf{Q}^{\mathbf{d}}(s, s') > 0$ indicates that by fixing a decision vector $\mathbf{d}$, a transition from s to s' is possible and that the timing of the transition is exponentially distributed with rate $\mathbf{Q}^{\mathbf{d}}(s, s')$. If there are several states s' such that $\mathbf{Q}^{\mathbf{d}}(s, s') > 0$, more than one transition will be possible. For each decision vector $\mathbf{d} \in \mathcal{D}$ and any $s \in S$, the total rate of taking an outgoing transition from state s when $\mathbf{d}$ is fixed is given by $E_{\mathbf{d}}(s) = \sum_{s' \neq s} \mathbf{Q}^{\mathbf{d}}(s, s')$, By fixing this decision vector $\mathbf{d}$, a transition from a state s into s' occurs within time t with probability

$$\mathbf{P}(s, s', t) = \frac{\mathbf{Q}^{\mathbf{d}}(s, s')}{E_{\mathbf{d}}(s)} . (1 - e^{-E_{\mathbf{d}}(s)t}), \quad t \geq 0.$$

Intuitively, $1 - e^{-E_{\mathbf{d}}(s)t}$ is the probability of taking an outgoing transition at s within time t (exponentially distributed with rate $E_{\mathbf{d}}(s)$) and $\mathbf{Q}^{\mathbf{d}}(s, s')/E_{\mathbf{d}}(s)$ is the probability of taking transition to s' among possible next states at s. Thus, the total probability of moving from s to s' under the decision $\mathbf{d}$ in one transition, written $\mathbf{P}_{\mathbf{d}}(s, s')$ is $\mathbf{Q}^{\mathbf{d}}(s, s')/E_{\mathbf{d}}(s)$. A state $s \in S$ is called *absorbing* if and only if $\mathbf{Q}^{\mathbf{d}}(s, s') = 0$ for all $s' \in S$ and all decision vectors $\mathbf{d} \in \mathcal{D}$. For an absorbing state, we have $E_{\mathbf{d}}(s) = 0$ for any decision vector $\mathbf{d}$ and no transitions are enabled. The initial state of a CTMDP is either fixed deterministically or selected randomly according to a probability distribution α over the set of states S.

Consider a time interval $[0, B]$ with time bound $B > 0$. Let Ω denote the set of all right-continuous step functions $f : [0, B] \to S$, i.e., there are time points $t_0 = 0 < t_1 < t_2 < \ldots < t_m = B$ such that $f(t') = f(t'')$ for all $t', t'' \in [t_i, t_{i+1})$ for all $i \in \{0, 1, \ldots, m-1\}$. Let $\mathcal{F}$ denote the sigma-algebra of the *cylinder sets*

$$\mathsf{Cyl}(s_0, I_0, \ldots, I_{m-1}, s_m) := \{f \in \Omega \,|\, \forall 0 \leq i \leq m \cdot f(t_i) = s_i \wedge i < m \Rightarrow (t_{i+1} - t_i) \in I_i\}. \quad (1)$$

for all m, $s_i \in S$ and non-empty time intervals $I_0, I_1, \ldots, I_{m-1} \subset [0, B]$.

▶ **Definition 3.** *A* policy π *is a function from* $[0,B]$ *into* $\mathcal{D}$*, which is assumed to be Lebesgue measurable. Any policy gives a decision vector* $\pi_t \in \mathcal{D}$ *at time* t *such that the action* $\pi_t(s)$ *is taken when the CTMDP is at state* s *at time* t*. The set of all such polices is denoted by* Π_B*.*

Any policy π together with an initial distribution α induces the probability space $(\Omega, \mathcal{F}, \mathbf{P}^\pi_\alpha)$. If the initial distribution is chosen deterministically as $s \in S$, we denote the probability measure by $\mathbf{P}^\pi_s$ instead of $\mathbf{P}^\pi_\alpha$.

A policy $\pi : [0,B] \to \mathcal{D}$ is *piecewise constant* if there exist a number $m \in \mathbb{N}$ and time points $t_0 = 0 < t_1 < t_2 < \ldots < t_m = B$ such that $\pi_{t'} = \pi_{t''}$ for all $t', t'' \in (t_i, t_{i+1}]$ and all $i \in \{0, 1, \ldots, m-1\}$. The policy is *stationary* if $m = 1$. We denote the class of stationary policies by $\Pi_{\mathtt{st}}$; observe that a stationary policy is given by a fixed decision vector, so $\Pi_{\mathtt{st}}$ is isomorphic with the set of decision vectors $\mathcal{D}$. In particular, it is a finite set.

▶ Remark 4. The policies in Def. 3 are called *timed positional* policies since the action is selected deterministically as a function of time and the state of the CTMDP at that time. A stationary policy is only positional since the selected action is independent of time.

▶ **Problem 1.** *Consider a CTMDP* $\mathcal{M} = (\{1, \ldots, n\} \uplus \{\mathbf{good}\}, \mathcal{D}, \mathbf{Q})$ *with a distinguished absorbing state named* **good** *and a time bound* $B > 0$*. Define the event*

$$\boldsymbol{reach} := \cup\{f \in \Omega \mid f(t) = \mathbf{good} \text{ for some } t \in [0,B]\}. \tag{2}$$

The time-bounded reachability problem *asks if for a rational vector* $r \in [0,1]^n$*, we have*

$$\sup_{\pi \in \Pi_B} \mathbf{P}^\pi_s(\boldsymbol{reach}) > r(s), \qquad \text{for all } s \in \{1, \ldots, n\}.$$

The event **reach** defined in (2) is written as a union of an uncountable number of functions but it is measurable in the probability space $(\Omega, \mathcal{F}, \mathbf{P}^\pi_\alpha)$ for any α. Since the state space is finite, **reach** can be written as a countable union of cylinder sets in the form of (1) by taking all the time intervals to be $[0,B]$ and enumerating over all possible sequence of states (which is countable) [6].

A policy $\pi^* \in \Pi_B$ is *optimal* if $P^{\pi^*}_s(\mathbf{reach}) = \sup_{\pi \in \Pi_B} \mathbf{P}^\pi_s(\mathbf{reach})$. Note that there are more general classes of policies that may depend also on the history of the states in the previous time points and which map the history to a distribution over $\mathcal{D}$. It is shown that piecewise constant timed positional policies are sufficient for the optimal reachability probability [19, 22, 24]. That is, if there is an optimal policy from the larger class of policies, there is already one from the class of piecewise constant, timed, positional policies.

A closely related problem is the existence of *stationary* optimal policies; here, it is possible that the optimal stationary policy performs strictly worse than an optimal policy.

▶ **Problem 2.** *Consider a CTMDP* $\mathcal{M} = (\{1, \ldots, n\} \uplus \{\mathbf{good}\}, \mathcal{D}, \mathbf{Q})$ *and a time bound* $B > 0$*. Decide whether there is an optimal policy* π^* *that is stationary, namely*

$$\exists \pi^* \in \Pi_{\mathtt{st}} \text{ s.t. } \sup_{\pi \in \Pi_B} \mathbf{P}^\pi_s(\boldsymbol{reach}) = \mathbf{P}^{\pi^*}_s(\boldsymbol{reach}), \qquad \text{for all } s \in \{1, \ldots, n\}.$$

In the following, we shall assume that the CTMDPs and all bounds in the above decision problems are given using rational numbers. That is, rates of transitions in each generator matrix is a rational number, and the time bound B is a rational number.

▶ **Theorem 5** ([10, 19])**.** *A policy* $\pi \in \Pi_B$ *is optimal if* $\mathbf{d}_t$*, the decision vector taken by* π *at time* $B - t$*, maximizes for almost all* $t \in [0,B]$

$$\max_{\mathbf{d}_t}(\mathbf{Q}^{\mathbf{d}_t} W^\pi_t) \text{ with } \frac{d}{dt} W^\pi_t = \mathbf{Q}^{\mathbf{d}_t} W^\pi_t, \tag{3}$$

with the initial condition $W^\pi_0(\mathbf{good}) = 1$ *and* $W^\pi_0(s) = 0$ *for all* $s \in \{1, 2, \ldots, n\}$*. There exists a piecewise constant policy* π *that maximizes the equations.*

The maximization in Equation (3) above is performed element-wise. Equation (3) should be solved forward in time to construct the policy π backward in time due to the definition $\mathbf{d}_t = \pi_{B-t}$. One can alternatively write down (3) directly backward in time based on π_t.

The proof of Theorem 5 is constructive [10, 19] and is based on the following sets for any vector W:

$$\begin{aligned} &\mathcal{F}_1(W) = \{\mathbf{d} \in \mathcal{D} \,|\, \mathbf{d} \text{ maximizes } \mathbf{Q}^{\mathbf{d}} W\},\\ &\mathcal{F}_2(W) = \{\mathbf{d} \in \mathcal{F}_1(W) \,|\, \mathbf{d} \text{ maximizes } [\mathbf{Q}^{\mathbf{d}}]^2 W\},\\ &\dots\\ &\mathcal{F}_j(W) = \{\mathbf{d} \in \mathcal{F}_{j-1}(W) \,|\, \mathbf{d} \text{ maximizes } [\mathbf{Q}^{\mathbf{d}}]^j W\}. \end{aligned} \tag{4}$$

The sets $\mathcal{F}_j(W)$ form a sequence of decreasing sets such that $\mathcal{F}_1(W) \supseteq \mathcal{F}_2(W) \supseteq \ldots \supseteq \mathcal{F}_{n+2}(W) = \mathcal{F}_{n+k}(W)$ for all $k > 2$. An optimal piecewise constant policy is the one that satisfies the condition $\mathbf{d}_t \in \mathcal{F}_{n+2}(W_t^\pi)$ for all $t \in [0, B]$. Note that if $\mathcal{F}_j(W_t^\pi)$ has only one element for some j, $\mathcal{F}_k(W_t^\pi) = \mathcal{F}_j(W_t^\pi)$ for all $k \geq j$ and that element is the optimal decision vector. The next proposition shows that when $\mathcal{F}_{n+2}(W_t^\pi)$ has more than one element, we can pick any one (and in fact, switch between them arbitrarily).

▶ **Proposition 6.** *Let π be an optimal policy satisfying Equation (3). Take any t^* such that $\mathcal{F}_{n+2}(W_{t^*}^\pi) \neq \lim_{t\uparrow t^*} \mathcal{F}_{n+2}(W_t^\pi)$. If $\mathcal{F}_{n+2}(W_{t^*}^\pi) = \{\mathbf{d}^1, \mathbf{d}^2, \ldots, \mathbf{d}^p\}$ for some $p > 1$ and*

$$\Delta_i := \sup\{\delta > 0 \,|\, \mathbf{d}^i \in \mathcal{F}_{n+2}(W_t^\pi) \text{ for all } t \in [t^*, t^* + \delta)\}, \quad \forall i \in \{1, 2, \ldots, p\}$$

Then, $\Delta_1 = \Delta_2 = \cdots = \Delta_p$.
Suppose there are points δ_1, δ_2 such that $t^* \leq \delta_1 < \delta_2 < t^* + \Delta_1$ and for all $t \in [\delta_1, \delta_2)$, we have $\pi_{B-t} = \mathbf{d}$ for some $\mathbf{d} \in \mathcal{F}_{n+1}(W_{t^*}^\pi)$. If π' is a policy that agrees with π on $[0, \delta_1)$ but for all $t \in [\delta_1, \delta_2)$, we have $\pi'_{B-t} = \mathbf{d}'$ for some $\mathbf{d}' \in \mathcal{F}_{n+1}(W_{t^*}^\pi) \setminus \{\mathbf{d}\}$, then π' also satisfies Equation (3) for almost all $t \in [0, \delta_2)$.

Proof. Since $\mathcal{F}_{n+2}(W_{t^*}^\pi) = \mathcal{F}_{n+k}(W_{t^*}^\pi)$ for all $k > 2$, for any $\mathbf{d}^i$ and $\mathbf{d}^j$ belonging to the set $\mathcal{F}_{n+2}(W_{t^*}^\pi)$, we have $[\mathbf{Q}^{\mathbf{d}^i}]^l W_{t^*}^\pi = [\mathbf{Q}^{\mathbf{d}^j}]^l W_{t^*}^\pi$ for all $l \geq 0$. Pick $\delta > 0$ sufficiently small such that $\{\mathbf{d}^1, \mathbf{d}^2, \ldots, \mathbf{d}^p\} \subseteq \mathcal{F}_{n+2}(W_t^\pi)$ for all $t \in [t^*, t^* + \delta)$. If the policy π selects $\mathbf{d}^i$ for all $t \in [t^*, t^* + \delta)$, we can write

$$W_t^\pi = e^{[\mathbf{Q}^{\mathbf{d}^i}](t-t^*)} W_{t^*}^\pi \text{ for } t \in [t^*, t^* + \delta),$$

where $e^\Gamma := \sum_{k=0}^{\infty} \frac{1}{k!}\Gamma^k$ denotes the exponential of a matrix Γ. Therefore, using the fact that $[\mathbf{Q}^{\mathbf{d}^i}]^l W_{t^*}^\pi = [\mathbf{Q}^{\mathbf{d}^j}]^l W_{t^*}^\pi$ for all $l \geq 0$ we have

$$e^{[\mathbf{Q}^{\mathbf{d}^i}](t-t^*)} W_{t^*}^\pi = e^{[\mathbf{Q}^{\mathbf{d}^j}](t-t^*)} W_{t^*}^\pi, \quad \forall\, t \geq t^*. \tag{5}$$

Similarly, we have

$$[\mathbf{Q}^{\mathbf{d}^i}]^l e^{[\mathbf{Q}^{\mathbf{d}^i}]\Delta} W_{t^*}^\pi = [\mathbf{Q}^{\mathbf{d}^j}]^l e^{[\mathbf{Q}^{\mathbf{d}^j}]\Delta} W_{t^*}^\pi, \quad \forall\, l \geq 0 \text{ and } \Delta \geq 0. \tag{6}$$

Now take any $i = \arg\min_j \Delta_j$, thus $\Delta_i \leq \Delta_j$ for all j. Also take $\mathbf{d}' \in \mathcal{F}_{n+2}(W_{t^*+\Delta_i}^\pi)$ and $\mathbf{d}' \neq \mathbf{d}^i$ (this is possible due to the definition of Δ_i). Denote by h the smallest integer for which $1 \leq h \leq n+2$ and

$$[\mathbf{Q}^{\mathbf{d}'}]^h W_{t^*+\Delta_i}^\pi > [\mathbf{Q}^{\mathbf{d}^i}]^h W_{t^*+\Delta_i}^\pi \Rightarrow [\mathbf{Q}^{\mathbf{d}'}]^h e^{[\mathbf{Q}^{\mathbf{d}^i}]\Delta_i} W_{t^*}^\pi > [\mathbf{Q}^{\mathbf{d}^i}]^h e^{[\mathbf{Q}^{\mathbf{d}^i}]\Delta_i} W_{t^*}^\pi.$$

Combining the above expression with Equation (6), we get

$$[\mathbf{Q}^{\mathbf{d}'}]^h e^{[\mathbf{Q}^{\mathbf{d}^i}]\Delta_i} W_{t^*}^\pi > [\mathbf{Q}^{\mathbf{d}^j}]^h e^{[\mathbf{Q}^{\mathbf{d}^j}]\Delta_i} W_{t^*}^\pi \;\Rightarrow\; [\mathbf{Q}^{\mathbf{d}'}]^h W_{t^*+\Delta_i}^\pi > [\mathbf{Q}^{\mathbf{d}^j}]^h W_{t^*+\Delta_i}^\pi,$$

which implies that $\Delta_j \leq \Delta_i$ for any j. The particular selection of i results in $\Delta_j = \Delta_i$ for all i, j. The second part of the proposition is obtained by setting $\Delta = (\delta_2 - \delta_1)$ in Equation (6) and using the definition of the exponential of a matrix. ◀

The above proposition highlights the fact that whenever $\mathcal{F}_{n+2}(W_t^\pi)$ contains more than one decision vector over a time interval, one can construct infinitely many optimal policies by arbitrarily switching between such decision vectors. In the rest of this paper, we restrict our attention to optimal policies that take only mandatory switches: the optimal policy will take an element of $\mathcal{F}_{n+2}(W_t^\pi)$ as long as possible. This does not influence Problems 1 and 2.

The major challenge in the computation of the optimal policy, thus answering the reachability problem, is the computation of the largest time $t \in [0, B)$ such that $\mathcal{F}_{n+2}(W_t^\pi) \neq \mathcal{F}_{n+2}(W_{t^-}^\pi)$, where $W_{t^-}^\pi$ denotes the value of $W_{t-\delta}^\pi$ with δ converging to zero from the right. Suppose a decision vector $\mathbf{d}_0 \in \mathcal{F}_{n+2}(W_0^\pi)$ is selected. The optimal policy will change at the following time point:

$$t'' := \sup\{t \mid \mathbf{d}_0 \in \mathcal{F}_{n+2}(W_{t'}^\pi) \text{ for all } t' \in [0, t)\}.$$

3 Conditional Decidability of Problems 1 and 2

3.1 Schanuel's Conjecture and its Implications

Our decidability results will assume Schanuel's Conjecture for the complex numbers, a unifying conjecture in transcendental number theory (see, e.g., [14]). Recall that a *transcendence basis* of a field extension L/K is a subset $S \subseteq L$ such that S is algebraically independent over K and L is algebraic over $K(S)$. The *transcendence degree* of L/K is the (unique) cardinality of some basis.

▶ **Conjecture 7** (**Schanuel's Conjecture (SC)**). *Let $a_1, \ldots, a_n$ be complex numbers that are linearly independent over rational numbers $\mathbb{Q}$. Then the field $\mathbb{Q}(a_1, \ldots, a_n, e^{a_1}, \ldots, e^{a_n})$ has transcendence degree at least n over $\mathbb{Q}$.*

An important consequence of Schanuel's conjecture is that the theory of reals $(\mathbb{R},0,1,+,\cdot,\leq)$ remains decidable when extended with the exponential and trigonometric functions over bounded domains.[1]

▶ **Theorem 8** (Macintyre and Wilkie (see [16, 17])). *Assume **SC**. For any $n \in \mathbb{N}$, the theory $\mathbb{R}_{\mathsf{MW}} := (\mathbb{R}, \exp \restriction [0, n], \sin \restriction [0, n], \cos \restriction [0, n])$ is decidable.*

Our main result will show that Problems 1 and 2 can be decided based on Theorem 8. In fact, Problem 2 can be decided directly from Schanuel's conjecture and recent results on exponential polynomials [11].

▶ **Theorem 9** (Main Result). *Assume **SC**. Then Problems 1 and 2 are decidable.*

In contrast, solving the time-bounded reachability problem for *stationary* policies is decidable unconditionally. This is because fixing a stationary policy reduces the time-bounded reachability problem to one on CTMCs, and one can use the decision procedure from [4].

[1] We note that while the result is claimed in several papers [16, 17], a complete proof of this result has never been published. Thus, it would be nice to have a "direct" proof of our main theorem (Theorem 1) starting with Schanuel's conjecture. We do not know such a proof.

3.2 Non-tangential Zeros

Recall that the solution to a first-order linear ODE of dimension n:

$$\frac{d}{dt}X_t = AX_t, \quad z_t = CX_t$$

with real matrices A and C and real initial condition $X_0 \in \mathbb{R}^n$, can be written as $z_t = Ce^{At}X_0$ where e^Γ denotes the exponential of a square matrix Γ, and defined as the infinite sum $e^\Gamma := \sum_{k=0}^{\infty} \frac{1}{k!}\Gamma^k$ that is guaranteed to converge for any matrix Γ. The function can be expressed as an exponential polynomial $z_t = \sum_{j=1}^{k} P_t(j)e^{\lambda_j t}$, where $\lambda_1, \ldots, \lambda_k$ are the distinct (real or complex) eigenvalues of A. Each function $P_t(j)$ is a polynomial function of t possibly with complex coefficients and has a degree one less than the multiplicity of the eigenvalue λ_j. Since the eigenvalues come in conjugate pairs, we can write the real-valued function z as

$$z_t = \sum_{j=1}^{k} e^{a_j t} \sum_{l=0}^{m_j-1} c_{j,l}t^l \cos(b_j t + \varphi_{j,l}), \tag{7}$$

where the eigenvalues are $a_j \pm \mathbf{i}b_j$ with multiplicity m_j. If A, X_0, and C are over the rational numbers, then a_j, b_j, $c_{j,l}$ are real algebraic and $\varphi_{j,l}$ is such that $e^{\mathbf{i}\varphi_{j,l}}$ is algebraic for all j and l. We can symbolically compute derivatives of z which also become functions with a similar closed-form as in (7).

▶ **Definition 10.** *The function z_t has a* zero *at $t = t^*$ if $z_{t^*} = 0$. The zero is said to be* non-tangential *if there is an $\varepsilon > 0$ such that $z_{t_1}z_{t_2} < 0$ for all $t_1 \in (t^* - \varepsilon, t^*)$ and all $t_2 \in (t^*, t^* + \varepsilon)$. The zero is called* tangential *if there is an $\varepsilon > 0$ such that $z_{t_1}z_{t_2} > 0$ for all $t_1 \in (t^* - \varepsilon, t^*)$ and all $t_2 \in (t^*, t^* + \varepsilon)$.*

Note that there are functions with zeros that are neither tangential nor non-tangential. Consider the function $z_t = t\sin\left(\frac{1}{t}\right)$ for $t \neq 0$ and $z_0 = 0$. The function does not satisfy the conditions of being tangential or non-tangential. For any $\varepsilon > 0$, there are $t_1 \in (-\varepsilon, 0)$ and $t_2 \in (0, \varepsilon)$, such that $z_{t_1}z_{t_2} = t_1t_2 \sin\left(\frac{1}{t_1}\right)\sin\left(\frac{1}{t_2}\right)$ is positive. There are also t_1 and t_2 in the respective intervals that make $z_{t_1}z_{t_2}$ negative. In this paper, we only work with functions of the form (7) that are analytic thus infinitely differentiable. Therefore, the first non-zero derivative of z_t at t^* will decide if t^* is tangential or not.

▶ **Proposition 11.** *For any function z_t of the form (7) such that $z_{t^*} = 0$ and $z \not\equiv 0$, there is a k_0 such that $\frac{d^k}{dt^k}z_t\big|_{t=t^*} = 0$ for all $k < k_0$ and $\frac{d^{k_0}}{dt^{k_0}}z_t\big|_{t=t^*} \neq 0$. Moreover, t^* is tangential if k_0 is an even number and is non-tangential if k_0 is an odd number.*

Proof. The proof is based on the Taylor series of z_t at $t = t^*$. Take k_0 the order of the first non-zero derivative of z_t at $t = t^*$. This k_0 always exists since otherwise $z \equiv 0$. The Taylor series of z_t will be

$$z_t = \sum_{k=k_0}^{\infty} \frac{(t-t^*)^k}{k!}\frac{d^k}{dt^k}z_t\big|_{t=t^*} = (t-t^*)^{k_0}\frac{d^{k_0}}{dt^{k_0}}z_t\big|_{t=t^*}\sum_{k=0}^{\infty}\alpha_k(t-t^*)^k, \tag{8}$$

for some $\{\alpha_0, \alpha_1, \ldots\}$ with $\alpha_0 = \frac{1}{k_0!}$. Define the function g by $g_t := \frac{z_t}{(t-t^*)^{k_0}}$ for $t \neq t^*$ and $g_{t^*} := \frac{1}{k_0!}\frac{d^{k_0}}{dt^{k_0}}z_t\big|_{t=t^*}$. Using (8), we get that g is continuous at t^* with $g_{t^*} \neq 0$. Therefore, there is an interval $(t^* - \varepsilon, t^* + \varepsilon)$ over which the function has the same sign as g_{t^*}. For all $t_1 \in (t^* - \varepsilon, t^*)$ and $t_2 \in (t^*, t^* + \varepsilon)$

$$
\begin{aligned}
&g_{t_1} g_{t^*} > 0 \Rightarrow \frac{z_{t_1}}{(t_1 - t^*)^{k_0}} g_{t^*} > 0 \Rightarrow (-1)^{k_0} z_{t_1} g_{t^*} > 0 \\
&g_{t_2} g_{t^*} > 0 \Rightarrow \frac{z_{t_2}}{(t_2 - t^*)^{k_0}} g_{t^*} > 0 \Rightarrow z_{t_2} g_{t^*} > 0 \\
&\Rightarrow (-1)^{k_0} z_{t_1} g_{t^*} z_{t_2} g_{t^*} > 0 \Rightarrow (-1)^{k_0} z_{t_1} z_{t_2} > 0.
\end{aligned}
$$

This means $z_{t_1} z_{t_2} > 0$ for even k_0 and t^* becomes tangential, and $z_{t_1} z_{t_2} < 0$ for odd k_0 and t^* becomes non-tangential. ◀

For any function $z_t = Ce^{At}X_0$, the predicate $\mathsf{NonTangentialZero}(z, l, u)$ stating the existence of a non-tangential zero in an interval (l, u) is expressible in $\mathbb{R}_{\mathsf{MW}}$:

$$\exists t^* \,.\, l < t^* < u \wedge z_{t^*} = 0 \wedge [\exists \varepsilon > 0 \,.\, \forall t_1 \in (t^* - \varepsilon, 0), t_2 \in (0, t^* + \varepsilon) \,.\, z_{t_1} z_{t_2} < 0]$$

3.3 Switch Points are Non-Tangential Zeroes

Given a CTMDP $\mathcal{M}$ and a piecewise constant optimal policy $\pi : [0, B] \to \mathcal{D}$ for the time-bounded reachability problem, a *switch point* t^* is a point of discontinuity of π. Consider a switch point t^* such that the optimal policy takes the decision vector $\mathbf{d}$ in the time interval $(t^* - \varepsilon)$ and then switches to another decision vector $\mathbf{d}'$ at time t^* for some $\varepsilon > 0$:

$$
\begin{aligned}
&\mathbf{d} \in \mathcal{F}_{n+2}(W_t^\pi) \text{ and } \mathbf{d}' \notin \mathcal{F}_{n+2}(W_t^\pi) \quad \forall t \in (t^* - \varepsilon, t^*), \\
&\mathbf{d} \notin \mathcal{F}_{n+2}(W_t^\pi) \text{ and } \mathbf{d}' \in \mathcal{F}_{n+2}(W_t^\pi) \quad \forall t \in (t^*, t^* + \varepsilon).
\end{aligned}
$$

Consider a (not necessarily unique) state $s \in S$ with actions $a, b \in \mathcal{D}_s$ such that $a \neq b$ and $\mathbf{d}(s) = a$, $\mathbf{d}'(s) = b$. Define the following set of first-order ODEs

$$\Sigma : \begin{cases} \frac{d}{dt} W_t^\pi = \mathbf{Q^d} W_t^\pi \\ z_t = (q^a - q^b) W_t^\pi \end{cases} \tag{9}$$

for $t \in (t^* - \varepsilon, t^* + \varepsilon)$, where q^a and q^b denote the s^{th} row of the matrices $\mathbf{Q^d}$ and $\mathbf{Q^{d'}}$, respectively. The optimal decision vector on an interval before t^* is $\mathbf{d}$, thus for all $t \in (t^* - \varepsilon, t^*)$,

$$\mathbf{d} \in \mathcal{F}_1(W_t^\pi) \Rightarrow \mathbf{Q^d} W_t^\pi \geq \mathbf{Q^{d'}} W_t^\pi \Rightarrow (\mathbf{Q^d} - \mathbf{Q^{d'}}) W_t^\pi \geq 0 \Rightarrow (q^a - q^b) W_t^\pi \geq 0 \Rightarrow z_t \geq 0.$$

The next lemma states that the switch point t^* corresponds to a non-tangential zero for z_t.

▶ **Lemma 12.** *Let π be an optimal piecewise constant policy for the time-bounded reachability problem with bound B. Suppose $\pi(B - t) = \mathbf{d}_t$ for all $t \in [0, B]$. Suppose that for a time point t^*, $\mathbf{d} \in \mathcal{D}$ is an optimal decision before t^* and $\mathbf{d}' \neq \mathbf{d}$ is optimal right after t^*. There is an ε such that for any $s \in S$ with $\mathbf{d}(s) \neq \mathbf{d}'(s)$, $z_t < 0$ for all $t \in (t^*, t^* + \varepsilon)$ with z_t defined in* (9)*.*

Proof. Take k_0 to be the smallest index $k \leq n$ with $\mathbf{d} \notin \mathcal{F}_{k+1}(W_{t^*}^\pi)$ and $\mathbf{d}' \in \mathcal{F}_{k+1}(W_{t^*}^\pi)$. Since $\mathbf{d}'$ is optimal at t^*, we have $\mathbf{d}, \mathbf{d}' \in \mathcal{F}_{k+1}(W_{t^*}^\pi)$ for all $k < k_0$. We show inductively that

$$[\mathbf{Q^d}]^{k+1} W_{t^*}^\pi = [\mathbf{Q^{d'}}]^{k+1} W_{t^*}^\pi \text{ and } \frac{d^k}{dt^k} z_{t^*} = 0 \text{ for all } 0 \leq k < k_0. \tag{10}$$

The claim is true for $k = 0$:

$$\begin{aligned}
&\mathbf{d}, \mathbf{d}' \in \mathcal{F}_1(W^\pi_{t^*}) \Rightarrow \mathbf{Q}^{\mathbf{d}} W^\pi_{t^*} = \mathbf{Q}^{\mathbf{d}'} W^\pi_{t^*}\\
&\Rightarrow (\mathbf{Q}^{\mathbf{d}} - \mathbf{Q}^{\mathbf{d}'}) W^\pi_{t^*} = \begin{bmatrix} \dots \\ q^a - q^b \\ \dots \end{bmatrix} W^\pi_{t^*} = 0 \Rightarrow (q^a - q^b) W^\pi_{t^*} = 0 \Rightarrow z_{t^*} = 0.
\end{aligned}$$

Now suppose (10) holds for $(k-1)$ with $k < k_0$. Then

$$\begin{aligned}
&\mathbf{d}, \mathbf{d}' \in \mathcal{F}_{k+1}(W^\pi_{t^*}) \Rightarrow [\mathbf{Q}^{\mathbf{d}}]^{k+1} W^\pi_{t^*} = [\mathbf{Q}^{\mathbf{d}'}]^{k+1} W^\pi_{t^*}\\
&\Rightarrow \mathbf{Q}^{\mathbf{d}} [\mathbf{Q}^{\mathbf{d}}]^{k} W^\pi_{t^*} = \mathbf{Q}^{\mathbf{d}'} [\mathbf{Q}^{\mathbf{d}'}]^{k} W^\pi_{t^*} \Rightarrow^{(*)} \mathbf{Q}^{\mathbf{d}} [\mathbf{Q}^{\mathbf{d}}]^{k} W^\pi_{t^*} = \mathbf{Q}^{\mathbf{d}'} [\mathbf{Q}^{\mathbf{d}}]^{k} W^\pi_{t^*}\\
&\Rightarrow [\mathbf{Q}^{\mathbf{d}} - \mathbf{Q}^{\mathbf{d}'}] [\mathbf{Q}^{\mathbf{d}}]^{k} W^\pi_{t^*} = 0 \Rightarrow^{(**)} [\mathbf{Q}^{\mathbf{d}} - \mathbf{Q}^{\mathbf{d}'}] \frac{d^k}{dt^k} X_{t^*} = 0\\
&\Rightarrow (q^a - q^b) \frac{d^k}{dt^k} X_{t^*} = 0 \Rightarrow \frac{d^k}{dt^k} z_{t^*} = 0,
\end{aligned}$$

where $(*)$ holds due to the induction assumption and $(**)$ is true due to the differential equation (9). Finally, we show that $\frac{d^{k_0}}{dt^{k_0}} z_{t^*} < 0$.

$$\begin{aligned}
&\mathbf{d} \notin \mathcal{F}_{k_0+1}(W^\pi_{t^*}) \text{ and } \mathbf{d}' \in \mathcal{F}_{k_0+1}(W^\pi_{t^*}) \Rightarrow [\mathbf{Q}^{\mathbf{d}}]^{k_0+1} W^\pi_{t^*} < [\mathbf{Q}^{\mathbf{d}'}]^{k_0+1} W^\pi_{t^*}\\
&\Rightarrow \mathbf{Q}^{\mathbf{d}} [\mathbf{Q}^{\mathbf{d}}]^{k_0} W^\pi_{t^*} < \mathbf{Q}^{\mathbf{d}'} [\mathbf{Q}^{\mathbf{d}'}]^{k_0} W^\pi_{t^*} \Rightarrow^{(\imath)} \mathbf{Q}^{\mathbf{d}} [\mathbf{Q}^{\mathbf{d}}]^{k_0} W^\pi_{t^*} < \mathbf{Q}^{\mathbf{d}'} [\mathbf{Q}^{\mathbf{d}}]^{k_0} W^\pi_{t^*}\\
&\Rightarrow [\mathbf{Q}^{\mathbf{d}} - \mathbf{Q}^{\mathbf{d}'}] \frac{d^{k_0}}{dt^{k_0}} W^\pi_{t^*} < 0 \Rightarrow (q^a - q^b) \frac{d^{k_0}}{dt^{k_0}} W^\pi_{t^*} < 0 \Rightarrow \frac{d^{k_0}}{dt^{k_0}} z_{t^*} < 0,
\end{aligned}$$

where $(\imath)$ holds due to (10) for $k_0 - 1$.

Since $z_{t^*} = 0$, we can select ε such that $z_t > 0$ for all $t \in (t^* - \varepsilon, t^*)$. Using Taylor expansion (8) and the facts that $\frac{d^{k_0}}{dt^{k_0}} z_{t^*} < 0$ and $z_t > 0$ for $t \in (t^* - \varepsilon, t^*)$, we have that k_0 must be an odd number, which means t^* is non-tangential by Prop. 11. The function z_t changes sign from positive to negative at t^*. ◀

3.4 Conditional Decidability

The decision procedure for Problem 1 is as follows. Fix a CTMDP $\mathcal{M} = (\{1, \dots, n\} \uplus \{\mathbf{good}\}, \mathcal{D}, \mathbf{Q})$ and a bound B. We inductively construct a piecewise constant optimal policy, going forward in time. To begin, we set the initial decision vector to $\mathbf{d}^1$, where $\mathbf{d}^1$ is selected such that $\mathbf{d}^1 \in \mathcal{F}_{n+2}(W^\pi_0)$ (Equation (4)) with W^π_0 set to the indicator vector that is 1 at the **good** state and 0 in other states.

Note that in general $\mathcal{F}_{n+2}(W^\pi_t)$ in (4) may have finitely many elements and the choice of optimal decision at time t, $\mathbf{d}_t \in \mathcal{F}_{n+2}(W^\pi_t)$ is not unique. Based on results of Proposition 6, any arbitrary element of $\mathcal{F}_{n+2}(W^\pi_t)$ can be chosen; but, we do not alter this choice until the picked decision vector does not belong to $\mathcal{F}_{n+2}(W^\pi_t)$ anymore. We know that there is a piecewise constant optimal policy π with finitely many switches obtained from the charactrization in Theorem 5. Denote the (unknown) number of switches by $k \in \mathbb{N}$.

We find k as follows. We inductively check the existence of a sequence of decision vectors $\mathbf{d}^1, \dots, \mathbf{d}^k$ and time points $t_1, \dots, t_{k-1}$ such that the optimal policy (given a lexicographical order on $\mathcal{D}$) switches from $\mathbf{d}^i$ to $\mathbf{d}^{i+1}$ at time t_i but does not have any switch between the time points. Then, we check if the optimal policy makes at least one additional switch point in the interval (t_k, B). The check reduces the question to a number of satisfiability questions in $\mathbb{R}_{\mathsf{MW}}$. If we find an additional switch, we know that the optimal strategy has at least $k+1$ switches and continue to check if there are further switch points. If not, we know that the optimal policy has k switch points.

We need some notation. A *prefix* $\sigma_k = (\mathbf{d}^1, t_1, \mathbf{d}^2, t_2, \ldots, t_{k-1}, \mathbf{d}^k) \in (\mathcal{D} \times (0, B))^* \times \mathcal{D}$ is a finite sequence of decision vectors from $\mathcal{D}$ and strictly increasing time points $0 < t_1 < t_2 < \ldots < t_{k-1} < B$ such that $\mathbf{d}^i \neq \mathbf{d}^{i+1}$ for $i \in \{1, \ldots, k-1\}$. Intuitively, it represents the prefix of a piecewise constant policy with the first $k-1$ switches. For two decision vectors $\mathbf{d}, \mathbf{d}'$, let $\Delta(\mathbf{d}, \mathbf{d}') := \{s \mid \mathbf{d}(s) \neq \mathbf{d}'(s)\}$ be the states at which the actions suggested by the decision vectors differ. For a decision vector $\mathbf{d}$, let $\mathbf{d}[s \mapsto b]$ denote the decision vector that maps state s to action b but agrees with $\mathbf{d}$ otherwise.

For a prefix $\sigma_k = (\mathbf{d}^1, t_1, \mathbf{d}^2, t_2, \ldots, t_{k-1}, \mathbf{d}^k)$, a state $s \in S$, and an action $b \in \mathcal{D}_s$, define

$$y_t^{s,b}(\sigma_k) = \mathbf{u}^T(s)([\mathbf{Q}^{\mathbf{d}^k}] - [\mathbf{Q}^{\mathbf{d}^k[s \mapsto b]}])e^{[\mathbf{Q}^{\mathbf{d}^k}](t-t_{k-1})}e^{[\mathbf{Q}^{\mathbf{d}^{k-1}}](t_{k-1}-t_{k-2})} \ldots e^{[\mathbf{Q}^{\mathbf{d}^1}]t_1}\mathbf{u}(\mathbf{good}),$$

where $\mathbf{u}(s)$ is a vector of dimension $n+1$ that assigns one to s and zero to every other entry. Observe that $y_t^{s,b}(\sigma_k)$ is a solution of a set of linear ODEs similar to z_t in Equation (9):

$$\begin{cases} \frac{d}{dt}W_t = [\mathbf{Q}^{\mathbf{d}^k}]W_t \\ y_t^{s,b}(\sigma_k) = \mathbf{u}^T(s)([\mathbf{Q}^{\mathbf{d}^k}] - [\mathbf{Q}^{\mathbf{d}^k[s \mapsto b]}])W_t, \end{cases} \tag{11}$$

with the condition $W_{t_{k-1}} = e^{[\mathbf{Q}^{\mathbf{d}^{k-1}}](t_{k-1}-t_{k-2})} \ldots e^{[\mathbf{Q}^{\mathbf{d}^1}]t_1}\mathbf{u}(\mathbf{good})$.

We shall use (variants of) the predicate $\mathsf{NonTangentialZero}(y^{\cdot,\cdot}, t_1, t_2)$, but write the predicates informally for readability. We need two additional predicates $\mathsf{Switch}(\sigma_k, t^*, \mathbf{d}')$ and $\mathsf{NoSwitch}(\sigma_{k+1})$. The predicate Switch states that, given a prefix σ_k, the first switch from $\mathbf{d}^k$ to a new decision vector $\mathbf{d}'$ occurs at time point $t^* > t_{k-1}$. This new switch requires three conditions. First, there is a simultaneous non-tangential zero at t^* for all dynamical systems of the form (11) associated with $y_t^{s,\mathbf{d}'(s)}(\sigma_k)$, $s \in \Delta(\mathbf{d}^k, \mathbf{d}')$. Second, t^* is the first time after t_{k-1} that any of the dynamical systems have a non-tangential zero. Finally, none of the states in $S \setminus \Delta(\mathbf{d}^k, \mathbf{d}')$ whose action remains the same before and after the switch, have a non-tangential zero in $(t_{k-1}, t^*]$ (up to and including t^*):

$$\mathsf{Switch}(\underbrace{(\mathbf{d}^1, t_1, \ldots, t_{k-1}, \mathbf{d}^k)}_{\sigma_k}, t^*, \mathbf{d}') \equiv$$

$$0 < t_1 < \ldots < t_{k-1} < B \wedge (B > t^* > t_{k-1}) \wedge (\Delta(\mathbf{d}^k, \mathbf{d}') \neq \emptyset) \wedge$$

$$\bigwedge_{s \in \Delta(\mathbf{d}^k, \mathbf{d}')} \left(\begin{array}{c} \text{“}y_t^{s,\mathbf{d}'(s)}(\sigma_k) \text{ has a non-tangential zero at } t^*\text{”} \wedge \\ \text{“}y_t^{s,\mathbf{d}'(s)}(\sigma_k) \text{ has no non-tangential zero in } (t_{k-1}, t^*)\text{”} \end{array} \right) \wedge$$

$$\bigwedge_{s \in S \setminus \Delta(\mathbf{d}^k, \mathbf{d}')} \text{“}y_t^{s,\mathbf{d}'(s)}(\sigma_k) \text{ has no non-tangential zero in } (t_{k-1}, t^*]\text{”}$$

The predicate $\mathsf{NoSwitch}(\sigma_{k+1})$ states that, given a prefix σ_{k+1}, the last decision vector $\mathbf{d}^{k+1}$ of (σ_{k+1}) stays optimal and does not switch to another decision vector within the interval (t_k, B). This is equivalent to stating that none of the dynamical systems of the from (11) associated with $y_t^{s,b}(\sigma_{k+1})$ for $s \in S, b \in \mathcal{D}_s \setminus \mathbf{d}^{k+1}(s)$ has a non-tangential zero in (t_k, B):

$$\mathsf{NoSwitch}(\sigma_{k+1}) \equiv \bigwedge_{s, b \neq \mathbf{d}^{k+1}(s)} \text{“}y_t^{s,b}(\sigma_{k+1}) \text{ has no non-tangential zero in } (t_k, B)\text{”}$$

We can now check if the optimal strategy has exactly k switches. The first part of the predicate written below sets up a proper σ and the last conjunct states that there is no further switch after the last one.

$$\exists t_1, \ldots, t_k.(0 < t_1 < t_2 \ldots < t_k < B) \wedge \bigwedge_{i=1}^{k} \mathsf{Switch}(\underbrace{\mathbf{d}^1, t_1, \ldots, \mathbf{d}^i, t_i, \mathbf{d}^{i+1}}_{\sigma_{i+1}}) \wedge \mathsf{NoSwitch}(\sigma_{k+1}).$$

We can enumerate these formulas with increasing k over all choices of decision vectors and stop when the above formula is valid. At this point, we know that there is a piecewise constant optimal policy with k switches, which plays the decision vectors $\mathbf{d}^1, \ldots, \mathbf{d}^k$. We can make one more query to check if the probability of reaching **good** when playing this strategy is at least a given rational vector $r \in [0,1]^n$:

$$\exists t_1, \ldots, t_k.(0 < t_1 < \ldots, t_k < B) \wedge \bigwedge_{i=1}^{k} \mathsf{Switch}(\mathbf{d}^1, t_1, \ldots, \mathbf{d}^i, t_i, \mathbf{d}^{i+1}) \wedge \mathsf{NoSwitch}(\sigma_{k+1}) \\ \wedge \bigwedge_{s=1}^{n} \mathbf{u}^T(s) e^{[\mathbf{Q}^{\mathbf{d}^{k+1}}](B-t_k)} e^{[\mathbf{Q}^{\mathbf{d}^k}](t_k - t_{k-1})} \ldots e^{[\mathbf{Q}^{\mathbf{d}^1}]t_1} \mathbf{u}(\mathbf{good}) > r(s) \tag{12}$$

This completes the proof of conditional decidability of Problem 1.

Conditional Decidability for Problem 2. A stationary policy $\mathbf{d}$ is not optimal if there is a switch point. Using the Switch predicate and conditional decidability of $\mathbb{R}_{\mathsf{MW}}$, this shows conditional decidability of Problem 2.

In fact, to check the presence of a single non-tangential zero, one can avoid Theorem 8 and get a direct construction based on Schanuel's conjecture. This construction is similar to [11] and is provided in Section 5. Unfortunately, when there are multiple switch points, we have to existentially quantify over previous switch points. Thus, the techniques of [11] cannot be straightforwardly extended to find a direct conditional decision procedure for Problem 1.

We do not know if there is a numerical procedure that only uses an oracle for non-tangential zeros. The problem is that, while numerical techniques can be used to bound each non-tangential zero with rational intervals with arbitrary precision as well as compute the reachability probability to arbitrary precision, we do not know how to numerically detect whether the reachability probability in (12) is exactly equal to a given r. By the Lindemann-Weierstrass Theorem [15], we already know that for CTMDPs with stationary optimal strategies, the value of reachability probability for any rational time bound $B > 0$ is transcendental and hence $\sup_{\pi \in \Pi_B} \mathbf{P}_s^{\pi}(\mathbf{reach}) \neq r(s)$ for all $s \in S$. However, we cannot prove that the reachability probability remains irrational in the general case.

4 Lower Bound: Continuous Skolem Problem

▶ **Problem 3** (Bounded Continuous-Time Skolem Problem). *Given a linear ordinary differential equation (ODE)*

$$\frac{d^n}{dt^n} z_t + a_{n-1} \frac{d^{n-1}}{dt^{n-1}} z_t + \cdots + a_1 \frac{d}{dt} z_t + a_0 z_t = 0 \tag{13}$$

with rational initial conditions $z_0, \frac{dz_t}{dt}|_{t=0}, \ldots, \frac{d^{n-1}z_t}{dt^{n-1}}|_{t=0} \in \mathbb{Q}$ *and rational coefficients* $a_{n-1}, a_{n-2}, \ldots, a_0 \in \mathbb{Q}$ *and a time bound* $B \in \mathbb{Q}$, *the* bounded continuous Skolem problem *asks whether there exists* $0 < t^* < B$ *such that it is a non-tangential zero for* z_t. *Further, we can assume w.l.o.g. that* $z_0 = 0$ *in the initial condition.*[2]

[2] The assumption is w.l.o.g. because given a linear ODE whose solution is z_t, one can construct another linear ODE whose solution is $y_t = t z_t$. Clearly, $y_0 = 0$ and there is a non-tangential zero of z in $(0, B)$ iff there is a non-tangential zero of y in $(0, B)$.

We note that our definition is slightly different from the usual definition of the problem, e.g., in [7, 11], which simply asks for any zero (i.e., $z_{t^*} = 0$), not necessarily a non-tangential one. Our version of the bounded continuous Skolem problem is also decidable assuming **SC** [11]. However, there is no unconditional decidability result known for this problem, even though we only look for a non-tangential zero.

We can encode any given linear ODE of order n in the form of (13) into a set of n first-order linear ODE on $X : [0, B] \to \mathbb{R}^n$ with

$$\begin{cases} \frac{d}{dt}X_t = AX_t, \quad X_0 = \left[z_0, \frac{dz_t}{dt}\Big|_{t=0}, \ldots, \frac{d^{n-1}z_t}{dt^{n-1}}\Big|_{t=0}\right]^T \\ z_t = CX_t, \end{cases} \tag{14}$$

with the state matrix A and output matrix C are

$$A = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ -a_0 & -a_1 & -a_2 & \cdots & -a_{n-1} \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 0 & \cdots & 0 \end{bmatrix}. \tag{15}$$

Using the representation (14), the solution of the linear ODE (13) can be written as $z_t = Ce^{At}X_0$. Therefore, the bounded continuous-time Skolem problem can be restated as whether the expression $Ce^{At}X_0$ has a non-tangential zero in the interval $(0, B)$.

We now reduce the bounded continuous-time Skolem problem to Problem 2. Given an instance (14)-(15) of the Skolem problem of dimension n, we shall construct a CTMDP over states $\{1, \ldots, 2n\} \cup \{\mathbf{good}, \mathbf{bad}\}$ and bound B, and just two decision vectors $\mathbf{d}^a$ and $\mathbf{d}^b$ that only differ in the available actions (a or b) at state 1. Our reduction will ensure that the answer of the Skolem problem has a non-tangential zero iff there is a switch in the optimal policy in the time-bounded reachability problem for bound B, and thus, iff stationary policies are not optimal.

▶ **Theorem 13.** *For every instance of the bounded continuous-time Skolem problem with dynamics $\frac{d}{dt}X_t = AX_t$, $z_t = CX_t$, initial condition X_0, and time bound B, there is a CTMDP $\mathcal{M}$ such that the dynamical system has a non-tangential zero in $(0, B)$ iff the optimal strategy of the CTMDP in the time-bounded reachability problem is not stationary.*

We sketch the main ideas of the proof here. Consider the linear differential equation described by the state space representation in (14) with the initial condition X_0 that has its first element equal to zero $X_0(1) = 0$. Given the time bound $B > 0$, to solve the bounded continuous Skolem problem, we are looking for the existence of a time $0 < t^* < B$ such that $z_{t^*} = 0$ is non-tangential. Equivalently, we want to find a non-tangential zero for the function $Ce^{At}X_0$, where $C = \begin{bmatrix} 1 & 0 & \cdots & 0 \end{bmatrix}$.

There are three obstacles to go from (14) to generator matrices for a CTMDP. Each generator matrix must have non-diagonal entries that are non-negative. The sum of each row of the matrix must be zero. Moreover, the last state of the CTMDP must be absorbing. None of these properties may hold for a general A. We show a series of transformations that take the matrix A to a matrix P that is sub-stochastic. Then we construct the generator matrices of the CTMDP using P that include the required absorbing state. We denote by $\mathbf{0}_\mathsf{m}$ and $\mathbf{1}_\mathsf{m}$ as row vectors of dimension m with all elements equal to zero and one, respectively.

▶ **Theorem 14.** *Suppose $A \in \mathbb{Q}^{n\times n}$, $X_0 \in \mathbb{Q}^n$ and $C = [1, \mathbf{0}_{n-1}]$ are given with $X_0(1) = 0$. There are positive constants γ, λ and a* generator *matrix $P \in \mathbb{Q}^{(2n+1)\times(2n+1)}$ such that*

$$Ce^{At}X_0 = \gamma e^{\lambda t}\left[C'e^{Pt}Y_0\right], \quad C' = [1, -1, \mathbf{0}_{2n-1}], \quad Y_0 = [\mathbf{0}_{2n}, 1]^T. \tag{16}$$

▶ Remark 15. The first equality in (16) ensures that nature of zeros of the two functions $Ce^{At}X_0$ and $C'e^{Pt}Y_0$ are the same. If one of them has a non-tangential zero at t^* the other one will also have a non-tangential zero at t^*. To see this, suppose $Ce^{At^*}X_0 = 0$ and $Ce^{At}X_0$ changes sign at t^*. The same things happen to $C'e^{Pt}Y_0$ due to the fact that the two functions are different with only a positive factor of $\gamma e^{\lambda t}$.

Without loss of generality, we assume the element A_{11} is negative. This assumption is needed when constructing the CTMDP in the sequel. If the assumption does not hold, we can always replace A with $A - \lambda_0\mathbb{I}_n$ for a sufficiently large λ_0 and merge λ_0 with λ in (16). Define the map $\phi_1 : \cup_n\mathbb{Q}^{n\times n} \to \cup_n\mathbb{Q}^{2n\times 2n}_{\geq 0}$ such that $\phi_1(A)$ is obtained by replacing each entry A_{ij} with the matrix $\begin{bmatrix}\alpha_{ij} & \beta_{ij}\\ \beta_{ij} & \alpha_{ij}\end{bmatrix}$, where $\alpha_{ij} = max(A_{ij}, 0)$ and $\beta_{ij} = max(-A_{ij}, 0)$. The map ϕ_1 maps any square matrix to another matrix with non-negative entries ([2]). Also define the map $\phi_2 : \cup_n\mathbb{Q}^n \to \cup_n\mathbb{Q}^{2n}$ such that $\phi_2(X)$ replaces each entry $X(i)$ with two entries $[X(i), 0]^T$.

▶ **Proposition 16.** *We have $C''e^{\phi_1(A)t}Y_2 = Ce^{At}X_0$ with $Y_2 := \phi_2(X_0)$ and $C'' := [1, -1, \mathbf{0}_{2n-2}]$.*

Proof. We can show inductively that for any $k \in \{0, 1, 2, \ldots\}$, $\{\alpha_1, \alpha_2, \ldots, \alpha_n\}$, and $[\beta_1, \beta_2, \ldots, \beta_n] := [\alpha_1, \alpha_2, \ldots, \alpha_n]A^k$, we have

$$[\alpha_1, -\alpha_1, \alpha_2, -\alpha_2, \ldots, \alpha_n, -\alpha_n]\phi_1(A)^k = [\beta_1, -\beta_1, \beta_2, -\beta_2, \ldots, \beta_n, -\beta_n].$$

Substitute $[\alpha_1, \alpha_2, \ldots, \alpha_n]$ by C and $[\beta_1, \beta_2, \ldots, \beta_n] = CA^k$ to get

$$C''\phi_1(A)^kY_2 = C''\phi_1(A)^k\phi_2(X_0) = [\beta_1, -\beta_1, \beta_2, -\beta_2, \ldots, \beta_n, -\beta_n]\phi_2(X_0)$$
$$= [\beta_1, \beta_2, \ldots, \beta_n]X_0 = CA^kX_0$$
$$\Rightarrow C''e^{\phi_1(A)t}Y_2 = \sum_{k=0}^{\infty}\frac{t^k}{k!}C''\phi_1(A)^kY_2 = \sum_{k=0}^{\infty}\frac{t^k}{k!}CA^kX_0 = Ce^{At}X_0.$$ ◀

Next, we define $\lambda := \max_i \sum_{j=1}^n |A_{ij}| + 1$, $P_2 := \phi_1(A) - \lambda\mathbb{I}_n$, and the vector $\boldsymbol{\beta} \in \mathbb{Q}^{2n}$ with

$$\boldsymbol{\beta}(2i-1) = \boldsymbol{\beta}(2i) = \max(0, -P_2Y_2(2i-1), -P_2Y_2(2i)) \quad 1 \leq i \leq n.$$

Note that the row sum of P_2 is at most -1 and $\boldsymbol{\beta} + P_2Y_2$ is element-wise non-negative with the maximum element

$$\gamma := \max_i P_2Y_2(i) + \boldsymbol{\beta}(i).$$

▶ **Proposition 17.** *The above choices of λ, γ and the matrix*

$$P := \begin{bmatrix} P_2 & \vdots & (P_2Y_2 + \boldsymbol{\beta})/\gamma \\ \cdots & \cdots & \cdots \\ \mathbf{0} & \vdots & 0 \end{bmatrix}$$

satisfy (16) *in Theorem 14. Moreover, P is row sub-stochastic.*

Proof. We can easily show by induction that

$$P^k Y_0 = \begin{bmatrix} P_2^{k-1}(P_2 Y_2 + \boldsymbol{\beta})/\gamma \\ 0 \end{bmatrix}, \quad \forall k \in \{1, 2, \ldots\}.$$

$$C' e^{Pt} Y_0 = \sum_{k=0}^{\infty} \frac{t^k}{k!} C' P^k Y_0 = C' Y_0 + C'' \sum_{k=1}^{\infty} \frac{t^k}{k!} P_2^{k-1}(P_2 Y_2 + \boldsymbol{\beta})/\gamma,$$

where $C'' := [1, -1, \mathbf{0}_{2n-2}]$ is the same vector as C' but the last element is eliminated.

$$C' e^{Pt} Y_0 = C' Y_0 + C'' e^{P_2 t} Y_2/\gamma - C'' Y_2/\gamma + \sum_{k=1}^{\infty} \frac{t^k}{k!} C'' P_2^{k-1} \boldsymbol{\beta}/\gamma.$$

The term $C' Y_0$ is zero by simple multiplication of the two vectors. $C'' Y_2 = C'' \phi_2(X_0) = X_0(1)$, which is also assumed to be zero. Finally, we see by induction that for all $k \in \{1, 2, \ldots\}$, the elements $(2i-1)$ and $2i$ of the matrix $P_2^{k-1}\boldsymbol{\beta}$ are equal due to the particular structure of P_2 and $\boldsymbol{\beta}$. Therefore, the last sum in the above is also zero and we get

$$\begin{aligned} C' e^{Pt} Y_0 &= C'' e^{P_2 t} Y_2/\gamma = C'' e^{\phi_1(A)t - \lambda \mathbb{I} t} \phi_2(X_0)/\gamma \\ &= C'' e^{\phi_1(A)t} \phi_2(X_0) e^{-\lambda t}/\gamma = C e^{AtX_0} e^{-\lambda t}/\gamma. \end{aligned}$$

To show that P is a sub-stochastic matrix, we recall that $P_2 Y_2 + \boldsymbol{\beta} \geq 0$ with maximum element γ. Then

$$P_2 \times \mathbf{1}_{2n} + (P_2 Y_2 + \boldsymbol{\beta})/\gamma \leq \phi_1(A)\mathbf{1}_{2n} - \lambda \mathbf{1}_{2n} + \mathbf{1}_{2n} = \phi_1(A)\mathbf{1}_{2n} - \max_i \sum_j |A_{ij}| \leq 0. \quad ◀$$

As the last step, we add an additional row and column to P to make it stochastic:

$$\mathbf{Q}^a := \begin{bmatrix} P_2 & \vdots & \Theta & \vdots & (P_2 Y_2 + \boldsymbol{\beta})/\gamma \\ \cdots & & \cdots & & \cdots \\ \mathbf{0}_{2\times 2n} & \vdots & \mathbf{0}_{2\times 1} & \vdots & \mathbf{0}_{2\times 1} \end{bmatrix}, \bar{C} = \begin{bmatrix} 1 & -1 & \mathbf{0}_{2n} \end{bmatrix}, \bar{Y}_0 = \begin{bmatrix} \mathbf{0}_{2n+1} \\ 1 \end{bmatrix},$$

where Θ has non-negative entries and is such that $\mathbf{Q}^a$ is stochastic (sum of elements of each row is zero). The added row and column correspond to an absorbing state for a CTMDP with no effect on reachability probability: $\bar{C} e^{t\mathbf{Q}^a} \bar{Y}_0 = C' e^{Pt} Y_0$.

Next, we obtain a second generator matrix for the CTMDP. Define $\mathbf{Q}^b := \mathbf{Q}^a + K$ with

$$K := \begin{bmatrix} -r & r & \mathbf{0}_{2n} \\ \mathbf{0}_{(2n+1)\times 1} & \mathbf{0}_{(2n+1)\times 1} & \mathbf{0}_{(2n+1)\times 2n} \end{bmatrix},$$

Note that $\mathbf{Q}^b$ has exactly the same transition rates as in $\mathbf{Q}^a$ except the transition from state 1 to state 2, which is changed by r.

▶ Remark 18. We assumed w.l.o.g. that A_{11} is negative. The construction of $P_2, P, \mathbf{Q}^a$ results in a positive value for $\mathbf{Q}^a_{12}$. Therefore, it is possible to select both negative and positive values for r such that $\mathbf{Q}^b_{12} = \mathbf{Q}^a_{12} + r \geq 0$.

Construction of the CTMDP. The CTMDP $\mathcal{M}$ has $2n+2$ states, corresponding to the rows of $\mathbf{Q}^a$ and $\mathbf{Q}^b$, with the absorbing state $2n+2$ associated with the **good** state and the absorbing state $2n+1$ with reachability probability equal to zero. We shall set the time bound to be B. $\mathcal{D}_s$ the set of actions that can be taken in state $s \in \{2, 3, \ldots, 2n+2\}$

is singleton and $\mathcal{D}_1 = \{a, b\}$. The set of decision vectors has two elements $\mathcal{D} = \{\mathbf{d}^a, \mathbf{d}^b\}$ corresponding to the actions a, b taken at state 1. For simplicity, we denote the generator matrices of these decision vectors by $\mathbf{Q}^a$ and $\mathbf{Q}^b$, respectively. Moreover, the two actions a, b have the same transition rates for jumping from state 1 to other states, except giving different rates r_a, r_b for jumping from 1 to 2 such that $r_b - r_a = r$.

The optimal policy π takes decision vector $\mathbf{d}_t \in \mathcal{D}$ at time $B - t$ such that $\mathbf{d}_t \in \mathcal{F}_{n+2}(W_t^\pi)$ for all $t \in [0, B]$ as defined in (4).

▶ **Proposition 19.** *Let r have the same sign of the first non-zero element of the set $\{\bar{C}\bar{Y}_0, \bar{C}\mathbf{Q}^a\bar{Y}_0, \bar{C}(\mathbf{Q}^a)^2\bar{Y}_0, \ldots\}$ and such that $\mathbf{Q}_{12}^a + r \geq 0$. This particular selection of r results in the optimality of $\mathbf{d}^a$ at $t = 0$.*

Proof. We have $W_0^\pi = \bar{Y}_0$ and $\mathcal{F}_k(W_0^\pi) = \arg\max_{\mathbf{d}}[\mathbf{Q}^{\mathbf{d}}]^k\bar{Y}_0$. Then, we need to compare $[\mathbf{Q}^a]^k\bar{Y}_0$ with $[\mathbf{Q}^b]^k\bar{Y}_0$ for different values of k and see which one gives the first highest value. These two are the same for $k = 1$ and $\mathcal{F}_1(W_0^\pi) = \arg\max_{\mathbf{d}} \mathbf{Q}^{\mathbf{d}}\bar{Y}_0 = \{\mathbf{d}^a, \mathbf{d}^b\}$. Suppose For $k_0 > 1$ is the smallest index such that $\bar{C}[\mathbf{Q}^a]^{k_0}\bar{Y}_0 \neq 0$. It can be shown inductively that $[\mathbf{Q}^b]^k\bar{Y}_0 = [\mathbf{Q}^a]^k\bar{Y}_0$ for all $1 \leq k \leq k_0$:

$$\begin{aligned}[\mathbf{Q}^b]^k\bar{Y}_0 &= \mathbf{Q}^b[\mathbf{Q}^b]^{k-1}\bar{Y}_0 = (\mathbf{Q}^a + K)[\mathbf{Q}^b]^{k-1}\bar{Y}_0 = (\mathbf{Q}^a + K)[\mathbf{Q}^a]^{k-1}\bar{Y}_0 \\ &= [\mathbf{Q}^a]^k\bar{Y}_0 + K[\mathbf{Q}^a]^{k-1}\bar{Y}_0 = [\mathbf{Q}^a]^k\bar{Y}_0 - r\begin{bmatrix}\bar{C}[\mathbf{Q}^a]^{k-1}\bar{Y}_0 \\ \mathbf{0}_{(2n+1)\times 2n}\end{bmatrix} = [\mathbf{Q}^a]^k\bar{Y}_0.\end{aligned}$$

This means $\mathcal{F}_k(W_0^\pi) = \arg\max_{\mathbf{d}}[\mathbf{Q}^{\mathbf{d}}]^k\bar{Y}_0 = \{\mathbf{d}^a, \mathbf{d}^b\}$ for all $1 \leq k \leq k_0$. We have for $k = k_0 + 1$

$$[\mathbf{Q}^b]^{k_0+1}\bar{Y}_0 = [\mathbf{Q}^a]^{k_0+1}\bar{Y}_0 - r\begin{bmatrix}\bar{C}[\mathbf{Q}^a]^{k_0}\bar{Y}_0 \\ \mathbf{0}_{(2n+1)\times 2n}.\end{bmatrix}$$

The first element of $[\mathbf{Q}^b]^{k_0+1}\bar{Y}_0$ is strictly less than the first element of $[\mathbf{Q}^a]^{k_0+1}\bar{Y}_0$ since r has the same sign as $\bar{C}[\mathbf{Q}^a]^{k_0}\bar{Y}_0$. Thus $\mathcal{F}_{k_0+1}(W_0^\pi) = \arg\max_{\mathbf{d}}[\mathbf{Q}^{\mathbf{d}}]^{k_0+1}\bar{Y}_0 = \{\mathbf{d}^a\}$. ◀

Note that the Skolem problem is trivial with the solution $z_t = 0$ for all $t \in [0, B]$ if all the elements of the set $\{\bar{C}\bar{Y}_0, \bar{C}\mathbf{Q}^a\bar{Y}_0, \bar{C}(\mathbf{Q}^a)^2\bar{Y}_0, \ldots\}$ are zero.

Prop. 19 guarantees existence of an $\varepsilon \in (0, B)$ such that W_t^π satisfies

$$\frac{d}{dt}W_t^\pi = \mathbf{Q}^a W_t^\pi \quad \forall t \in (0, \varepsilon),$$

with the initial condition $W_0^\pi(2n+2) = 1$ and $W_0^\pi(s) = 0$ for all $s \in \{1, 2, \ldots, 2n+1\}$.

To check if the optimal policy switches to $\mathbf{d}^b$ at some time point, we should check if there is $t^* < B$ such that $\mathbf{d}^b \in \mathcal{F}_{n+2}(W_{t^*}^\pi)$. This is equivalent to having t^* being non-tangential for the maximization in $\mathcal{F}_1(W_t^\pi)$, which means t^* is non-tangential for the equation

$$\mathbf{Q}^a W_t^\pi = \mathbf{Q}^b W_t^\pi \Leftrightarrow K W_t^\pi = 0 \Leftrightarrow \bar{C} W_t^\pi = 0.$$

Summarizing the above derivations, we have the following set of ODEs

$$\frac{d}{dt}W_t^\pi = \mathbf{Q}^a W_t^\pi \quad , W_0^\pi = \bar{Y}_0, \quad z_t = \bar{C} W_t^\pi. \tag{17}$$

The optimal policy for CTMDP $\mathcal{M}$ switches from $\mathbf{d}^a$ to $\mathbf{d}^b$ at some time point t^* if and only if z_t in (17) has a non-tangential zero in $(0, B)$ if and only if the original dynamics $Ce^{At}X_0$ has a non-tangential zero in $(0, B)$. This completes the proof of Theorem 13.

5 A Direct Algorithm for Problem 2

We now show a "direct" method for decidability of Problem 2 based on Schanuel's conjecture but without relying on the decidability of $\mathbb{R}_{\mathsf{MW}}$. As stated before, a switch point in a strategy corresponds to the existence of a non-tangential zero for the functions $y_t^{s,b}(\mathbf{d}^1)$ for $s \in S$ and $b \in \mathcal{D}_s \setminus \mathbf{d}^1(s)$. We know $y_t^{s,b}(\mathbf{d}^1)$ is an exponential polynomial of the form (7). Thus, deciding Problem 2 reduces to checking if an exponential polynomial of the form (7) in one free variable t has a non-tangential zero in a bounded interval. We use the following result from [11].

▶ **Theorem 20** ([11]). *Assume* ***SC****. It is decidable whether an exponential polynomial of the form* (7) *has a zero in the interval* (t_1, t_2) *with* $t_1, t_2 \in \mathbb{Q}$.

Theorem 20 decides whether a zero, not necessarily a non-tangential one, exists. We shall use the characterization of Proposition 11 to check if a non-tangential zero of $y_t := y_t^{s,b}(\mathbf{d}^1)$ exists in $(0, B)$. Define the functions

$$z_t^k = y_t^2 + \sum_{j=1}^{k} \left(\frac{d^j}{dt^j} y_t \right)^2, \quad k \in \{0, 1, 2, \ldots\}. \tag{18}$$

▶ **Theorem 21.** *Fix rational numbers* $t_1 < t_2$*. Suppose* y_t *has a zero in the interval* (t_1, t_2) *and* y_t *is not identically zero over this interval. There is* k_0 *as the smallest* k *such that* z_t^k *in* (18) *does* not *have any zero in* (t_1, t_2)*. Moreover, the zero of* y_t *in* (t_1, t_2) *is non-tangential if* k_0 *is odd and is tangential if* k_0 *is even.*

Intuitively, the above theorem states that if y_t has at least one zero in (t_1, t_2), we can check for the existence of a tangential or non-tangential zero by a finite number of applications of Theorem 20 to functions z_t^k in (18). Note that y_t may have both tangential and non-tangential zeros; Theorem 21 gives a way of identifying the type of one of the zeros (the one with the largest order).

Proof of Theorem 21. Since y_t is an exponential polynomial, so is z_t^k for all k. Thus, we can use Theorem 20 to check if z_t^k has a zero in (t_1, t_2). Note that z_t^k is the sum of squares of $\frac{d^j}{dt^j} y_t$, which means

$$z_{t^*}^k = 0 \;\Rightarrow\; y_{t^*} = \frac{dy_t}{dt}\Big|_{t=t^*} = \cdots = \frac{d^k y_t}{dt^k}\Big|_{t=t^*} = 0. \tag{19}$$

The first part of the theorem is proved by showing that if for each k, z_t^k has a zero in (t_1, t_2), then y_t is identically zero. Suppose $z_t^k = 0$ for some $t = t_k^*$ in the interval (t_1, t_2), for any $k \in \{0, 1, 2, \ldots\}$. Using (19), we get that $y_t = 0$ for all $t \in \{t_0^*, t_1^*, t_2^*, \ldots\}$. If the set $\{t_0^*, t_1^*, t_2^*, \ldots\}$ is not finite, we get that y_t is identically zero according to the identity theorem [1]. If the set of zeros is finite, there is some t^* that appears infinitely often in the sequence $(t_0^*, t_1^*, t_2^*, \ldots)$. Therefore, $z_{t^*}^k = 0$ for infinitely many indices, which means $\frac{d^k y_t}{dt^k}\big|_{t=t^*} = 0$ for all k. Having y_k as an analytic function, this again implies that y_t is identically zero.

Since y_t is not identically zero, take k_0 such that $z_t^{k_0}$ does not have a zero in (t_1, t_2) but $z_t^{k_0 - 1}$ does. Then, there is $t^* \in (t_1, t_2)$ such that y_t and all its derivatives up to order $k_0 - 1$ are zero at t^* but $\frac{d^{k_0}}{dt^{k_0}} y_t\big|_{t=t^*} \neq 0$. This t^* and k_0 satisfy the conditions of Proposition 11. Thus, t^* is a non-tangential zero for y_t if k_0 is odd and a tangential zero if k_0 is even. ◀

To check if there is a non-tangential zero in an interval $(0, B)$, we apply Theorem 21 to each zero of y_t individually. Suppose y_t has at least one zero. We can localize all zeros of y_t as follows:

1. Set $(t_1, t_2) := (0, B)$;
2. Set k_0 to be the smallest index such that z_t^k in (18) does not have any zero in (t_1, t_2);
3. If $k_0 > 0$, do the next steps:
   - Use bisection to find an interval $(t', t'') \subset (t_1, t_2)$ such that over this interval, $z_t^{k_0-1}$ has a zero and $z_t^{k_0}$ and $\frac{d^{k_0}}{dt^{k_0}} y_t$ do not have any zero;
   - Store (t', t'');
   - Repeat Steps 2-3 with $(t_1, t_2) := (t_1, t')$;
   - Repeat Steps 2-3 with $(t_1, t_2) := (t'', t_2)$.

The bisection used in the above algorithm sequentially splits the interval into two sub-intervals and picks the one that contains the zero of $z_t^{k_0-1}$. It stops when $\frac{d^{k_0}}{dt^{k_0}} y_t$ does not have any zero over the selected sub-interval. The splitting terminates after a finite number of iterations due to the fact that $\frac{d^{k_0}}{dt^{k_0}} y_t$ is a continuous function and non-zero at the zero of y_t. The whole algorithm terminates after a finite number of iterations since y_t has a finite number of zeros in $(0, B)$ (note that if y_t has infinite number of zeros in $(0, B)$, it will be identically zero according to the identity theorem [1]). The output of the algorithm is a set of intervals. Within each interval, y_t has a single zero. Applying Theorem 21 to each such interval will decide whether the zero is tangential or non-tangential.

References

1 Mark J. Ablowitz and Athanassios S. Fokas. *Complex Variables: Introduction and Applications*. Cambridge Texts in Applied Mathematics. Cambridge University Press, 2003. doi:10.1017/CBO9780511791246.

2 S. Akshay, Timos Antonopoulos, Joël Ouaknine, and James Worrell. Reachability problems for Markov chains. *Inf. Process. Lett.*, 115(2):155–158, 2015. doi:10.1016/j.ipl.2014.08.013.

3 Ana Medina Ayala, Sean B. Andersson, and Calin Belta. Formal synthesis of control policies for continuous time Markov processes from time-bounded temporal logic specifications. *IEEE Trans. Automat. Contr.*, 59(9):2568–2573, 2014. doi:10.1109/TAC.2014.2309033.

4 Adnan Aziz, Kumud Sanwal, Vigyan Singhal, and Robert K. Brayton. Model-checking continous-time Markov chains. *ACM Trans. Comput. Log.*, 1(1):162–170, 2000. doi:10.1145/343369.343402.

5 Christel Baier, Holger Hermanns, Joost-Pieter Katoen, and Boudewijn R. Haverkort. Efficient computation of time-bounded reachability probabilities in uniform continuous-time Markov decision processes. *Theor. Comput. Sci.*, 345(1):2–26, 2005. doi:10.1016/j.tcs.2005.07.022.

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

7 Paul C. Bell, Jean-Charles Delvenne, Raphaël M. Jungers, and Vincent D. Blondel. The continuous Skolem-Pisot problem. *Theor. Comput. Sci.*, 411(40-42):3625–3634, 2010. doi:10.1016/j.tcs.2010.06.005.

8 Tomás Brázdil, Vojtech Forejt, Jan Krcál, Jan Kretínský, and Antonín Kucera. Continuous-time stochastic games with time-bounded reachability. *Inf. Comput.*, 224:46–70, 2013. doi:10.1016/j.ic.2013.01.001.

9 Peter Buchholz, Ernst Moritz Hahn, Holger Hermanns, and Lijun Zhang. Model checking algorithms for CTMDPs. In *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, volume 6806 of *Lecture Notes in Computer Science*, pages 225–242. Springer, 2011. doi:10.1007/978-3-642-22110-1_19.

10 Peter Buchholz and Ingo Schulz. Numerical analysis of continuous time Markov decision processes over finite horizons. *Comput. Oper. Res.*, 38(3):651–659, 2011. doi:10.1016/j.cor.2010.08.011.

11 Ventsislav Chonev, Joël Ouaknine, and James Worrell. On the Skolem problem for continuous linear dynamical systems. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 100:1–100:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.ICALP.2016.100.

12 Laure Daviaud, Marcin Jurdzinski, Ranko Lazic, Filip Mazowiecki, Guillermo A. Pérez, and James Worrell. When is containment decidable for probabilistic automata? In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 121:1–121:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.121.

13 John Fearnley, Markus N. Rabe, Sven Schewe, and Lijun Zhang. Efficient approximation of optimal control for continuous-time Markov games. *Inf. Comput.*, 247:106–129, 2016. doi:10.1016/j.ic.2015.12.002.

14 Serge Lang. *Introduction to transcendental numbers.* Addison-Wesley series in mathematics. Addison-Wesley Pub. Co., 1966.

15 Serge Lang. Transcendental numbers and Diophantine approximations. *Bull. Amer. Math. Soc.*, 77(5):635–677, September 1971.

16 Angus Macintyre. Model theory of exponentials on Lie algebras. *Math. Struct. Comput. Sci.*, 18(1):189–204, 2008. doi:10.1017/S0960129508006622.

17 Angus Macintyre. Turing meets Schanuel. *Ann. Pure Appl. Log.*, 167(10):901–938, 2016. doi:10.1016/j.apal.2015.10.003.

18 Angus Macintyre and Alex J. Wilkie. On the decidability of the real exponential field. In *Kreiseliana. About and Around Georg Kreisel*, pages 441–467. A K Peters, 1996.

19 Bruce L. Miller. Finite state continuous time Markov decision processes with a finite planning horizon. *SIAM Journal on Control*, 6(2):266–280, 1968.

20 Martin R. Neuhäußer, Mariëlle Stoelinga, and Joost-Pieter Katoen. Delayed nondeterminism in continuous-time Markov decision processes. In *Foundations of Software Science and Computational Structures, 12th International Conference, FOSSACS 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings*, volume 5504 of *Lecture Notes in Computer Science*, pages 364–379. Springer, 2009. doi:10.1007/978-3-642-00596-1_26.

21 Martin R. Neuhäußer and Lijun Zhang. Time-bounded reachability probabilities in continuous-time Markov decision processes. In *QEST 2010, Seventh International Conference on the Quantitative Evaluation of Systems, Williamsburg, Virginia, USA, 15-18 September 2010*, pages 209–218. IEEE Computer Society, 2010. doi:10.1109/QEST.2010.47.

22 Martin R. Neuhäußer. *Model checking nondeterministic and randomly timed systems.* PhD thesis, Univ. Twente, Enschede, 2010.

23 Jakob Piribauer and Christel Baier. On Skolem-hardness and saturation points in Markov decision processes. In *ICALP*, 2020.

24 Markus N. Rabe and Sven Schewe. Finite optimal control for time-bounded reachability in CTMDPs and continuous-time Markov games. *Acta Inf.*, 48(5-6):291–315, 2011. doi:10.1007/s00236-011-0140-0.

25 Markus N. Rabe and Sven Schewe. Optimal time-abstract schedulers for CTMDPs and continuous-time Markov games. *Theor. Comput. Sci.*, 467:53–67, 2013. doi:10.1016/j.tcs.2012.10.001.

26 Mahmoud Salamati, Sadegh Soudjani, and Rupak Majumdar. A Lyapunov approach for time-bounded reachability of CTMCs and CTMDPs. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)*, 5(1):1–29, 2020.

27 A. J. Wilkie. Schanuel's conjecture and the decidability of the real exponential field. In *Algebraic Model Theory*, pages 223–230. Springer Netherlands, Dordrecht, 1997.

28 Nicolás Wolovick and Sven Johr. A characterization of meaningful schedulers for continuous-time Markov decision processes. In *Formal Modeling and Analysis of Timed Systems, 4th International Conference, FORMATS 2006, Paris, France, September 25-27, 2006, Proceedings*, volume 4202 of *Lecture Notes in Computer Science*, pages 352–367. Springer, 2006. doi:10.1007/11867340_25.

When Is a Bottom-Up Deterministic Tree Translation Top-Down Deterministic?

Sebastian Maneth
Universität Bremen, Germany
maneth@uni-bremen.de

Helmut Seidl
TU München, Germany
seidl@in.tum.de

Abstract

We consider two natural subclasses of deterministic top-down tree-to-tree transducers, namely, linear and uniform-copying transducers. For both classes we show that it is decidable whether the translation of a transducer *with* look-ahead can be realized by a transducer *without* look-ahead. The transducers constructed in this way, may still make use of *inspection*, i.e., have an additional tree automaton restricting the domain. We provide a second procedure which decides whether inspection can be removed and if so, constructs an equivalent transducer *without* inspection. The construction relies on a fixpoint algorithm that determines inspection requirements and on dedicated earliest normal forms for linear as well as uniform-copying transducers which can be constructed in polynomial time. As a consequence, equivalence of these transducers can be decided in polynomial time. Applying these results to deterministic bottom-up transducers, we obtain that it is decidable whether or not their translations can be realized by deterministic uniform-copying top-down transducers without look-ahead (but with inspection) – or without both look-ahead and inspection.

2012 ACM Subject Classification Theory of computation → Models of computation; Theory of computation → Formal languages and automata theory

Keywords and phrases Top-Down Tree Transducers, Earliest Transformation, Linear Transducers, Uniform-copying Transucers, Removal of Look-ahead, Removal of Inspection

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.134

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

1 Introduction

Even though top-down and bottom-up tree transducers are well-studied formalisms that were introduced already in the 1970's (by Rounds [13] and Thatcher [14] independently, and by Thatcher [15], respectively), some fundamental questions have remained open until today. A prominent example of such a question is: can we decide for a given deterministic bottom-up tree transducer whether or not its translation can be realized by a top-down tree transducer? We answer this question affirmatively, however, for a slight restriction on the considered top-down tree transducers: they must be *uniform-copying* (*uc*). This means that all copies of the same input subtree must be processed by the same state.

It is well-known that for every deterministic bottom-up tree transducer an equivalent deterministic top-down tree transducer can be constructed which, however, makes use of regular look-ahead [7]. That transducer indeed is *uc*. The question which we ask therefore is: can regular look-ahead in *uc* transducers be eliminated? In order to answer this question, we provide a canonical earliest normal form for *uc* (as well as for linear) deterministic top-down transducers with and without look-ahead. We prove that if an earliest such transducer A can be realized by such a transducer A' *without* look-ahead (but with input inspection),

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 134; pp. 134:1–134:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

then A must be synchronizing, twinning, and erasing. To understand the *synchronizing* property, consider a transducer with look-ahead that translates an input tree of the form $a(f(t_1,t_2)))$ into the tree $\langle a,f\rangle(t_1,t_2)$, where $\langle a,f\rangle$ is a binary output symbol. Clearly this translation *can* be done by a transducer without look-ahead: it outputs nothing at the root node, keeps the node label in its state, and at its child outputs the corresponding binary symbol. Now consider that $a(f(g(t_1),t_2)))$ is translated to $\langle a,g\rangle(f(t_1,t_2))$. Such a translation can*not* be realized by a transducer without look-ahead. The information about the g-node cannot be "synchronized" at the f-node because it comes after the output must be produced (contradicting the order of origins of output nodes [12], see also [10, 11]).

The *twinning* property is similar to the string case [5] (see also [2, 3]), but now for paths. To understand the *erasing* property, consider a transducer with the following rules.

$$\begin{array}{lcllcl} q_0(a(x_1:h_e)) & \to & a(a(e)) & q_{\mathrm{id}}(a(x_1:h_f)) & \to & a(q_{\mathrm{id}}(x_1)) \\ q_0(a(x_1:h_f)) & \to & a(q_{\mathrm{id}}(x_1)) & q_{\mathrm{id}}(f) & \to & f \\ q_0(f) & \to & f & & & \end{array}$$

Here, input trees are of the form $a(\cdots a(e)\cdots)$ or $a(\cdots a(f)\cdots)$. The look-ahead automaton has two states h_e and h_f, indicating that the input tree is of the first form (e-leaf) or the second form (f-leaf). The transducer translates input trees of the first form to the fixed tree $a(a(e))$ and realizes the identity on trees of the second form. This translation can*not* be realized by a transducer without look-ahead. The *erasing* property demands that if an input path depends on two different look-ahead states h_1,h_2, where for h_1 a constant output tree is produced (viz. the tree $a(a(e))$), then for h_2 no output may be produced in any loop.

Given a *uc* transducer A with look-ahead that is synchronizing, twinning, and erasing we construct an equivalent *uc* transducer with *inspection* (if it exists), i.e., a *uc* transducer where the domain is given separately via some top-down deterministic tree automaton.

The *third highlight* of our contribution is a procedure that removes inspection (if possible). The idea here is quite different from what we have discussed until now. Let us consider an example. The domain automaton accepts trees of the form $f(t_1,t_2)$ where t_2 is an arbitrary binary tree (with internal nodes labeled f and leaves labeled a or b) and t_1 is a tree which has a left-most leaf labeled a and a right-most leaf labeled b. The transducer has this rule:

$$q_0(f(x_1,x_2)) \to f(f(b,b),q_{\mathrm{id}}(x_2)),$$

where state q_{id} realizes the identity. Does there exist an equivalent top-down tree transducer *without* inspection? As it turns out, the answer is "yes". The idea is that the output subtree $f(b,b)$ can be used to simulate inspection! These are the rules of an equivalent transducer without inspection:

$$\begin{array}{lll} q_0(f(x_1,x_2)) \to f(q(x_1),q_{\mathrm{id}}(x_2)) & q_a(f(x_1,x_2)) \to q_a(x_1) & q_b(f(x_1,x_2)) \to q_b(x_2) \\ q(f(x_1,x_2)) \to f(q_a(x_1),q_b(x_2)) & q_a(a) \to b & q_b(b) \to b \end{array}$$

We show that it is decidable for a given top-down deterministic tree language, whether or not it can be simulated on a given output tree. The challenge now is that it may be necessary to *delay* outputting certain output subtrees, until rules are encountered which require these output trees for simulating their inspection needs. Similar as before, such delay is only possible along input paths and must stop when two input subtrees of an input node are processed. Using a fixpoint algorithm we are able to determine whether or not sufficiently large output subtrees can be made available in order to satisfy all inspection needs. The approach we have sketched here is quite different from earlier methods for look-ahead removal [9] that rely on *difference bounds*, i.e., the differences in the translation with respect to different look-ahead states. In order to obtain an effective construction, however, so far a variety of technical restrictions had to be introduced – implying that even for *linear* deterministic top-down tree transducers, look-ahead removal remained an open problem. To the best of our knowledge, this paper is the first to present look-ahead removal for natural and known subclasses of top-down tree transducers.

2 Basics

Let Σ denote a ranked alphabet. Then Σ_k is the set of all symbols in Σ of rank k. As usual, we define the set $\mathcal{T}_\Sigma$ of all (finite) *trees* over Σ as the set of all strings $t = f(t_1, \ldots, t_k)$ where $f \in \Sigma_k$ for some $k \geq 0$ and $t_1, \ldots, t_k \in \mathcal{T}_\Sigma$. For convenience, we also write f for $f()$ if f is of rank 0. A subtree of that form is also called *leaf*. Let $X = \{x_i \mid i \in \mathbb{N}\}$ denote an infinite set of distinct *variables* which is disjoint from any other occurring ranked alphabet (be it the input or the output alphabet of a transducer). All elements of the set X are assumed to have rank 0. For a finite set $J \subseteq \mathbb{N}$ we denote by X_J the set of variables $\{x_j \mid j \in J\}$, and we write $\mathcal{T}_\Sigma(X_J)$ for the set of all trees t over $\Sigma \cup X_J$. E.g., $f(h(x_2), a) \in \mathcal{T}_\Sigma(\{x_2\})$ where $f \in \Sigma_2$, $h \in \Sigma_1$, and $a \in \Sigma_0$. Trees in $\mathcal{T}_\Sigma(X_J)$ are also called *patterns*. Of particular importance is the set of *unary* patterns $\mathcal{T}_\Sigma(\{x_1\}) = \mathcal{T}_\Sigma(x_1)$. This set forms a *free monoid* where the monoid operation "$\cdot$" is substitution into the variable x_1. The tree $t = f(h(x_1), g(a, h(x_1)))$, e.g., can be uniquely factored into $f(x_1, g(a, x_1))$ and $h(x_1)$. We thus write $f(h(x_1), g(a, h(x_1))) = f(x_1, g(a, x_1)) \cdot h(x_1)$. We also consider the set $\mathcal{C}_\Sigma \subseteq \mathcal{T}_\Sigma(x_1)$ of *contexts* over Σ which is the subset of unary patterns which contain exactly one occurrence of x_1. Technically, this means that each context t either is equal to x_1, or is of the form $t = f(t_1, \ldots, t_k)$ for some $f \in \Sigma_k$ for some $k \geq 1$ and $1 \leq j, j' \leq k$ so that t_j is a context and $t_{j'} \in \mathcal{T}_\Sigma$ for all $j' \neq j$.

In the following, Σ and Δ denote fixed non-empty ranked alphabets of input and output symbols, respectively. In this paper, we consider deterministic top-down tree transducers with *uniform copying*, or *uc*-transducers for short. Intuitively, *uniform copying* means that each subtree of the input is processed at most once – while the produced output may be copied arbitrarily often. This restriction is trivially met by *linear* deterministic top-down transducers – but also by those that arise from the top-down simulation of deterministic *bottom-up* transducers by means of regular look-ahead. Here, we refrain from introducing transducers with *look-ahead* and *inspection* separately, as this would result in awkward duplication of almost identical definitions. Instead, we find it convenient to introduce yet another model, namely, deterministic transducers with (unambiguous) *advice* – which later can be instantiated either with top-down deterministic inspection (no interference with the computation of the transducer, only restriction to relevant input) or bottom-up deterministic look-ahead (interference with the computation as well as restriction to relevant input).

A *finite tree automaton over* Σ, (for short, *TA*) B consists of

1. a finite set H of states,
2. a subset $F \subseteq H$ of accepting states, and a transition relation $\delta \subseteq \bigcup_{k\geq 0} H \times \Sigma_k \times H^k$.

The computation of B on some input tree t can be represented by a tree in $\mathcal{T}_T$ where the ranked alphabet T consists of all transitions $\tau = \langle h, f, h_1 \ldots h_k\rangle \in \delta$ where the rank of τ equals the rank of the input symbol f. For $h \in H$, an h-computation ϕ for some $t = f(t_1, \ldots, t_k) \in \mathcal{T}_\Sigma$ is a tree $\phi = \tau(\phi_1, \ldots, \phi_k)$ where ϕ_i is a h_i-computation for t_i for all $i = 1, \ldots, k$. We write $h : t$ to indicate that there is an h-computation for t. We write $\mathsf{dom}_B(h) = \{t \in \mathcal{T}_\Sigma \mid h : t\}$ and define the set of trees accepted by B as $\mathcal{L}(B) = \{t \in \mathcal{T}_\Sigma \mid \exists h_0 \in F \text{ such that } h_0 : t\}$. An (h, h')-computation ϕ of B on some context $t \in \mathcal{C}_\Sigma$ is analogously defined as a context in $\mathcal{C}_T$ where h is the state at the root and h' is assumed at the variable leaf. We write $(h, h') : t$ to indicate that a (h, h')-computation for t exists. In particular, $(h, h) : x_1$ for every state h of B. In general, we assume that every occurring TA B is *trim*, i.e., every transition of B occurs in some *accepting* computation, i.e., some h-computation with $h \in F$. We call B

- *bottom-up deterministic*, when for every tuple $(f, h_1 \dots h_k) \in \Sigma_k \times H^k$, there is at most one $h \in H$ so that $\langle h, f, h_1 \dots h_k\rangle \in \delta$;
- *top-down deterministic* when F consists of a single state only, and for every $k \geq 0$ and pair $(h, f) \in H \times \Sigma_k$, there is at most one tuple $(h_1 \dots h_k) \in H^k$ such that $\langle h, f, h_1 \dots h_k\rangle \in \delta$;
- *unambiguous*, if for each $t \in \mathcal{T}_\Sigma$, there is at most one $h \in F$ and at most one h-computation ϕ of B for t.

It is well-known that B is unambiguous whenever B is bottom-up deterministic, or top-down deterministic. In the following definition, we assume that the TA B is trim, unambiguous, and has a single accepting state h_0.

A *deterministic uniform-copying top-down tree transducer with advice over* Σ *and* Δ (for short, a $\mathsf{DT}^{\mathrm{A}}_{\mathrm{uc}}$ *transducer*, or *uc-transducer*, or a $\mathsf{DT}^{\mathrm{A}}_{\mathrm{uc}}$) A is a tuple (B, Q, ι, T_0, R) where

1. B is an unambiguous *advice* TA with a single final state h_0;
2. Q is a finite set of states together with a mapping $\iota : Q \to H$
3. T_0 is an axiom which is either a tree from $\mathcal{T}_\Delta$, or of the form $T_0 = p \cdot q_0(x_1)$ where $p \in \mathcal{T}_\Delta(x_1)$ with $q_0 \in Q$ and $\iota(q_0) = h_0$;
4. R is the set of rules such that for every transition $\langle h, f, h_1 \dots h_k\rangle$ of B and every state $q \in Q$ with $\iota(q) = h$, R contains one rule

$$q(f(x_1 : h_1, \dots, x_k : h_k)) \to T \tag{1}$$

where $f \in \Sigma_k$ for some $k \geq 0$, and $T = p\{x_j \mapsto q_j(x_j) \mid j \in J\}$ where $p \in T_\Delta(X_J)$ for some subset $J \subseteq \{1, \dots, k\}$ and for all $j \in J$, $q_j \in Q$ with $\iota(q_j) = h_j$ (thus $\{x_j \mapsto \dots\}$ is our notation of substituting leaves labeled x_j by the correspoding trees).

A *uc*-transducer is *linear*, if each input variable x_i occurs at most once in the right-hand side of every rule (we also say "$\mathsf{DT}^{\mathrm{A}}_{\mathrm{lin}}$ transducer").

We remark that we view the set Q of states of A as symbols of rank 1 distinct from all symbols in Δ. Given an h-computation ϕ of B on some input tree $t \in \mathcal{T}_\Sigma$, the rule (in R) at each node of t is uniquely determined by the state $q \in Q$ (with $\iota(q) = h$) at the root of t, and the transitions chosen in ϕ. Assume that $t = f(t_1, \dots, t_k)$, and $\phi = \tau(\phi_1, \dots, \phi_k)$ for $\tau = \langle h, f, h_1 \dots h_k\rangle$. A q-computation ψ of A on t with output s is given by $\rho(\sigma_1, \dots, \sigma_k)$ provided the following holds:

1. ρ is a rule of the form (1);
2. if $j \in J$, then σ_j is a q_j-computation for t_j with some output s_j and otherwise, $\sigma_j = \phi_j$;
3. $s = T\{x_j \mapsto s_j \mid j \in J\}$.

If such a q-computation for t exists with output s, we write $q : t \to s$.

As for TAs, we not only require the notion of a q-computation of A for input trees $t \in \mathcal{T}_\Sigma$ with output s, but also the notion of a (q, h)-computation of A on a context $t \in \mathcal{C}_\Sigma$ with output s. Let ϕ denote a $(\iota(q), h)$-computation of B. If $t = x_1$, then x_1 is a (q, h)-computation for x_1 with output $s = q(x_1)$ whenever $\iota(q) = h$. Assume that $t = f(t_1, \dots, t_k)$ and $t_j \in \mathcal{C}_\Sigma$ is a context, and let $\phi = \tau(\phi_1, \dots, \phi_k)$ denote the corresponding $(\iota(q), h)$-computation of B. Assume that the rule ρ is of the form (1). Then $\rho(\psi_1, \dots, \psi_k)$ is a (q, h)-computation for t with output $s = p\{x_j \mapsto s_j \mid j \in J\}$, if the following holds:

1. If $j' \notin J$, then $\psi_{j'} = \phi_{j'}$;
2. If $j' \in J \setminus \{j\}$, then $\psi_{j'}$ is a q_j-computation for $t_{j'}$ with output $s_{j'}$;
3. If $j' = j \in J$, then $\psi_{j'}$ is a (q_j, h)-computation for t_j with output s_j.

We remark that if s is non-ground, i.e., is not contained in $\mathcal{T}_\Delta$, then $s = s' \cdot q'(x_1)$ with $\iota(q') = h$. If such a (q, h)-computation exists, we write $(q, h) : t \to s$.

The *translation* of A is the partial mapping $[\![.]\!]_A : \mathcal{T}_\Sigma \to \mathcal{T}_\Delta$ defined by $[\![t]\!]_A = p_0$ if the axiom of A is the ground tree p_0 and $t \in \mathcal{L}(B)$, and $[\![t]\!]_A = p_0 \cdot s$ if the axiom is of the form $p_0 \cdot q_0(x_1)$ and $q_0 : t \to s$ holds.

Let us briefly list instances of *uc*-transducers with advice that are of interest here.

Transducers with Look-ahead. If the advice automaton B is chosen as bottom-up deterministic, the transducer A is a *uc*-transducer with regular look-ahead (for short, a $\mathsf{DT}^{\mathrm{R}}_{\mathrm{uc}}$ transducer, or a $\mathsf{DT}^{\mathrm{R}}_{\mathrm{uc}}$). We remark that the single axiom does not impose a severe restriction. Consider the generalization of a look-ahead automaton B with a non-singleton set F of accepting states and, accordingly, equip A with one dedicated axiom T_h for each accepting state $h \in F$. Instead, we may introduce a fresh unary input symbol \$ and define a new look-ahead automaton B' with a single fresh accepting state h_0 and a transducer A' with a single axiom such that $\mathcal{L}(B') = \{\$(t) \mid t \in \mathcal{L}(B)\}$ and $[\![\$(t)]\!]_{A'} = s$ holds iff $[\![t]\!]_A = s$ holds. In order to achieve that, we add to the set of transitions of B, all transitions $\langle h_0, \$, h\rangle$, $h \in F$, and likewise introduce a fresh axiom $q_0(x_1)$ for A' together with a fresh state q_0 where $\iota(q_0) = h_0$ and the rules $q_0(\$(x_1 : h)) \to T_h$ whenever $h \in F$ and T_h is the axiom of A for h.

Transducers with Inspection. If the advice automaton B is chosen as top-down deterministic, the transducer A can be considered as a *uc*-transducer with inspection automaton B (for short, a $\mathsf{DT}^{\mathrm{I}}_{\mathrm{uc}}$ transducer, or a $\mathsf{DT}^{\mathrm{I}}_{\mathrm{uc}}$). In this case, the state annotations h_i in the rule (1), can be dropped since these are obtained from $\iota(q)$ (q the current state of the transducer) and the input symbol f. A deterministic *bottom-up* tree transducer in the classical sense, e.g., as in [6] is obtained in our model as a $\mathsf{DT}^{\mathrm{A}}_{\mathrm{uc}}$ transducer where $Q = H$ and ι is the identity. A classical deterministic *top-down* tree transducer with uniform copying, on the other hand, is obtained as a $\mathsf{DT}^{\mathrm{I}}_{\mathrm{uc}}$ transducer where the inspection does not restrict the domain. This can be achieved, e.g., by setting $H = Q \cup \{\top\}$ for a fresh symbol $\top$ and $\iota(q) = q$. Moreover, for each $f \in \Sigma_k$, $\langle \top, f, \top^k\rangle \in \delta$ as well as $\langle q, f, q'_1 \dots q'_k\rangle \in \delta$ whenever there is a rule $q(f(x_1, \dots, x_k)) \to T$ such that for each $i = 1, \dots, k$, $q'_i = q_i$ if $q_i(x_i)$ occurs in T and $q'_i = \top$ otherwise.

We remark that in the same way, deterministic *linear* top-down tree transducers with look-ahead as well as deterministic linear bottom-up transducers and deterministic linear top-down transducers with inspection are instances of *linear* $\mathsf{DT}^{\mathrm{A}}_{\mathrm{uc}}$ transducers.

3 A Dedicated Earliest Normal Form for *uc*-Transducers with Advice

In this section we present the construction of earliest normal-forms for *uc*-transducers with advice. We also indicate how a corresponding construction is obtained for *linear* transducers. In the following, we fix some unambiguous TA B for advice. A construction of *earliest* top-down transducers has already been provided in [8] for top-down deterministic domain automata B and as well as in [4] for bottom-up deterministic B. Here, we are slightly more liberal by allowing *unambiguous* B to generalize both cases. The constructions from [8, 4], on the other hand, neither preserve linearity nor uniform-copying.

▶ **Example 1.** Consider a linear top-down transducer with the rules:

$$\begin{array}{lcllcl} q_0(g(x_1)) & \to & q_1(x_1) & \qquad q_0(a) & \to & a \\ q_1(f(x_1,x_2)) & \to & f(q_0(x_1), q_0(x_2)) & \qquad q_0(b) & \to & b \end{array}$$

and the axiom $q_0(x_1)$. The (canonical) earliest transducer constructed according to the methods in [8] has the same axiom $q_0(x_1)$, but the rules:

$$\begin{array}{lllllll}
q_0(g(x_1)) & \to & f(q_{11}(x_1), q_{12}(x_1)) & & q_0(a) & \to & a \\
q_{11}(f(x_1, x_2)) & \to & q_0(x_1) & & q_0(b) & \to & b \\
q_{12}(f(x_1, x_2)) & \to & q_0(x_2) & & & &
\end{array}$$

where no inspection automaton is required. The non-linearity in the first rule arises inevitably, because the output f-node is already determined at this point and therefore must be output for the transducer to be *earliest*.

We introduce *dedicated* constructions which allow to construct equivalent *canonical* transducers, but retain linearity or uniform-copying. It turns out that these normal forms can be obtained in polynomial time. Our key insight is that *dedicated* notions should be provided for the notion of *maximal common prefix* of a given set S of trees. In [8], a pattern such as $p = a(x_1, g(x_1))$ was used to represent the common part of trees in S (note that they do not use the symbol x_1, but the symbol $\top$ to denote "any tree"). This meant for an element $t \in S$ such as $t = a(b, g(c))$ that different occurrences of the symbol x_1 in p could correspond to not necessarily isomorphic subtrees of t, in the example, b and c, respectively. In that point, we will now be more restrictive and only allow *substitutions*, i.e., equal replacements of the occurrences of the single variable x_1 in patterns. For a distinction, we call such patterns *uniform.* Let us denote by $\mathcal{P}_\Delta$ the set $\mathcal{T}_\Delta \cup \mathcal{T}_\Delta(x_1) \cup \{\bot\}$ of all ground trees and unary patterns, extended with one specific element $\bot$. This set forms a *partial order* where for $t_1, t_2 \in \mathcal{P}_\Delta$, $t_1 \sqsubseteq t_2$ iff $t_1 = \bot$ or $t_1 = t_2\{x_1 \mapsto s\}$ for some $s \in \mathcal{T}_\Delta \cup \mathcal{T}_\Delta(x_1)$. In fact, $\mathcal{P}_\Delta$, partially ordered in this way, forms a *complete lattice* with finite ascending chains. In particular, the top-most element is x_1, and the binary least upper bound operation $\sqcup$ for incomparable elements $t_1, t_2 \neq \bot$, is given by $t_1 \sqcup t_2 = s$ where s is the maximal prefix such that $s\{x_1 \mapsto t'_i\} = t_i$ for suitable trees t'_i $(i = 1, 2)$.

We remark that uniform patterns may contain more than one occurrence of x_1 – all representing, though, isomorphic subtrees. Let $\mathcal{P}^{(1)}_\Delta \subseteq \mathcal{P}_\Delta$ denote the subset $\mathcal{T}_\Delta \cup \mathcal{C}_\Delta \cup \{\bot\}$ of all elements which either equal $\bot$ or contain at most one occurrence of x_1. Patterns in that set are also called *1-patterns.* For the induced partial ordering on $\mathcal{P}^{(1)}_\Delta$, we again obtain a complete lattice with finite ascending chains only. For a distinction, let us denote the least upper bound operation with respect to $\mathcal{P}^{(1)}_\Delta$ with $\sqcup^{(1)}$.

▶ **Example 2.** The difference between the two least upper bound operations becomes apparent when considering trees which differ in more than one subtree:

$$\begin{array}{lll}
f(g(a,a),c) \sqcup f(g(b,b),c) & = & f(g(x_1,x_1),c) \\
f(g(a,a),c) \sqcup^{(1)} f(g(b,b),c) & = & f(x_1,c).
\end{array}$$

On the other hand, $f(g(a,a),c) \sqcup f(g(b,b),d) = f(g(a,a),c) \sqcup^{(1)} f(g(b,b),d) = x_1$. We remark that the earliest construction in [8] would return $f(g(x_1,x_1),x_1)$ in the latter case – implying that the place holder x_1 no longer represents *isomorphic* subtrees.

For $q \in Q$, let

$$\mathsf{pref}_A(q) = \bigsqcup\{s \in \mathcal{T}_\Delta \mid \exists t \in \mathcal{T}_\Sigma.\, q : t \to s\} \text{ and } \mathsf{pref}^{(1)}_A(q) = \bigsqcup{}^{(1)}\{s \in \mathcal{T}_\Delta \mid \exists t \in \mathcal{T}_\Sigma.\, q : t \to s\}$$

In the following, we show that $\mathsf{pref}_A : Q \to \mathcal{P}_\Delta$ as the least solutions of the set $\mathcal{C}_A$ of constraints. The case of $\mathsf{pref}^{(1)}_A$ is analogous. The set $\mathcal{C}_A$ consists of one constraint $c(\rho)$ for each rule ρ of A. Assume that $\tau \equiv q(f(\ldots)) \to T$ of A where $T = p\{x_j \mapsto q_j(x_j) \mid j \in J\}$ for some $p \in \mathcal{T}_\Delta(X_J)$ and suitable $q_j \in Q$. Then the constraint $c(\tau)$ is given by

$$\sigma(q)^\sharp \quad \sqsupseteq \quad [\![T]\!]^\sharp \sigma^\sharp \tag{2}$$

where the value $[\![T]\!]^\sharp\sigma^\sharp$ returns $\bot$ if for any state $q_j, j \in J$, $\sigma^\sharp(q_j) = \bot$. Otherwise, assume that p' is obtained from p and $\sigma^\sharp$ by replacing $q_j(x_j)$ with $\sigma^\sharp(q_j) \cdot x_j$ where $j \in J$ and with $\sigma^\sharp(q_j)$ whenever $\sigma^\sharp(q_j)$ is ground. If p' is ground, we set $[\![T]\!]^\sharp\sigma^\sharp = p'$. If p' contains occurrences of x_j for a single $j \in J$, i.e., is of the form $p' = p'' \cdot x_j$ for some $p'' \in \mathcal{T}_\Delta(x_1)$, then $[\![T]\!]^\sharp\sigma^\sharp = p''$. Otherwise, i.e., if p' contains occurrences of more than one variable, then $[\![T]\!]^\sharp\sigma^\sharp = u$ where $u \in \mathcal{T}_\Delta(x_1)$ is the maximal prefix such that $p' = u \cdot p''$ for some p'' containing all x_j, i.e., u has maximal size with this property and thus is *least* with respect to the ordering on patterns.

▶ **Example 3.** Let $T = f(g(q_2(x_2), a), g(q_2(x_2), q_1(x_1))$, and thus $p = f(g(x_2, a), g(x_2, x_1))$. Then for $\sigma^\sharp = \{q_1 \mapsto a, q_2 \mapsto h(x_1)\}$, we have that $p' = f(g(h(x_2), a), g(h(x_2), a))$, and thus, $[\![T]\!]^\sharp\sigma^\sharp = f(x_1, x_1) \cdot g(x_1, a) \cdot h(x_1) = f(g(h(x_1), a), g(h(x_1), a))$.

We remark that each right-hand side of a constraint in $\mathcal{C}_A$ represents a function which is *distributive* in each argument, i.e., commutes with the binary operator $\sqcup$ in each accessed argument $\sigma^\sharp(q_j)$. Recall that any distributive function is also monotonic. Since the partial ordering on $\mathcal{P}_\Delta$ and likewise on $\mathcal{P}_\Delta^{(1)}$ are complete lattices with finite ascending chains, the constraint system (2) as well as the respective system for linear transducers and 1-patterns, has a least solution. We thus obtain:

▶ **Lemma 4.** *Let $\mathsf{pref}_A(q), q \in Q$, denote the least solution of the set of constraints* (2) *over the complete lattice $\mathcal{P}_\Delta$ ($\mathcal{P}_\Delta^{(1)}$). Then for every $a \in Q$,*

$$\mathsf{pref}_A(q) = \bigsqcup\{s \mid \exists t \in \mathcal{T}_\Sigma.\ q : t \to s\} \tag{3}$$

holds. Moreover, this least solution can be computed in polynomial time.

Proof. Recall that by our assumption, the advice automaton is trim. Therefore, according to our construction, there is a q-computation for every state $q \in Q$, i.e., $\mathsf{pref}_A(q) \neq \bot$ for each $q \in Q$. The equality in equation (3) then is due to the fixpoint transfer lemma [1]. More explicitly, let $X_q^{(i)}$ denote the ith iterate of the fixpoint iteration for the constraint system for $i \geq 0$. By induction on i, it can be verified that $X_q^{(i)}$ equals the maximal common prefix of all s such that $q : t \to s$ for trees $t \in \mathcal{T}_\Sigma$ of depth less than i. Thereby, the prefixes $X_q^{(i+1)}$ can be determined from the preixes $X_q^{(i)}$ in polynomial time. This is obvious for *linear* transducers A. When A is *uniform copying*, and general uniform patterns are used, polynomial time can be obtained when trees are represented as *dags* where isomorphic subtrees are represented only once. Since the number of iterations required for reaching the least fixpoint of the constraint system is bounded by the size of the transducer A, the overall complexity statement follows. ◀

Now let A denote some *uc*-transducer (*linear* transducer) A with advice and a non-empty set of states. In particular, the axiom of A contains an occurrence of some state q_0 (with $\iota(q_0) = h_0$). We call A an earliest *uc*-transducer (*linear* transducer), if $\mathsf{pref}_A(q) = x_1$ ($\mathsf{pref}_A^{(1)}(q) = x_1$) for all states q of A. If this is not yet the case, we construct a transducer A' of the same kind as A as follows where we only present the construction for *uc* transducers (the linear case is analogous). The set Q' of states of A' is obtained from the set Q of states of A by $Q' = \{q \in Q \mid \mathsf{pref}_A(q) \notin \mathcal{T}_\Delta\}$. Assume that the axiom T_0 of A equals $T_0 = p \cdot q_0(x_1)$. If $\mathsf{pref}_A(q_0) = s \in \mathcal{T}_\Delta$, then the axiom T_0' of A' is given by $T_0' = p \cdot s$. Otherwise, the new axiom T_0' is given by $T_0' = p \cdot \mathsf{pref}_A(q_0) \cdot q_0(x_1)$. Now assume that $q \in Q'$, and $\mathsf{pref}_A(q) = u$. Then for each rule $q(f(x_1 : h_1, \ldots, x_k : h_k)) \to p\{x_j \mapsto q_j(x_j) \mid j \in J\}$ of A, A' has a rule $q(f(x_1 : h_1, \ldots, x_k : h_k) \to T'$ where T' is defined as follows. For $j \in J$, let $s_j = \mathsf{pref}_A(q_j)$ if $\mathsf{pref}_A(q_j) \in \mathcal{T}_\Delta$, and $s_j = u_j \cdot q_j(x_j)$ if $\mathsf{pref}_A(q_j) = u_j \in \mathcal{T}_\Delta(x_1)$. Then u must be a prefix of $p\{x_j \mapsto s_j \mid j \in J\}$, and we choose T' such that $p\{x_j \mapsto s_j \mid j \in J\} = u \cdot T'$ holds.

▶ **Lemma 5.** *Assume that A is a DT^A_{uc} and A' the DT^A_{uc} as constructed above. Then,*

1. *For each state q of A' with $u = \mathsf{pref}_A(q)$, it holds that*
   - **a.** *If $q : t \to s$ holds for A, then $q : t \to s'$ holds for A' for some $s' \in \mathcal{T}_\Delta$ so that $s = u \cdot s'$, and vice versa,*
   - **b.** *If $q : t \to s'$ holds for A', then $q : t \to u \cdot s'$ holds for A.*
   - **c.** *A and A' are equivalent where $\mathsf{pref}_{A'}(q) = x_1$ holds for all states q of A'.*
2. *A' can be constructed from A in polynomial time.*

The analogous properties hold for DT^A_{lin} transducers.

Due to this lemma, we obtain that for each *uc*-transducer (*linear* transducer) an equivalent earliest transducer can be constructed in polynomial time. In fact that transducer can further be *minimized*. For that, we define $\equiv$ as the coarsest equivalence relation on states such that $q \equiv q'$ implies that $\iota(q) = \iota(q')$, and for each input symbol $f \in \Sigma$ there is a rule $q(f(x_1 : h_1, \ldots, x_k : h_k)) \to T$ iff there is a rule $q'(f(x_1 : h_1, \ldots, x_k : h_k)) \to T'$ such that $T = p\{x_j \mapsto q_j(x_j) \mid j \in J\}$ and $T' = p\{x_j \mapsto q'_j(x_j) \mid x_j \in X_J\}$ for some common pattern $p \in \mathcal{T}_\Delta(X_J)$ and states $q_j, q'_j \in Q$ such that for all $j \in J$, $q_j \equiv q'_j$ holds.

▶ **Lemma 6.** *Let A be an earliest* uc*-transducer (*linear *transducer) and $\equiv$ the equivalence relation as defined above. Then the following holds:*

1. *$q \equiv q'$ iff for all $[\![q]\!]_A = [\![q']\!]_A$;*
2. *$\equiv$ can be constructed in polynomial time.*

The proof of Lemma 6 follows closely the corresponding proof of Theorem 13 of [8]. Putting Lemmas 5 and 6 together, we obtain:

▶ **Theorem 7.** *For each DT^A_{uc} (DT^A_{lin}) transducer A, a unique canonical earliest DT^A_{uc} (DT^A_{lin}) transducer A' can be constructed such that (1) A' has at most as many states as A, (2) A' is equivalent to A, and (3) A' can be constructed in polynomial time.*

4 How to Remove Look-ahead

In the following, we assume that we are given a deterministic top-down tree transducer A with regular look-ahead. By Theorem 7, we may assume that A is earliest. Our goal is to decide whether the translation of A can be realized by a deterministic top-down tree transducer *without* look-ahead (but with inspection). A necessary condition for the latter is that the domain of the given translation can be accepted by a top-down deterministic automaton. By assumption, the domain of the translation of A is given by the set $\mathcal{L}(B)$ of all trees accepted by B. If the translation can be realized by a *uc*-transducer with inspection only, $\mathcal{L}(B) = \mathcal{L}(B')$ for some top-down deterministic automaton B'. One such B' can be obtained by means of the *powerset* construction. The set H' of states of B' are subsets of states of B where in particular, $\{h_0\} \in H'$ is the accepting state. Moreover, if $S \subseteq H$ is a state in H', then for every input symbol $f \in \Sigma_k$ and every $j \in \{1, \ldots, k\}$,

$$S_j = \{h_j \in H \mid \exists h \in S,\ h_1, \ldots, h_{j-1}, h_{j+1}, \ldots, h_k \in H.\, \langle h, f, h_1 \ldots h_k\rangle \in \delta\} \ \in\ H'$$

and $\langle S, f, S_1 \ldots S_k\rangle$ is in the transition relation of B'. As B is assumed to be trim, the automaton B' constructed in this way, is trim as well. Checking whether or not $\mathcal{L}(B) = \mathcal{L}(B')$ is decidable. In fact, the two automata are equivalent iff for each transition $\langle S, f, S_1 \ldots S_k\rangle$ constructed for B', and every tuple of states $(h_1, \ldots, h_k) \in S_1 \times \ldots \times S_k$ there is some $h \in S$ such that $\langle h, f, h_1 \ldots h_k\rangle$ is a transition of B.

Let us thus assume that B and B' are equivalent. The following construction is given for uniform copying transducers (the construction for linear transducers is analogous). Let us assume that the earliest $\mathsf{DT}^{\mathrm{R}}_{\mathrm{uc}}$ transducer A is canonical and equivalent to a $\mathsf{DT}^{\mathrm{I}}_{\mathrm{uc}}$ transducer. As the case where the axiom T_0 of A is ground, is trivial, we now assume that the axiom is non-ground, i.e., $T_0 = s_0 \cdot q_0(x_1)$ with $\iota(q_0) = h_0$. Then we construct a $\mathsf{DT}^{\mathrm{I}}_{\mathrm{uc}}$ transducer A' as follows. The states of A' are given by $\langle\rho\rangle$ for mappings ρ which assign to the states h in some set $S \in H'$, trees $\rho(h)$ which are either in $\mathcal{T}_\Delta$ or of the form $\rho(h) = s \cdot q(x_1)$ where $s \in \mathcal{T}_\Delta(x_1)$ and $q \in Q$ is a state of A with $\iota(q) = h$. For that mapping ρ, we define $\iota'(\langle\rho\rangle) = S$. The axiom of A' is given by $T'_0 = s_0 \cdot \langle\{h_0 \mapsto q_0(x_1)\}\rangle$.

Assume now that $\langle\rho\rangle$ is a state of A' with domain S. Consider some input symbol $f \in \Sigma$ of rank $k \geq 0$ where $\langle S, f, S_1 \ldots S_k\rangle$ is a transition of B'. Let p' denote a pattern in $\mathcal{T}_\Delta(X_{J'})$ for some $J' \subseteq \{1, \ldots, k\}$. Let ρ_i, $i = 1, \ldots, k$ be mappings with domains S_i such that for $h \in S$ and $\langle h, f, h_1 \ldots h_k\rangle \in \delta$,

1. If $\rho(h) \in \mathcal{T}_\Delta$, then $h_j \in S_j$ for all $j \in \{1, \ldots, k\}$, and $\rho(h) = p'\{x_j \mapsto \rho_j(h_j) \mid j \in J'\}$;
2. If $\rho(h) = s \cdot q(x_1)$, and A has a rule of the form (1), then $J \subseteq J'$ and $s \cdot p = p'\{x_j \mapsto u_j \mid j \in J'\}$ where $u_j = \rho_j(h_j)$ if $\rho_j(h_j)$ is ground, and $u_j = \rho_j(h_j) \cdot x_j$ otherwise.
3. For each $j \in J'$, the mapping ρ_j is (up to states in Q) prefix-free, i.e., the longest common prefix of $\rho_j(h), h \in S_j$, in $\mathcal{T}_\Delta(x_1)$ is x_1.

If A is equivalent to some $\mathsf{DT}_{\mathrm{uc}}$ transducer, p' and ρ_j with these properties must always exist, and then are uniquely defined.

▶ **Example 8.** Assume that $\rho = \{h_1 \mapsto f(a, g(c)), h_2 \mapsto f(b, g(c)), h_3 \mapsto f(a, b), h_4 \mapsto f(b, b), h_5 \mapsto c\}$ and for the binary input symbol f, B has the transitions $\langle h_1, f, h_a h_c\rangle, \langle h_2, f, h_b h_c\rangle, \langle h_3, f, h_a h_b\rangle, \langle h_4, f, h_b h_b\rangle$ while there is no transition for f resulting in state h_5. By comparing the outputs for h_1 and h_2, we identify the subtrees a and b whose outputs cannot be decided depending on the input symbol f alone, but require information about the first child of f. Likewise, by comparing the outputs for h_3 and h_4, we identify the corresponding subtrees $g(a)$ and b whose outputs can be discriminated only depending on the second child of f in the input. Accordingly, the pattern is given by $p' = f(x_1, x_2)$ where $\rho_1 = \{h_a \mapsto a, h_b \mapsto b\}$ and $\rho_2 = \{h_c \mapsto g(c), h_b \mapsto b\}$.

Example 8 illustrates the perhaps most complicated case, namely, when all outputs stored in ρ are ground. Given that J', p' and $\rho_j, j \in J'$, with the given properties exist, we add to A' the states $\langle\rho_j\rangle, j \in J'$, together with the rule

$$\langle\rho\rangle(f(x_1 : S_1, \ldots, x_k : S_k)) \to p'\{x_j \mapsto \langle\rho_j\rangle(x_j) \mid j \in J'\}. \tag{4}$$

The resulting transducer is a $\mathsf{DT}^{\mathrm{I}}_{\mathrm{uc}}$ transducer A' which is equivalent to A. Now assume that the construction successfully terminates. The following two lemmas summarize the properties of the resulting transducer A'.

▶ **Lemma 9.** *Consider a state $\langle\rho\rangle$ of A' for some mapping ρ with domain S, $h \in S$ and $t \in \mathcal{T}_\Sigma$ with $h : t$.*

1. *If $\rho(h) = u$ is ground, then $\langle\rho\rangle : t \to u$;*
2. *If $\rho(h) = u \cdot q(x_1)$ for some $u \in \mathcal{T}_\Delta(x_1)$, and $q : t \to s$, then $\langle\rho\rangle : t \to u \cdot s$.*

▶ **Lemma 10.** *Assume that $t \in \mathcal{C}_\Sigma$ is a context where $(S_0, S) : t$ holds in B' for $S_0 = \{h_0\}$.*

1. *$S = \{h \in H \mid \exists s_h. (q_0, h) : t \to s_h\}$, i.e., S is the set of all h such that there is a (q_0, h)-computation of A for t;*
2. *Assume that for each $h \in S$, $(q_0, h) : t \to s_h$. Then $(\langle\rho_0\rangle, S) : t \to s$ for $\rho_0 = \{h_0 \mapsto q_0(x_1)\}$ so that the following holds:*

- *If s is ground, then s_h is ground for each $h \in S$, and $s = s_h$ holds for each $h \in S$;*
- *If $s = s' \cdot \langle\rho\rangle(x_1)$, then ρ is a mapping with domain S, and for each $h \in S$,*
  - *$\rho(h)$ is ground iff s_h is ground where $s' \cdot \rho(h) = s_h$.*
  - *If $\rho(h) = u_h \cdot q(x_1)$ for some u_h, then $s' \cdot u_h \cdot q(x_1) = s_h$.*

The proof of these two lemmas is by induction on the structure of t, following the definition of A'. We conclude that, upon successful termination, A and A' are equivalent. It thus suffices to prove successful termination whenever A is equivalent to *some* DT_{uc} transducer.

We introduce three properties. The $\mathsf{DT}^{\text{R}}_{\text{uc}}$ transducer A is called *synchronizing*, if for every input tree $t_0 \in \mathcal{C}_\Sigma$, input symbol $f \in \Sigma$ of rank $k \geq 0$ and look-ahead states h'_1, h'_2 so that $(q_0, h_i) : t \to s_i \cdot q_i(x_1)$ for $i = 1, 2$, the following holds. Let $q_i(f(x_1 : h_{i,1}, \ldots, x_k : h_{i,k})) \to T_i$ be rules of A according to (1) where both T_1 and T_2 are non-ground. Then one of the following two cases occurs.

Case 1. There are patterns $p_1, p_2 \in \mathcal{T}_\Delta(X_J)$ which agree in their sets of occurring variables, and factorizations $T_i = p_i\{x_j \mapsto u_{i,j} \mid j \in J\}$ for $i = 1, 2$ such that

- $s_1 \cdot p_1 = s_2 \cdot p_2$,
- for each $j \in J$, each $u_{i,j}$ is either ground, or is of the form $s' \cdot q'(x_j)$ for suitable s', q'.

Case 2. There is some j such that both $T_1 = s'_1 \cdot q'_1(x_j)$ and $T_2 = s'_2 \cdot q'_2(x_j)$ for suitable $s'_1, s'_2 \in \mathcal{T}_\Delta(x_1)$ and states $q'_1, q'_2 \in Q$.

Secondly, the $\mathsf{DT}^{\text{R}}_{\text{uc}}$ transducer A is called *twinning*, if the following holds for all states q_1, q_2 and contexts $t, t' \in \mathcal{C}_\Sigma$ such that $(q_0, \iota(q_i)) : t \to s_i \cdot q_i(x_1)$ and $(q_i, \iota(q_i)) : t' \to s'_i \cdot q_i(x_1)$.

- Either $s'_1 = s'_2 = x_1$,
- or there are trees $u, v \in \mathcal{T}_\Delta(x_1)$ such that $s_1 = s_2 \cdot w$, $s'_1 = v \cdot w$ and $s'_2 = w \cdot v$ or vice versa, $s_2 = s_1 \cdot w$, $s'_2 = v \cdot w$ and $s'_1 = w \cdot v$ for suitable $w, v \in \mathcal{T}_\Delta(x_1)$.

Finally, the $\mathsf{DT}^{\text{R}}_{\text{uc}}$ transducer A is called *erasing*, if the following holds for all input trees $t, t' \in \mathcal{C}_\Sigma$ and states $h_1, h_2 \in H$. Assume that $(q_0, h_i) : t \to s_i$ for $i = 1, 2$ where s_1 is of the form $s'_1 \cdot q(x_1)$ (thus, $\iota(q) = h_1$) and s_2 is ground. Then $(q, h_1) : t' \to u \cdot q(x_1)$ for some $u \in \mathcal{T}_\Delta(x_1)$ and $(h_2, h_2) : t'$ implies that $u = x_1$.

The *variation* $\|t_1, t_2\|$ of $t_1, t_2 \in \mathcal{T}_\Delta(x_1)$ is the minimal depth of u_1, u_2 such that $t_i = t_0 \cdot u_i, i = 1, 2$ for a $t_0 \in \mathcal{T}_\Delta(x_1)$. The $\mathsf{DT}^{\text{R}}_{\text{uc}}$ transducer A has *bounded variation* if $\exists K \geq 0$ such that for every $t \in \mathcal{C}_\Sigma$ and $h_1, h_2 \in H$ with $(q_0, h_i) : t \to s_i$ for $i = 1, 2$, $\|s'_1, s'_2\| \leq K$ holds. – Assume that A is synchronizing, erasing and twinning. Then the outputs of any two computations for the same input tree, cannot not differ much. Intuitively, the variation is synchronized at branching rules, does not increase in monadic loops and may increase only marginally once one of the outputs is ground. Let us define the *size* $|A|$ of some $\mathsf{DT}^{\text{A}}_{\text{uc}}$ A as the sum of the sizes of all rules of A where the size of the rule (1) is $k + 1$ plus the number on nodes in the right-hand side. Altogether, we prove:

▶ **Lemma 11.** *Assume that the domain of A is top-down deterministic where the bottom-up deterministic look-ahead automaton B has $m \geq 1$ states.*

1. *If A is equivalent to some DT^{I}_{uc} A', then A is synchronizing, erasing and twinning.*
2. *If the DT^{R}_{uc} A is synchronizing, erasing and twinning, then A has bounded variation where the bound is given by $|A| \cdot (|A| + m)$.*

Proof. The proof that A then must be twinning follows along the same lines as for word transducers. Here, we only consider synchronization. Assume that A is equivalent to some $\mathsf{DT}^{\text{I}}_{\text{uc}}$ A', and $(\{h_0\}, S) : t$ for some $t \in \mathcal{C}_\Sigma$ and state S of B'. Then $(\bar{q}, S) : t \to s_0 \cdot \bar{q}(x_1)$ holds for the initial state $\bar{q}_0$ of A' and some state $\bar{q}$ of A' where $\iota(\bar{q}_0) = \{h_0\}$ and $\iota(\bar{q}) = S$. Assume that there is a transition $\bar{q}(f(\ldots)) \to T$ of A'. Consider any $h \in S$ so that $\langle h, f, h_1 \ldots h_k\rangle$ is a transition of B, and assume that $(q_0, h) : t_0 \to s \cdot q(x_1)$ for some $s \in \mathcal{T}_\Delta(x_1)$. Since A is assumed to be earliest, we have that $s_0 \cdot u_0 = s$ for some $u_0 \in \mathcal{T}_\Delta(x_1)$. Moreover, there is a rule $q(f(x_1 : h_1, \ldots, x_k : h_k)) \to T'$ of A for f. We consider three cases.

1. T is ground. Then T' is ground with $s_0 \cdot T = s \cdot T' = s_0 \cdot u_0 \cdot T'$. Consequently by top-cancellation, $T = u_0 \cdot T'$ holds.
2. T contains occurrences of a single variable x_j only, i.e., is of the form $p \cdot \bar{q}'(x_j)$ for some state $\bar{q}'$ of A'. If T' is ground, then $s_0 \cdot p \cdot v = s \cdot T' = s_0 \cdot u \cdot T'$ for some $v \in \mathcal{T}_\Delta$, i.e., $p \cdot v = u_0 \cdot T'$. If T' in non-ground, it necessarily is of the form $T' = p' \cdot q'(x_j)$ where (again by top-cancellation) $p \cdot u' = u_0 \cdot p'$ holds for some $u' \in \mathcal{T}_\Delta(x_1)$.
3. T contains occurrences of variables $x_{j_1} \neq x_{j_2}$. Let J denote the set of indices j so that x_j occurs in T. Then $T = u_0 \cdot p\{x_j \mapsto \bar{q}_j(x_j) \mid j \in J\}$ holds for some $p \in \mathcal{T}_\Delta(X_J)$ so that $T' = p\{x_j \mapsto u_j \mid j \in J\}$ for some u_j which are either ground or of the form $s_j \cdot q_j(x_j)$ with $q_j \in Q$.

Now assume that we have $(q_0, h_i) : t_0 \to s_i \cdot q_i(x_1)$, and let $(\bar{q}_0, S) : t \to s_0 \cdot \bar{q}(x_1)$ denote the corresponding computation of the $\mathsf{DT}^{\mathrm{I}}_{\mathrm{uc}}$ A'. In particular, this means that $h_1, h_2 \in S$, and we can apply the observations listed above. Let $q_i(f_i(x_1 : h_{i,1}, \ldots, x_k : h_{i,k}) \to T_i, i = 1, 2$, be rules of A for q_i and f such that T_1, T_2 are both non-ground. Then there also must be a rule $\bar{q}(f(\ldots)) \to T$ of A' where T is not ground as well, i.e., the first of the three cases does not apply. Now, assume that the monadic second case of the synchronization property also does not apply. Then T contains at least two variables, and there are factorizations $T = u_i \cdot p_i$ for $i = 1, 2$ so that $T_i = p_i\tau_i$ for substitutions of τ_i mapping each x_j to some ground tree or expression $s_{i,j} \cdot q_{i,j}(x_j)$ where $s_i = s_0 \cdot u_i$. We conclude that

$$s_1 \cdot p_1 = s_0 \cdot u_1 \cdot p_1 = s_0 \cdot u_2 \cdot p_2 = s_2 \cdot p_2$$

holds. Finally, consider some j where $h_{1,j} = h_{2,j} = h'$ for some h'. Assume for a contradiction that $\tau_1(h') \neq \tau_2(h')$, and consider any input tree t' so that $h' : t'$. Since $h_1 \neq h_2$ holds, some $j' \neq j$ exists so that $h_{1,j'} \neq h_{2,j'}$ holds. In particular, this means that the right-hand side T of ρ for f contains an occurrence of $\bar{q}'(x_j)$ for some state $\bar{q}'$ of A'. Then $\bar{q}' : t' \to s'$ for some ground tree s'. If $\tau_i(h) = v_i \cdot q'_i(x_j)$ for states q'_i of A, then $q'_i : t' \to s'_i$ with $s' = v_i \cdot s'_i$. Now since A is earliest, it follows that $v_1 = v_2$ must hold while q'_1 and q'_2 are equivalent, as their outputs coincide for each input. Since A is canonical, this further means that $q'_1 = q'_2$. Likewise, if $\tau_1(h') = s'_1$ is ground, then necessarily $\tau_2(h')$ also must be ground and coincide. Thus, the synchronization property follows.

It remains to prove the second assertion of lemma, namely, that every $\mathsf{DT}^{\mathrm{R}}_{\mathrm{uc}}$ A which is synchronizing, erasing and twinning, has a variation bounded by $|A| \cdot (|A| + m)$. Let B' denote the top-down deterministic automaton accepting the domain of A, and assume for a contradiction that $t \in \mathcal{C}_\Sigma$ is a context with a minimal number of nodes violating the claim of the lemma. Let S denote the state of B' such that $(\{h_0\}, S) : t$ holds. For $h_1, h_2 \in S$, assume that $(h_f, h_\nu) : t \to s_\nu$, $\nu = 1, 2$, holds for A. Assume that $t = t_1 \cdot \ldots \cdot t_m$ where $t_i = f_i(u_{i,1}, \ldots, u_{i,j_i-1}, x_1, u_{i,j_i+1}, \ldots, u_{i,k_i})$ for some some $1 \leq j_i \leq k_i$, some $f_i \in \Sigma_{k_i}$ and ground trees $u_{i,j'}, j' \neq j_i$. Let $h_{i,0}, \ldots, h_{i,m}$ states of B so that $(h_{i-1}, h_i) : t_i$ holds for $i = 1, \ldots, m$. Clearly, if $m = 0$, $s_1 = s_2 = x_1$ and the assertion holds. First, we consider the case that there is a maximal $m' \leq m$ where $(q_0, h'_m) : t_1 \ldots t_{m'-1} \to s'_i \cdot q_i(x_0)$ holds such that $(q_i, h_{i,m'}) : p_i\tau_i$ for some $p_1, p_2 \in \mathcal{T}_\Delta(X_J)$ and substitutions τ_i, where

- $s'_1 \cdot p_1 = s'_2 \cdot p_2$;
- $\tau_1(x_{j'}) = \tau_2(x_j) \in \mathcal{T}_\Delta$ for $j' \neq j_{m'}$;
- $\tau_i(x_{j_{m'}})$ is of the form $v'_i \cdot q'_i(x_1)$ where $v'_i \cdot q'_i(x_{j_{m'}})$ is a subtree of some right-hand side of A.

If $m' = m$, then obviously, the claim of the lemma holds. Therefore, either there is no such m' or $m' < m$. Assume that u is obtained from p_1 by substituting x_j with $\tau_1(x_j)$ for all $j \neq j_{m'}$. Thus, $s' = s'_1 \cdot u$ is a common prefix of s_1 and s_2. In case that the properties above are never satisfied, we set $m' = 0$, $s' = x_1$, and let $v'_i \cdot q'_i(x_1)$ equal $q_i(x_1)$.

Assume that $m'' \geq m'$ is chosen maximal so that $(q'_i, h_{i,m''}) : t_{m'+1} \dots t_{m''} \to s''_i \cdot q''_i(x_i)$, i.e., both outputs are non-ground. The variation for the context $t_1 \cdot \ldots \cdot t_{m''}$ then is given by $\|v'_1 \cdot s''_1 \cdot q''_1(x_1), v'_2 \cdot s''_2 \cdot q''_2(x_1)\|$. We claim that this variation is bounded by $|A|^2$. Assume for a contradiction that this were not the case. Then due to the synchronization property and the choice of m', this implies that $m'' - m' > n^2$ where n is the number of states of A. Therefore, at least one pair of states occurs at least twice. But then, due to the twinning property, the same variation is attained with a smaller context – contradicting the minimality of t.

To continue with our argument, we conclude that m'' must be less than m. By the definition of m'' this means that one of the right-hand sides chosen for q''_i and $f_{m''+1}$ must be ground. W.l.o.g., assume that this is the right-hand side T_1 for q''_1. But then due to the erasing property, the depth of the output for $t_{m''+1} \cdot \ldots \cdot t_m$ is bounded by $|A| \cdot m$. Altogether therefore, the variation is bounded by $|A|^2 + |A| \cdot m = |A| \cdot (|A| + m)$ – in contradiction to our assumption. This concludes the proof. ◀

In summary, we obtain:

▶ **Theorem 12.** *Let A be a DT^R_{uc}. It is decidable whether or not the translation of A can be realized by a DT^I_{uc}, and if so an equivalent DT^I_{uc} A' can be constructed.*

A corresponding theorem also holds for $\mathsf{DT}^{\mathrm{R}}_{\mathrm{lin}}$ transducers.

5 How to Inspect Top-Down Deterministic Languages

Now consider a $\mathsf{DT}^{\mathrm{I}}_{\mathrm{uc}}$ A with underlying top-down deterministic automaton B which is assumed to be canonical earliest. For the following, we denote the unique state h of B with $\mathsf{dom}_B(h) = \mathcal{T}_\Sigma$ (given that there is such a state), by $\top$. The $\mathsf{DT}^{\mathrm{I}}_{\mathrm{uc}}$ ($\mathsf{DT}^{\mathrm{I}}_{\mathrm{lin}}$) A is *without inspection* (denoted by $\mathsf{DT}_{\mathrm{uc}}$ and $\mathsf{DT}_{\mathrm{lin}}$) if for every rule $q(f(x_1 : h_1, \dots, x_k : h_k) \to T$ of A, $h_j = \top$ whenever x_j does not occur in T. When B does not have a state $\top$, then A is without inspection only if the right-hand side of every rule of A contains all variables x_j occurring in its left-hand side, i.e., A is *non-deleting*. Note that a $\mathsf{DT}_{\mathrm{uc}}$ can easily be changed in such a way that no advice automaton is present at all (and still the same translation is realized).

Consider a fixed output tree $s \in \mathcal{T}_\Delta$. A language $L \subseteq \mathcal{T}_\Sigma$ is called $\mathsf{DT}_{\mathrm{uc}}$ ($\mathsf{DT}_{\mathrm{lin}}$) *output recognizable by* s iff there is a $\mathsf{DT}_{\mathrm{uc}}$ ($\mathsf{DT}_{\mathrm{lin}}$) A such that $[\![t]\!]_A$ is defined iff $t \in L$, where $[\![t]\!]_A = s$ for all $t \in L$. It turns out that a language is output recognizable via some $\mathsf{DT}_{\mathrm{uc}}$ iff it is output recognizable via some $\mathsf{DT}_{\mathrm{lin}}$. Hence, we drop the qualification. The language L is called *output recognizable* (without further mentioning of an output tree) if L is out recognizable by some $s \in \mathcal{T}_\Delta$. Assume that the language L is accepted by the trim top-down deterministic TA B. Then it can be decided in *polynomial time* whether or not L is output recognizable, and if so, whether or not L is output recognizable by a particular given tree s.

▶ **Lemma 13.** *$\mathcal{L}(B)$ is output recognizable iff for every strongly connected component H' of the transition relation of B, every transition $\langle h', f, h_1 \dots h_k\rangle$ of B, and i with $h', h_i \in H'$, it holds that $h_j = \top$ for all $j \neq i$.*

Let s denote any output tree in $\mathcal{T}_\Delta$ and h a state of B. Then $\mathsf{dom}_B(h)$ is output recognizable by s if for every transition $\langle h, f, h_1 \dots h_k\rangle$ with subsequence $h_{i_1} \dots h_{i_r}$ of states different from $\top$, there is a pattern $s' \in \mathcal{T}_\Delta(X_r)$ such that $s = s'\{x_j \mapsto s_j \mid j = 1, \dots, r\}$ and $\mathsf{dom}_B(h_{i_j})$ is output recognizable by s_j for all $j = 1, \ldots, r$. In case that h and one of the h_j are contained in the same strongly connected component of B, then r must equal 1. Accordingly, then s' can be chosen as x_1. We further remark that the pattern s' can be chosen to be *linear*, i.e., each variable x_j occurs exactly once. The constructed transducer thus is in fact, a $\mathsf{DT}_{\rm lin}$. Finally we note that the set of all states h such that $\mathsf{dom}_B(h)$ is output recognizable by s, can be determined by a TA running over s.

▶ **Theorem 14.** *For a given state h of a top-down deterministic TA B,*

1. *it can be decided in polynomial time whether or not $\mathsf{dom}_B(h)$ is output recognizable;*
2. *it can be decided in polynomial time whether or not $\mathsf{dom}_B(h)$ is output recognizable by a particular tree s and if so, a DT_{lin} $A_{B,h,s}$ without inspection can be constructed in polynomial time with domain $\mathsf{dom}_B(h)$ such that $[\![t]\!]_{A_{B,h,s}} = s$ for all $t \in \mathsf{dom}_B(h)$.*

6 How to Satisfy Inspection Needs

In the following, we consider an arbitrary $\mathsf{DT}^{\rm I}_{\rm uc}$ transducer A with underlying top-down deterministic TA B as *inspection* automaton. Consider a rule τ of the form $q(f(x_1 : h_1, \ldots, x_k : h_k)) \to T$ of A. For every x_i not occurring in T, it must be verified that the corresponding subtree of the input is contained in $\mathsf{dom}_B(h_i)$. This verification is trivial if $\mathsf{dom}_B(h_i) = \mathcal{T}_\Sigma$. Such a state h_i (if present) has been denoted by $\top$. Accordingly, let J_τ denote the set of indices j such that x_j does *not* occur in the right-hand side of T while at the same time, $h_j \neq \top$. Let us thus call the *multiset* $\eta_\tau = \{h_j \mid j \in J_\tau\}$ the *inspection need* of the rule τ. Assume that T has disjoint ground subtrees $s_j, j \in J_\tau$, such that $\mathsf{dom}_B(h_j)$ is output recognizable by s_j for $j \in J_\tau$. Then the rule τ can equivalently be replaced by a rule without inspection need. In this case, we say that τ *satisfies* its inspection need.

▶ **Example 15.** Consider the rule $q(f(x_1 : h_1, x_2 : h_2)) \to g(x_1, r(b))$ where $\mathsf{dom}_B(h_2)$ equals the set $L = \{g(a,t) \mid t \in \mathcal{T}_\Sigma\}$. Then $J_\tau = \{2\}$ where the language L is output realizable with respect to $r(b)$. The latter can be seen by means of the rules $q_1(g(x_1, x_2)) \to r(q_2(x_1))$ and $q_2(a) \to b$. Accordingly, the given rule for q and f satisfies its inspection need.

Our goal is to construct for a given $\mathsf{DT}^{\rm I}_{\rm uc}$ transducer A an equivalent $\mathsf{DT}_{\rm uc}$ transducer A' such that each rule of A' satisfies its inspection need. If each rule of A satisfies its inspection need, this need no longer be the case for the *earliest* transducer equivalent to A. The reason is that some ground subtrees of prefixes of right-hand sides may have been moved to the right-hand sides of other rules. Satisfying inspection needs of rules therefore requires to partly *revert* the *earliest* transformation. In the following, we call a state q of the $\mathsf{DT}^{\rm I}_{\rm uc}$ A *constant*, if there is a single output tree s such that $s = s'$ whenever $q : t \to s'$ holds.

▶ **Lemma 16.** *For a partial mapping $\mu : \mathcal{T}_\Sigma \to \mathcal{T}_\Delta$, the following are equivalent: (1) μ is realized by a DT_{uc} without inspection; (2) μ is realized by a DT^{I}_{uc} without constant states, but where all inspection needs are satisfied.*

Assume that A' is a $\mathsf{DT}_{\rm uc}$ and A the corresponding $\mathsf{DT}^{\rm I}_{\rm uc}$ in canonical earliest normal form. This means that for each state q of A and each state $q' \in q$ of A' with $\mathsf{pref}_{A'}(q') = p$, $q' : t \to s'$ holds for A' iff $q : t \to s$ with $s' = p \cdot s$ holds. In particular, the constant outputs for some states of A' may occur as subtrees in p and thus are already produced before A processes t. In order to recover the (yet unknown) $\mathsf{DT}_{\rm uc}$ A' without inspection from A, we determine the minimal suffix p' of p so that all inspections possibly encountered when q processes its input, can be satisfied. Such a *generalized inspection need* of a q-computation is represented by a sequence $(M_1, \emptyset) \ldots (M_{r-1}, \emptyset)(M_r, \phi)$, $r \geq 0$, where $M_1, \ldots, M_r$ are multisets of inspection states and ϕ is a downward-closed subset of sub-multisets of M_r with $M_r \notin \psi$. Intuitively, a generalized inspection need is the *sequence* of future inspections yet to be simulated. The pair (M_r, ϕ) to the right is meant to occur farest in the future. Thereby, the set ϕ describes which sub-multisets of individual inspections of M_r can already be accomplished (the available ground terms might be used in more than one way). If $M_r \in \phi$, then *all* languages $\mathsf{dom}_B(h), h \in M_r$, are simultaneously realizable – implying that the whole pair (M_r, ϕ) can be dropped.

For a finite multiset G of trees in $\mathcal{T}_\Delta$ and a finite multiset M of states in H, define $\langle\langle G, M\rangle\rangle$ as the set of all sub-multisets M' of M so that the multiset of languages $\{\mathsf{dom}_B(h) \mid h \in M'\}$ are simultaneously output recognizable by means of disjoint subtrees of trees in G.

Now assume that $p \in \mathcal{T}_\Delta(x_1)$ where $p = s_1 \cdot \ldots \cdot s_m$ where each of the $s_j \in \mathcal{T}_\Delta(x_1)$ is *irreducible*, i.e., cannot be written as a product of patterns different from x_1. For p, we define the auxiliary transformation $\llbracket p\rrbracket^{\sharp\sharp}$ as the composition $\llbracket s_1\rrbracket^{\sharp\sharp} \circ \ldots \circ \llbracket s_m\rrbracket^{\sharp\sharp}$ where for an irreducible pattern $s \in \mathcal{T}_\Delta(x_1)$, the transformation is defined as follows. First, $\llbracket s\rrbracket^{\sharp\sharp}\epsilon = \epsilon$. For $\alpha = \alpha'\,(M, \phi)$, the pattern s can only be used to satisfy the inspection need of the *last* pair in α. Let G denote the set of maximal distinct ground subtrees of s. Let ϕ' denote the set of multisets of inspection needs which become satisfiable when the ground terms from G are additionally available, i.e., $\phi' = \{R_1 \cup R_2 \mid R_1 \in \phi, R_2 \in \langle\langle G, M \setminus R_1\rangle\rangle\}$. Then $\llbracket s\rrbracket^{\sharp\sharp}\alpha = \alpha'$ if $M \in \phi'$ and $\llbracket s\rrbracket^{\sharp\sharp}\alpha = \alpha'\,(M, \phi')$ otherwise. Let I the set of all possible generalized inspection needs. Let us introduce some notation. For a particular state q and input tree t, let us define the inspection need $\eta_q(t)$ of q for t as follows. Assume that $t = f(t_1, \ldots, t_k)$ and τ is the rule of A of the form $q(f(x_1 : h_1, \ldots, x_k : h_k)) \to p\{x_j \mapsto q_j(x_j) \mid j \in J\}$ such that $p \in \mathcal{T}_\Delta(X_J)$ holds with $h_i : t_i$ for $i = 1, \ldots, k$. For the rule τ, we define the transformation

$$\llbracket\tau\rrbracket^\sharp : (X_J \to I) \to I \quad \text{such that} \quad \eta_q(t) = \llbracket\tau\rrbracket^\sharp\{x_j \mapsto \eta_{q_j}(t_j) \mid j \in J\} \quad \text{holds.}$$

The transformation $\llbracket\tau\rrbracket^\sharp$ is defined by case distinction. If $J = \emptyset$, i.e., p is ground, we check in how far p itself is sufficient for η_τ to be output realizable. Let $\phi = \langle\langle\{p\}, \eta_\tau\rangle\rangle$. Then

$$\llbracket\tau\rrbracket^\sharp\emptyset = \left\{ \begin{array}{ll} \epsilon & \text{if } \eta_\tau \in \phi \\ (\eta_\tau, \phi) & \text{otherwise} \end{array} \right.$$

$$\llbracket\tau\rrbracket^\sharp\{x_j \mapsto \alpha\} = \llbracket p\rrbracket^{\sharp\sharp}((\eta_\tau, \emptyset)\alpha)$$

Finally, assume that J contains more than one index. Let $p = p'\{x_j \mapsto p_j \cdot x_j \mid j \in J\}$ for *maximal* patterns $p_j \in \mathcal{T}_\Delta(x_1)$. For $\alpha_j, j \in J$, assume that $p_j = p'_j \cdot u_j$ for some *minimal* suffix $u_j \in \mathcal{T}_\Delta(x_1)$ with $\llbracket u_j\rrbracket^{\sharp\sharp}\alpha_j = \epsilon$. These suffixes must exist, whenever A is equivalent to some $\mathsf{DT}_{\mathrm{uc}}$ without inspection. Let G denote the set of distinct maximal ground subtrees of $p'\{x_j \mapsto p'_j \cdot x_j \mid j \in J\}$, and $\phi = \langle\langle G, \eta_\tau\rangle\rangle$. Then we define

$$\llbracket\tau\rrbracket^\sharp\{x_j \mapsto \alpha_j \mid x_j \in X_J\} = \left\{ \begin{array}{ll} \epsilon & \text{if } \eta_\tau \in \phi \\ (\eta_\tau, \phi) & \text{otherwise} \end{array} \right.$$

We have:

▶ **Lemma 17.** *Assume that A is the canonical earliest normal form of some DT_{uc} A' without inspection. Let q denote some state of A, i.e., an equivalence class of states of A'. Let $q' \in q$ be state of A', let $p = \mathsf{pref}_{A'}(q')$ the maximal common prefix of outputs of A' for q', and $t \in \mathsf{dom}_B(\iota(q))$. Then (1) $\eta_q(t)$ is defined and (2) $\llbracket p\rrbracket^{\sharp\sharp}(\eta_q(t)) = \epsilon$.*

The proof is by induction on the structure of t where we use that for $q' : t \to s'$ and $q : t \to s$ we have that $s' = p \cdot s$ for $p = \mathsf{pref}_{A'}(q')$.

By computing $\eta_q(t)$, we partly recover information on the (unknown) common prefix p of the (yet to be constructed) $\mathsf{DT}_{\mathrm{uc}}$ A' for q'. By computing the set of *all* these inspection needs for q, we determine the maximal requirement on a suffix of output already produced by A when reaching q, whose delay then is sufficient to satisfy all possible future inspection needs. Therefore, let $S[q] = \{\eta_q(t) \mid t \in \mathsf{dom}_B(\iota(q)\}$. In order to calculate this set, we construct a constraint system $\mathcal{C}_I$ with unknowns X_q, where q is state of A. The system consists of one constraint per rule of A. Assume that τ is the rule $q(f(\ldots)) \to p\{x_j \mapsto q_j(x_j) \mid j \in J\}$ where $p \in \mathcal{T}_\Delta(X_J)$ and $q_j, j \in J$ are states of A. Then the constraint system $\mathcal{C}_I$ has the constraint

$$X_q \supseteq \{[\![\tau]\!]^\sharp\{x_j \mapsto \alpha_j \mid j \in J\} \mid \forall j \in J.\, \alpha_j \in X_{q_j}\}$$

The given constraint system is over subsets of I where all right-hand sides are monotonic w.r.t. subset inclusion. Therefore, it has a *least* solution. Moreover, we have:

▶ **Lemma 18.** *Assume that A is a canonical DT^I_{uc} transducer which is equivalent to some DT_{uc} A' without inspection. Let X_q, q a state of A, denote the least solution of the system of constraints $\mathcal{C}_I$. Then for each state q of A: (1) $X_q = S[q]$ and (2) the length of each $\alpha \in S[q]$ is bounded by $|A|$.*

Proof. In order to verify the first statement, we prove by induction that the ith iterate $X_q^{(i)}$ of the constraint system exactly equals the set of all $\eta_q(t)$ for trees t of depth less than i. For a proof of the second statement, we note that each $\alpha \in S[q]$ is the inspection need of some execution starting in q which must be accomplished by every pattern available at q. More precisely, for every state q of A there is a context $t \in \mathcal{C}_\Sigma$ such that $(q_0, h) : t \to u \cdot q(x_1)$ holds where $\iota(q) = h$. Here, we rely on the minimality of A, implying that each state q can be reached in this way. Let $T'_0 \cdot q_0(x_1)$ denote the axiom of A. Then t can be chosen in such a way that $T_0 \cdot u$ consists of at most $|A|$ factors. Since at least one factor of the pattern is required to realize the inspection at one rule, the upper bound to the lengthes of inspection needs $\alpha \in S[q]$ follows. ◀

As a consequence, the sets $S[q]$, q a state of A, are effectively computable.

For a finite set $S \subseteq I$, let t^S denote the minimal suffix v of t such that $[\![v]\!]^{\sharp\sharp}\alpha = \epsilon$ for all $\alpha \in S$. The states of the new $\mathsf{DT}_{\mathrm{uc}}$ A' are pairs $\langle q, s\rangle$, q a state of A and $s \in \mathcal{T}_\Delta(x_1)$ an output pattern for A, which will also be called the *buffer*. Assume that we are given for each state q of A, the (finite) set $S[q]$ of inspection needs which are to be satisfied by q-computations. Assume that $T'_0 = u \cdot v$ where $v = (T'_0)^{S[q_0]}$. Then the axiom of A' is given by $u \cdot \langle q_0, v\rangle(x_1)$. Assume that state $\langle q, u\rangle$ of A' has already been constructed, and τ is a rule of A of the form $q(f(\ldots)) \to p\{x_j \mapsto q_j(x_j) \mid j \in J\}$ where $p \in \mathcal{T}_\Delta(X_J)$.

If $J = \emptyset$, A' has a rule $\langle q, u\rangle(f(\ldots)) \to u \cdot p$. In case that $\eta_\tau \neq \epsilon$, $u \cdot p$ must be sufficient to satisfy the inspection needs incurred by the rule τ, i.e., $\langle\langle\{u \cdot t\}, \eta_\tau\rangle\rangle$ must contain η_τ.

Next assume that $J = \{j\}$, i.e., $p = p' \cdot x_j$. Then $[\![u \cdot p']\!]^{\sharp\sharp}((\eta_\tau, \emptyset)\,\alpha) = \epsilon$ must hold for all $\alpha \in S[q_j]$. Therefore, there is a factorization such that $u \cdot p' = u' \cdot v$ where $v = (u \cdot p')^{S[q_j]}$; we add the rule $\langle q, u\rangle(f(\ldots)) \to u' \cdot \langle q', v\rangle(x_i)$ to A'. Finally, assume that J contains at least two elements. Then p is of the form $p = p'\{x_j \mapsto p_j \cdot x_j \mid j \in J\}$ for $p' \in \mathcal{T}_\Delta(X_J)$ and maximal patterns $p_j \in \mathcal{T}_\Delta(x_1)$. For each $j \in J$, there must be a factorization $p_j = u_j \cdot v_j$ where $v_j = (p_j)^{S[q_j]}$. Moreover, the subset G of ground subtrees of u, p' and $u_j, j \in J$, must be sufficient to satisfy the inspection need of τ itself, i.e., $\langle\langle G, \eta_\tau\rangle\rangle$ contains η_τ. Then add the rule $\langle q, u\rangle(f(\ldots)) \to u \cdot p'\{x_j \mapsto u_j \cdot \langle q_j, v_j\rangle(x_j) \mid j \in J\}$ to A'. Correctness and termination of the construction follows from the following lemma.

▶ **Lemma 19.** *Assume that A is the canonical earliest DT^{I}_{uc} for a DT_{uc} A'' without inspection. Then the construction in Section 6 terminates with some A' so that the following properties are satisfied:*

1. *A is equivalent to A';*
2. *Each inspection need η_τ of A' is satisfiable;*
3. *For each constructed state $\langle q, u\rangle$ of A', $u \in \mathcal{T}_\Delta(x_1)$ has length at most $|A|^2 \cdot (a-1)$ if a is the maximal rank of an input symbol.*

Proof. Assume that A is equivalent to some A'' without inspection, i.e., is the canonical earliest normal form of A''. This means that for states q_1, q_2 of A, $t_1 \in \mathcal{C}_\Sigma$ and $t_2 \in \mathcal{T}_\Sigma$ with $(q_1, \iota(q_2)) : t_1 \to s_1 \cdot q_2(x_1)$, and $q_2 : t_2 \to s_2$ and all states q'_1, q'_2 of A'' so that $q'_i \in q_i$, $(q'_1, \iota(q'_2)) : t \to s'_1 \cdot q'_2(x_1)$ and $q'_2 : t \to s'_2$, for some s'_1, s'_2, it holds that $v_1 \cdot s_1 = s'_1 \cdot v_2$ and $v_2 \cdot s_2 = s'_2$ for $v_i = \mathsf{pref}_{A'}(q'_i)$.

Consider a pair $\langle q, u\rangle$ returned by the construction and let $q' \in q$ denote a state of A'' contained in q with $v_{q'} = \mathsf{pref}_{A''}(q')$. Then u is a suffix of $v_{q'}$. Consequently, the set of all constructed states $\langle q, u\rangle$ is finite, and all pre-conditions at the construction of rules are met. This means that all inspection needs of the resulting transducer are satisfiable. Moreover, we have that $q : t \to s$ holds for A iff $\langle q, u\rangle : t \to s'$ holds for A' where $u \cdot s = s'$ – implying that A and A' are equivalent.

It remains to prove item (3), i.e., to provide an upper bound for the lengthes of the patterns u occurring in the construction as second components of states $\langle q, u\rangle$ – not in terms of the transducer A'', but in terms of the transducer A, which serves as the input to the construction. According to thr construction, we know that u is a minimal pattern to satisfy all generalized inspection needs in $S[q]$.

First, assume that $(q, \iota(q)) : t \to s \cdot q(x_1)$ holds for A some context $t \in \mathcal{C}_\Sigma$ so that only rules τ are applied whose right-hand sides have occurrences of single variables only. In that case, $t = t_1 \cdot \ldots \cdot t_m$ for irreducible contexts $t_i \in \mathcal{C}_\Sigma$ where each computation for t_i consists in the application of a single rule τ_i. *Case 1*: $s = x_1$. Then η_{τ_i} must be $\emptyset$ for all $i = 1, \ldots, m$. *Case 2*: $s \neq x_1$. Then $u \cdot s = u_1 \cdot v$ so that all inspections to be satisfied by the τ_i together can be satisfied by u_1 while v is sufficient to satisfy all generalized inspection needs in $S[q]$. Accordingly, $[\![s^{|A|\cdot(a-1)}]\!]^{\sharp\sharp}\alpha = \epsilon$ for each $\alpha \in S[q]$. Therefore, s (independently of u) is at least able to satisfy each (M, ϕ) occurring inside some $\alpha \in S[q]$.

Now assume for a contradiction, that a state $\langle q, u\rangle$ is constructed where u is of length exceeding $|A|^2 \cdot (a-1)$. Then there is a minimal context $t \in \mathcal{C}_\Sigma$ of the form $t = t' \cdot t_1 \cdot \ldots \cdot t_N \cdot t''$ for $N = |A|^2 \cdot (a-1) + 1$ such that $(q_1, \iota(q')) : t' \to s' \cdot q'(x_1)$, $(q', \iota(q')) : t_i \to s_i \cdot q'(x_1)$ for $i = 1, \ldots, N$ with $s_i \neq x_1$, and $(q', \iota(q)) : t' \to s'' \cdot q(x_1)$ so that for some $v_0 \in \mathcal{T}_\Delta(x_1)$, one of the following two conditions holds:

- The axiom of A is of the form $s_0 \cdot v_1 \cdot q_1(x_1)$; or
- there is a right-hand side of a rule of A which is of the form $p\{x_j \mapsto v'_j \cdot q'_j(x_j) \mid j \in J\}$ for some J of cardinality exceeding 1, where $v_1 = v'_{j'}$ and $q_1 = q'_{j'}$ for some $j' \in J$.

By construction, all inspection needs along the way, are satisfied by $v_1 \cdot s'$. According to our observation above, though, u must be a suffix of $s_2 \cdot \ldots \cdot s_N \cdot s''$ – contradicting the minimality of the context t. We conclude that the maximal length of the buffer is bounded by $|A|^2 \cdot (a-1)$. ◀

In summary, we have shown:

▶ **Theorem 20.** *(1) For a DT^{I}_{uc} (DT^{I}_{lin}) A it is decidable if is equivalent to a DT_{uc} (DT_{lin}) A', and if so, such A' can be constructed. (2) For a DT^{R}_{uc} (DT^{R}_{lin}) A is decidable if A is equivalent to a DT_{uc} (DT_{lin}) A', and if so, such A' can be constructed.*

In Section 2 we have seen that a bottom-up deterministic transducer (DB) can be seen as a particularly simple $\mathsf{DT}^{\mathrm{R}}_{\mathrm{uc}}$ transducer. Accordingly, we obtain as a corollary:

▶ **Corollary 21.** *Let A be a (linear) DB. It is decidable if an equivalent DT^{I}_{uc} (DT^{I}_{lin}) or an equivalent DT_{uc} (DT_{lin}) exists, and if so, such a transducer can be constructed.*

7 Conclusion

We showed for two natural subclasses of deterministic top-down tree transducers how to remove (bottom-up deterministic) look-ahead and replace it whenever possible, with (top-down deterministic) inspection. We then also showed for the given classes how to remove inspection (if possible). The constructions are technically intricate, but crucially rely on canonical earliest normal forms for the transducers in question. As a corollary we obtain that for a given deterministic bottom-up transducer it is decidable whether or not it can be realized by a deterministic top-down tree transducer that is either *uc* or linear.

One may wonder if our results imply that for a given deterministic bottom-up tree transducer U it is decidable whether or not it can be realized by an *arbitrary* deterministic top-down tree transducer. I.e., if U *can* be realized by top-down transducer can be realized by a *uc* such transducer? Interestingly, this is *not* the case: let h_a, h_b be look-ahead states that indicate that the left-most leaf of the input tree is labeled a and b, respectively and consider a transducer which has these rules (for every $h \in \{h_a, h_b\}$):

$$q_0(f(x_1 : h_a, x_2 : h)) \to g(a, b, q_{\mathrm{id}}(x_2)) \qquad q_0(f(x_1 : h_b, x_2 : h)) \to g(c, d, q_{\mathrm{id}}(x_2))$$

The corresponding translation *can* be realized by a deterministic top-down tree transducer! However, the transducer is *not uc* (viz. the output leaves a and b must be produced by different states, but both on the input variable x_1).

References

1 Krzysztof R. Apt and Gordon D. Plotkin. Countable nondeterminism and random assignment. *J. ACM*, 33(4):724–767, 1986. doi:10.1145/6490.6494.

2 Marie-Pierre Béal and Olivier Carton. Determinization of transducers over finite and infinite words. *Theor. Comput. Sci.*, 289(1):225–251, 2002. doi:10.1016/S0304-3975(01)00271-7.

3 Jean Berstel. *Transductions and context-free languages*, volume 38 of *Teubner Studienbücher : Informatik*. Teubner, 1979. URL: http://www.worldcat.org/oclc/06364613.

4 Adrien Boiret, Aurélien Lemay, and Joachim Niehren. Learning top-down tree transducers with regular domain inspection. In *Proceedings of the 13th International Conference on Grammatical Inference, ICGI 2016, Delft, The Netherlands, October 5-7, 2016*, pages 54–65, 2016. URL: http://proceedings.mlr.press/v57/boiret16.html.

5 Christian Choffrut. Une caracterisation des fonctions sequentielles et des fonctions sous-sequentielles en tant que relations rationnelles. *Theor. Comput. Sci.*, 5(3):325–337, 1977. doi:10.1016/0304-3975(77)90049-4.

6 Joost Engelfriet. Bottom-up and top-down tree transformations - A comparison. *Mathematical Systems Theory*, 9(3):198–231, 1975. doi:10.1007/BF01704020.

7 Joost Engelfriet. Top-down tree transducers with regular look-ahead. *Mathematical Systems Theory*, 10:289–303, 1977. doi:10.1007/BF01683280.

8 Joost Engelfriet, Sebastian Maneth, and Helmut Seidl. Deciding equivalence of top-down XML transformations in polynomial time. *J. Comput. Syst. Sci.*, 75(5):271–286, 2009. doi:10.1016/j.jcss.2009.01.001.

9 Joost Engelfriet, Sebastian Maneth, and Helmut Seidl. Look-ahead removal for total deterministic top-down tree transducers. *Theor. Comput. Sci.*, 616:18–58, 2016. doi:10.1016/j.tcs.2015.12.005.

10 Emmanuel Filiot, Sebastian Maneth, Pierre-Alain Reynier, and Jean-Marc Talbot. Decision problems of tree transducers with origin. *Inf. Comput.*, 261(Part):311–335, 2018. doi:10.1016/j.ic.2018.02.011.

11 Zoltán Fülöp and Andreas Maletti. Linking theorems for tree transducers. *J. Comput. Syst. Sci.*, 82(7):1201–1222, 2016. doi:10.1016/j.jcss.2016.05.003.

12 Aurélien Lemay, Sebastian Maneth, and Joachim Niehren. A learning algorithm for top-down XML transformations. In *Proceedings of the Twenty-Ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2010, June 6-11, 2010, Indianapolis, Indiana, USA*, pages 285–296, 2010. doi:10.1145/1807085.1807122.

13 William C. Rounds. Mappings and grammars on trees. *Mathematical Systems Theory*, 4(3):257–287, 1970. doi:10.1007/BF01695769.

14 James W. Thatcher. Generalized sequential machine maps. *J. Comput. Syst. Sci.*, 4(4):339–367, 1970. doi:10.1016/S0022-0000(70)80017-4.

15 James W. Thatcher. There's a lot more to finite automata theory than you would have thought. In *Proc. 4th Ann. Princeton Conf. on Informations Sciences and Systems*, pages 263–276, 1970. Also published in revised form under the title "Tree automata: an informal survey" in Currents in the Theory of Computing (ed. A. V. Aho), Prentice-Hall, 1973, 143-172.

Implicit Automata in Typed λ-Calculi I: Aperiodicity in a Non-Commutative Logic

Lê Thành Dũng Nguyễn
Laboratoire d'informatique de Paris Nord, Villetaneuse, France
Laboratoire Cogitamus (http://www.cogitamus.fr/indexen.html)
nltd@nguyentito.eu

Pierre Pradic
Department of Computer Science, University of Oxford, UK
pierre.pradic@cs.ox.ac.uk

Abstract

We give a characterization of *star-free languages* in a λ-calculus with support for *non-commutative affine types* (in the sense of linear logic), via the algebraic characterization of the former using *aperiodic monoids*. When the type system is made commutative, we show that we get *regular languages* instead. A key ingredient in our approach – that it shares with higher-order model checking – is the use of *Church encodings* for inputs and outputs. Our result is, to our knowledge, the first use of non-commutativity in implicit computational complexity.

2012 ACM Subject Classification Theory of computation → Algebraic language theory; Theory of computation → Linear logic

Keywords and phrases Church encodings, ordered linear types, star-free languages

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.135

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

Related Version Full version with appendices: https://hal.archives-ouvertes.fr/hal-02476219

Acknowledgements The initial trigger for this line of work was twofold: the serendipitous rediscovery of Hillebrand and Kanellakis's theorem by Damiano Mazza, Thomas Seiller and the first author, and Mikołaj Bojańczyk's suggestion to the second author to look into connections between transducers and linear logic. Célia Borlido proposed looking at star-free languages at the Topology, Algebra and Categories in Logic 2019 summer school.
During its long period of gestation, this work benefited from discussions with many other people: Pierre Clairambault, Amina Doumane, Marie Fortin, Jérémy Ledent, Paolo Pistone, Lorenzo Tortora de Falco, Noam Zeilberger and others that we may have forgotten to mention.
We also thank the anonymous reviewers for their constructive feedback, especially for their highly relevant pointers to the literature.

1 Introduction

A type-theoretic implicit automata theory. This paper explores connections between the languages recognized by automata and those definable in certain typed λ-calculi (minimalistic functional programming languages). It is intended to be the first in a series, whose next installments will investigate the functions computable by transducers (automata with output, see e.g. [15, 36]). Insofar as programming language theory is related to proof theory, via the Curry–Howard correspondence, we are therefore trying to *bridge logic and automata.* That said, our work does not fit in the "logics as specification languages" paradigm, exemplified by the equivalence of recognition by finite-state automata and Monadic Second-Order Logic (MSO). One could sum up the difference by analogy with the two main approaches to machine-free complexity: *implicit computational complexity (ICC)* and *descriptive complexity.*

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 135; pp. 135:1–135:20
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Both aim to characterize complexity classes without reference to a machine model, but the methods of ICC have a more computational flavor.

programming paradigm	declarative	functional
complexity classes	Descriptive Complexity	Implicit Computational Complexity
automata theory	subsystems of MSO	**this paper** (and planned sequels)

To our knowledge, very few works have looked at this kind of "type-theoretic" or "proof-theoretic" ICC for automata. Let us mention a few recent papers (that we will discuss further in §7) concerning transducers [13, 10] and multi-head automata [46, 28] and, most importantly, a remarkable result from 1996 that provides our starting point:

▶ **Theorem 1.1** (Hillebrand & Kanellakis [24, Theorem 3.4]). *A language $L \subseteq \Sigma^*$ can be defined in the* simply typed λ-calculus *by some* closed λ-term *of type* $\mathtt{Str}_\Sigma[A] \to \mathtt{Bool}$ *for some type A (that may depend on L) if and only if it is a* regular language.

Let us explain this statement. We consider a grammar of simple types with a single base type: $A, B ::= o \mid A \to B$, and use the *Church encodings* of booleans and strings:

$$\mathtt{Bool} = o \to o \to o \qquad\qquad \mathtt{Str}_\Sigma = (o \to o) \to \ldots \to (o \to o) \to o \to o$$

with $|\Sigma|$ arguments of type $(o \to o)$, where Σ is a finite alphabet. Moreover, given any other chosen type A, one can form the type $\mathtt{Str}_\Sigma[A]$ by substituting A for the ground type o:

▶ **Notation 1.2.** For types A and B, we denote by $B[A]$ the substitution $B\{o := A\}$ of every occurrence of o in B by A.

Every closed λ-term t of type $\mathtt{Str}_\Sigma$ can also be seen as a term of type $\mathtt{Str}_\Sigma[A]$. (This is a way to simulate a modicum of parametric polymorphism in a monomorphic type system.) It follows that any closed λ-term of type $\mathtt{Str}_\Gamma[A] \to \mathtt{Bool}$ in the simply typed λ-calculus defines a predicate on strings, i.e. a language $L \subseteq \Sigma^*$.

Although little-known[1], Hillebrand and Kanellakis's theorem should not be surprising in retrospect: there are strong connections between Church encodings and automata (see e.g. [45, 48, 34]), that have been exploited in particular in *higher-order model checking* for the past 15 years [2, 38, 25, 21, 23, 49]. This is not a mere contrivance: these encodings have been a canonical data representation for λ-calculi for much longer[2].

Star-free languages. We would like to extend this result by characterizing strict subclasses of regular languages, the most famous being the *star-free languages.* Recall that the canonicity of the class of regular languages is firmly established by its various definitions: regular expressions, finite automata, definability in MSO and the algebraic characterization.

▶ **Theorem 1.3** (cf. [44, §II.2.]). *A language $L \subseteq \Sigma^*$ is regular if and only if for some* finite monoid M*, some subset $P \subseteq M$ and some* monoid morphism $\varphi \in \mathrm{Hom}(\Sigma^*, M)$*, $L = \varphi^{-1}(P)$.*

Similarly, the seminal work of Schützenberger, Petrone, McNaughton and Papert in the 1960s (see [47] for a historical discussion) has led to many equivalent definitions for star-free languages, with the algebraic notion of *aperiodicity* playing a key role:

[1] See e.g. Damiano Mazza's answer to this MathOverflow question: `https://mathoverflow.net/q/296879`

[2] They were introduced for booleans and integers by Church in the 1930s, and later generalized by Böhm and Berarducci [12], see also `http://okmij.org/ftp/tagless-final/course/Boehm-Berarducci.html`. (Similar ideas appear around the same time in [31].) As for the refined encodings with linear types that we use later, they already appear in Girard's founding paper on linear logic [17, §5.3.3].

▶ **Definition 1.4.** *A monoid M is* aperiodic *when any sequence of iterated powers is eventually constant, i.e. for any $x \in M$ there exists an exponent $n \in \mathbb{N}$ such that $x^n = x^{n+1}$.*

▶ **Theorem 1.5** (cf. [47])**.** *For a language $L \subseteq \Sigma^*$, the following conditions are equivalent:*

- *L is defined by some* star-free regular expression: $E, E' ::= \varnothing \mid \{a\} \mid E \cup E' \mid E \cdot E' \mid E^c$ *where a can be any letter in Σ and E^c denotes the* complement *of E ($[\![E^c]\!] = \Sigma^* \setminus [\![E]\!]$);*
- *$L = \varphi^{-1}(P)$ for some finite and* aperiodic *monoid M, some subset $P \subseteq M$ and some monoid morphism $\varphi \in \mathrm{Hom}(\Sigma^*, M)$;*
- *L is recognized by a deterministic finite automaton whose transition monoid is* aperiodic*;*
- *L is definable in first-order logic.*

Attempting to capture star-free languages in a λ-calculus presents a serious methodological challenge: they form a strict subclass of uniform AC^0, and, as far as we know, type-theoretic ICC has never managed before to characterize complexity classes as small as this.

Non-commutative affine types. Monoids appear in typed λ-calculi when one looks at the functions from a type A to itself, i.e. at the (closed) terms of type $A \to A$. At first glance, it seems difficult indeed to enforce the aperiodicity of such monoids via a type system. For instance, one needs to rule out $\texttt{not} = \lambda b.\, \lambda x.\, \lambda y.\, b\, y\, x : \texttt{Bool} \to \texttt{Bool}$ since it "has period two": its iteration yields the sequence (modulo $\beta\eta$-conversion) $\texttt{not}, \texttt{id}, \texttt{not}, \texttt{id}, \ldots$ (where $\texttt{id} = \lambda b.\, b$) which is not eventually constant. Observe that $\texttt{not}$ essentially *exchanges* the two arguments of b; to exclude it, we are therefore led to require functions to *use their arguments in the same order that they are given in.*

It is well-known that in order to make such a *non-commutative* λ-calculus work – in particular to ensure that non-commutative λ-terms are closed under β-reduction – one needs to make the type system *affine*, that is, to restrict the duplication of data. This is achieved by considering a type system based on Girard's linear[3] logic [17], a system whose "resource-sensitive" nature has been previously exploited in ICC [20, 19]. Not coincidentally, the theme of non-commutativity first appeared in a form of linear logic *ante litteram*, namely the Lambek calculus [29], and resurfaced shortly after the official birth of linear logic: it is already mentioned by Girard in a 1987 colloquium [18].

We shall therefore introduce and use a variant of Polakow and Pfenning's Intuitionistic Non-Commutative Linear Logic [39, 40], making a distinction between two kinds of function arrows: $A \multimap B$ and $A \to B$ are, respectively, the types of affine functions and non-affine functions from A to B. Accordingly:

▶ **Definition 1.6.** *A type is said to be* purely affine *if it does not contain the "$\to$" connective.*

In our system that we call the *$\lambda\wp$-calculus*, the types of Church encodings become

$$\texttt{Bool} = o \multimap o \multimap o \qquad\qquad \texttt{Str}_\Sigma = (o \multimap o) \to \ldots \to (o \multimap o) \to (o \multimap o)$$

where $\texttt{Str}_\Sigma$ has $|\Sigma|$ arguments[4] of type $(o \multimap o)$. Setting $\texttt{true} = \lambda^\circ x.\, \lambda^\circ y.\, x : \texttt{Bool}$ and $\texttt{false} = \lambda^\circ x.\, \lambda^\circ y.\, y : \texttt{Bool}$ for the rest of the paper, we can now state our main result:

▶ **Theorem 1.7.** *A language $L \subseteq \Sigma^*$ is* star-free *if and only if it can be defined by a closed $\lambda\wp$-term of type $\texttt{Str}_\Sigma[A] \multimap \texttt{Bool}$ for some* purely affine *type A (that may depend on L).*

[3] The main difference between so-called linear and affine type systems is that the latter allow *weakening*, that is, to not use some argument. Typically, $\lambda x.\, \lambda y.\, x$ is affine but not linear while $\lambda x.\, x\, x$ is neither linear nor affine. The type system that we use in this paper is affine, not strictly linear.

[4] $o \multimap o$ occurs $|\Sigma| + 1$ times in $\texttt{Str}_\Sigma$: $|\Sigma|$ arguments plus the output.

However, if we use the *commutative* variant of the $\lambda\wp$-calculus instead, then what we get is the class of regular languages (Theorem 5.1), just as in Hillebrand and Kanellakis's theorem.

As far as we know, non-commutative type systems have never been applied to implicit complexity before (but they have been used to control the expressivity of a domain-specific programming language [26]). Previous works indeed tend to see non-commutative λ-terms (or proof nets) as static objects, and to focus on their topological aspects (e.g. [6, 51, 35]), though there is another tradition relating self-dual non-commutativity to process algebras[5] [41, 22].

Proof strategy. As usual in implicit computational complexity, the proof of Theorem 1.7 consists of a *soundness* part – "every $\lambda\wp$-definable language is star-free" – and an *extensional completeness* part – the converse implication. In our case, soundness is a corollary of the following property of the purely affine fragment of the $\lambda\wp$-calculus – what one might call the *planar*[6] *affine λ-calculus* (cf. [1, 51]):

▶ **Theorem 1.8** (proved in §3). *For any* purely affine *type A, the set of closed $\lambda\wp$-terms of type $A \multimap A$, quotiented by $\beta\eta$-convertibility and endowed with function composition ($f \circ g = \lambda^{\circ}x.\, f\,(g\,x)$), is a* finite and aperiodic monoid*.*

Extensional completeness turns out here to be somewhat deeper than the "programming exercise of limited theoretical interest" [33, p. 137] that one generally finds in ICC. Indeed, we have only managed to encode star-free languages in the $\lambda\wp$-calculus by relying on a powerful tool from semigroup theory: the *Krohn–Rhodes decomposition* [27].

Plan of the paper. After having defined the $\lambda\wp$-calculus in §2, we prove Theorem 1.7: soundness is treated in §3 and extensional completeness in §4. Then we discuss the analogous results for the commutative variant of the $\lambda\wp$-calculus and its extension with additives (§5), our plans for the next papers in the series (§6) and finally some related work (§7).

Prerequisites. We assume that the reader is familiar with the basics of λ-calculi and type systems, but require no prior knowledge of automata theory. This choice is motivated by the impression that it is more difficult to introduce the former than the latter in a limited number of pages. Nevertheless, we hope that our results will be of interest to both communities.

2 Preliminaries: the $\lambda\wp$-calculus and Church encodings

The terms and types of the $\lambda\wp$-calculus are defined by the respective grammars

$$A, B ::= o \mid A \to B \mid A \multimap B \qquad t, u ::= x \mid t\,u \mid \lambda^{\to}x.\,t \mid \lambda^{\circ}x.\,t$$

As always, the $\lambda\wp$ terms are identified up to α-equivalence (both $\lambda^{\to}$ and λ° are binders). There are two rules for β-reduction (closed under contexts)

$$(\lambda^{\to}x.\,t)\,u \longrightarrow_{\beta} t\{x := u\} \qquad (\lambda^{\circ}x.\,t)\,u \longrightarrow_{\beta} t\{x := u\}$$

[5] This connection with the sequential composition of processes can be seen as a sort of embodiment of Girard's slogan "time is the contents of non-commutative linear logic" [18, IV.6]. But generally, these works follow a "proof search as computation" paradigm (logic programming) rather than "normalization as computation" (functional programming).

[6] Hence our choice of name: the "Weierstraß P" character "$\wp$" in "$\lambda\wp$" stands for "planar".

$$\frac{}{\Gamma \uplus \{x : A\} \mid \varnothing \vdash x : A} \qquad \frac{\Gamma \mid \Delta \vdash t : A \to B \qquad \Gamma \mid \varnothing \vdash u : A}{\Gamma \mid \Delta \vdash t\,u : B} \qquad \frac{\Gamma \uplus \{x : A\} \mid \Delta \vdash t : B}{\Gamma \mid \Delta \vdash \lambda^{\to} x.\, t : A \to B}$$

$$\frac{}{\Gamma \mid x : A \vdash x : A} \qquad \frac{\Gamma \mid \Delta \vdash t : A \multimap B \qquad \Gamma \mid \Delta' \vdash u : A}{\Gamma \mid \Delta \cdot \Delta' \vdash t\,u : B} \qquad \frac{\Gamma \mid \Delta \cdot (x : A) \vdash t : B}{\Gamma \mid \Delta \vdash \lambda^{\circ} x.\, t : A \multimap B}$$

$$\frac{\Gamma \mid \Delta \vdash t : A}{\Gamma \mid \Delta' \vdash t : A} \text{ when } \Delta \text{ is a subsequence of } \Delta'$$

Figure 1 The typing rules of the $\lambda\wp$-calculus (see Appendix C in [37] for examples of derivations).

and the remaining conversion rules are the expected η-reduction/η-expansion rules.

The typing judgements make use of dual contexts (a common feature originating in [7]): they are of the form $\Gamma \mid \Delta \vdash t : A$ where t is a term, A is a type, Γ is a set of bindings of the form $x : B$ (x being a variable and B a type), and Δ is an *ordered list* of bindings – this order is essential for non-commutativity. The typing rules are given in Figure 1, where $\Delta \cdot \Delta'$ denotes the *concatenation of the ordered lists* Δ *and* Δ'. For both $\Gamma, \Gamma', \ldots$ and $\Delta, \Delta', \ldots$ we require each variable to appear at most once on the left of a colon.

▶ Remark 2.1. Unlike Polakow and Pfenning's system [39, 40], the $\lambda\wp$-calculus:

- contains two function types instead of four[7], with the top two rows of Figure 1 corresponding almost exactly[8] to the rules given for those connectives in [39];
- is affine instead of linear, as expressed by the "ordered weakening" rule at the bottom of Figure 1 – this seems important to get enough expressive power for our purposes[9].

▶ Remark 2.2. Morally, the non-affine variables "commute with everything". More formally, one could translate the $\lambda\wp$-calculus into a non-commutative version of Intuitionistic Affine Logic whose exponential modality "!" incorporates the customary rules (see e.g. [50])

$$\frac{\Gamma, !A, B, \Delta \vdash C}{\Gamma, B, !A, \Delta \vdash C} \qquad\qquad \frac{\Gamma, B, !A, \Delta \vdash C}{\Gamma, !A, B, \Delta \vdash C}$$

▶ **Proposition 2.3.** *The $\lambda\wp$-calculus enjoys subject reduction and admits normal forms (that is, every well-typed $\lambda\wp$-term is convertible to a β-normal η-long one).*

Proof sketch. This is routine: subject reduction follows from a case analysis, while the fact that the simply typed λ-calculus has normal forms entails that the $\lambda\wp$-calculus also does (the obvious translation preserves the β-reduction and η-expansion relations). ◀

We have already seen the type $\mathtt{Str}_\Sigma = (o \multimap o) \to \ldots \to (o \multimap o) \to (o \multimap o)$ of Church-encoded strings in the introduction. Let us now introduce the term-level encodings:

[7] Our "$\to$" and "$\multimap$" are called "intuitionistic functions" and "right ordered functions" in [39]; we have no counterpart for the "linear [commutative] functions" and "left ordered functions" in the $\lambda\wp$-calculus.

[8] The only difference is that we drop the linear commutative context.

[9] Usually, the linear/affine distinction does not matter for implicit computational complexity if we allow collecting the garbage produced during the computation in a designated part of the output, as in e.g. [30]. But non-commutativity obstructs the free movement of garbage.

▶ **Definition 2.4.** *Let Σ be a finite alphabet, $w = w[1] \dots w[n] \in \Sigma^*$ be a string, and for each $c \in \Sigma$, let t_c be a $\lambda\wp$-term (on which the next proposition will add typing assumptions). We abbreviate $(t_c)_{c\in\Sigma}$ as $\vec{t_\Sigma}$, and define the $\lambda\wp$-term $w^\dagger(\vec{t_\Sigma}) = \lambda^\circ x.\, t_{w[1]}\,(\dots\,(t_{w[n]}\,x)\dots)$.*

Given a total order $c_1 < \dots < c_{|\Sigma|}$ on the alphabet $\Sigma = \{c_1, \dots, c_{|\Sigma|}\}$, the Church encoding *of any string $w \in \Sigma^*$ is $\overline{w} = \lambda^{\to} f_{c_1}.\ \dots\ \lambda^{\to} f_{c_{|\Sigma|}}.\, w^\dagger(\vec{f_\Sigma})$.*

This is simpler than the notation might suggest: as an example, for $\Sigma = \{a, b\}$ with $a < b$, $\overline{baa} = \lambda^{\to} f_a.\ \lambda^{\to} f_b.\ \lambda^\circ x.\ f_b\ (f_a\ (f_a\ x))$. Our choice of presentation is meant to stress the role of the *open* subterm $(baa)^\dagger(\vec{f_{\{a,b\}}}) = \lambda^\circ x.\ f_b\ (f_a\ (f_a\ x))$, cf. Remark 2.9.

We now summarize the classical properties of the Church encoding of strings.

▶ **Proposition 2.5.** *We reuse the notations of the above definition.*

- *Assume that there is a type A and a typing context $\Gamma \mid \Delta$ such that for all $c \in \Sigma$, $\Gamma \mid \Delta \vdash t_c : A \multimap A$. Then $\Gamma \mid \Delta \vdash w^\dagger(\vec{t_\Sigma}) : A \multimap A$.*
- *In particular, $\{f_c : o \multimap o \mid c \in \Sigma\} \mid \varnothing \vdash w^\dagger(\vec{f_\Sigma}) : o \multimap o$ for any* variables *$(f_c)_{c\in\Sigma}$.*
- *Furthermore, in the case of variables, $w \in \Sigma^* \mapsto w^\dagger(\vec{f_\Sigma})$ is in fact a* bijection *between the strings over Σ and the $\lambda\wp$-terms u such that $\{f_c : o \multimap o \mid c \in \Sigma\} \mid \varnothing \vdash u : o \multimap o$ and considered up to $\beta\eta$-conversion*[10]*.*
- *It follows from the above that $w \in \Sigma^* \mapsto \overline{w}$ is a bijection from Σ^* to the set of* closed *$\lambda\wp$-terms of type $\texttt{Str}_\Sigma$ modulo $\beta\eta$.*
- *Finally, with the assumptions on t_c of the first item, we have $\overline{w}\,t_{c_1}\ \dots\ t_{c_{|\Sigma|}} \longrightarrow^*_\beta w^\dagger(\vec{t_\Sigma})$.*

▶ **Example 2.6.** Given two closed $\lambda\wp$-terms $t_a, t_b : \texttt{Bool} \multimap \texttt{Bool}$, one can define the term $g = \lambda^\circ s.\ s\ t_a\ t_b\ \texttt{false} : \texttt{Str}_{\{a,b\}}[\texttt{Bool}] \multimap \texttt{Bool}$. Then for any $w = w[1] \dots w[n] \in \{a, b\}^*$, we have $g\ \overline{w} \longrightarrow^*_\beta w^\dagger(\vec{t_{\{a,b\}}})\ \texttt{false} \longrightarrow^*_\beta t_{w[1]}\ (\dots\ (t_{w[n]}\ \texttt{false}))$.

- For $t_a = \lambda^\circ x.\ \texttt{true}$ and $t_b = \lambda^\circ x.\ x$, g decides the language of words in $\{a, b\}^*$ that contain at least one a; this language is indeed star-free as it can be expressed as $\varnothing^c a \varnothing^c$.
- Coming back to a point raised in the introduction, if negation were definable by a $\lambda\wp$-term $\texttt{not} : \texttt{Bool} \multimap \texttt{Bool}$, then for $t_a = t_b = \texttt{not}$, the language decided by g would consist of words of odd length: a standard example of regular language that is not star-free.

▶ Remark 2.7. Actually, the $\lambda\wp$-term $\texttt{not}' : \lambda^\circ b.\ b\ \texttt{false}\ \texttt{true} : \texttt{Bool}[\texttt{Bool}] \multimap \texttt{Bool}$ does "define negation". A point of utmost importance is that because of the heterogeneity of the input and output types, this term does not contradict Theorem 1.8 and *cannot be iterated by a Church-encoded string.* Monomorphism is therefore crucial for us: if our type system had actual polymorphism, one could give $\texttt{not}'$ the type $(\forall\alpha.\ \texttt{Bool}[\alpha]) \multimap (\forall\alpha.\ \texttt{Bool}[\alpha])$, whose input and output types are equal, and then the words of odd length would be $\lambda\wp$-definable.

An analogous phenomenon in the simply typed λ-calculus is that one can define $n \mapsto 2^n$ on the type of Church numerals $\texttt{Nat}$ by a term of type $\texttt{Nat}[o \to o] \to \texttt{Nat}$, but not by a term of type $\texttt{Nat} \to \texttt{Nat}$ (since iterating it would give rise to a tower of exponentials of variable height, which is known to be inexpressible by any $\texttt{Nat}[A] \to \texttt{Nat}$).

Yet our ersatz of polymorphism still allows for some form of compositionality that will prove useful in several places in §4 (the proof may be found in Appendix B in [37]):

▶ **Lemma 2.8.** *If $\vdash t : A[T] \multimap B$ and $\vdash u : B[U] \multimap C$, then $\vdash \lambda^\circ x.\, u\,(t\,x) : A[T[U]] \multimap C$.*

▶ Remark 2.9. One final observation on Church encodings: when the context Γ of non-affine variables contains $f_c : o \multimap o$ for each $c \in \Sigma$, then any string $w \in \Sigma^*$ can be represented as

[10] η-conversion is necessary to identify $\lambda^{\to} f.\, f : \texttt{Str}_{\{a\}}$ with $\overline{a} = \lambda^{\to} f.\ \lambda^\circ x.\ f\ x : \texttt{Str}_{\{a\}}$.

the open $\lambda\wp$-term $\Gamma \mid \ldots \vdash w^\dagger(\vec{f_\Sigma}) : o \multimap o$ in that context, and such strings can even be concatenated by function composition. The point is that this gives us a kind of *purely affine type of strings*, which will allow us in §4.2 to encode sequential transducers as $\lambda\wp$-terms of type $\mathtt{Str}_\Sigma[A] \multimap \mathtt{Str}_\Pi$ for some purely affine type A (compare Theorem 1.7).

3 Proof of soundness

As stated in the introduction, the soundness part of our main Theorem 1.7 will follow from Theorem 1.8, so we start this section by proving the latter. First, the monoid structure on the closed $\lambda\wp$-terms of *any* type $A \multimap A$ can be verified routinely: both $(f \circ g) \circ h$ and $f \circ (g \circ h)$ β-reduce to $\lambda^\circ x.\, f\ (g\ (h\ x))$, and $\lambda^\circ x.\, x$ provides the identity element. The finiteness of this monoid for A *purely affine* comes from a slightly more general statement:

▶ **Proposition 3.1.** *For any purely affine type B, there are finitely many $\beta\eta$-equivalence classes of closed $\lambda\wp$-terms of type B.*

Proof. This is a well-known property of affine type systems: here, non-commutativity plays no role. We provide a proof in Appendix B in [37]. ◀

The substantial part of Theorem 1.8 is the *aperiodicity* of this monoid. It is here that non-commutativity comes into play. Morally, it is a kind of monotonicity condition that $\lambda\wp$-terms obey. A first idea would therefore be to seek to exploit the fact that the monoid of monotone functions on an ordered set is aperiodic. What we end up using is closely related:

▶ **Lemma 3.2.** *For any $k \in \mathbb{N}$, the monoid of* partial non-decreasing *functions from $\{1, \ldots, k\}$ to itself (endowed with usual function composition) is aperiodic.*

Proof. Let $f : \{1, \ldots, k\} \rightharpoonup \{1, \ldots, k\}$ be non-decreasing. For any $i \in \{1, \ldots, k\}$, the sequence $(f^n(i))_{n\in\mathbb{N}}$ is either non-increasing or non-decreasing as long as it is defined (depending on whether $i \geq f(i)$ or $i \leq f(i)$); so at some $n = N_i$, either it becomes undefined or it reaches a fixed point of f. By taking $N = \max_{1\leq i\leq k} N_i$, we have $f^N = f^{N+1}$. ◀

This underlies the proof of the key lemma below, that allows one to reduce the aperiodicity of some $t : A \multimap A$ to the aperiodicity of $\lambda\wp$-terms at smaller types.

▶ **Notation 3.3.** $\Delta \vdash t : A$ is an abbreviation for $\varnothing \mid \Delta \vdash t : A$ (indeed, the context of non-affine variables will be generally empty in our proof).

▶ **Notation 3.4.** Let $u_1, \ldots, u_k$ and $v_1, \ldots, v_l$ be $\lambda\wp$-terms. The notation $\vec{v}[\vec{y} := \vec{u}]$ denotes the componentwise parallel substitution $(v_i[y_1 := u_1, \ldots, y_k := u_k])_{1\leq i\leq l}$.

▶ **Lemma 3.5.** *Let $t = \lambda^\circ x.\, \lambda^\circ y_1. \ldots \lambda^\circ y_m.\, x\, u_1 \ldots u_k$ be a well-typed closed $\lambda\wp$-term of type $A \multimap A$ in η-*long form*, so that $x : A,\ y_1 : B_1,\ \ldots,\ y_k : B_k \vdash x\, u_1 \ldots u_k : o$ with $A = B_1 \multimap \ldots \multimap B_k \multimap o$. Then:*

- *$t^n = t \circ \ldots \circ t$ (n times) is β-convertible to $\lambda^\circ x.\, \lambda^\circ y_1. \ldots \lambda^\circ y_k.\, x\, u_1^{(n)} \ldots u_k^{(n)}$ where $\vec{u}^{(0)} = (y_1, \ldots, y_k)$, $\vec{u}^{(n+1)} = \vec{u}^{(n)}[\vec{y} := \vec{u}]$;*
- *For large enough $n \in \mathbb{N}$, each $u_i^{(n+1)}$ depends only on $u_i^{(n)}$ for the same $i \in \{1, \ldots, k\}$. More precisely, there exists $N \in \mathbb{N}$ such that for all $i \in \{1, \ldots, k\}$ there exists a well-typed closed $\lambda\wp$-term $t'_i : B_i \multimap B_i$ such that for all $n \geq N$, $u_i^{(n+1)} = t'_i\ u_i^{(n)}$.*

Proof. The first item is established by induction: abbreviating $\lambda^{\circ} y_1. \ldots \lambda^{\circ} y_k.$ as $\lambda^{\circ}\vec{y}.$,

$$\begin{aligned} t \circ (\lambda^{\circ} x. \lambda^{\circ}\vec{y}. x\, u_1^{(n)} \ \ldots\ u_k^{(n)}) &=_\beta \lambda^{\circ} x. \lambda^{\circ}\vec{y}. (\lambda^{\circ}\vec{y}. x\, u_1^{(n)} \ \ldots\ u_k^{(n)})\, u_1 \ \ldots\ u_k \\ &=_\beta \lambda^{\circ} x. \lambda^{\circ}\vec{y}. x\, (u_1^{(n)}[\vec{y} := \vec{u}]) \ \ldots\ u_k^{(n)}([\vec{y} := \vec{u}]) \end{aligned}$$

(We invite to reader to reproduce the full computation to check that no spurious capture of free variables happens.)

For the second item, let us define the partial function $\mu_{\vec{u}} : \{1, \ldots, k\} \rightharpoonup \{1, \ldots, k\}$ by $\mu_{\vec{u}}(i) = j \iff y_i \in \mathrm{FV}(u_j)$. ($\mathrm{FV}(u)$ denotes the set of free variables of u.) The relation on the right-hand side of the equivalence is indeed a partial function because of the affineness of $t = \lambda^{\circ} x. \lambda^{\circ} y_1. \ldots \lambda^{\circ} y_k. z\, u_1 \ldots u_k$. One can also show that for all $n \in \mathbb{N}$, $\mathrm{FV}(u_i^{(n)}) = \{y_j \mid (\mu_{\vec{u}})^n(j) = i\}$.

As a consequence of non-commutativity, $\mu_{\vec{u}}$ is non-decreasing. This is because for the typing judgment on $x\, u_1 \ldots u_k$ to hold, there must exist $\Delta_1, \ldots, \Delta_k$ such that:

- for all $j \in \{1, \ldots, k\}$, $\Delta_j \vdash u_j$ and $\forall i,\ y_i \in \mathrm{FV}(u_j) \iff (y_i : B_i) \in \Delta_j$;
- $\Delta_1 \cdot \ldots \cdot \Delta_k$ is an *ordered subsequence* of $(y_1 : B_1) \cdot \ldots \cdot (y_k : B_k)$.

Therefore, by Lemma 3.2, there exists $N \in \mathbb{N}$ such that $(\mu_{\vec{u}})^N = (\mu_{\vec{u}})^{N+1}$.

Next, let $i \in \{1, \ldots, k\}$. We may reformulate our goal as finding $t'_i : B_i \multimap B_i$ such that $t'_i\ u_i^{(N+n)} =_{\beta\eta} u^{(N+n+1)}$ for all $n \in \mathbb{N}$. The simple case is when $i \notin (\mu_{\vec{u}})^N(\{1, \ldots, k\})$: $u_i^{(N)}$ has no free variables, so $u_i^{(N+1)} = u_i^{(N)}[\vec{y} := \vec{u}] = u_i^{(N)}$: we may then take $t'_i = \lambda^{\circ} z.\ z$. For the remainder of the proof we assume otherwise, that is, we take i in the range of $(\mu_{\vec{u}})^N$.

First, $\vec{u}^{(n+1)} = \vec{u}[\vec{y} := \vec{u}^{(n)}]$ because parallel substitution is associative[11]. Thus,

$$\forall n \in \mathbb{N},\ u_i^{(N+n+1)} = u_i \left[y_j := u_j^{(N+n)} \text{ for } j \in \{1, \ldots, k\} \text{ such that } \mu_{\vec{u}}(j) = i \right]$$

Any $j \in \{1, \ldots, k\} \setminus \{i\}$ such that $\mu_{\vec{u}}(j) = i$ is not a fixed point of $\mu_{\vec{u}}$, and therefore is not in the range of $(\mu_{\vec{u}})^N$ since $(\mu_{\vec{u}})^N = (\mu_{\vec{u}})^{N+1} = \mu_{\vec{u}} \circ (\mu_{\vec{u}})^N$. By the simple case already treated, we then have $u_j^{(N+n)} = u_j^{(N)}$. This allows us to write the above equation as

$$u_i^{(N+n+1)} = r_i[y_i := u_i^{(N+n)}] \quad \text{where} \quad r_i = u_i \left[y_j := u_j^{(N)} \text{ for } j \neq i \text{ s.t. } \mu_{\vec{u}}(j) = i \right]$$

Using β-conversion, $u_i^{(N+n+1)} =_\beta (\lambda^{\circ} y_i. r_i)\, u_i^{(N+n)}$. We conclude by setting $t'_i = (\lambda^{\circ} y_i. r_i)$. It is clear that this $\lambda\wp$-term is closed, but one should check that it is well-typed; to do so, one convenient observation is that the $u_j^{(N)}$ are closed (because $j \notin (\mu_{\vec{u}})^N(\{1, \ldots, k\})$) and well-typed (as closed subterms of a reduct of the N-fold composition t^N). ◀

The remainder of the proof of Theorem 1.8 is essentially bureaucratic.

Proof of the aperiodicity part of Theorem 1.8. Let $t : A \multimap A$; our goal is to show that the sequence $t^n = t \circ \ldots \circ t$ is eventually constant modulo $\beta\eta$. We shall do so by *induction on the size of A*. The type A is *purely affine* by assumption, and can therefore be written as $B_1 \multimap \ldots \multimap B_m \multimap o$ where the B_i are also purely affine for $i \in \{1, \ldots, m\}$. The base case $m = 0$ being trivial, we assume $m \geq 1$. In this case, by Proposition 2.3, t has an η-long β-normal form $t = \lambda^{\circ} x. \lambda^{\circ} y_1. \ldots \lambda^{\circ} y_m. z\, u_1 \ldots u_k$ where z is a variable. There are two cases:

- $z = y_i$ for some i. Then $(y_i : B_i) \cdot \Delta \vdash z\, u_1 \ldots u_k$ by application rule (we omit the non-affine context Γ which will always be empty during this proof). The abstraction rule only allows introducing $\lambda^{\circ} y_i$ when $(y_i : B_i)$ is on the right, so by then Δ must have been

[11] More precisely, $(\vec{t_1}[\vec{x} := \vec{t_2}])[\vec{y} := \vec{t_3}] = \vec{t_1}[\vec{x} := \vec{t_2}[\vec{y} := \vec{t_3}]]$ when $\vec{y} \cap (\mathrm{FV}(\vec{t_1}) \setminus \vec{x}) = \varnothing$.

entirely emptied out by previous abstractions. This means that $\lambda^{\circ} y_i. \ldots \lambda^{\circ} y_m. z\, u_1 \ldots u_k$ is a closed term, so in particular it contains no free occurrence of x: t is a constant function from A to A. So the sequence of iterations stabilizes from $n = 1$.

- $z = x$, which entails $k = m$ since the variable x is of type $A = B_1 \multimap \ldots \multimap B_m \multimap o$ and we must have $x : A,\ y_1 : B_1,\ \ldots,\ y_m : B_m \vdash x\, u_1 \ldots u_k : o$. Lemma 3.5 gives us closed $\lambda\wp$-terms $t'_i : B_i \multimap B_i$ $(i \in \{1, \ldots, k\})$ whose iterates eventually determine those of t. Since the type B_i has size strictly smaller than A, the induction hypothesis applies: each $((t'_i)^n)_{n\in\mathbb{N}}$ is eventually constant modulo $\beta\eta$. Therefore, this is also the case for t. ◀

Let us now apply Theorem 1.8 to the $\lambda\wp$-terms defining languages.

▶ **Lemma 3.6.** *Let $\Sigma = \{c_1, \ldots, c_{|\Sigma|}\}$ be a finite alphabet, A be a* purely affine *type and $t : \mathtt{Str}_\Sigma[A] \multimap \mathtt{Bool}$ be a closed $\lambda\wp$-term. Then there exist some closed $\lambda\wp$-terms $g_c : A \multimap A$ for $c \in \Gamma$ and $h : (A \multimap A) \multimap \mathtt{Bool}$ such that $t =_{\beta\eta} \lambda^{\circ} s.\, h\,(s\, g_{c_1} \ldots g_{c_{|\Sigma|}})$.*

Proof. By inspection of the normal form of t, see Appendix B in [37]. ◀

Reusing the notations of this lemma, let us define $\varphi : \Sigma^* \to \{v \mid v : A \multimap A\}/{=_{\beta\eta}}$ to be the monoid morphism such that $\varphi(c) = g_c$ for $c \in \Sigma$. Then for all $w \in \Sigma^*$, $\varphi(w) = w^\dagger(\vec{g}_\Sigma)$ (in the quotient): by a similar computation than for $f \circ (g \circ h) =_{\beta\eta} (f \circ g) \circ h$, we have $g_{w[1]} \circ \ldots \circ g_{w[n]} \longrightarrow^*_\beta w^\dagger(\vec{g}_\Sigma)$. Therefore, by Proposition 2.5, $\varphi^{-1}(\{v \mid h\, v =_{\beta\eta} \mathtt{true}\})$ is none other than the language defined by the $t : \mathtt{Str}_\Sigma[A] \multimap \mathtt{Bool}$ in the lemma. Thus, L fits the second definition of star-free languages given in Theorem 1.5: indeed, the codomain of φ is finite and aperiodic by Theorem 1.8. This proves the soundness part of Theorem 1.7.

4 Expressiveness of the $\lambda\wp$-calculus

We now turn to the *extensional completeness* part in Theorem 1.7: our goal is to construct, for any star-free language, a closed $\lambda\wp$-term of type $\mathtt{Str}_\Sigma[A] \multimap \mathtt{Bool}$ (for some purely affine A) that defines this language. To do so, the most convenient way that we have found is to take a detour through automata that compute an output string instead of a single bit (acceptance/rejection). We will recall the notion of *aperiodic sequential function* (Definition 4.4), and then establish that:

▶ **Theorem 4.1.** *Any* aperiodic sequential function $\Sigma^* \to \Pi^*$ *can be expressed by a $\lambda\wp$-term of type $\mathtt{Str}_\Sigma[A] \multimap \mathtt{Str}_\Pi$ for some* purely affine *type A.*

The advantage of working with this class of functions is that they can be assembled from small "building blocks" by function composition, as the *Krohn–Rhodes decomposition* (Theorem 4.8) tells us. Our proof strategy for the above theorem will consist in encoding these blocks (Lemma 4.10) and composing them together (as a special case of Lemma 2.8).

To deduce the desired result, we rely on two lemmas (proved in Appendix B in [37]):

▶ **Lemma 4.2.** *If a language $L \subseteq \Sigma^*$ is* star-free, *then its* (string-valued) indicator function *$\chi_L : \Sigma^* \to \{1\}^*$, defined by $\chi_L(w) = 1$ if $w \in L$ and $\chi_L(w) = \varepsilon$ otherwise, is aperiodic sequential.*

▶ **Lemma 4.3.** *There exists a $\lambda\wp$-term $\mathtt{nonempty} : \mathtt{Str}_{\{1\}}[\mathtt{Bool}] \multimap \mathtt{Bool}$ that tests whether its input string is non-empty.*

Let L be a star-free language. Combining Lemma 4.2 and Theorem 4.1, χ_L is definable by some $\lambda\wp$-term $\mathtt{indic}_L : \mathtt{Str}_\Sigma[A] \multimap \mathtt{Str}_{\{1\}}$ where A is purely affine. To compose this with the non-emptiness test of Lemma 4.3, we use Lemma 2.8 again: the $\lambda\wp$-term

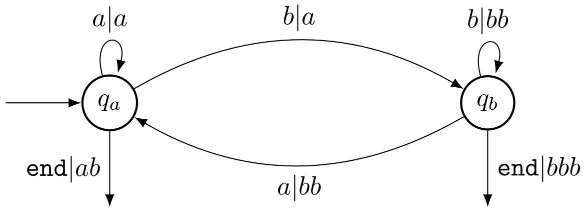


Figure 2 A schematic representation of a sequential transducer whose formal definition is $Q = \{q_a, q_b\}$, $\delta(q, a) = (q_a, a)$ and $\delta(q, b) = (q_b, bb)$ for $q \in Q$, $q_I = q_a$, $F(q_a) = ab$ and $F(q_b) = bbb$.

$t_L = \lambda^\circ x.\ \mathtt{nonempty}\ (\mathtt{indic}_L\ x) : \mathtt{Str}_\Sigma[A[\mathtt{Bool}]] \multimap \mathtt{Bool}$ defines L. Since A and $\mathtt{Bool}$ are purely affine, so is $A[\mathtt{Bool}]$: we just deduced extensional completeness from Theorem 4.1. Proving the latter is the goal of the rest of this section.

4.1 Reminders on automata theory

Sequential transducers are among the simplest models of automata with output. They are deterministic finite automata which can append a word to their output at each transition, and at the end, they can add a suffix to the output depending on the final state. The definition is classical; a possible reference is [44, Chapter V].

▶ **Definition 4.4.** *A* sequential transducer *with input alphabet* Σ *and output alphabet* Π *consists of a set of* states Q, *a* transition function $\delta : Q \times \Sigma \to Q \times \Pi^*$, *an* initial state $q_I \in Q$, *and a* final output function $F : Q \to \Pi^*$. *We abbreviate* $\delta_i = \pi_i \circ \delta$ *for* $i \in \{1, 2\}$, *where* $\pi_1 : Q \times \Pi^* \to Q$ *and* $\pi_2 : Q \times \Pi^* \to \Pi^*$ *are the projections of the product.*

Given an input string $w = w[1] \dots w[n] \in \Sigma^*$, *the* run *of the transducer over* w *is the sequence of states* $q_0 = q_I$, $q_1 = \delta_{\mathrm{st}}(q_0, w[1])$, $\dots$, $q_n = \delta_{\mathrm{st}}(q_{n-1}, w[n])$. *Its* output *is obtained as the concatenation* $\delta_{\mathrm{out}}(q_0, w[1]) \cdot \ldots \cdot \delta_{\mathrm{out}}(q_{n-1}, w[n]) \cdot F(q_n)$.

A sequential function *is a function* $\Sigma^* \to \Pi^*$ *computed as described above by some sequential transducer.*

▶ **Definition 4.5.** *The* transition monoid *of a sequential transducer is the submonoid of* $Q \to Q$ *(endowed with reverse function composition:* $fg = g \circ f$*) generated by the maps* $\{\delta_{\mathrm{st}}(-, c) \mid c \in \Sigma\}$ *(where* $\delta_{\mathrm{st}}(-, c)$ *stands for* $q \mapsto \delta_{\mathrm{st}}(q, c)$*).*

A sequential transducer is said to be aperiodic *when its transition monoid is aperiodic. A function that can be computed by such a transducer is called an* aperiodic sequential function.

▶ **Example 4.6.** The transducer in Figure 2 computes $f : w \in \{a, b\}^* \mapsto a \cdot \psi(w) \cdot b$ where ψ is the monoid morphism that doubles every b: $\psi(a) = a$ and $\psi(b) = bb$. Its transition monoid T is generated by $G = \{(\delta_{\mathrm{st}}(-, a) : q \mapsto q_a), (\delta_{\mathrm{st}}(-, b) : q \mapsto q_b)\}$; one can verify that $T = G \cup \{\mathrm{id}\}$ and therefore $\forall h \in T,\ h \circ h = h$. Thus, f is an aperiodic sequential function.

▶ Remark 4.7. The converse to Lemma 4.2 is also true; more generally, the preimage of a star-free language by an aperiodic sequential function is star-free, and the preimage of a regular language is regular. But we will not need this here.

▶ **Theorem 4.8** (Krohn–Rhodes decomposition, aperiodic case, cf. Appendix A in [37])**.** *Any* aperiodic sequential function $f : \Sigma^* \to \Pi^*$ *can be realized as a composition* $f = f_1 \circ \ldots \circ f_n$ *(with* $f_i : \Xi_i^* \to \Xi_{i-1}^*$, $\Xi_0 = \Pi$ *and* $\Xi_n = \Sigma$*) where each function* f_i *is computed by some aperiodic sequential transducer* with 2 states.

Figure 2 gives an example of aperiodic transducer with two states.

▶ Remark 4.9. This is not the standard way to state this theorem, though one may find it in the literature, usually without proof (e.g. [10, §1.1]); see [8] for a tutorial containing a proof sketch of this version. In Appendix A in [37], we show how Theorem 4.8 follows from the more usual statement on wreath products of monoid actions.

4.2 Encoding aperiodic sequential transducers

Thanks to the Krohn–Rhodes decomposition and to the fact that the string functions definable in the $\lambda\wp$-calculus (as specified by Theorem 4.1) are closed under composition (by Lemma 2.8), the following entails Theorem 4.1, thus concluding our completeness proof.

▶ **Lemma 4.10.** *Any function $\Sigma^* \to \Pi^*$ computed by some aperiodic sequential transducer* with 2 states *can be expressed by some $\lambda\wp$-term of type $\mathtt{Str}_\Sigma[A] \multimap \mathtt{Str}_\Pi$, for a* purely affine *type A depending on the function.*

Let us start by exposing the rough idea of the encoding's trick using set-theoretic maps. We reuse the notations of Definition 4.4 and assume w.l.o.g. that the set of states is $Q = \{1, 2\}$.

Suppose that at some point, after processing a prefix of the input, the transducer has arrived in state 1 (resp. 2) and in the meantime has outputted $w \in \Pi^*$. We can represent this "history" by the pair (κ_w, ζ) (resp. (ζ, κ_w)) where

$$\zeta, \kappa_w : \Pi^* \to \Pi^* \qquad \zeta : x \mapsto \varepsilon \qquad \kappa_w : x \mapsto w \cdot x$$

For instance, in the case of Example 4.6, after reading a string $s = s'b$, the transducer is in the state q_b and has outputted[12] $w = a \cdot \psi(s')$, which we represent as $(\zeta, \kappa_{a\cdot\psi(s')})$ (taking $q_a = 1$ and $q_b = 2$; ψ is described in Example 4.6). In general, some key observations are

$$\zeta \circ \kappa_w = \zeta \qquad \kappa_w \circ \kappa_{w'} = \kappa_{ww'} \qquad \kappa_w(w')\zeta(w'') = \zeta(w'')\kappa_w(w') = ww'$$

Now, consider an input letter $c \in \Sigma$; how to encode the corresponding transition $\delta(-, c)$ as a transformation on the pair encoding the current state and output history? It depends on the state transition $\delta_{\mathrm{st}}(-, c)$; we have thanks to the above identities:

- $(h, g) \mapsto (h \circ \kappa_{\delta_{\mathrm{out}}(1,c)}, g \circ \kappa_{\delta_{\mathrm{out}}(2,c)})$ when $\delta_{\mathrm{st}}(-, c) = \mathrm{id}$;
- $(h, g) \mapsto (\kappa_{h(\delta_{\mathrm{out}}(1,c))g(\delta_{\mathrm{out}}(2,c))}, \zeta)$ when $\delta_{\mathrm{st}}(-, c) : q' \mapsto 1$ (note that $h = \zeta$ xor $g = \zeta$);
- $(h, g) \mapsto (\zeta, \kappa_{h(\delta_{\mathrm{out}}(1,c))g(\delta_{\mathrm{out}}(2,c))})$ when $\delta_{\mathrm{st}}(-, c) : q' \mapsto 2$;
- The remaining case $\delta_{\mathrm{st}}(-, c) : q \mapsto 3 - q$ is *excluded by aperiodicity.* This point is crucial: this case would correspond to $(h, g) \mapsto (g \circ \kappa_{\delta_{\mathrm{out}}(2,c)}, h \circ \kappa_{\delta_{\mathrm{out}}(1,c)})$ which morally "uses its arguments h, g in the wrong order".

Coming back to Example 4.6, let us say that after the transducer has read a prefix $s = s'b$ of its input string as we previously described, the next letter is a. Then the expression $h(\delta_{\mathrm{out}}(1, c))g(\delta_{\mathrm{out}}(2, c))$ above is in this case $\zeta(a)\kappa_{a\cdot\psi(s')}(bb) = \varepsilon \cdot a \cdot \psi(s') \cdot bb = a \cdot \psi(s)$ which is indeed the output that the transducer produces after reading the input prefix $sa = s'ba$.

Next, we must transpose these ideas to the setting of the $\lambda\wp$-calculus.

[12] This is indeed $a \cdot \psi(s')$ and not $a \cdot \psi(s) = a \cdot \psi(s') \cdot bb$. If the input turns out to end there, the final output function will provide the missing suffix $F(q_b) = bbb$ to obtain $f(s) = a \cdot \psi(s) \cdot b = a \cdot \psi(s') \cdot bbb$.

Proof of Lemma 4.10. We define the $\lambda\wp$-term meant to compute our sequential function as

$$\lambda^{\circ} s.\, \lambda^{\to} f_{a_1}. \;\ldots\; \lambda^{\to} f_{a_{|\Pi|}}. \, \mathtt{out}\,(s\,\mathtt{trans}_{c_1}\;\ldots\;\mathtt{trans}_{c_{|\Sigma|}}) : \mathtt{Str}_{\Sigma}[A] \multimap \mathtt{Str}_{\Pi}$$

where $\Sigma = \{c_1, \ldots, c_{|\Sigma|}\}$, $\Pi = \{a_1, \ldots, a_{|\Pi|}\}$ and, writing $\Gamma = \{f_a : o \multimap o \mid a \in \Pi\}$,

$$\Gamma \mid \varnothing \vdash \mathtt{trans}_c : A \multimap A \quad (\text{for all } c \in \Sigma) \qquad\qquad \Gamma \mid \varnothing \vdash \mathtt{out} : (A \multimap A) \multimap (o \multimap o)$$

In the presence of this non-affine context Γ, the type $S = o \multimap o$ morally serves as a purely affine type of strings, as mentioned in Remark 2.9. Moreover this "contextual encoding of strings" supports concatenation (by function composition), leading us to represent the maps ζ and κ_w as open terms of type $T = S \multimap S$ that use non-affinely the variables f_a for $a \in \Pi$.

We shall take the type A, at which the input $\mathtt{Str}_{\Sigma}$ is instantiated, to be $A = T \multimap T \multimap S$, which is indeed purely affine as required by the theorem statement. This can be seen morally as a type of continuations [42] taking pairs of type $T \otimes T$ (although our $\lambda\wp$-calculus has no actual $\otimes$ connective). Without further ado, let us program (the typing derivations for some of the following $\lambda\wp$-terms are given in Appendix C in [37]):

- $\mathtt{cat} = \lambda^{\circ} w.\, \lambda^{\circ} w'.\, \lambda^{\circ} x.\, w\,(w'\,x) : S \multimap S \multimap o \multimap o = S \multimap S \multimap S = S \multimap T$ plays the roles of both the concatenation operator and of $w \mapsto \kappa_w$ (thanks to currying)
- $\mathtt{zeta} = \lambda^{\circ} w'.\, \lambda^{\circ} x.\, x : S \multimap o \multimap o = T$
- $u_q = \delta_{\mathrm{out}}(q,c)^{\dagger}(\vec{f_{\Pi}}) : o \multimap o$ (by Proposition 2.5) represents the output word $\delta_{\mathrm{out}}(q,c)$ that corresponds to a given input letter $c \in \Sigma$ and state $q \in Q = \{1,2\}$
- case $\delta_{\mathrm{st}}(q,c) = q$: $\mathtt{trans}_c = \lambda^{\circ} k.\, \lambda^{\circ} h.\, \lambda^{\circ} g.\, k\,(\lambda^{\circ} y.\, h\,(\mathtt{cat}\,u_1\,y))\,(\lambda^{\circ} z.\, g\,(\mathtt{cat}\,u_2\,z))$ – if we wanted to handle the excluded case $\delta_{\mathrm{st}}(q,c) = 3 - q$, we would write a similar term with the occurrences of h and g exchanged $(\lambda^{\circ} k.\, \lambda^{\circ} h.\, \lambda^{\circ} g.\, k\,(\lambda^{\circ} y.\, g\;\ldots)\,(\lambda^{\circ} z.\, h\;\ldots))$, violating the non-commutativity requirement (contrast with the proof of Theorem 5.4);
- case $\delta_{\mathrm{st}}(q,c) = 1$: $\mathtt{trans}_c = \lambda^{\circ} k.\, \lambda^{\circ} h.\, \lambda^{\circ} g.\, k\,(\mathtt{cat}\,(\mathtt{cat}\,(h\,u_1)\,(g\,u_2)))\,\mathtt{zeta}$
- case $\delta_{\mathrm{st}}(q,c) = 2$: $\mathtt{trans}_c = \lambda^{\circ} k.\, \lambda^{\circ} h.\, \lambda^{\circ} g.\, k\,\mathtt{zeta}\,(\mathtt{cat}\,(\mathtt{cat}\,(h\,u_1)\,(g\,u_2)))$
- $\mathtt{out} = \lambda^{\circ} j.\, j\,(\lambda^{\circ} h.\, \lambda^{\circ} g.\, \mathtt{cat}\,(h\,v_1)\,(g\,v_2))\,(\lambda^{\circ} x.\, x)\,\mathtt{zeta}$, where $v_q = F(q)^{\dagger}(\vec{f_{\Pi}})$ represents the output suffix for state $q \in \{1,2\}$, assuming w.l.o.g. that the initial state is 1 (also, here $\lambda^{\circ} x.\, x$ represents κ_{ε} since the latter is the identity on Π^*)

We leave it to the reader to check that these $\lambda\wp$-terms have the expected computational behavior; again, see Appendix C in [37] for typing derivations. Note that in functional programming terms, the use of continuations turns the "right fold" of the Church-encoded input string into a "left fold", and the latter fits with the left-to-right processing of a sequential transducer. ◀

5 Regular languages in extensions of the $\lambda\wp$-calculus

5.1 The commutative case

The $\lambda\wp$-calculus adds two restrictions to the simply typed λ-calculus, namely affineness and non-commutativity, with the latter depending on the former as already mentioned. One could wonder whether affineness by itself would be enough to characterize star-free languages. We now show that it is not the case.

The *commutative* variant of the $\lambda\wp$-calculus – let us call this variant the *λa-calculus*[13] – has the same grammar of types and terms as the $\lambda\wp$-calculus (cf. §2). The typing rules are also given by Figure 1, but their interpretation differs from the previous one as follows:

[13] *a* standing for "affine".

Δ, Δ' stand for *sets* of bindings $x : A$, $\Delta \cdot \Delta'$ denotes the *disjoint union* of sets, and one must read "subset" instead of "subsequence". In other words, the main difference is that in the λa-calculus, the affine context Δ does not keep track of the *ordering* of variables.

By plugging this commutative system in the statement of our main result (Theorem 1.7), we get *regular languages* instead of star-free languages:

▶ **Theorem 5.1.** *A language $L \subseteq \Sigma^*$ is* regular *if and only if it can be defined by a closed λa-term of type $\mathtt{Str}_\Sigma[A] \multimap \mathtt{Bool}$ for some purely affine type A (that may depend on L).*

Proof. Soundness is a consequence of Hillebrand and Kanellakis's Theorem 1.1, by a simple translation from the λa-calculus to the simply typed λ-calculus which "forgets affineness".

For extensional completeness, consider a regular language $L = \varphi^{-1}(P)$ where P is a subset of a finite monoid M and $\varphi : \Sigma^* \to M$ is a morphism (cf. Theorem 1.3). If we represent an element $m \in M$ by a M-indexed bit vector v_m such that $v_m[i] = 1 \iff i = m$, then a translation $m \mapsto mp$ can be represented by a *purely disjunctive* formula:

$$v_{mp}[i] = v_m[j_1] \vee \ldots \vee v_m[j_k] \text{ where } \{j_1, \ldots, j_k\} = \{j \in M \mid jp = i\}$$

Moreover, this is *linear* in the following sense: given a fixed $p \in M$, each index $j \in M$ is involved in the right-hand side of this formula for exactly one $i \in M$.

Let $\mathtt{ttt} = \lambda^\circ x.\,\mathtt{true} : \mathtt{Bool} \multimap \mathtt{Bool}$ and $\mathtt{fff} = \lambda^\circ x.\,x : \mathtt{Bool} \multimap \mathtt{Bool}$. This makes the type $B = \mathtt{Bool} \multimap \mathtt{Bool}$ into a kind of type of booleans that supports a disjunction of type $B \multimap B \multimap B$ (by function composition) and a type-cast function of type $B \multimap \mathtt{Bool}$ (by applying to $\mathtt{false}$). (Of course B has other closed inhabitants besides $\mathtt{ttt}$ and $\mathtt{fff}$, but we only use those two.) Using this type and the "iteration+continuations" recipe of the proof of Lemma 4.10, one can define a λa-term of type $\mathtt{Str}_\Sigma[A] \multimap \mathtt{Bool}$ that decides the language L with $A = B \multimap \ldots \multimap B \multimap \mathtt{Bool}$ (with $|M|$ arguments of type B). ◀

Let us go further. According to Theorem 4.1, the $\lambda\wp$-calculus can define all aperiodic sequential functions; we show that as one can expect, the aperiodicity condition is lifted when moving to the commutative λa-calculus. However, the trick used in the direct encoding of the above proof does not work, and we have only managed to encode general sequential functions by resorting to the Krohn–Rhodes theorem.

▶ **Theorem 5.2** (Krohn–Rhodes decomposition, non-aperiodic case, cf. Appendix A in [37])**.** *Any* sequential function $f : \Sigma^* \to \Pi^*$ *can be realized as a composition $f = f_1 \circ \ldots \circ f_n$ (with $f_i : \Xi_i^* \to \Xi_{i-1}^*$, $\Xi_0 = \Pi$ and $\Xi_n = \Sigma$) where each function f_i is computed by some sequential transducer whose transition monoid is either* aperiodic *or a* group*.*

▶ Remark 5.3. By Theorem 4.8, the aperiodic transducers among the f_i can be further decomposed into two-state aperiodic transducers.

▶ **Theorem 5.4.** *Any sequential function $\Sigma^* \to \Pi^*$ can be expressed by some λa-term of type $\mathtt{Str}_\Sigma[A] \multimap \mathtt{Str}_\Pi$, for a* purely affine *type A depending on the function.*

Proof sketch. First, by Theorem 4.1, we can already encode aperiodic sequential functions, since every well-typed $\lambda\wp$-term is also a well-typed λa-term. One can also show that Lemma 2.8 applies to the λa-calculus. By the general Krohn–Rhodes theorem, we just need to handle the case of a sequential transducer whose transition monoid is a group.

The idea, in terms of set-theoretic maps as in our explanation of the proof of Lemma 4.10 (whose notations we borrow here), is as follows. The current state $q \in Q$ and output history $w \in \Pi^*$ is represented by a Q-indexed family $(g_{q'})_{q' \in Q}$ of functions such that $g_q = \kappa_w$ and for

$q' \neq q$, $g_{q'} = \zeta$. The transition $\delta(-,c)$ is represented by $(g_q)_{q\in Q} \mapsto (g_{\sigma(q)} \circ \kappa_{\delta_{\mathrm{out}}(\sigma(q),c)})_{q\in Q}$ where $\sigma = (\delta_{\mathrm{st}}(-,c))^{-1}$ – the latter is well-defined because the group assumption means that $\delta_{\mathrm{st}}(-,c)$ is a permutation of Q. The final output is obtained at the end as the concatenation $g_{q_1}(F(q_1)) \dots g_{q_n}(F(q_n))$ where $Q = \{q_1, \dots, q_n\}$ (with an arbitrary enumeration of Q).

The elaboration of the corresponding λa-term is left to the reader. Keep in mind that the reason this term will not be well-typed for the $\lambda\wp$-calculus is that the inversions in the permutation $\delta_{\mathrm{st}}(-,c)$ correspond to violations of non-commutative typing. ◀

5.2 Extension with additive pairs

Let's look at what happens if we add the *additive conjunction* connective of linear logic to the $\lambda\wp$-calculus. The $\lambda\wp^{\&}$*-calculus* is obtained by adding $A, B ::= \dots \mid A \mathbin{\&} B$ to the grammar of types and $t, u ::= \dots \mid \langle t, u\rangle \mid \pi_1\, t \mid \pi_2\, t$ for terms, with the typing rules

$$\frac{\Gamma \mid \Delta \vdash t : A \quad \Gamma \mid \Delta \vdash u : B}{\Gamma \mid \Delta \vdash \langle t, u\rangle : A \mathbin{\&} B} \qquad \frac{\Gamma \mid \Delta \vdash t : A_1 \mathbin{\&} A_2}{\Gamma \mid \Delta \vdash \pi_i\, t : A_i} \qquad \text{(see [39, §4])}$$

the β-reduction rules $\pi_i\,\langle t_1, t_2\rangle \longrightarrow_\beta t_i$, and the corresponding η-conversion rules.

Recall that we discussed both in the introduction and in Remark 2.7 the need to prevent the existence of a $\lambda\wp$-term of type $\texttt{Bool} \multimap \texttt{Bool}$ for negation. However, if we use the additive conjunction to define the type $\texttt{Bool}^{\&} = (o \mathbin{\&} o) \multimap o$, the following are well-typed $\lambda\wp^{\&}$-terms:

$$\texttt{true}^{\&} = \lambda^{\circ} p.\ \pi_1\, p \qquad \texttt{false}^{\&} = \lambda^{\circ} p.\ \pi_2\, p \qquad \texttt{not}^{\&} = \lambda^{\circ} b.\ \lambda^{\circ} p.\ b\ \langle \pi_2\, p, \pi_1\, p\rangle$$

More generally:

▶ **Proposition 5.5.** *Let* $\texttt{Fin}^{\&}(n) = (o \mathbin{\&} \dots \mathbin{\&} o) \multimap o$*. For all* $n \in \mathbb{N}$*, there is a canonical bijection between* $\{1, \dots, n\}$ *and the closed* $\lambda\wp^{\&}$*-terms of type* $\texttt{Fin}^{\&}(n)$*. Furthermore, using this encoding, every map* $\{1, \dots, n_1\} \times \dots \times \{1, \dots, n_k\} \to \{1, \dots, m\}$ *can be defined by a closed* $\lambda\wp^{\&}$*-term of type* $\texttt{Fin}^{\&}(n_1) \multimap \dots \texttt{Fin}^{\&}(n_k) \multimap \texttt{Fin}^{\&}(m)$*.*

▶ **Corollary 5.6.** *Every* regular *language can be defined by a closed* $\lambda\wp^{\&}$*-term of type* $\texttt{Str}_\Sigma[A] \multimap \texttt{Bool}$ *for some purely affine type* A *– we consider "&" as an affine connective and therefore allow it in* A*.*

Proof idea. Take $A = \texttt{Fin}^{\&}(|M|)$ where M is any finite monoid that recognizes the language as specified in Theorem 1.3. (We could also prove the converse by relying on an extension of Hillebrand and Kanellakis's Theorem 1.1 to the simply typed λ-calculus with products.) ◀

Similarly, one could show that the addition of the *additive disjunction* "$\oplus$" of linear logic to the $\lambda\wp$-calculus would be sufficient to encode all regular languages.

5.3 On regular and first-order tree languages: a discussion

There is a rich theory of *tree automata* that extends the notion of regular language to trees over ranked alphabets instead of strings. Such trees admit Church encodings; for instance, for an alphabet with arities $(a : 2, b : 2, x : 0)$ (i.e. for trees with two kind of binary nodes and one kind of leaf) one would have $\texttt{Tree}_{(2,2,0)} = (o \multimap o \multimap o) \to (o \multimap o \multimap o) \to o \to o$.

▶ Remark 5.7. A string over an alphabet $\Sigma = \{c_1, \dots, c_{|\Sigma|}\}$ can be seen as a tree with arities $(c_1 : 1,\ \dots,\ c_{|\Sigma|} : 1, \varepsilon : 0)$. This would lead to defining the type of Church-encoded strings as $\texttt{Str}'_\Sigma = (o \multimap o) \to \dots \to (o \multimap o) \to o \to o$. Our type $\texttt{Str}_\Sigma$, which is the traditional choice in linear logic (see the discussion on Church numerals in [17, §5.3.2]), is a bit more precise since it expresses that such a "unary tree" can only contain one ε node. But as there exist conversion functions $\mathtt{Str}_\Sigma \multimap \mathtt{Str}'_\Sigma$ and $\mathtt{Str}'_\Sigma[o \multimap o] \multimap \mathtt{Str}_\Sigma$, this choice does not make much difference (thanks again to Lemma 2.8).

We shall not go into the details of tree automata here, but the knowledgeable reader may check that Proposition 5.5 can be used to encode all *regular tree languages* over $(a : 2, b : 2, x : 0)$ as closed $\lambda\wp^{\&}$-terms of type $\mathtt{Tree}_{(2,2,0)}[A] \multimap \mathtt{Bool}$ for purely affine A. Predictably, this fails for the $\lambda\wp$-calculus without additive connectives. More noteworthy is the failure of the trick used to prove Theorem 5.1 for the commutative λa-calculus when one replaces strings with trees. Thus, it seems (though this remains conjectural) that *the additives of linear logic might be required to express some regular tree languages.*

We believe that this is no accident and that some fundamental difficulty of automata theory is being manifested here. Indeed, if we had a characterization of regular tree languages in the λa-calculus, we could expect that moving to the $\lambda\wp$-calculus would yield the *first-order tree languages*, which are the commonly accepted counterpart of star-free languages for trees. (Recall from Theorem 1.5 that definability in first-order logic is among the equivalent definitions of star-free languages.) However, while Theorem 1.5 demonstrates that star-free languages are well-understood, the situation is quite different for first-order tree languages: there is no known algebraic characterization, and neither is there any known algorithm to decide whether a tree automaton recognizes a first-order language (see e.g. [9, §3]). Another argument for the necessity of additives, discussed in the next section, comes from transducers.

6 Next episode preview: transducers in typed λ-calculi

We started from Hillebrand and Kanellakis's Theorem 1.1 and obtained an analogous statement for star-free languages instead of regular languages. Another direction that we could have pursued is to replace languages by *functions*, by looking at the type $\mathtt{Str}_\Sigma[A] \to \mathtt{Str}_\Pi$. Indeed, an immediate consequence of this "regular = λ-definable" result is:

▶ **Corollary 6.1.** *If $f : \Sigma^* \to \Pi^*$ is definable by a closed simply typed λ-term of type $\mathtt{Str}_\Sigma[A] \to \mathtt{Str}_\Pi$, then for any regular language $L \subseteq \Pi^*$, $f^{-1}(L) \subseteq \Sigma^*$ is also regular.*

Proof idea. Let $u : \mathtt{Str}_\Pi[B] \to \mathtt{Bool}$ and $t : \mathtt{Str}_\Sigma[A] \to \mathtt{Str}_\Pi$ be simply typed λ-terms defining L and f respectively. Then $f^{-1}(L)$ is defined by $\lambda x.\, u\,(t\,x)$ which is well-typed with type $\mathtt{Str}_\Sigma[A[B]] \to \mathtt{Bool}$ (analogously to Lemma 2.8). ◀

This suggests a connection between these λ-definable string functions and automata theory. But while it is not too hard to define functions of hyperexponential growth in the simply typed λ-calculus, most classes of string functions from automata theory (see [36] for a recent survey) grow much more slowly (polynomially or even linearly in the input size). The challenge then becomes to *restrict the expressiveness via types* to capture such classes. This calls for the recipes that have worked here, namely affine types and non-commutativity.

▷ Claim 6.2 (to be proved in a sequel). *The functions definable by closed terms of type $\mathtt{Str}_\Sigma[A] \multimap \mathtt{Str}_\Pi$, for purely affine A, are the MSO transductions*[14] [14] (a.k.a. *regular functions*[15]) *in the λa-calculus and the FO transductions in the $\lambda\wp$-calculus.*

[14] MSO stands for Monadic Second-Order Logic while FO stands for First-Order Logic, cf. the introduction.

[15] This name is somewhat confusing, since there are multiple classes of string functions that collapse to the single class of regular languages when we consider indicator functions. For example, in-between the sequential functions (Definition 4.4) and the regular (MSO-definable) functions, there is a widely

This goes beyond the encodings of sequential transducers presented in this paper (Theorem 4.1 and Theorem 5.4). But the latter are an important stepping stone, since we do not know how to prove the above claim without using the Krohn–Rhodes decomposition somewhere. To summarize the results of the present paper together with its planned sequel:

calculus	affine	commutative	$\mathtt{Str}_\Sigma[A] \multimap \mathtt{Bool}$	$\mathtt{Str}_\Sigma[A] \multimap \mathtt{Str}_\Pi$
$\lambda\wp$	yes	no	star-free (FO-definable) languages	FO transductions
λa	yes	yes	regular (MSO-definable) languages	MSO transductions

While the connection between non-commutativity and aperiodicity came as a surprise to us, we had more reasons to suspect that affine types should have something to do with transducers. Indeed, the term "linear" itself has been used to describe the *copyless assignment* condition on streaming string transducers (SSTs) [5], a machine model for MSO transductions, e.g. "updates should make a linear use of registers" [15, §5] (in our terminology, the register assignments of SSTs are in fact affine, not strictly linear). Moreover, it seems (informally speaking) that the more sophisticated *single-use-restricted assignments* of streaming *tree* transducers [3] correspond to a form of linearity that incorporates an *additive conjunction*, whereas copyless assignments are purely *multiplicative*; compare with the discussion of §5.3.

7 Related work

We have already mentioned in the introduction several lines of tangentially related research, such as higher-order model checking or the topology of non-commutative proofs. In this section, we discuss a few references that we deemed to be more directly relevant.

Automata as circular proofs. Aside from Hillebrand and Kanellakis's Theorem 1.1, perhaps our most direct precursors in "implicit automata theory" are the works by DeYoung and Pfenning [13] on sequential transducers (their version seems to be equivalent to Definition 4.4) and by Kuperberg, Pinault and Pous [28] characterizing regular languages and deterministic logarithmic space complexity. Both rely on a proofs-as-programs interpretation of *circular*[16] *proof systems* for some variants of linear logic with fixed points.

The Church encoding of strings is obtained by a systematic procedure [12] from the inductive definition $s ::= \varepsilon \mid c_1 \cdot s \mid \ldots \mid c_{|\Sigma|} \cdot s$ ($\Sigma = \{c_1, \ldots, c_{|\Sigma|}\}$). Using fixed points of formulas, one can instead turn it into the recursive type[17] $\mathtt{Str}^\mu_\Sigma = 1 \oplus \mathtt{Str}^\mu_\Sigma \oplus \ldots \oplus \mathtt{Str}^\mu_\Sigma$; this is the definition of the type of strings in [13], and it is also implicitly at work in[18] [28].

So both our approach (following Hillebrand and Kanellakis [24]) and those using fixed point logics morally work because the consumption of strings represented as inductive data types is similar to their traversal by automata. However, while the use of the "right fold" provided by a Church-encoded string involves an "inversion of control" (in programming jargon) that, in the case of the simply typed λ-calculus, has drastic effects on expressive power[19] (contrast Theorem 1.1 with the fact that $\beta\eta$-convertibility of simply typed λ-terms is not elementary recursive [32]), circular proofs seem to give the programmer more degrees of freedom: Kuperberg et al. do not need to add polymorphism to go beyond regular languages.

studied strictly intermediate class called the *rational functions*. (The adjective "rational" is used to refer to regular languages in a French tradition going back to Nivat and Schützenberger.)

[16] These are sometimes called "cyclic" proofs, but in our context, this would create a confusion with an unrelated non-commutative logic, *cyclic linear logic* [50].

[17] Formally, this is expressed as the least fixed point $\mathtt{Str}^\mu_\Sigma = \mu\alpha.\ 1 \oplus \alpha \oplus \ldots \oplus \alpha$.

[18] The left rules given in [28, Figure 1] for A and A^* correspond to $A = 1 \oplus \ldots \oplus 1$ and $A^* = 1 \oplus (A \otimes A^*)$.

[19] To overcome those limits and express any elementary recursive function as a simply typed λ-term, Hillebrand and Kanellakis use an alternative representation of inputs inspired by database theory [24].

Recognizable languages of λ-terms. A modern point of view on Hillebrand and Kanellakis's Theorem 1.1 can be implicitly found in a paper by Terui [48] emphasizing the method of *evaluation in a finite denotational semantics* used to prove it. Along these lines, general notions of recognizable languages of closed λ-terms of a given type (specializing to regular languages for the type of Church-encoded strings) have been proposed, based on finite semantics, in the simply-typed λ-calculus by Salvati [45] and in an infinitary λ-calculus by Melliès [34]. It is plausible that Theorem 1.1 can be extended to give an equivalent syntactic definition for Salvati's recognizable languages: for a simple type B they would be the languages definable by $B[A] \to \texttt{Bool}$. An interesting question would be whether one can give an encoding of *higher-dimensional trees* in the simply typed λ-calculus so that this notion of recognizability coincides with Rogers's automata for those trees [43, 16].

Other implicit automata results. In a recent preprint, Bojańczyk [10] introduces a new class of string-to-string functions that admits several equivalent definitions (see also [11]). One of them uses the simply typed λ-calculus enriched with a ground type of lists and several primitive functions on lists. Strings are represented as lists of characters, which differs from our use of functional encodings in a λ-calculus without any primitive data type.

Using a computational model inspired by denotational semantics of linear logic, Seiller [46] gives a characterization of each level of the k-head two-way non-deterministic automata hierarchy. The lowest level ($k = 1$) corresponds to regular languages, while the union over $k \in \mathbb{N}_{\geq 1}$ gives the complexity class NL (non-deterministic logarithmic space). Something in common with our work is that the representation of strings used by [46] is more or less a semantic version of Church encodings (see [46, §3.2]). There is one main difference with what one usually calls implicit complexity: Seiller's result does not take place inside a syntactically defined programming language (and it is far from obvious how to turn this model into a similarly expressive syntax, because of the previously mentioned inversion of control).

Controlling expressible functions with non-commutativity. The tree-processing programming language of Kodama, Suenaga and Kobayashi [26] uses non-commutative types to force programs to process their input in a depth-first, left-to-right fashion. This allows them to be compiled into a target language that works on a stream of tokens, suggesting a possible connection with nested word automata [4]. The non-commutativity is restricted to arguments of ground type in [26], whereas it is important for our $\lambda\wp$-calculus that it applies at all orders (indeed, since we encode data as functions, higher-order functions are pervasive).

References

1 Samson Abramsky. Temperley–Lieb Algebra: From Knot Theory to Logic and Computation via Quantum Mechanics. In Goong Chen, Louis Kauffman, and Samuel Lomonaco, editors, *Mathematics of Quantum Computation and Quantum Technology*, volume 20074453, pages 515–558. Chapman and Hall/CRC, September 2007. doi:10.1201/9781584889007.ch15.

2 Klaus Aehlig. A Finite Semantics of Simply-Typed Lambda Terms for Infinite Runs of Automata. *Logical Methods in Computer Science*, 3(3), July 2007. doi:10.2168/LMCS-3(3:1)2007.

3 Rajeev Alur and Loris D'Antoni. Streaming Tree Transducers. *Journal of the ACM*, 64(5):1–55, August 2017. doi:10.1145/3092842.

4 Rajeev Alur and P. Madhusudan. Adding nesting structure to words. *Journal of the ACM*, 56(3):16:1–16:43, 2009. doi:10.1145/1516512.1516518.

5 Rajeev Alur and Pavol Černý. Expressiveness of streaming string transducers. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010)*, pages 1–12, 2010. `doi:10.4230/LIPIcs.FSTTCS.2010.1`.

6 Jean-Marc Andreoli, Gabriele Pulcini, and Paul Ruet. Permutative logic. In C.-H. Luke Ong, editor, *Computer Science Logic, 19th International Workshop, CSL 2005, 14th Annual Conference of the EACSL, Oxford, UK, August 22-25, 2005, Proceedings*, volume 3634 of *Lecture Notes in Computer Science*, pages 184–199. Springer, 2005. `doi:10.1007/11538363_14`.

7 Andrew Barber. Dual Intuitionistic Linear Logic. Technical report ECS-LFCS-96-347, LFCS, University of Edinburgh, 1996. URL: `http://www.lfcs.inf.ed.ac.uk/reports/96/ECS-LFCS-96-347/`.

8 Mikołaj Bojańczyk. The simplest transducer models and their Krohn-Rhodes decompositions. `https://www.mimuw.edu.pl/~bojan/slides/transducer-course/krohn-rhodes.html`. Slides of a lecture given at FSTTCS '19, accessed on 11-02-2020.

9 Mikołaj Bojańczyk. Automata column: Some Open Problems in Automata and Logic. *ACM SIGLOG News*, 1(2):3–12, October 2014. `doi:10.1145/2677161.2677163`.

10 Mikołaj Bojańczyk. Polyregular Functions. *CoRR*, abs/1810.08760, October 2018. `arXiv:1810.08760`.

11 Mikołaj Bojańczyk, Sandra Kiefer, and Nathan Lhote. String-to-String Interpretations With Polynomial-Size Output. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 106:1–106:14. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019. `doi:10.4230/LIPIcs.ICALP.2019.106`.

12 Corrado Böhm and Alessandro Berarducci. Automatic synthesis of typed λ-programs on term algebras. *Theoretical Computer Science*, 39:135–154, January 1985. `doi:10.1016/0304-3975(85)90135-5`.

13 Henry DeYoung and Frank Pfenning. Substructural proofs as automata. In Atsushi Igarashi, editor, *Programming Languages and Systems - 14th Asian Symposium, APLAS 2016, Hanoi, Vietnam, November 21-23, 2016, Proceedings*, volume 10017 of *Lecture Notes in Computer Science*, pages 3–22, 2016. `doi:10.1007/978-3-319-47958-3_1`.

14 Joost Engelfriet and Hendrik Jan Hoogeboom. MSO definable string transductions and two-way finite-state transducers. *ACM Transactions on Computational Logic*, 2(2):216–254, April 2001. `doi:10.1145/371316.371512`.

15 Emmanuel Filiot and Pierre-Alain Reynier. Transducers, Logic and Algebra for Functions of Finite Words. *ACM SIGLOG News*, 3(3):4–19, August 2016. `doi:10.1145/2984450.2984453`.

16 Neil Ghani and Alexander Kurz. Higher dimensional trees, algebraically. In Till Mossakowski, Ugo Montanari, and Magne Haveraaen, editors, *Algebra and Coalgebra in Computer Science, Second International Conference, CALCO 2007, Bergen, Norway, August 20-24, 2007, Proceedings*, volume 4624 of *Lecture Notes in Computer Science*, pages 226–241. Springer, 2007. `doi:10.1007/978-3-540-73859-6_16`.

17 Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–101, January 1987. `doi:10.1016/0304-3975(87)90045-4`.

18 Jean-Yves Girard. Towards a geometry of interaction. In J. W. Gray and A. Scedrov, editors, *Categories in Computer Science and Logic*, volume 92 of *Contemporary Mathematics*, pages 69–108. American Mathematical Society, Providence, RI, 1989. Proceedings of a Summer Research Conference held June 14–20, 1987. `doi:10.1090/conm/092/1003197`.

19 Jean-Yves Girard. Light Linear Logic. *Information and Computation*, 143(2):175–204, June 1998. `doi:10.1006/inco.1998.2700`.

20 Jean-Yves Girard, Andre Scedrov, and Philip J. Scott. Bounded linear logic: a modular approach to polynomial-time computability. *Theoretical Computer Science*, 97(1):1–66, April 1992. `doi:10.1016/0304-3975(92)90386-T`.

21 Charles Grellois. *Semantics of linear logic and higher-order model-checking*. PhD thesis, Université Paris 7, April 2016. URL: `https://tel.archives-ouvertes.fr/tel-01311150/`.

22 Alessio Guglielmi. A system of interaction and structure. *ACM Transactions on Computational Logic*, 8(1), January 2007. `doi:10.1145/1182613.1182614`.

23 Matthew Hague, Andrzej S. Murawski, C.-H. Luke Ong, and Olivier Serre. Collapsible Pushdown Automata and Recursion Schemes. *ACM Transactions on Computational Logic*, 18(3):25:1–25:42, August 2017. `doi:10.1145/3091122`.

24 Gerd G. Hillebrand and Paris C. Kanellakis. On the Expressive Power of Simply Typed and Let-Polymorphic Lambda Calculi. In *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science*, pages 253–263. IEEE Computer Society, 1996. `doi:10.1109/LICS.1996.561337`.

25 Naoki Kobayashi. Model Checking Higher-Order Programs. *Journal of the ACM*, 60(3):1–62, June 2013. `doi:10.1145/2487241.2487246`.

26 Koichi Kodama, Kohei Suenaga, and Naoki Kobayashi. Translation of tree-processing programs into stream-processing programs based on ordered linear type. *Journal of Functional Programming*, 18(3):333–371, 2008. `doi:10.1017/S0956796807006570`.

27 Kenneth Krohn and John Rhodes. Algebraic theory of machines. I. Prime decomposition theorem for finite semigroups and machines. *Transactions of the American Mathematical Society*, 116:450–464, 1965. `doi:10.1090/S0002-9947-1965-0188316-1`.

28 Denis Kuperberg, Laureline Pinault, and Damien Pous. Cyclic Proofs and Jumping Automata. In Arkadev Chattopadhyay and Paul Gastin, editors, *39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019)*, volume 150 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 45:1–45:14. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019. `doi:10.4230/LIPIcs.FSTTCS.2019.45`.

29 Joachim Lambek. The mathematics of sentence structure. *American Mathematical Monthly*, 65(3):154–170, 1958.

30 Olivier Laurent. Polynomial time in untyped elementary linear logic. *Theoretical Computer Science*, 813:117–142, April 2020. `doi:10.1016/j.tcs.2019.10.002`.

31 Daniel Leivant. Reasoning about functional programs and complexity classes associated with type disciplines. In *24th Annual Symposium on Foundations of Computer Science (FOCS 1983)*, pages 460–469, Tucson, AZ, USA, November 1983. `doi:10.1109/SFCS.1983.50`.

32 Harry G. Mairson. A simple proof of a theorem of Statman. *Theoretical Computer Science*, 103(2):387–394, September 1992. `doi:10.1016/0304-3975(92)90020-G`.

33 Damiano Mazza. *Polyadic Approximations in Logic and Computation*. Habilitation à diriger des recherches, Université Paris 13, November 2017. URL: `https://lipn.fr/~mazza/papers/Habilitation.pdf`.

34 Paul-André Melliès. Higher-order parity automata. In *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–12, Reykjavik, Iceland, June 2017. IEEE. `doi:10.1109/LICS.2017.8005077`.

35 Paul-André Melliès. Ribbon Tensorial Logic. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science - LICS '18*, pages 689–698, Oxford, United Kingdom, 2018. ACM Press. `doi:10.1145/3209108.3209129`.

36 Anca Muscholl and Gabriele Puppis. The Many Facets of String Transducers. In Rolf Niedermeier and Christophe Paul, editors, *36th International Symposium on Theoretical Aspects of Computer Science (STACS 2019)*, volume 126 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 2:1–2:21. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019. `doi:10.4230/LIPIcs.STACS.2019.2`.

37 Lê Thành Dũng Nguyễn and Pierre Pradic. Implicit automata in typed λ-calculi I: aperiodicity in a non-commutative logic, 2020. Full version of this article. URL: `https://hal.archives-ouvertes.fr/hal-02476219`.

38 C.-H. Luke Ong. On Model-Checking Trees Generated by Higher-Order Recursion Schemes. In *21st Annual IEEE Symposium on Logic in Computer Science (LICS'06)*, pages 81–90, Seattle, WA, USA, 2006. IEEE. doi:10.1109/LICS.2006.38.

39 Jeff Polakow and Frank Pfenning. Natural deduction for intuitionistic non-communicative linear logic. In Jean-Yves Girard, editor, *Typed Lambda Calculi and Applications, 4th International Conference, TLCA'99, L'Aquila, Italy, April 7-9, 1999, Proceedings*, volume 1581 of *Lecture Notes in Computer Science*, pages 295–309. Springer, 1999. doi:10.1007/3-540-48959-2_21.

40 Jeff Polakow and Frank Pfenning. Relating Natural Deduction and Sequent Calculus for Intuitionistic Non-Commutative Linear Logic. *Electronic Notes in Theoretical Computer Science*, 20:449–466, January 1999. doi:10.1016/S1571-0661(04)80088-4.

41 Christian Retoré. Pomset logic: A non-commutative extension of classical linear logic. In Philippe de Groote, editor, *Typed Lambda Calculi and Applications, Third International Conference on Typed Lambda Calculi and Applications, TLCA '97, Nancy, France, April 2-4, 1997, Proceedings*, volume 1210 of *Lecture Notes in Computer Science*, pages 300–318. Springer, 1997. doi:10.1007/3-540-62688-3_43.

42 John C. Reynolds. The discoveries of continuations. *LISP and Symbolic Computation*, 6(3):233–247, November 1993. doi:10.1007/BF01019459.

43 James Rogers. Syntactic Structures as Multi-dimensional Trees. *Research on Language and Computation*, 1(3):265–305, September 2003. doi:10.1023/A:1024695608419.

44 Jacques Sakarovitch. *Elements of Automata Theory.* Cambridge University Press, 2009. Translated by Reuben Thomas. doi:10.1017/CBO9781139195218.

45 Sylvain Salvati. Recognizability in the simply typed lambda-calculus. In Hiroakira Ono, Makoto Kanazawa, and Ruy J. G. B. de Queiroz, editors, *Logic, Language, Information and Computation, 16th International Workshop, WoLLIC 2009, Tokyo, Japan, June 21-24, 2009. Proceedings*, volume 5514 of *Lecture Notes in Computer Science*, pages 48–60. Springer, 2009. doi:10.1007/978-3-642-02261-6_5.

46 Thomas Seiller. Interaction Graphs: Non-Deterministic Automata. *ACM Transactions on Computational Logic*, 19(3):21:1–21:24, August 2018. doi:10.1145/3226594.

47 Howard Straubing. First-order logic and aperiodic languages: a revisionist history. *ACM SIGLOG News*, 5(3):4–20, 2018. doi:10.1145/3242953.3242956.

48 Kazushige Terui. Semantic Evaluation, Intersection Types and Complexity of Simply Typed Lambda Calculus. In *23rd International Conference on Rewriting Techniques and Applications (RTA'12)*, pages 323–338, 2012. doi:10.4230/LIPIcs.RTA.2012.323.

49 Igor Walukiewicz. LambdaY-calculus with priorities. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Vancouver, BC, Canada, June 24-27, 2019*, pages 1–13. IEEE, 2019. doi:10.1109/LICS.2019.8785674.

50 David N. Yetter. Quantales and (noncommutative) linear logic. *The Journal of Symbolic Logic*, 55(1):41–64, March 1990. doi:10.2307/2274953.

51 Noam Zeilberger and Alain Giorgetti. A correspondence between rooted planar maps and normal planar lambda terms. *Logical Methods in Computer Science*, 11(3), September 2015. doi:10.2168/LMCS-11(3:22)2015.

Computing Measures of Weak-MSO Definable Sets of Trees

Damian Niwiński
Institute of Informatics, University of Warsaw, Poland
https://www.mimuw.edu.pl/~niwinski/
niwinski@mimuw.edu.pl

Marcin Przybyłko
Fachbereich Informatik, University of Bremen, Germany
przybyl@uni-bremen.de

Michał Skrzypczak
Institute of Informatics, University of Warsaw, Poland
https://www.mimuw.edu.pl/~mskrzypczak/
mskrzypczak@mimuw.edu.pl

Abstract

This work addresses the problem of computing measures of recognisable sets of infinite trees. An algorithm is provided to compute the probability measure of a tree language recognisable by a weak alternating automaton, or equivalently definable in weak monadic second-order logic. The measure is the uniform coin-flipping measure or more generally it is generated by a branching stochastic process. The class of tree languages in consideration, although smaller than all regular tree languages, comprises in particular the languages definable in the alternation-free μ-calculus or in temporal logic CTL. Thus, the new algorithm may enhance the toolbox of probabilistic model checking.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects; Mathematics of computing → Markov processes

Keywords and phrases infinite trees, weak alternating automata, coin-flipping measure

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.136

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

Funding *Damian Niwiński*: Polish National Science Centre grant no. 2014-13/B/ST6/03595.
Marcin Przybyłko: ERC consolidator grant 647289 CODA.
Michał Skrzypczak: Polish National Science Centre grant no. 2016/22/E/ST6/00041.

1 Introduction

The non-emptiness problem asks if an automaton accepts at least one object. From a logical perspective, it is an instance of the consistency question: does a given specification have a model? Sometimes it is also relevant to ask a quantitative version of this question: whether a *non-negligible* set of models satisfy the specification. When taken to the realm of probability theory, this boils down to estimating the probability that a random object is accepted by a given automaton. In this paper, models under consideration are infinite binary trees labelled by a finite alphabet. Our main problem of interest is the following.

▶ **Problem 1.** *Given a regular tree language L, compute the probability that a randomly generated tree belongs to L.*

In other words, we ask for the probability measure of L. Here, the tree language L might be given by a formula of monadic second-order logic, but for complexity reasons it is more suitable to present it by a tree automaton or by a formula of modal μ-calculus, see e.g. [9,13]. By default, the considered measure is the uniform *coin-flipping* measure, where each letter

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 136; pp. 136:1–136:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

is chosen independently at random; but also more specific measures are of interest. If the computed probability is rational then it can be represented explicitly, but the measure can be irrational, see e.g. [15], and may require more complex representation. One of the possible choices, exploited in this paper, is a formula over the field of reals $\mathbb{R}$.

Chen et al. [6] addressed Problem 1 in the case where the tree language L is recognised by a deterministic top-down automaton and the measure is induced by a stochastic branching process, which then makes also a part of the input data. Their algorithm compares the probability with any given rational number in polynomial space and with 0 or 1 in polynomial time. The limitations of this result come from the deterministic nature of the considered automata: deterministic top-down tree automata are known to have limited expressive power within all regular tree languages.

Michalewski and Mio [15] stated Problem 1 explicitly and solved it for languages L given by so-called *game automata* and the coin-flipping measure. This class of automata subsumes deterministic ones and captures some important examples including the game languages, cf. [10], but even here the strength of non-determinism is limited; in particular, the class is not closed under finite union. The algorithm from [15] reduces the problem to computing the value of a Markov branching play, and uses Tarski's decision procedure for the theory of reals. These authors also discover that the measure of a regular tree language can be irrational, which stays in contrast with the case of ω-regular languages, i.e. regular languages of infinite words, where the coin-flipping measure is always rational, cf. [5].

Another step towards a solution to Problem 1 was made by the second author of the present article, who proposed an algorithm to compute the coin-flipping measure of tree languages definable in fragments of first-order logic [20]. This work is subsumed in a report [21] (accepted for publication in a journal) co-authored with the third author, where a new class of languages L is also resolved: tree languages recognised by safety automata, i.e. non-deterministic automata with a trivial accepting condition.

An analogue of Problem 1 can be stated for ω-regular languages. As noted by [6], the problem then reduces to a well-known question in verification solved by Courcoubetis and Yannakakis [8] already in the 1990s, namely whether a run of a finite-state Markov chain satisfies an ω-regular property. The algorithm runs in single-exponential time w.r.t. the automaton (and linear w.r.t. the Markov chain). A related question was also studied by Staiger [24], who gave an algorithm to compute Hausdorff dimension and Hausdorff measure of a given ω-regular language.

In general, Problem 1 remains unsolved. At first sight, one may even wonder if it is well-stated, as regular tree languages need not in general be Borel, cf. [18]. However, due to [12, 16], we know that regular languages of trees are always universally measurable.

In the present paper, we solve Problem 1 in the case where the language L is recognised by a weak alternating automaton or, equivalently, defined by a formula of weak monadic second-order logic, cf. [17]. The class of tree languages in consideration is incomparable with the one considered by Michalewski and Mio [15], but subsumes those considered in [20, 21]. Yet another presentation of this class can be given in terms of alternation-free fragment of modal μ-calculus, see [1] for details. This fragment is known to be useful in verification and model checking, in particular, temporal logic CTL embeds into this fragment.

We consider the coin-flipping measure as our primary case, but we also show how to extend our approach to measures generated by stochastic branching processes, as in [6]. The computed probability is presented by a first-order formula in prenex normal form over the field of reals. The provided formula is exponential in the size of the automaton and polynomial in the size of the branching process. Moreover, the quantifier alternation of the computed formula is constant (equal 4). Combined with the known decision procedures for the theory of reals, this gives the following.

▶ **Theorem 2.** *There is an algorithm that inputs a weak alternating parity automaton $\mathcal{A}$, a branching process $\mathcal{P}$, and a rational number q encoded in binary; and decides if the measure generated by $\mathcal{P}$ of the language recognised by $\mathcal{A}$ is equal, smaller, or greater than q. The algorithm works in time polynomial in q, doubly exponential in $\mathcal{A}$, and singly exponential in $\mathcal{P}$.*

Similarly to the approach taken in [21], we reduce the problem to computation of an appropriate probability distribution over the powerset of the automaton's states. To do so, we consider the set of all such distributions $\mathcal{D}\mathsf{P}(Q)$ with a suitable ordering $\preceq$. The structure is in fact a finitary case of a probabilistic powerdomain introduced by Saheb-Djahromi [22] (see also [14]), but we do not exploit category-theoretic concepts in this paper. The key step is an approximation of the target language L by two families of tree languages representing safety and reachability properties, respectively. Then we can apply fixed-point constructions thanks to a kind of synergy between the order and topological properties of $\mathcal{D}\mathsf{P}(Q)$.

2 Trees, topology, and measure

The set of natural numbers $\{0, 1, 2, \ldots\}$ is denoted by $\mathbb{N}$, or by ω whenever we treat it as an ordinal. A finite non-empty set A is called an *alphabet*. By $\mathsf{P}(X)$ we denote the family of all subsets of a set X. The set of *finite words* over an alphabet A (including the *empty word* ε) is denoted by A^*, and the set of ω-words by A^ω. The *length* of a finite word $w \in A^*$ is denoted by $|w|$. A *full infinite binary tree* over an alphabet A (or simply a *tree* if confusion does not arise) is a mapping $t\colon \{\mathtt{L}, \mathtt{R}\}^* \to A$. The set of all such trees, denoted by Tr_A, can be equipped with a topology induced by a metric

$$d(t_1, t_2) = \begin{cases} 0 & \text{if } t_1 = t_2 \\ 2^{-n} \text{ with } n = \min\{|w| \mid t_1(w) \neq t_2(w)\} & \text{otherwise.} \end{cases}$$

It is well-known that this topology coincides with the product topology on A^ω, where A is a discrete topological space. The topology can be generated by a basis consisting of all the sets U_f, where $f\colon \mathrm{dom}(f) \to A$ is a function with a finite domain $\mathrm{dom}(f) \subset \{\mathtt{L}, \mathtt{R}\}^*$, and U_f consists of all trees t that coincide with f on $\mathrm{dom}(f)$. If A has at least 2 elements then this topology is homeomorphic to the Cantor discontinuum $\{0, 1\}^\omega$ (see, e.g. [19]).

The set of trees can be further equipped with a probabilistic measure μ_0, which is the standard Lebesgue measure on the product space defined on the basis by $\mu_0\left(U_f\right) = \left|A\right|^{-|\mathrm{dom}(f)|}$.

We note a useful property of this measure, which intuitively amounts to saying that events happening in incomparable nodes are independent. For $t \in \mathrm{Tr}_A$ and $v \in \{\mathtt{L}, \mathtt{R}\}^*$, the subtree of t induced by v is a tree $t{\restriction_v} \in \mathrm{Tr}_A$ defined by $t{\restriction_v}(w) = t(vw)$, for $w \in \{\mathtt{L}, \mathtt{R}\}^*$.

▶ Remark 3. If $v_1, \ldots, v_k \in \{\mathtt{L}, \mathtt{R}\}^*$ are pairwise incomparable nodes (i.e., none is a prefix of another) and $V_1, \ldots, V_k \subseteq \mathrm{Tr}_A$ are Borel sets then

$$\mu_0\left(\{t \in \mathrm{Tr}_A \mid t{\restriction_{v_i}} \in V_i \text{ for } i = 1, \ldots, k\}\right) \quad = \quad \mu_0(V_1) \cdot \ldots \cdot \mu_0(V_k). \tag{1}$$

We refer to e.g. [12] for more detailed considerations of measures on sets of infinite trees.

3 Tree automata and games

An *alternating parity automaton* over infinite trees can be presented as a tuple $\mathcal{A} = \langle A, Q, q_{\mathrm{I}}, \delta, \Omega\rangle$, where A is a finite alphabet; Q a finite set of *states*; $q_{\mathrm{I}} \in Q$ an *initial state*; $\delta\colon Q \times A \to \mathrm{BC}^+\big(\{\mathtt{L}, \mathtt{R}\} \times Q\big)$ a *transition function* that assigns to a pair $(q, a) \in Q \times A$ a finite positive Boolean combination of pairs $(d, q') \in \{\mathtt{L}, \mathtt{R}\} \times Q$; and finally $\Omega\colon Q \to \mathbb{N}$ is a *priority mapping*.

In this paper, we assume that automata are *weak*, i.e. the priorities $\Omega(q)$ are non-increasing along transitions. More precisely, if (d, q') is an atom that appears in the formula $\delta(q, a)$ then $\Omega(q) \geq \Omega(q')$. Given $n \in \mathbb{N}$, we denote by $Q_{<n}$ and $Q_{\geq n}$ the subsets of Q consisting of those states whose priority is respectively strictly smaller or greater than n.

The semantics of an automaton can be given in a terms of a game played by two players $\exists$ and $\forall$ over a tree t in Tr_A from a state $p \in Q$. Let Γ be the set of all sub-formulae of the formulae in $\delta(q, a)$, for all $(q, a) \in Q \times A$. The set of *positions* of the game is the set $\big(Q \sqcup \Gamma\big) \times \{\mathrm{L}, \mathrm{R}\}^*$ and the *initial position* is $\big(p, \varepsilon\big)$. The positions of the form $\big(q, v\big)$, $\big(\phi_1 \vee \phi_2, v\big)$, and $\big((d, q), v\big)$ are controlled by $\exists$, while the positions of the form $\big(\phi_1 \wedge \phi_2, v\big)$ are controlled by $\forall$. The edges connect the following types of positions:

- $\big(q, v\big)$ and $\big(\delta(q, t(v)), v\big)$,
- $\big(\phi_1 \vee \phi_2, v\big)$ and $\big(\phi_i, v\big)$ for $i = 1, 2$,
- $\big(\phi_1 \wedge \phi_2, v\big)$ and $\big(\phi_i, v\big)$ for $i = 1, 2$,
- $\big((d, q), v\big)$ and $\big(q, v \cdot d\big)$.

We assume that every formula in the image $\delta(Q \times A)$ is non-trivial and, thus, every position is a source of some edge.

The directed graph described above forms the *arena* of our game that we denote by $\mathcal{G}(t, p)$. A *play* in the arena is any infinite path starting from the initial position $\big(p, \varepsilon\big)$. We call the positions of the form (q, v) *state positions*. Given a play π, the *states* of the play denoted states(π) is the sequence of states $(q_0, q_1, \ldots) \in Q^\omega$ such that the successive state positions visited during π are $\big(q_i, v_i\big)$, for $i = 0, 1, \ldots$, and some $(v_i)_{i \in \omega}$.

To complete the definition of the game, we specify a winning criterion for $\exists$. The default is the *parity condition*, but we will also consider other criteria. Let

$$\mathrm{Runs} \overset{\text{def}}{=} \{\rho \in Q^\omega \mid \forall i \in \omega.\ \Omega(\rho(i)) \geq \Omega(\rho(i{+}1))\}$$

be the set that contains all sequences of states that induce non-increasing sequences of priorities. Notice that since $\mathcal{A}$ is weak, only such sequences may arise in the game. In general, a *winning condition* is any set $W \subseteq \mathrm{Runs}$. That is, a play π is *winning* for $\exists$ with respect to W if, and only if, states$(\pi) \in W$. The game with a winning set W is denoted by $\mathcal{G}(t, p, W)$.

The *parity condition* $W_P \subseteq \mathrm{Runs}$ for a weak automaton amounts to: $(q_0, q_1, \ldots) \in W_P$ if $\lim_{i \to \infty} \Omega(q_i) \equiv 0 \mod 2$, i.e. the limit priority of states visited in a play is even. Let $\mathrm{L}(\mathcal{A}, p)$ be the set of trees such that $\exists$ has a winning strategy in $\mathcal{G}(t, p, W_P)$. Then, the language of an automaton $\mathcal{A}$ is the set $\mathrm{L}(\mathcal{A}) \overset{\text{def}}{=} \mathrm{L}(\mathcal{A}, q_{\mathrm{I}})$, where q_{I} is the initial state of $\mathcal{A}$.

As mentioned above, we will consider games with various winning criteria. The following simple observation is useful.

▶ Remark 4. If $W \subseteq W' \subseteq \mathrm{Runs}$ then the following implication holds: if $\exists$ wins $\mathcal{G}(t, p, W)$ then $\exists$ wins also $\mathcal{G}(t, p, W')$.

Since the winning criteria in consideration will always be ω-regular languages of infinite words, we implicitly rely on the following classical fact (cf. [13]).

▶ **Proposition 5.** *Games on graphs with ω-regular winning conditions are finite memory determined.*

We will also use the following fact, cf. e.g. [17, 23].

▶ **Proposition 6.** *For a weak alternating parity automaton $\mathcal{A}$, all tree languages $\mathrm{L}(\mathcal{A}, p)$ are Borel and, consequently, measurable with respect to the uniform measure μ_0 (and also any other Borel measure on Tr_A).*

Note that measurability holds for non-weak automata as well [12].

4 Approximations

For the sake of this section we fix a weak alternating parity automaton $\mathcal{A}$. Our aim is to provide some useful approximations for the tree languages $\mathrm{L}(\mathcal{A}, p)$. The approximations are simply some families of tree languages indexed by states $p \in Q$. Those families, called *Q-indexed families*, or *Q-families* for short, are represented by functions $\mathcal{L}\colon Q \to \mathsf{P}(\mathrm{Tr}_A)$. By the construction, we will guarantee that the tree languages $\mathcal{L}(q)$ will themselves be recognisable by some weak alternating automata. Each Q-family naturally possesses a dual representation by a mapping $\mathrm{Tr}_A \to \mathsf{P}(Q)$ that we denote by the same letter (but with different brackets)

$$\mathcal{L}[t] \stackrel{\text{def}}{=} \{q \in Q \mid t \in \mathcal{L}(q)\} \in \mathsf{P}(Q).$$

If $\rho \in \mathrm{Runs} \subseteq Q^\omega$ is an infinite sequence of states then $\lim_{i\to\infty} \Omega(\rho(i))$ (denoted by $\mathrm{limit}(\rho)$) exists, because by the definition of Runs the priorities are non-increasing and bounded. Recall that $W_P \subseteq \mathrm{Runs}$ is the set of runs satisfying the parity condition, i.e. $W_P = \{\rho \in \mathrm{Runs} \mid \mathrm{limit}(\rho) \equiv 0 \mod 2\}$. For $i, n \in \mathbb{N}$, consider the following subsets of Runs:

$$S_i^n \stackrel{\text{def}}{=} W_P \cup \big\{\rho \in \mathrm{Runs} \mid \Omega(\rho(i)) \geq n\big\},$$
$$S_\infty^n \stackrel{\text{def}}{=} W_P \cup \big\{\rho \in \mathrm{Runs} \mid \mathrm{limit}(\rho) \geq n\big\},$$
$$R_i^n \stackrel{\text{def}}{=} W_P \cap \big\{\rho \in \mathrm{Runs} \mid \Omega(\rho(i)) < n\big\},$$
$$R_\infty^n \stackrel{\text{def}}{=} W_P \cap \big\{\rho \in \mathrm{Runs} \mid \mathrm{limit}(\rho) < n\big\}.$$

Connotatively, the name of the sets S_i^n comes from the condition of *safety*, while the sets R_i^n are named after *reachability*. More precisely, S_i^n is an over-approximation of W_P, that makes $\exists$ win also if she manages to reach a priority $\geq n$ in the ith visited node of a given tree. Analogously, R_i^n is an under-approximation of W_P that makes $\forall$ win in the above case.

Based on the above definitions, we define the respective Q-families. For $p \in Q$, let $\mathcal{S}_i^n(p)$, $\mathcal{S}_\infty^n(p)$, $\mathcal{R}_i^n(p)$, and $\mathcal{R}_\infty^n(p)$ be the sets of trees such that $\exists$ has a winning strategy in the game $\mathcal{G}(t, p, W)$, where W is respectively S_i^n, S_∞^n, R_i^n, and R_∞^n. Figure 1 below depicts the way these Q-families are used in the general construction.

It is easy to see that all the tree languages above can be recognised by weak parity alternating automata.

▶ **Lemma 7.** *For every $n \in \mathbb{N}$ and $i \in \mathbb{N}$, we have*

$$S_i^n \supseteq S_{i+1}^n \supseteq S_\infty^n \text{ and } R_i^n \subseteq R_{i+1}^n \subseteq R_\infty^n.$$

Analogously, for every $p \in Q$,

$$\mathcal{S}_i^n(p) \supseteq \mathcal{S}_{i+1}^n(p) \supseteq \mathcal{S}_\infty^n(p) \text{ and } \mathcal{R}_i^n(p) \subseteq \mathcal{R}_{i+1}^n(p) \subseteq \mathcal{R}_\infty^n(p).$$

Proof. The first property follows directly from the definition of Runs, which guarantees that $\Omega(\rho(0)) \geq \Omega(\rho(1)) \geq \ldots \geq \mathrm{limit}(\rho)$. Then, the second property follows from Remark 4. ◀

It is straightforward to see that $S_\infty^n = \bigcap_{i\in\mathbb{N}} S_i^n$ and $R_\infty^n = \bigcup_{i\in\mathbb{N}} R_i^n$. However, it is not clear that these equalities imply the desired properties for the respective sets of trees. Lemma 9 below implies that it is the case. The proof relies on combinatorics of binary trees, namely on König's Lemma.

▶ **Lemma 8.** *Take $n \in \mathbb{N}$ and $p \in Q$. Let $B^n_\infty = \{\rho \in \mathrm{Runs} \mid \mathrm{limit}(\rho) < n\}$ and, for $i \in \mathbb{N}$, let $B^n_i = \{\rho \in \mathrm{Runs} \mid \Omega(\rho(i)) < n\}$. If σ is a winning strategy of $\exists$ in $\mathcal{G}(t, p, B^n_\infty)$ then there exists a number $J \in \mathbb{N}$, such that σ is actually winning in $\mathcal{G}(t, p, B^n_J)$. An analogous property holds if σ is a winning strategy for $\forall$.*

Proof. Let σ be a winning strategy of $\exists$ in $\mathcal{G}(t, p, B^n_\infty)$ (the case of $\forall$ is completely analogous). Let $T \subseteq \big(Q \times \{\mathtt{L}, \mathtt{R}\}\big)^*$ be the set of sequences $(q_i, d_i)_{i \leq \ell}$, with $q_0 = p$, such that there exists a play consistent with σ that visits all the positions $(q_i, d_0 \cdots d_{i-1})$ for $i = 0, 1, \ldots, \ell$, and additionally $\Omega(q_\ell) \geq n$. Observe that T is prefix-closed. Thus, we can treat T as a tree. Moreover, as $Q \times \{\mathtt{L}, \mathtt{R}\}$ is finite, T is finitely branching. If T is finite then there exists J such that all the sequences in T have length at most J. In that case σ is winning in $\mathcal{G}(t, p, B^n_J)$, and we are done.

For the sake of contradiction, suppose that T is infinite. Apply König's Lemma to obtain an infinite path $(q_i, d_i)_{i \in \omega}$ in T. By the definition of T, it implies that there exists an infinite play consistent with σ such that $(q_i)_{i \in \omega}$ is the sequence of states visited during the play. But this is a contradiction, because $\mathrm{limit}\big((q_i)_{i \in \omega}\big) \geq n$ by the definition of T and, therefore, the considered play is losing for $\exists$. ◀

▶ **Lemma 9.** *Using the above notions, for every state $p \in Q$, we have*

$$\mathcal{S}^n_\infty(p) = \bigcap_{i \in \mathbb{N}} \mathcal{S}^n_i(p) \quad \textit{and} \quad \mathcal{R}^n_\infty(p) = \bigcup_{i \in \mathbb{N}} \mathcal{R}^n_i(p).$$

Proof. Consider the first claim and take a tree $t \in \mathrm{Tr}_A$ such that for every $i \in \mathbb{N}$ we have $t \in \mathcal{S}^n_i(p)$. We need to prove that $t \in \mathcal{S}^n_\infty(p)$. Assume contrarily, that $t \notin \mathcal{S}^n_\infty(p)$. By determinacy, see Proposition 5, it means that there exists a strategy σ' of $\forall$ such that for every play π consistent with σ', we have $\mathrm{limit}(\mathrm{states}\,(\pi)) < n$ and $\mathrm{limit}(\mathrm{states}\,(\pi))$ is odd. Hence, in particular, σ' is winning for $\forall$ in $\mathcal{G}(t, p, B^n_\infty)$. Therefore, by Lemma 8, we know that there exists a number $J \in \mathbb{N}$ such that, for every π consistent with σ' with $\mathrm{states}\,(\pi) = (q_0, q_1, \ldots)$, we have $\Omega(q_J) < n$. Therefore, the strategy σ' witnesses that $t \notin \mathcal{S}^n_J(p)$, a contradiction.

We now prove the second claim. Take a tree $t \in \mathcal{R}^n_\infty(p)$. We need to prove that $t \in \mathcal{R}^n_i(p)$ for some $i \in \mathbb{N}$. Let σ be a strategy of $\exists$ witnessing that $t \in \mathcal{R}^n_\infty(p)$. Again, Lemma 8 guarantees that there exists a number $J \in \mathbb{N}$ such that if π is a play consistent with σ with $\mathrm{states}\,(\pi) = (q_0, q_1, \ldots)$ then $\Omega(q_J) < n$. Thus, $t \in \mathcal{R}^n_J(p)$. ◀

The following lemma provides another characterisation of the above Q-families.

▶ **Lemma 10.** *For each $p \in Q$, we have $\mathcal{S}^0_i(p) = \mathrm{Tr}_A$ and $\mathcal{R}^0_i(p) = \emptyset$. Take $n > 0$. If $\Omega(p) \geq n$ then $\mathcal{S}^n_0(p) = \mathrm{Tr}_A$ and $\mathcal{R}^n_0(p) = \emptyset$. If $\Omega(p) < n$ then*

$$\begin{aligned} &\mathrm{L}(\mathcal{A}, p) = \mathcal{S}^n_0(p), && \mathrm{L}(\mathcal{A}, p) = \mathcal{R}^n_0(p), \\ &\mathrm{L}(\mathcal{A}, p) = \mathcal{S}^{n-1}_\infty(p) \quad \textit{for odd } n, && \mathrm{L}(\mathcal{A}, p) = \mathcal{R}^{n-1}_\infty(p) \quad \textit{for even } n. \end{aligned}$$

Proof. The cases of $n = 0$ are trivial. The first two claims in the case $\Omega(p) \geq n$ follow directly from the definitions. Take p such that $\Omega(p) < n$. Notice that in that case the sequence of states ρ in a play in $\mathcal{G}(t, p)$ satisfies

$$\begin{aligned} &\rho \in W_P \iff \rho \in S^n_0, && \rho \in W_P \iff \rho \in R^n_0, \\ &\rho \in W_P \iff \rho \in S^{n-1}_\infty \quad \text{for odd } n, && \rho \in W_P \iff \rho \in R^{n-1}_\infty \quad \text{for even } n. \end{aligned}$$

where the first two equivalences follow from the fact that $\Omega(\rho(0)) = \Omega(p) < n$. The last two equivalences can be derived from the fact that $\mathrm{limit}(\rho) \leq \Omega(p) < n$. First, we have $\mathrm{limit}(\rho) \geq n{-}1 \Leftrightarrow \mathrm{limit}(\rho) = n{-}1$. Thus, if n is odd and $\mathrm{limit}(\rho) \geq n{-}1$ then we know that $\mathrm{limit}(\rho)$ is even. Analogously, if n is even then $n{-}1$ is odd and the fact that $\mathrm{limit}(\rho)$ is even guarantees that $\mathrm{limit}(\rho) < n{-}1$.

Clearly, the above equivalences imply that, under the assumption of the lemma, a strategy winning for the condition W_P is winning for the respective conditions and vice-versa. ◀

Our aim now is to define a function $\Delta\colon \mathsf{P}(Q) \times A \times \mathsf{P}(Q) \to \mathsf{P}(Q)$ that will allow us to form equations over Q-families. An ordered pair of sets of states $P_{\mathtt{L}}, P_{\mathtt{R}} \in \mathsf{P}(Q)$ induces a *valuation* $v_{P_{\mathtt{L}},P_{\mathtt{R}}}$ to the atoms in $\{\mathtt{L},\mathtt{R}\} \times Q$ defined by: $v_{P_{\mathtt{L}},P_{\mathtt{R}}}(d,p)$ is true if $p \in P_d$. Now, consider additionally a letter $a \in A$ and put

$$\Delta(P_{\mathtt{L}}, a, P_{\mathtt{R}}) = \big\{q \in Q \mid v_{P_{\mathtt{L}},P_{\mathtt{R}}} \models \delta(q,a)\big\}.$$

Equivalently, $q \in \Delta(P_{\mathtt{L}}, a, P_{\mathtt{R}})$ if $\exists$ can play the finite game represented by $\delta(q,a)$ in such a way to reach only such atoms (d,p) that satisfy $p \in P_d$.

▶ **Lemma 11.** *The function $\Delta\colon \mathsf{P}(Q) \times A \times \mathsf{P}(Q) \to \mathsf{P}(Q)$ is monotone, i.e. if $P_{\mathtt{L}} \subseteq P'_{\mathtt{L}}$ and $P_{\mathtt{R}} \subseteq P'_{\mathtt{R}}$ then $\Delta(P_{\mathtt{L}}, a, P_{\mathtt{R}}) \subseteq \Delta(P'_{\mathtt{L}}, a, P'_{\mathtt{R}})$.*

Proof. It follows directly from the fact that the Boolean formulae in $\delta(q,a)$ are positive. ◀

Recall that $t{\restriction}_v \in \mathrm{Tr}_A$ denotes the subtree of t induced by a node v, cf. Section 2. The following lemma shows how to increase the index i of the above Q-families $\mathcal{S}^n_i$ and $\mathcal{R}^n_i$.

▶ **Lemma 12.** *Take $n \in \mathbb{N}$, $i \in \mathbb{N}$, and a tree $t \in \mathrm{Tr}_A$. Then we have:*

$$\mathcal{S}^n_{i+1}[t] = \Delta\big(\mathcal{S}^n_i[t{\restriction}_{\mathtt{L}}], t(\varepsilon), \mathcal{S}^n_i[t{\restriction}_{\mathtt{R}}]\big),$$
$$\mathcal{R}^n_{i+1}[t] = \Delta\big(\mathcal{R}^n_i[t{\restriction}_{\mathtt{L}}], t(\varepsilon), \mathcal{R}^n_i[t{\restriction}_{\mathtt{R}}]\big).$$

The proof of this lemma is based on a standard technique of merging strategies: the game $\mathcal{G}(t,p)$ can be split into a finite game corresponding to the formula $\delta\big(p, t(\varepsilon)\big)$ that leads to the sub-games $\mathcal{G}(t{\restriction}_{\mathtt{L}}, p_{\mathtt{L}})$ and $\mathcal{G}(t{\restriction}_{\mathtt{R}}, p_{\mathtt{R}})$ for some states $p_{\mathtt{L}}, p_{\mathtt{R}} \in Q$.

Proof. Take a play π in the arena $\mathcal{G}(t,p)$ for some state $p \in Q$. Recall that, by the definition of the game, the initial position of the play is (p,ε) and the next state position will have the form (q,d), for some $q \in Q$ and $d \in \{\mathtt{L},\mathtt{R}\}$. Consider the suffix of the play π starting from that position. Clearly, this suffix induces a play, say π', in the arena $\mathcal{G}(t{\restriction}_d, q)$, starting from the position (q,ε) (technically, to satisfy our definition, we need also to replace every position (α, dv) by (α, v) in the original play). Moreover, the sequence of states visited by π', $\mathrm{states}(\pi')$, is a suffix of the sequence $\mathrm{states}(\pi)$ obtained by removing just the first element. By the definition of S^n_i and R^n_i we have therefore

$$\mathrm{states}(\pi) \in S^n_{i+1} \iff \mathrm{states}(\pi') \in S^n_i, \text{ and } \mathrm{states}(\pi) \in R^n_{i+1} \iff \mathrm{states}(\pi') \in R^n_i. \tag{2}$$

We will now provide the proof for $\mathcal{S}^n_{i+1}$, the case of $\mathcal{R}^n_{i+1}$ is analogous. Let $P_{\mathtt{L}}$ and $P_{\mathtt{R}}$ equal respectively $\mathcal{S}^n_i[t{\restriction}_{\mathtt{L}}]$ and $\mathcal{S}^n_i[t{\restriction}_{\mathtt{R}}]$. Put $a = t(\varepsilon)$. Recall that by the duality of the two representations of Q-families, $p \in \mathcal{S}^n_{i+1}[t]$ iff $t \in \mathcal{S}^n_{i+1}(p)$. So we need to prove that for every $p \in Q$ we have $t \in \mathcal{S}^n_{i+1}(p)$ if and only if $p \in \Delta(P_{\mathtt{L}}, a, P_{\mathtt{R}})$.

Assume that $t \in \mathcal{S}^n_{i+1}(p)$. Take a strategy σ witnessing that. Notice that if a position of the form (q,d) can be reached by σ then by (2) we know that $t{\restriction}_d \in \mathcal{S}^n_i(q)$, i.e. $q \in P_d$. Thus, the strategy σ witnesses that $p \in \Delta(P_{\mathtt{L}}, a, P_{\mathtt{R}})$.

$$\begin{tikzcd}
\vec{\mu_0}(\mathcal{S}^0_0) \arrow[d, "\mathcal{F}"'] & \vec{\mu_0}(\mathcal{R}^1_\infty) \arrow[r, mapsto, "\mathcal{Q}_{\geq 2}"'] & \vec{\mu_0}(\mathcal{S}^2_0) \arrow[d, "\mathcal{F}"'] & \vec{\mu_0}(\mathcal{R}^3_\infty) \arrow[r, mapsto, "\mathcal{Q}_{\geq 4}"'] & \vec{\mu_0}(\mathcal{S}^4_0) \arrow[d, "\mathcal{F}"'] & \cdots \\
\vec{\mu_0}(\mathcal{S}^0_1) \arrow[d, "\mathcal{F}"'] & \vec{\mu_0}(\mathcal{R}^1_4) \arrow[u, "\mathcal{F}"] & \vec{\mu_0}(\mathcal{S}^2_1) \arrow[d, "\mathcal{F}"'] & \vec{\mu_0}(\mathcal{R}^3_4) \arrow[u, "\mathcal{F}"] & \vec{\mu_0}(\mathcal{S}^4_1) \arrow[d, "\mathcal{F}"'] & \\
\vec{\mu_0}(\mathcal{S}^0_2) \arrow[d, "\mathcal{F}"'] & \vec{\mu_0}(\mathcal{R}^1_3) \arrow[u, "\mathcal{F}"] & \vec{\mu_0}(\mathcal{S}^2_2) \arrow[d, "\mathcal{F}"'] & \vec{\mu_0}(\mathcal{R}^3_3) \arrow[u, "\mathcal{F}"] & \vec{\mu_0}(\mathcal{S}^4_2) \arrow[d, "\mathcal{F}"'] & \cdots \\
\vec{\mu_0}(\mathcal{S}^0_3) \arrow[d, "\mathcal{F}"'] & \vec{\mu_0}(\mathcal{R}^1_2) \arrow[u, "\mathcal{F}"] & \vec{\mu_0}(\mathcal{S}^2_3) \arrow[d, "\mathcal{F}"'] & \vec{\mu_0}(\mathcal{R}^3_2) \arrow[u, "\mathcal{F}"] & \vec{\mu_0}(\mathcal{S}^4_3) \arrow[d, "\mathcal{F}"'] & \\
\vec{\mu_0}(\mathcal{S}^0_4) \arrow[d, "\mathcal{F}"'] & \vec{\mu_0}(\mathcal{R}^1_1) \arrow[u, "\mathcal{F}"] & \vec{\mu_0}(\mathcal{S}^2_4) \arrow[d, "\mathcal{F}"'] & \vec{\mu_0}(\mathcal{R}^3_1) \arrow[u, "\mathcal{F}"] & \vec{\mu_0}(\mathcal{S}^4_4) \arrow[d, "\mathcal{F}"'] & \\
\vec{\mu_0}(\mathcal{S}^0_\infty) \arrow[r, mapsto, "\mathcal{Q}_{<1}"] & \vec{\mu_0}(\mathcal{R}^1_0) \arrow[u, "\mathcal{F}"] & \vec{\mu_0}(\mathcal{S}^2_\infty) \arrow[r, mapsto, "\mathcal{Q}_{<3}"] & \vec{\mu_0}(\mathcal{R}^3_0) \arrow[u, "\mathcal{F}"] & \vec{\mu_0}(\mathcal{S}^4_\infty) \arrow[r, mapsto, "\mathcal{Q}_{<5}"] & \cdots
\end{tikzcd}$$

Figure 1 A schematic presentation of the relationship between the distributions used in the proof. The vertical axis represents the order $\preceq$, i.e. $\vec{\mu_0}(\mathcal{S}^0_0) \succeq \vec{\mu_0}(\mathcal{S}^0_\infty)$. The edges marked $\mathcal{F}$, $\mathcal{Q}_{<n}$, and $\mathcal{Q}_{\geq n}$ represent applications of the respective operations. The vertical convergence is understood in terms of pointwise limits in $\mathbb{R}^{\mathsf{P}(Q)}$.

For the opposite direction, assume that $p \in \Delta(P_{\mathtt{L}}, a, P_{\mathtt{R}})$. This means that there exists a finite strategy of $\exists$ that allows her to resolve the formula $\delta(p, a)$ in such a way that for every atom (d, q) that can be reached by this strategy, we have $(d, q) \in P_d$. The last means that $\exists$ has a winning strategy in the game $\mathcal{G}(t{\restriction}_d, q, \mathcal{S}^n_i)$. Now we can combine all above strategies in a strategy in the game $\mathcal{G}(t, p, S^n_{i+1})$, which by Equation (2) is again winning for $\exists$. Hence, $t \in \mathcal{S}^n_{i+1}(p)$, as desired. ◀

5 Measures and distributions

Following an approach started in [21], we transfer the problem of computing measures of tree languages $\mathrm{L}(\mathcal{A}, p)$ to computing a suitable probability distribution on the sets of the automaton states. We start with a general construction. For a finite set X, consider the set of probability distributions over X, $\mathcal{D}X \overset{\text{def}}{=} \{\alpha \colon X \to [0,1] \mid \sum_{x \in X} \alpha(x) = 1\}$. Observe that, if X is partially ordered by a relation $\leq$ then $\mathcal{D}X$ is partially ordered by a relation $\preceq$ defined as follows: $\alpha \preceq \beta$ if for every upward-closed[1] set $U \subseteq X$, we have $\sum_{x \in U} \alpha(x) \leq \sum_{x \in U} \beta(x)$. In this article, we are interested in $\langle X, \leq \rangle$ being the powerset $\mathsf{P}(Q)$ ordered by inclusion $\subseteq$.

▶ Remark 13. The relation $\preceq$ is a partial order on $\mathcal{D}X$ (as an intersection of a finite family of partial orders).

Every Q-family $\mathcal{L}$ for a weak alternating automaton $\mathcal{A}$ induces naturally a member of $\mathcal{D}\mathsf{P}(Q)$, which is a distribution $\vec{\mu_0}(\mathcal{L})$ defined by

$$\vec{\mu_0}(\mathcal{L})(P) = \mu_0\{t \in \mathrm{Tr}_A \mid \mathcal{L}[t] = P\}.$$

Here μ_0 is the uniform probability measure on Tr_A. The sets in consideration are measurable thanks to Proposition 6.

Note that if the language family is exactly $\mathcal{L}(q) = \mathrm{L}(\mathcal{A}, q)$ then the probability assigned to a set of states P amounts to the probability that a randomly chosen tree, with respect to μ_0, is accepted precisely from the states in P.

▶ **Lemma 14.** *If for each $q \in Q$ we have $\mathcal{L}(q) \subseteq \mathcal{L}'(q)$ then $\vec{\mu_0}(\mathcal{L}) \preceq \vec{\mu_0}(\mathcal{L}')$ in $\mathcal{D}\mathsf{P}(Q)$.*

[1] That is if $x \leq y$ and $x \in U$ then $y \in U$.

Proof. Take any upward-closed family $U \subseteq \mathsf{P}(Q)$. Then

$$\sum_{P\in U} \vec{\mu_0}\big(\mathcal{L}\big)(P) = \sum_{P\in U} \mu_0\big\{t\in \mathrm{Tr}_A \mid \mathcal{L}[t]=P\big\} = \mu_0\big\{t\in \mathrm{Tr}_A \mid \mathcal{L}[t]\in U\big\} \leq$$
$$\leq \mu_0\big\{t\in \mathrm{Tr}_A \mid \mathcal{L}'[t]\in U\big\} = \sum_{P\in U} \mu_0\big\{t\in \mathrm{Tr}_A \mid \mathcal{L}'[t]=P\big\} = \sum_{P\in U} \vec{\mu_0}\big(\mathcal{L}'\big)(P),$$

where the middle inequality follows from the assumption that $\mathcal{L}(q)\subseteq\mathcal{L}'(q)$ and the fact that the family U is upward-closed. ◀

We now examine the sequences of distributions $\vec{\mu_0}\big(\mathcal{S}^n_i\big)$, $\vec{\mu_0}\big(\mathcal{R}^n_i\big)$, $\vec{\mu_0}\big(\mathcal{S}^n_\infty\big)$, and $\vec{\mu_0}\big(\mathcal{R}^n_\infty\big)$ arising from the Q-families introduced in the previous section. Our aim is to bind them by equations computable within $\mathcal{D}\mathsf{P}(Q)$. As an analogue to the operation Δ, we introduce a function $\mathcal{F}\colon \mathcal{D}\mathsf{P}(Q)\to\mathcal{D}\mathsf{P}(Q)$ defined for $\beta\in\mathcal{D}\mathsf{P}(Q)$ and $P\in\mathsf{P}(Q)$ by

$$\mathcal{F}(\beta)(P) = \frac{1}{|A|}\cdot \sum_{(P_{\mathtt{L}},a,P_{\mathtt{R}})\in\Delta^{-1}(P)} \beta(P_{\mathtt{L}})\cdot\beta(P_{\mathtt{R}}). \tag{3}$$

Note that the formula guarantees that $\mathcal{F}(\beta)$ is indeed a probabilistic distribution in $\mathcal{D}\mathsf{P}(Q)$. The operator $\mathcal{F}$ allows us to lift the inductive definitions of the Q-families $\mathcal{S}^n_{i+1}$ and $\mathcal{R}^n_{i+1}$ given by Lemma 12, to their counterparts in the level of probability distributions.

▶ **Lemma 15.** *For each $n\in\mathbb{N}$ and $i\in\mathbb{N}$ we have*

$$\vec{\mu_0}\big(\mathcal{S}^n_{i+1}\big) = \mathcal{F}\Big(\vec{\mu_0}\big(\mathcal{S}^n_i\big)\Big) \text{ and } \vec{\mu_0}\big(\mathcal{R}^n_{i+1}\big) = \mathcal{F}\Big(\vec{\mu_0}\big(\mathcal{R}^n_i\big)\Big).$$

Proof. Take $P\in\mathsf{P}(Q)$ and observe that

$$\begin{aligned}
\mathcal{F}\Big(\vec{\mu_0}\big(\mathcal{S}^n_i\big)\Big)(P) &\overset{(1)}{=} \frac{1}{|A|}\cdot \sum_{(P_{\mathtt{L}},a,P_{\mathtt{R}})\in\Delta^{-1}(P)} \vec{\mu_0}\big(\mathcal{S}^n_i\big)(P_{\mathtt{L}})\cdot\vec{\mu_0}\big(\mathcal{S}^n_i\big)(P_{\mathtt{R}})\\
&\overset{(2)}{=} \sum_{(P_{\mathtt{L}},a,P_{\mathtt{R}})\in\Delta^{-1}(P)} \mu_0\big\{t_{\mathtt{L}} \mid \mathcal{S}^n_i[t_{\mathtt{L}}]{=}P_{\mathtt{L}}\big\}\cdot\frac{1}{|A|}\cdot\mu_0\big\{t_{\mathtt{R}} \mid \mathcal{S}^n_i[t_{\mathtt{R}}]{=}P_{\mathtt{R}}\big\}\\
&\overset{(3)}{=} \sum_{(P_{\mathtt{L}},a,P_{\mathtt{R}})\in\Delta^{-1}(P)} \mu_0\big\{t \mid \mathcal{S}^n_i[t{\restriction}_{\mathtt{L}}]{=}P_{\mathtt{L}}\wedge t(\varepsilon){=}a\wedge\mathcal{S}^n_i[t{\restriction}_{\mathtt{R}}]{=}P_{\mathtt{R}}\big\}\\
&\overset{(4)}{=} \mu_0\left(\bigcup_{(P_{\mathtt{L}},a,P_{\mathtt{R}})\in\Delta^{-1}(P)} \big\{t \mid \mathcal{S}^n_i[t{\restriction}_{\mathtt{L}}]{=}P_{\mathtt{L}}\wedge t(\varepsilon){=}a\wedge\mathcal{S}^n_i[t{\restriction}_{\mathtt{R}}]{=}P_{\mathtt{R}}\big\}\right)\\
&\overset{(5)}{=} \mu_0\Big\{t\in\mathrm{Tr}_A \mid \Delta\big(\mathcal{S}^n_i[t{\restriction}_{\mathtt{L}}],t(\varepsilon),\mathcal{S}^n_i[t{\restriction}_{\mathtt{R}}]\big){=}P\Big\}\\
&\overset{(6)}{=} \mu_0\Big\{t\in\mathrm{Tr}_A \mid \mathcal{S}^n_{i+1}[t]{=}P\Big\} \overset{(7)}{=} \vec{\mu_0}\big(\mathcal{S}^n_{i+1}\big)(P),
\end{aligned}$$

where: (1) is just the definition of $\mathcal{F}\Big(\vec{\mu_0}\big(\mathcal{S}^n_i\big)\Big)$; (2) follows from the definition of $\vec{\mu_0}\big(\mathcal{S}^n_i\big)$; (3) follows from Remark 3 and the independence of $\{t(\varepsilon)=a\}$ from the other events in consideration; (4) follows from the fact that the measured sets are pairwise disjoint; (5) follows simply from the definition of Δ; (6) follows from Lemma 12; and (7) is just the definition of $\vec{\mu_0}\big(\mathcal{S}^n_{i+1}\big)$.

The proof for $\mathcal{R}^n_{i+1}$ follows the same steps, except it uses the $\mathcal{R}^n_i$ variant of Lemma 12. ◀

Now, recall that $Q_{\geq n}$ and $Q_{<n}$ are sets of states of respective priorities. Let the functions $\mathcal{Q}_{<n}, \mathcal{Q}_{\geq n}\colon \mathcal{D}\mathsf{P}(Q) \to \mathcal{D}\mathsf{P}(Q)$ be defined by

$$\mathcal{Q}_{<n}(\beta)(P) \overset{\text{def}}{=} \textstyle\sum_{P'\colon P'\cap Q_{<n}=P} \beta(P'), \tag{4}$$

$$\mathcal{Q}_{\geq n}(\beta)(P) \overset{\text{def}}{=} \textstyle\sum_{P'\colon P'\cup Q_{\geq n}=P} \beta(P'). \tag{5}$$

Again, the formulae guarantee that $\mathcal{Q}_{<n}(\beta)$ and $\mathcal{Q}_{\geq n}(\beta)$ are both probabilistic distributions in $\mathcal{D}\mathsf{P}(Q)$. The following lemma shows the relation between these functions and the limit distributions $\vec{\mu_0}\big(\mathcal{S}^{n-1}_\infty\big)$ and $\vec{\mu_0}\big(\mathcal{R}^{n-1}_\infty\big)$.

▶ **Lemma 16.** *For each $n \in \mathbb{N}$ we have*

$$\mathcal{Q}_{<n}\Big(\vec{\mu_0}\big(\mathcal{S}^{n-1}_\infty\big)\Big) = \vec{\mu_0}\big(\mathcal{R}^{n}_0\big) \qquad \textit{if } n \textit{ is odd,}$$

$$\mathcal{Q}_{\geq n}\Big(\vec{\mu_0}\big(\mathcal{R}^{n-1}_\infty\big)\Big) = \vec{\mu_0}\big(\mathcal{S}^{n}_0\big) \qquad \textit{if } n \textit{ is even.}$$

This lemma follows from Lemma 10 in a similar way as Lemma 15 follows from Lemma 12.

Proof. Consider the case of even n and a tree $t \in \mathrm{Tr}_A$. We need to show that

$$\mathcal{Q}_{\geq n}\Big(\vec{\mu_0}\big(\mathcal{R}^{n-1}_\infty\big)\Big) = \vec{\mu_0}\big(\mathcal{S}^{n}_0\big).$$

Lemma 10 implies that

$$\mathcal{S}^n_0[t] = \big(\mathcal{R}^{n-1}_\infty[t]\big) \cup Q_{\geq n}. \tag{6}$$

Therefore, for each $P \in \mathsf{P}(Q)$ we have

$$\begin{aligned}
\vec{\mu_0}\big(\mathcal{S}^n_0\big)(P) &= \mu_0\big\{t \in \mathrm{Tr}_A \mid \mathcal{S}^n_0[t] = P\big\}\\
&= \mu_0\big\{t \in \mathrm{Tr}_A \mid \big(\mathcal{R}^{n-1}_\infty[t]\big) \cup Q_{\geq n} = P\big\}\\
&= \mu_0\left(\bigcup_{P'\colon P'\cup Q_{\geq n}=P} \big\{t \in \mathrm{Tr}_A \mid \mathcal{R}^{n-1}_\infty[t] = P'\big\}\right)\\
&= \sum_{P'\colon P'\cup Q_{\geq n}=P} \mu_0\big\{t \in \mathrm{Tr}_A \mid \mathcal{R}^{n-1}_\infty[t] = P'\big\}\\
&= \sum_{P'\colon P'\cup Q_{\geq n}=P} \vec{\mu_0}\big(\mathcal{R}^{n-1}_\infty\big)(P')\\
&= \mathcal{Q}_{\geq n}\big(\vec{\mu_0}\big(\mathcal{R}^{n-1}_\infty\big)\big)(P)
\end{aligned}$$

The case of odd n is entirely analogous. ◀

The two above lemmata express the properties of the operators $\mathcal{F}$, $\mathcal{Q}_{<n}$, and $\mathcal{Q}_{\geq n}$ as depicted on Figure 1.

6 Limit distributions $\vec{\mu_0}\big(\mathcal{S}^n_\infty\big)$ and $\vec{\mu_0}\big(\mathcal{R}^n_\infty\big)$

In this section we show how to represent the distributions $\vec{\mu_0}\big(\mathcal{S}^n_\infty\big)$ and $\vec{\mu_0}\big(\mathcal{R}^n_\infty\big)$ as fixed points. We begin by proving that these distributions are limits in $\mathbb{R}^{\mathsf{P}(Q)}$ of the sequences of vectors $\big(\vec{\mu_0}\big(\mathcal{S}^n_i\big)\big)_{i\in\mathbb{N}}$ and $\big(\vec{\mu_0}\big(\mathcal{R}^n_i\big)\big)_{i\in\mathbb{N}}$ respectively. This is a consequence of Lemmata 7 and 9.

▶ **Lemma 17.** *For each $n \in \mathbb{N}$ and $P \in \mathsf{P}(Q)$ we have*

$$\lim_{i\to\infty} \vec{\mu_0}\big(\mathcal{S}^n_i\big)(P) = \vec{\mu_0}\big(\mathcal{S}^n_\infty\big)(P) \text{ and } \lim_{i\to\infty} \vec{\mu_0}\big(\mathcal{R}^n_i\big)(P) = \vec{\mu_0}\big(\mathcal{R}^n_\infty\big)(P).$$

Proof. We consider case of $\vec{\mu_0}\big(\mathcal{S}^n_\infty\big)$, the case of $\vec{\mu_0}\big(\mathcal{R}^n_\infty\big)(P)$ is entirely dual. First, we show that the respective limits agree when taking sums over any upward closed family $U \subseteq \mathsf{P}(Q)$, see (7) below. For $i \in \mathbb{N}$ let $X_i = \bigcup_{P'\in U}\{t \in \mathrm{Tr}_A \mid \mathcal{S}^n_i[t] = P'\}$ and $X_\infty = \bigcup_{P'\in U}\{t \in \mathrm{Tr}_A \mid \mathcal{S}^n_\infty[t] = P'\}$. Lemma 7 together with the fact that U is upward-closed imply that $X_0 \supseteq X_1 \supseteq \ldots \supseteq X_\infty$. Lemma 9 and finiteness of Q imply that for every tree t there exists an index J such that $\mathcal{S}^n_J[t] \subseteq \mathcal{S}^n_\infty[t]$. Therefore, $\bigcap_{i\in\mathbb{N}} X_i = X_\infty$. By continuity of the measure μ_0 we get that $\lim_{i\to\infty}\mu_0(X_i) = \mu_0(X_\infty)$. This means that

$$\lim_{i\to\infty} \sum_{P'\in U} \vec{\mu_0}\big(\mathcal{S}^n_i\big)(P') = \lim_{i\to\infty} \mu_0(X_i) = \mu_0(X_\infty) = \sum_{P'\in U} \vec{\mu_0}\big(\mathcal{S}^n_\infty\big)(P'). \tag{7}$$

Clearly, $\{P\} = \{P' \in \mathsf{P}(Q) \mid P' \supseteq P\} \setminus \{P' \in \mathsf{P}(Q) \mid P' \supsetneq P\}$ with both these families upward closed. Therefore, we can apply (7) twice and obtain the desired equation. ◀

The monotonicity of Δ, see Lemma 11, implies the following lemma.

▶ **Lemma 18.** *The operator $\mathcal{F}\colon \mathcal{D}\mathsf{P}(Q) \to \mathcal{D}\mathsf{P}(Q)$, see Equation (3), is monotone in $\preceq$.*

Proof. We need to prove that $\mathcal{F}$ is monotone w.r.t. the order $\preceq$. Thus, for every $\alpha \preceq \beta \in \mathcal{D}\mathsf{P}(Q)$ and an upward-closed family $U \subseteq \mathsf{P}(Q)$ we should have $\sum_{P\in U}\mathcal{F}(\alpha)(P) \leq \sum_{P\in U}\mathcal{F}(\beta)(P)$.

After multiplying by $\frac{1}{|A|}$ and splitting the sum over separate letters $a \in A$ (see the definition of $\mathcal{F}$, cf. (3)), it is enough to show that for each $a \in A$ and $O_a \stackrel{\text{def}}{=} \{(P_{\mathtt{L}}, P_{\mathtt{R}}) \mid \Delta(P_{\mathtt{L}}, a, P_{\mathtt{R}}) \in U\}$ we have

$$\sum_{(P_{\mathtt{L}},P_{\mathtt{R}})\in O_a} \alpha(P_{\mathtt{L}})\cdot\alpha(P_{\mathtt{R}}) \leq \sum_{(P_{\mathtt{L}},P_{\mathtt{R}})\in O_a} \beta(P_{\mathtt{L}})\cdot\beta(P_{\mathtt{R}}).$$

Now, by monotonicity of Δ (see Lemma 11) and the fact that U is upward-closed, we know that if $P_{\mathtt{L}} \subseteq P'_{\mathtt{L}}$, $P_{\mathtt{R}} \subseteq P'_{\mathtt{R}}$, and $(P_{\mathtt{L}}, P_{\mathtt{R}}) \in O_a$ then also $(P'_{\mathtt{L}}, P'_{\mathtt{R}}) \in O_a$. By $P_{\mathtt{L}}^{-1}\cdot O_a$ and $O_a \cdot P_{\mathtt{R}}^{-1}$ we will denote the sections $\{P_{\mathtt{R}} \mid (P_{\mathtt{L}}, P_{\mathtt{R}}) \in O_a\}$ and $\{P_{\mathtt{L}} \mid (P_{\mathtt{L}}, P_{\mathtt{R}}) \in O_a\}$ respectively. Notice that both of them are upward-closed. Thus, using the assumption that $\alpha \preceq \beta$ twice, we obtain

$$\begin{aligned}
\sum_{(P_{\mathtt{L}},P_{\mathtt{R}})\in O_a} \alpha(P_{\mathtt{L}})\cdot\alpha(P_{\mathtt{R}}) &= \sum_{P_{\mathtt{L}}\in\mathsf{P}(Q)} \alpha(P_{\mathtt{L}})\cdot\left(\sum_{P_{\mathtt{R}}\in P_{\mathtt{L}}^{-1}\cdot O_a} \alpha(P_{\mathtt{R}})\right)\\
&\leq \sum_{P_{\mathtt{L}}\in\mathsf{P}(Q)} \alpha(P_{\mathtt{L}})\cdot\left(\sum_{P_{\mathtt{R}}\in P_{\mathtt{L}}^{-1}\cdot O_a} \beta(P_{\mathtt{R}})\right)\\
&= \sum_{(P_{\mathtt{L}},P_{\mathtt{R}})\in O_a} \alpha(P_{\mathtt{L}})\cdot\beta(P_{\mathtt{R}}) = \sum_{(P_{\mathtt{L}},P_{\mathtt{R}})\in O_a} \beta(P_{\mathtt{R}})\cdot\alpha(P_{\mathtt{L}})\\
&= \sum_{P_{\mathtt{R}}\in\mathsf{P}(Q)} \beta(P_{\mathtt{R}})\cdot\left(\sum_{P_{\mathtt{L}}\in O_a\cdot P_{\mathtt{R}}^{-1}} \alpha(P_{\mathtt{L}})\right)\\
&\leq \sum_{P_{\mathtt{R}}\in\mathsf{P}(Q)} \beta(P_{\mathtt{R}})\cdot\left(\sum_{P_{\mathtt{L}}\in O_a\cdot P_{\mathtt{R}}^{-1}} \beta(P_{\mathtt{L}})\right)\\
&= \sum_{(P_{\mathtt{L}},P_{\mathtt{R}})\in O_a} \beta(P_{\mathtt{R}})\cdot\beta(P_{\mathtt{L}}) = \sum_{(P_{\mathtt{L}},P_{\mathtt{R}})\in O_a} \beta(P_{\mathtt{L}})\cdot\beta(P_{\mathtt{R}}).
\end{aligned}$$

◀

With the two lemmata above, we are ready to conclude this section: we characterise the distributions $\vec{\mu_0}\big(\mathcal{S}^n_\infty\big)$ and $\vec{\mu_0}\big(\mathcal{R}^n_\infty\big)$, see Figure 1, by a specialised variant of the Knaster-Tarski fixed point theorem.

▶ **Proposition 19.** *For each $n\in\mathbb{N}$ the distribution $\vec{\mu_0}\big(\mathcal{S}^n_\infty\big)$ is the $\preceq$-greatest distribution β satisfying $\beta=\mathcal{F}(\beta)$ and $\beta\preceq\vec{\mu_0}\big(\mathcal{S}^n_0\big)$. Similarly, $\vec{\mu_0}\big(\mathcal{R}^n_\infty\big)$ is the $\preceq$-least distribution β satisfying $\beta=\mathcal{F}(\beta)$ and $\beta\succeq\vec{\mu_0}\big(\mathcal{R}^n_0\big)$.*

Proof. Consider the case of $\vec{\mu_0}\big(\mathcal{S}^n_\infty\big)$. Observe that $\mathcal{F}$ is continuous in $\mathbb{R}^{\mathsf{P}(Q)}$. Indeed, it is given by a vector of quadratic polynomials from $\mathbb{R}^{\mathsf{P}(Q)}$ to $\mathbb{R}^{\mathsf{P}(Q)}$. Now, take $P\in\mathsf{P}(Q)$ and observe that

$$\begin{aligned}
\vec{\mu_0}\big(\mathcal{S}^n_\infty\big)(P) = \lim_{i\to\infty}\vec{\mu_0}\big(\mathcal{S}^n_i\big)(P) = \lim_{i\to\infty}\mathcal{F}\big(\vec{\mu_0}\big(\mathcal{S}^n_i\big)\big)(P) =\\
\mathcal{F}\Big(\lim_{i\to\infty}\vec{\mu_0}\big(\mathcal{S}^n_i\big)(P)\Big) = \mathcal{F}\Big(\vec{\mu_0}\big(\mathcal{S}^n_\infty\big)(P)\Big)
\end{aligned}$$

where the first equality follows from Lemma 17; the second from Lemma 15; the third from continuity of $\mathcal{F}$; and the last from Lemma 17, again. Therefore, $\beta=\vec{\mu_0}\big(\mathcal{S}^n_\infty\big)$ satisfies $\beta=\mathcal{F}(\beta)$. Moreover, Lemmata 7 and 14 imply that $\beta\preceq\vec{\mu_0}\big(\mathcal{S}^n_0\big)$.

Consider now any distribution $\beta\in\mathcal{D}\mathsf{P}(Q)$ such that $\beta=\mathcal{F}(\beta)$ and $\beta\preceq\vec{\mu_0}\big(\mathcal{S}^n_0\big)$. We need to prove that $\beta\preceq\vec{\mu_0}\big(\mathcal{S}^n_\infty\big)$. Lemma 18 states that $\mathcal{F}$ is monotone. Therefore, by inductively applying Lemma 15 for $i=0,\ldots$, we infer that

$$\beta=\mathcal{F}(\beta)\leq\mathcal{F}\Big(\vec{\mu_0}\big(\mathcal{S}^n_i\big)\Big)=\vec{\mu_0}\big(\mathcal{S}^n_{i+1}\big).$$

Take any upward-closed family $U\subseteq\mathsf{P}(Q)$. The above inequality implies that for each $i\in\mathbb{N}$ we have $\sum_{P\in U}\beta(P)\leq\sum_{P\in U}\vec{\mu_0}\big(\mathcal{S}^n_i\big)(P)$. By taking the limit as in Lemma 17 we obtain that $\sum_{P\in U}\beta(P)\leq\sum_{P\in U}\vec{\mu_0}\big(\mathcal{S}^n_\infty\big)(P)$. This implies that $\beta\preceq\vec{\mu_0}\big(\mathcal{S}^n_\infty\big)$.

The case of $\vec{\mu_0}\big(\mathcal{R}^n_\infty\big)$ is similar, we utilise the opposite monotonicity $\beta\succeq\vec{\mu_0}\big(\mathcal{R}^n_{i+1}\big)$. ◀

7 Computing measures

In this section, we conclude our solution to Problem 1 for weak alternating automata. This is achieved by a reduction to the first-order theory of the real numbers $\mathcal{R} = \langle \mathbb{R}, 0, 1, +, \cdot \rangle$. The theory is famously decidable thanks to Tarski-Seidenberg theorem, see e.g. [25].

Throughout this section, we assume that the reader is familiar with the syntax and semantics of first-order logic. We say that a formula $\varphi(x_1, \ldots, x_k)$ *represents* a relation $r \subseteq \mathbb{R}^k$ if it holds in $\mathbb{R}$ according to an evaluation v of the free variables $x_1, \ldots, x_n$, precisely when the tuple $\langle v(x_1), \ldots, v(x_k) \rangle$ belongs to r. For example, the formula $\exists z.\ x + (z \cdot z) = y$ represents the standard ordering $\leq$ on real numbers. A formula represents a number $a \in \mathbb{R}$ if it represents the singleton $\{a\}$; for example the formula $x \cdot x = 1{+}1 \wedge \exists z.\ x = z \cdot z$, represents the number $\sqrt{2}$.

▶ **Theorem 20.** *Given a weak alternating automaton $\mathcal{A}$ one can compute a formula $\psi_{\mathcal{A}}(x)$ that represents the number $\mu_0\big(\mathrm{L}(\mathcal{A})\big)$. Moreover, the formula is in a prenex normal form, its size is exponential in the size of $\mathcal{A}$, and the quantifier alternation of $\psi_{\mathcal{A}}(x)$ is constant.*

Proof. Fix a weak alternating automaton $\mathcal{A} = \langle A, Q, q_{\mathrm{I}}, \delta, \Omega \rangle$. Let $N > \Omega(q_{\mathrm{I}})$ be an even number (either $\Omega(q_{\mathrm{I}}){+}1$ or $\Omega(q_{\mathrm{I}}){+}2$). Fix an enumeration $\{P_1, \ldots, P_K\}$ of $\mathsf{P}(Q)$ with $K = 2^{|Q|}$. We will identify a distribution $\alpha \in \mathcal{D}\mathsf{P}(Q)$ with its representation $\alpha = (a_1, \ldots, a_K) \in \mathbb{R}^K$ as a vector of real numbers. Following this identification, $\alpha(P_k)$ stands for a_k. Clearly the properties that $\mathcal{F}(\alpha) = \beta$, $\mathcal{Q}_{<n}(\alpha) = \beta$, and $\mathcal{Q}_{\geq n}(\alpha) = \beta$ are definable by quantifier free formulae of size polynomial in K.

The following formula defines the fact that $\alpha \in \mathcal{D}\mathsf{P}(Q)$.

$$\mathrm{dist}(\alpha) \equiv \sum_{k=1}^{K} \alpha(P_k) = 1 \ \wedge \ \bigwedge_{k=1}^{K} 0 \leq \alpha(P_k) \leq 1$$

Analogously to our representation of distributions, every subset $U \subseteq \mathsf{P}(Q)$ can be represented by its indicator: a vector of numbers $\iota = (i_1, \ldots, i_K)$ such that $\iota(P)$ is either 0 (if $P \notin U$) or 1 (if $P \in U$). Note that if U is upward closed then whenever $P \subseteq P'$ and $\iota(P) = 1$ then $\iota(P') = 1$. The following formula defines the fact that ι represents an upward-closed set.

$$\mathrm{upward}(\iota) \equiv \bigwedge_{k=1}^{K} \big(\iota(P_k){=}0 \vee \iota(P_k){=}1\big) \wedge \bigwedge_{P \subseteq P'} \iota(P){=}1 \to \iota(P'){=}1.$$

Thus, to check if $\alpha \preceq \beta$ one can use the following formula (see Claim 21 below)

$$\mathrm{minor}(\alpha, \beta, \iota) \equiv \sum_{k=1}^{K} \alpha(P_k) \cdot \iota(P_k) \leq \sum_{k=1}^{K} \beta(P_k) \cdot \iota(P_k).$$

Notice that all the above formulae: $\mathrm{dist}(\alpha)$, $\mathrm{upward}(\iota)$, and $\mathrm{minor}(\alpha, \beta, \iota)$ are quantifier free: the $\bigwedge$ there are just explicitly written as finite conjunctions. Therefore, these formulae can be used to relativise quantifiers in a prenex normal form of a formula: for instance we write $\forall \alpha \colon \mathrm{dist}(\alpha).\ \exists \beta \colon \mathrm{dist}(\beta).\ \psi(\alpha, \beta)$ to denote $\forall \alpha.\ \exists \beta.\ \mathrm{dist}(\alpha) \to \big(\mathrm{dist}(\beta) \wedge \psi(\alpha, \beta)\big)$.

▷ Claim 21. Given two distributions α and β, we have $\alpha \preceq \beta$ if and only if

$$\forall \iota \colon \mathrm{upward}(\iota).\ \mathrm{minor}(\alpha, \beta, \iota).$$

$$\begin{tikzcd}
\alpha_0 = \vec{\mu}_0(\mathcal{S}_0^0) \arrow[dd, "\heartsuit"'] & & \beta_1 = \vec{\mu}_0(\mathcal{R}_\infty^0) \arrow[r, "\mathcal{Q}_{\geq 2}"'] & \alpha_2 = \vec{\mu}_0(\mathcal{S}_0^0) \arrow[dd, "\heartsuit"'] & & \beta_3 = \vec{\mu}_0(\mathcal{R}_\infty^0) \arrow[r, "\mathcal{Q}_{\geq 4}"'] & \cdots \\
 & & & & & & \cdots \\
\beta_0 = \vec{\mu}_0(\mathcal{S}_\infty^0) \arrow[r, "\mathcal{Q}_{<1}"] & \alpha_1 = \vec{\mu}_0(\mathcal{R}_0^0) \arrow[uur, "\spadesuit"] & & \beta_2 = \vec{\mu}_0(\mathcal{S}_\infty^0) \arrow[r, "\mathcal{Q}_{<3}"] & \alpha_3 = \vec{\mu}_0(\mathcal{R}_0^0) \arrow[uur, "\spadesuit"] & & \cdots
\end{tikzcd}$$

Figure 2 A diagram of the distributions α_n and β_n in the formula $\psi_{\mathcal{A}}(x)$, cf. Figure 1. The symbol $\heartsuit$ represents applications of Proposition 19 in the case of $\mathcal{S}_0^n$ and $\mathcal{S}_\infty^n$; while $\spadesuit$ corresponds to the dual case of $\mathcal{R}_0^n$ and $\mathcal{R}_\infty^n$.

The formula $\psi_{\mathcal{A}}(x)$ is indented to specify the distributions $(\alpha_n, \beta_n)_{n=0,\ldots,N}$ in a way depicted on Figure 2. The value $\vec{\mu}_0\big(\mathcal{S}_0^0(P)\big)$ is 1 if $P = Q$ and 0 otherwise, see Lemma 10. Proposition 19 allows us to define $\vec{\mu}_0\big(\mathcal{S}_\infty^n\big)$ (resp. $\vec{\mu}_0(\mathcal{R}_\infty^n)$) using $\vec{\mu}_0\big(\mathcal{S}_0^n\big)$ (resp. $\vec{\mu}_0\big(\mathcal{R}_0^n\big)$) as specific fixed points of the operation $\mathcal{F}$. Finally, Lemma 16 allows us to define $\vec{\mu}_0\big(\mathcal{R}_0^n\big)$ using $\mathcal{Q}_{<n}$, and $\vec{\mu}_0\big(\mathcal{S}_0^n\big)$ using $\mathcal{Q}_{\geq n}$. The value of x is related to those distributions based on Lemma 10 which implies that $\mu_0\big(\mathrm{L}(\mathcal{A})\big) = \sum_{P\colon q_{\mathrm{I}}\in P\in \mathsf{P}(Q)} \vec{\mu}_0\big(\mathcal{S}_0^N\big)(P)$.

The following equation defines the formula $\psi_{\mathcal{A}}(x)$.

$$\psi_{\mathcal{A}}(x) \equiv \exists \alpha_0, \beta_0\colon \mathrm{dist}(\alpha_0), \mathrm{dist}(\beta_0), \beta_0{=}\mathcal{F}(\beta_0).$$

$$\vdots \tag{8}$$

$$\exists \alpha_N, \beta_N\colon \mathrm{dist}(\alpha_N), \mathrm{dist}(\beta_N), \beta_N{=}\mathcal{F}(\beta_N).$$

$$\forall \theta\colon \mathrm{dist}(\theta), \theta{=}\mathcal{F}(\theta). \tag{9}$$

$$\exists \iota_0\colon \mathrm{upward}(\iota_0).$$

$$\vdots \tag{10}$$

$$\exists \iota_N\colon \mathrm{upward}(\iota_N).$$

$$\forall \gamma_0\colon \mathrm{upward}(\gamma_0).$$

$$\vdots \tag{11}$$

$$\forall \gamma_N\colon \mathrm{upward}(\gamma_N).$$

$$\left(\alpha_0(Q) = 1 \wedge \bigwedge_{P\neq Q} \alpha_0(P) = 0\right) \wedge \tag{12}$$

$$\left(\bigwedge_{n=1}^{N} [n \text{ is odd}] \rightarrow \alpha_n = \mathcal{Q}_{<n}(\beta_{n-1})\right) \wedge \tag{13}$$

$$\left(\bigwedge_{n=1}^{N} [n \text{ is even}] \rightarrow \alpha_n = \mathcal{Q}_{\geq n}(\beta_{n-1})\right) \wedge \tag{14}$$

$$\left(\bigwedge_{n=0}^{N} [n \text{ is odd}] \rightarrow \mathrm{minor}(\alpha_n, \beta_n, \gamma_n)\right) \wedge \tag{15}$$

$$\left(\bigwedge_{n=0}^{N} [n \text{ is even}] \rightarrow \mathrm{minor}(\beta_n, \alpha_n, \gamma_n)\right) \wedge \tag{16}$$

$$\left(\bigwedge_{n=0}^{N} [n \text{ is odd}] \rightarrow \Big(\neg\mathrm{minor}(\alpha_n, \theta, \iota_n) \vee \mathrm{minor}(\beta_n, \theta, \gamma_n)\Big)\right) \wedge \tag{17}$$

$$\left(\bigwedge_{n=0}^{N}[n \text{ is even}] \rightarrow \Big(\neg\mathrm{minor}(\theta,\alpha_n,\iota_n) \vee \mathrm{minor}(\theta,\beta_n,\gamma_n)\Big)\right) \quad \wedge \tag{18}$$

$$\left(\sum_{P\ni q_{\mathrm{I}}} \alpha_N(P) = x\right) \tag{19}$$

Observe that the size of this formula is polynomial in K and N (in fact it is $\mathcal{O}(N\cdot K^2)$), i.e. exponential in the size of the automaton $\mathcal{A}$. Moreover, the formula is in prenex normal form and its quantifier alternation is 4 (the sub-formulae that involve $\bigwedge$ are written explicitly as conjunctions).

We begin by proving soundness of the formula: we assume that $\psi_{\mathcal{A}}(x)$ holds and show that $x = \mu_0\big(\mathrm{L}(\mathcal{A})\big)$. Consider a sequence of distributions $(\alpha_n,\beta_n)_{n=0,\ldots,N}$ witnessing (8). The following two lemmata prove inductively that for $n = 0,\ldots,N$ we have

$$\begin{aligned} &\alpha_n = \vec{\mu_0}(\mathcal{S}_0^n) \text{ and } \beta_n = \vec{\mu_0}(\mathcal{S}_\infty^n) && \text{for even } n,\\ &\alpha_n = \vec{\mu_0}(\mathcal{R}_0^n) \text{ and } \beta_n = \vec{\mu_0}(\mathcal{R}_\infty^n) && \text{for odd } n. \end{aligned} \tag{20}$$

▶ **Lemma 22.** *Using the above notations and the assumption that $\psi_{\mathcal{A}}(x)$ holds:*

for even n, if $\alpha_n = \vec{\mu_0}(\mathcal{S}_0^n)$ then $\beta_n = \vec{\mu_0}(\mathcal{S}_\infty^n)$,
for odd n, if $\alpha_n = \vec{\mu_0}(\mathcal{R}_0^n)$ then $\beta_n = \vec{\mu_0}(\mathcal{R}_\infty^n)$.

Proof. Both claims follow from Proposition 19. Take n odd and assume that $\alpha_n = \vec{\mu_0}(\mathcal{R}_0^n)$. We know that $\beta_n = \mathcal{F}(\beta_n)$ by (8). Moreover, by Claim 21, the arbitrary choice of γ_n, and (15) we know that $\alpha_n \preceq \beta_n$. It is enough to prove that if θ is any distribution satisfying $\alpha_n \preceq \theta$ and $\theta = \mathcal{F}(\theta)$ then $\beta_n \preceq \theta$.

Assume contrarily that θ is a distribution such that $\alpha_n \preceq \theta$ and $\theta = \mathcal{F}(\theta)$ but $\beta_n \not\preceq \theta$. We know that θ must satisfy the sub-formula in (9). Take the upward closed sets $(\iota_\ell)_{\ell=0,\ldots,N}$ given by (10). Now let $(\gamma_\ell)_{\ell=0,\ldots,N}$ be any sequence of upward closed sets such that γ_n witnesses the fact that $\beta_n \not\preceq \theta$, i.e. $\neg\mathrm{minor}(\beta_n,\theta,\gamma_n)$ holds. But this is a contradiction with (17) because $\mathrm{minor}(\alpha_n,\theta,\iota_n)$ is true as $\alpha_n \preceq \theta$ and $\mathrm{minor}(\beta_n,\theta,\gamma_n)$ is false.

The case of even n is analogous. ◀

▶ **Lemma 23.** *Using the above notations and the assumption that $\psi_{\mathcal{A}}(x)$ holds:*

$\alpha_n = \vec{\mu_0}(\mathcal{S}_0^n)$ for even n and $\alpha_n = \vec{\mu_0}(\mathcal{R}_0^n)$ for odd n.

Proof. The proof is inductive in n. First, $\alpha_0 = \vec{\mu_0}(\mathcal{S}_0^n)$ because of (12) and the statement for $n = 0$ in Lemma 10 (we can take $\theta = \beta_0$ and $\gamma_\ell = \iota_\ell$ for $\ell = 0,\ldots,N$ to check that Condition (12) holds).

Now assume that the above conditions are true for $n-1$ for some $n \in \{1,\ldots,N\}$. Again, by the symmetry we assume that n is odd, i.e. $\alpha_{n-1} = \vec{\mu_0}(\mathcal{S}_0^{n-1})$. By Lemma 22 we know that $\beta_{n-1} = \vec{\mu_0}(\mathcal{S}_\infty^{n-1})$. Condition (13) says that $\alpha_n = \mathcal{Q}_{<n}(\beta_{n-1}) = \mathcal{Q}_{<n}\big(\vec{\mu_0}(\mathcal{S}_\infty^{n-1})\big)$. Now Lemma 16 implies that $\mathcal{Q}_{<n}\big(\vec{\mu_0}(\mathcal{S}_\infty^{n-1})\big) = \vec{\mu_0}\big(\mathcal{R}_0^n\big)$ and the induction step is complete. ◀

Equation (20) together with Condition (19), imply that $x = \mu_0\{t \in \mathrm{Tr}_A \mid q_{\mathrm{I}} \in \mathcal{S}_0^N[t]\}$. Since $N > \Omega(q_{\mathrm{I}})$ is even, Lemma 10 implies that $\mathcal{S}_0^N(q_{\mathrm{I}}) = \mathrm{L}(\mathcal{A}, q_{\mathrm{I}})$ and therefore, $q_{\mathrm{I}} \in \mathcal{S}_0^N[t]$ if and only if $t \in \mathrm{L}(\mathcal{A})$. This guarantees that $x = \mu_0\big(\mathrm{L}(\mathcal{A})\big)$.

We will now prove completeness of the formula: if $x = \mu_0\big(\mathrm{L}(\mathcal{A})\big)$ then $\psi_{\mathcal{A}}(x)$ holds. Choose the distributions $(\alpha_n,\beta_n)_{n=0,\ldots,N}$ in (8) as in (20). We will show that then the rest of the formula holds. Consider any distribution θ. For each $n = 0,\ldots,N$ let ι_n be an upward-closed set witnessing that $\alpha_n \not\preceq \theta$ for n odd (resp. $\theta \not\preceq \alpha_n$ for n even); or any upward closed set if the respective inequality holds.

Take any $(\gamma_n)_{n=0,\ldots,N}$ that are upward closed. We need to check that the sub-formula starting in (12) holds. Conditions (12) – (16) and (19) hold by the same lemmata as mentioned in the previous section. To check Conditions (17) and (18) one again invokes Proposition 19: either ι_n witnesses that $\alpha_n \not\preceq \theta$ (resp. $\theta \not\preceq \alpha_n$) or, if ι_n was chosen arbitrarily, then Proposition 19 implies that also the respective inequality with β_n holds. ◀

7.1 Branching processes

This section is devoted to the variant of Theorem 20 in the case when instead of the standard measure μ_0 one uses a measure $\mu_{\mathcal{P}}$ generated by a given branching process $\mathcal{P}$. For the sake of simplicity we define only binary branching processes, the case of a fixed higher arity can be solved analogously.

A *branching process* is a tuple $\mathcal{P} = \langle A, \tau, \alpha_{\mathrm{I}} \rangle$ where A is a finite alphabet; $\tau \colon A \to \mathcal{D}A^2$ a *branching function* that assigns a probability distribution over A^2 to every letter in A; and $\alpha_{\mathrm{I}} \in \mathcal{D}A$ an *initial distribution.* We assume that all probabilities occurring in these distributions are rational. By the *size* of $\mathcal{P}$ we understand the size of its binary representation.

A branching process $\mathcal{P}$ can be seen as a generator of random trees: it defines a complete Borel measure $\mu_{\mathcal{P}}$ over the set of infinite trees in the following way. Let $f\colon \mathrm{dom}(f) \to A$ be a complete finite tree of depth $d \geq 0$ i.e. $\mathrm{dom}(f) = \{u \in \{\mathtt{L},\mathtt{R}\}^* \mid |u| \leq d\} = \{\mathtt{L},\mathtt{R}\}^{<d+1}$. Then the measure $\mu_{\mathcal{P}}$ of the basic set U_f, see Section 2, is defined by

$$\mu_{\mathcal{P}}(U_f) \overset{\text{def}}{=} \alpha_{\mathrm{I}}(f(\varepsilon)) \cdot \prod_{u \in \{\mathtt{L},\mathtt{R}\}^{<d}} \tau(f(u))\big(f(u\mathtt{L}), f(u\mathtt{R})\big). \tag{21}$$

Now, $\mu_{\mathcal{P}}$ can be extended in a standard way to a complete Borel measure on the set of all infinite trees Tr_A. Intuitively, a tree $t \in \mathrm{Tr}_A$ that is chosen according to $\mu_{\mathcal{P}}$ is generated in a top-down fashion: the root label $t(\varepsilon)$ is chosen according to the initial distribution α_{I}; and the labels of the children $u\mathtt{L}$ and $u\mathtt{R}$ of a node u are chosen according to the distribution $\tau(t(u)) \in \mathcal{D}A^2$ defined for the label of their parent u.

Observe that the uniform measure μ_0 over trees Tr_A equals the measure $\mu_{\mathcal{P}_0}$ defined by the branching process $\mathcal{P}_0 = \langle A, \tau_0, \alpha_0 \rangle$, where $\alpha_0(a) = |A|^{-1}$ and $\tau_0(a)(a_{\mathtt{L}}, a_{\mathtt{R}}) = |A|^{-2}$ for each $a, a_{\mathtt{L}}, a_{\mathtt{R}} \in A$.

▶ **Theorem 24.** *Given a weak alternating automaton $\mathcal{A}$ and a branching process $\mathcal{P}$ one can compute a formula $\psi_{\mathcal{A},\mathcal{P}}(x)$ that represents the number $\mu_{\mathcal{P}}\big(\mathrm{L}(\mathcal{A})\big)$. Moreover, the formula is in a prenex normal form; its size is exponential in the size of $\mathcal{A}$ and polynomial in the size of $\mathcal{P}$; and the quantifier alternation of $\psi_{\mathcal{A},\mathcal{P}}$ is constant.*

To prove Theorem 24, we construct the formula $\psi_{\mathcal{A},\mathcal{P}}(x)$ directly along the same lines as the formula $\psi_{\mathcal{A}}(x)$ in Theorem 20. The difference is that instead of working in the space $\mathcal{D}\mathsf{P}(Q)$ with the order $\preceq$, our computations are performed in the space $(\mathcal{D}\mathsf{P}(Q))^A$ ordered by the order $\preceq$ considered coordinate-wise. A complete presentation of this more general construction will be given in the full version of this article.

7.2 Representing algebraic numbers

We now use the formulae $\psi_{\mathcal{A}}$ and $\psi_{\mathcal{A},\mathcal{P}}$ constructed above to find the measure of the language $\mathrm{L}(\mathcal{A})$. We use the celebrated result of Tarski [25] and its two algorithmic improvements.

▶ **Theorem 25** ([2, 3])**.** *Given a formula ψ of first-order logic over $\mathbb{R}$, one can decide if ψ holds in deterministic exponential space. Moreover, if ψ is in a prenex normal form and the alternation of quantifiers $\forall$ and $\exists$ in ψ is bounded then the algorithm works in single exponential time in the size of ψ.*

Proof of Theorem 2. Input a weak alternating automaton $\mathcal{A}$, a branching process $\mathcal{P}$, and a rational number q. Consider the formula $\psi \equiv \exists x.\ \psi_{\mathcal{A},\mathcal{P}}(x) \wedge q \bowtie x$, where $\bowtie$ is one of $<$, $=$, or $>$. Notice that ψ is in prenex normal form; its size is exponential in the size of $\mathcal{A}$ and polynomial in the size of $\mathcal{P}$; and its quantifier alternation is constant. Apply the algorithm from Theorem 25 to check whether ψ is true in $\mathbb{R}$. ◀

We can also compute a representation of the measure $\mu_{\mathcal{P}}\big(\mathrm{L}(\mathcal{A})\big)$. The *quantifier elimination* procedure due to Tarski [25] transforms a formula $\psi(x_1,\ldots,x_n)$ into an equivalent quantifier-free formula $\widehat{\psi}(x_1,\ldots,x_n)$, which moreover can be represented by a *semialgebraic* set, see [4, Chapter 2].

▶ **Theorem 26** ([7]). *Given a formula $\psi(x_1,\ldots,x_n)$ of first-order logic over $\mathbb{R}$, one can construct a representation of the set of tuples $(x_1,\ldots,x_n)$ satisfying ψ, as a semialgebraic set. Moreover, this algorithm works in time doubly-exponential in the size of ψ.*

Theorems 20 and 24 together with the above results imply the following claim.

▶ **Corollary 27.** *Given a weak alternating automaton $\mathcal{A}$ of size n, one can compute a representation of the value $\mu_0\big(\mathrm{L}(\mathcal{A})\big)$ as a singleton semialgebraic set in time triply exponential in n. Moreover, given a branching process of size m, one can compute a representation of the value $\mu_{\mathcal{P}}\big(\mathrm{L}(\mathcal{A})\big)$ as a singleton semialgebraic set in time triply exponential in n and doubly exponential in m.*

8 Conclusions

We have shown how to compute the probability measure of a tree language L recognised by a weak alternating automaton. The crucial trait is *continuity* of certain approximations of the measure of L in a properly chosen order $\preceq$, see Lemma 17. This continuity relies on König's lemma, cf. Lemma 9. In terms of μ-calculus, it stems from both the absence of alternation between least and greatest fixed points in formulae and the boundedness of branching in models (for a study of continuity in μ-calculus see [11]).

Whether our techniques can be extended beyond weak automata – hopefully to all tree automata or, equivalently, full MSO logic, or full μ-calculus – remains open. The question is of interest as, e.g. translation of the logic CTL* into μ-calculus requires at least one alternation between least and greatest fixed points (cf. [9], Exercise 10.13). On the other hand, fixed point formulas over binary trees are not continuous in general, and may require ω_1 iterations to reach stabilisation, already on the second level of the fixed-point hierarchy.

This problem has been already successfully tackled in the context of measurability of regular tree languages – Mio [16] uses Martin's axiom to control the behaviour of measure when taking limits of sequences of length ω_1. Such behaviour cannot be directly simulated in $\mathcal{D}X$, because each well-founded chain of distributions has a countable length. However, this need not be an absolute obstacle as it might be the case that the values of the measure of the iterations stabilise before the actual fixed point is reached, possibly in ω steps.

References

1 André Arnold and Damian Niwiński. Fixed point characterisation of weak monadic logic definable sets of trees. In *Tree Automata and Languages*, pages 159–188, 1992.

2 Saugata Basu, Richard Pollack, and Marie-Françoise Roy. *Algorithms in Real Algebraic Geometry*, volume 10 of *Algorithms and Computation in Mathematics*. Springer-Verlag Berlin Heidelberg, 2006.

3 Michael Ben-Or, Dexter Kozen, and John Reif. The complexity of elementary algebra and geometry. *Journal of Computer and System Sciences*, 32(2):251–264, 1986.

4 Jacek Bochnak, Michel Coste, and Marie-Françoise Roy. *Real Algebraic Geometry*, volume 36 of *A Series of Modern Surveys in Mathematics.* Springer-Verlag Berlin Heidelberg, 1998.

5 Krishnendu Chatterjee, Marcin Jurdzinski, and Thomas A. Henzinger. Quantitative stochastic parity games. In *SODA*, pages 121–130, 2004.

6 Taolue Chen, Klaus Dräger, and Stefan Kiefer. Model checking stochastic branching processes. In *MFCS*, pages 271–282, 2012.

7 George E. Collins. Hauptvortrag: Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Automata Theory and Formal Languages*, pages 134–183, 1975.

8 Costas Courcoubetis and Mihalis Yannakakis. The complexity of probabilistic verification. *J. ACM*, 42(4):857–907, 1995.

9 Stéphane Demri, Valentin Goranko, and Martin Lange. *Temporal Logics in Computer Science: Finite-State Systems.* Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2016.

10 Alessandro Facchini, Filip Murlak, and Michał Skrzypczak. Rabin-Mostowski index problem: A step beyond deterministic automata. In *LICS*, pages 499–508, 2013.

11 Gaëlle Fontaine. Continuous fragment of the mu-calculus. In *CSL*, pages 139–153, 2008.

12 Tomasz Gogacz, Henryk Michalewski, Matteo Mio, and Michał Skrzypczak. Measure properties of regular sets of trees. *Information and Computation*, 256:108–130, 2017.

13 Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002.

14 Claire Jones and Gordon D. Plotkin. A probabilistic powerdomain of evaluations. In *LICS*, pages 186–195, 1989.

15 Henryk Michalewski and Matteo Mio. On the problem of computing the probability of regular sets of trees. In *FSTTCS*, pages 489–502, 2015.

16 Matteo Mio. On the equivalence of game and denotational semantics for the probabilistic mu-calculus. *Logical Methods in Computer Science*, 8(2), 2012.

17 David E. Muller, Ahmed Saoudi, and Paul E. Schupp. Alternating automata. the weak monadic theory of the tree, and its complexity. In *ICALP*, volume 226 of *Lecture Notes in Computer Science*, pages 275–283, 1986.

18 Damian Niwiński and Igor Walukiewicz. A gap property of deterministic tree languages. *Theoretical Computer Science*, 1(303):215–231, 2003.

19 Dominique Perrin and Jean-Éric Pin. *Infinite Words: Automata, Semigroups, Logic and Games.* Elsevier, 2004.

20 Marcin Przybyłko. On computing the measures of first-order definable sets of trees. In *GandALF*, pages 206–219, 2018.

21 Marcin Przybyłko and Michał Skrzypczak. The uniform measure of simple regular sets of infinite trees. *CoRR*, abs/2001.11576, 2020. `arXiv:2001.11576`.

22 Nasser Saheb-Djahromi. Cpo's of measures for nondeterminism. *Theor. Comput. Sci.*, 12:19–37, 1980.

23 Jerzy Skurczyński. The Borel hierarchy is infinite in the class of regular sets of trees. *Theoretical Computer Science*, 112(2):413–418, 1993.

24 Ludwig Staiger. The Hausdorff measure of regular omega-languages is computable. *Bulletin of the EATCS*, 66:178–182, 1998.

25 Alfred Tarski. *A Decision Method for Elementary Algebra and Geometry.* University of California Press, 1951.

Finite Sequentiality of Finitely Ambiguous Max-Plus Tree Automata

Erik Paul
Institute of Computer Science, Leipzig University, Germany
epaul@informatik.uni-leipzig.de

Abstract

We show that the finite sequentiality problem is decidable for finitely ambiguous max-plus tree automata. A max-plus tree automaton is a weighted tree automaton over the max-plus semiring. A max-plus tree automaton is called finitely ambiguous if the number of accepting runs on every tree is bounded by a global constant. The finite sequentiality problem asks whether for a given max-plus tree automaton, there exist finitely many deterministic max-plus tree automata whose pointwise maximum is equivalent to the given automaton.

2012 ACM Subject Classification Theory of computation → Quantitative automata; Theory of computation → Tree languages

Keywords and phrases Weighted Tree Automata, Max-Plus Tree Automata, Finite Sequentiality, Decidability, Finite Ambiguity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.137

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

Funding This work was partially supported by Deutsche Forschungsgemeinschaft (DFG), Graduiertenkolleg 1763 (QuantLA).

1 Introduction

A *max-plus automaton* is a finite automaton whose transitions are weighted by real numbers. A max-plus automaton assigns a weight to each of its runs by adding the weights of the transitions which constitute the run and it assigns a weight to every word by taking the maximum over the weights of all runs on the given word. Max-plus automata are weighted automata [40, 39, 27, 6, 13] over the max-plus semiring. In the form of min-plus automata, they were originally introduced by Imre Simon as a means to show the decidability of the *finite power property* [42, 43] and they enjoy a continuing interest [26, 19, 22, 7, 12, 16, 28]. They have found applications in many different contexts, for example to determine the star height of a language [18], to prove the termination of certain string rewriting systems [44], and to model discrete event systems [23]. They also appear in the context of natural language processing [29], where probabilities are often computed in the min-plus semiring as negative log-likelihoods for reasons of numerical stability.

Like finite automata, max-plus automata are by definition non-deterministic devices. However, while every finite automaton can be determinized [36], the same is in general not true for max-plus automata [22]. Actually, it is a long-standing open question whether given a max-plus automaton, the existence of an equivalent deterministic automaton can be decided. This problem is commonly known as the *sequentiality problem* and is one of the most prominent open questions about max-plus automata. For practical applications, the execution of a deterministic automaton is of course much more efficient than the execution of a non-deterministic one, so being able to decide whether a given automaton can be determinized is very much desirable. While open in general, the sequentiality problem has been shown to be decidable for some important subclasses of max-plus automata, namely for *unambiguous* [29], *finitely ambiguous* [22], and *polynomially ambiguous* [21] max-plus automata. Here, we call a max-plus automaton *unambiguous* if there exists at most one run on every word, *finitely ambiguous* if the number of runs on each word is bounded by a global constant, and *polynomially ambiguous* if the number of runs on each word is bounded polynomially in the length of the word. Note that the classes of deterministic, unambiguous, finitely ambiguous, polynomially ambiguous, and arbitrary max-plus automata form a strictly ascending hierarchy [22, 20, 28]. Also, deciding the degree of ambiguity of a max-plus automaton can easily be reduced to deciding the degree of ambiguity of a finite automaton. It is trivial to decide whether a finite automaton is deterministic. Polynomial time algorithms to decide whether a finite automaton is unambiguous, finitely ambiguous, or polynomially ambiguous can be found in [8, 45, 41].

While a given max-plus automaton may not be equivalent to a single deterministic max-plus automaton, this does not exclude the possibility that it is equivalent to the pointwise maximum of finitely many deterministic automata. The problem of deciding whether a max-plus automaton possesses such a *finitely sequential* representation is known as the *finite sequentiality problem.* The decidability of the finite sequentiality problem was posed as an open question in [19] and has been solved only recently for unambiguous [4] and finitely ambiguous [3] max-plus automata. Note that the class of max-plus automata which possess a finitely sequential representation lies strictly between the classes of deterministic and finitely ambiguous max-plus automata, and it is incomparable to the class of unambiguous max-plus automata [22].

In this paper, we show that the finite sequentiality problem is decidable for finitely ambiguous *max-plus tree automata.* Operating on trees instead of words, max-plus tree automata are a generalization of max-plus word automata and more generally, they are weighted tree automata [1, 5, 14, 17] over the max-plus semiring. Applications of max-plus tree automata include proving the termination of certain term rewriting systems [24] and they are commonly employed in natural language processing [35] in the form of *probabilistic context-free grammars.* Our approach to proving the decidability of the finite sequentiality problem for finitely ambiguous max-plus tree automata employs ideas from Bala's proof of the corresponding result for finitely ambiguous max-plus word automata [3]. However, due to lack of space, formal proofs had to be omitted in [3] and Bala's informal description of his methods does not suffice for reconstruction. Also, no other published version of [3] exists. We provide an honest attempt to compare our approach to his but note that our interpretation might not be accurate.

In his proof for max-plus word automata, Bala first introduces the *A-Fork property* and then proceeds to show that this property is a decidable criterion characterizing the finite sequentiality of a finitely ambiguous max-plus automaton. More precisely, he shows that a finitely ambiguous max-plus automaton possesses a finitely sequential representation if and only if the A-Fork property is not satisfied. To show the decidability of the A-Fork property, he shows its expressibility in a decidable fragment of Presburger arithmetic. To show that an automaton is not finitely sequential if the A-Fork property is satisfied, he uses pumping techniques similar to those employed in [4] for the finite sequentiality problem of unambiguous max-plus word automata. This part of his proof most likely employs Ramsey's Theorem [37] as it involves "colorings of finite hypercubes". His proof for the existence of a finitely sequential representation in case that the A-Fork property is not satisfied employs transducers and the notions of *critical pairs* and *close approximations*, none of which occur in our approach. We are thus unsure about the nature of this particular part of the proof, but it most likely uses a reduction to the decidability of the finite sequentiality problem for unambiguous automata.

Our approach is as follows. First, we introduce the *separation property*, a twofold modification of the A-Fork property. On the one hand, we endow our new property with a criterion accounting for the non-linear structure of trees. This new criterion is inspired by the criterion we added in [34] to the *fork property* [4], the property characterizing finite sequentiality of unambiguous max-plus word automata, in order to obtain the *tree fork property*, the property characterizing finite sequentiality of unambiguous max-plus tree automata. On the other hand, we strengthen the A-Fork property as with only the first modification, our new property would wrongly characterize some finitely sequential automata as not being finitely sequential. We then show that the separation property is decidable by employing Parikh's Theorem [31, 15] for a reduction to the decidability of the satisfiability of systems of linear inequalities over the rational numbers with integer solutions [30, 9]. This means in particular that we show the decidability of the finite sequentiality problem only for automata with weights in the rationals. Then we employ Ramsey's Theorem to show that no finitely sequential representation exists whenever the separation property is satisfied. Due to the criterion accounting for the non-linearity of trees, this is considerably more difficult than in [3] and it is in fact the most technical and the most challenging aspect of our result. Finally, we show that if the separation property is not satisfied for a given max-plus tree automaton, then we can construct finitely many unambiguous max-plus tree automata which all do not satisfy the tree fork property and whose pointwise maximum is equivalent to the automaton. By [34], these unambiguous automata then possess finitely sequential representations. Combining these, we obtain a finitely sequential representation of the original automaton.

2 Preliminaries

For a set X, we denote the power set of X by $\mathcal{P}(X)$ and the cardinality of X by $|X|$. For two sets X and Y and a mapping $f\colon X \to Y$, we call X the *domain* of f, denoted by $\mathrm{dom}(f)$. For a subset $X' \subseteq X$, we call the set $f(X') = \{y \in Y \mid \exists x \in X'\colon f(x) = y\}$ the *image* or *range of* X' *under* f. The *restriction of* f *to* X', denoted by $f{\restriction_{X'}}$, is the mapping $f{\restriction_{X'}}\colon X' \to Y$ defined by $f{\restriction_{X'}}(x) = f(x)$ for every $x \in X'$. For an element $y \in Y$, we call the set $f^{-1}(y) = \{x \in X \mid f(x) = y\}$ the *preimage of* y *under* f. For a second mapping $g\colon X \to Y$, we write $f = g$ if for all $x \in X$ we have $f(x) = g(x)$.

We let $\mathbb{N} = \{0, 1, 2, \ldots\}$. By $\mathbb{N}^*$ we denote the set of all finite words over $\mathbb{N}$. The empty word is denoted by ε, and the length of a word $w \in \mathbb{N}^*$ by $|w|$. The set $\mathbb{N}^*$ is partially ordered by the prefix relation $\leq_{\mathrm{P}}$ and totally ordered with respect to the lexicographic ordering $\leq_{\mathrm{L}}$. Two words from $\mathbb{N}^*$ are called *prefix-dependent* if they are in prefix relation, and otherwise they are called *prefix-independent*.

A *ranked alphabet* is a pair $(\Gamma, \mathrm{rk}_\Gamma)$, often abbreviated by Γ, where Γ is a finite set and $\mathrm{rk}_\Gamma\colon \Gamma \to \mathbb{N}$ a mapping which assigns a *rank* to every symbol. For every $m \geq 0$ we define $\Gamma^{(m)} = \mathrm{rk}_\Gamma^{-1}(m)$ as the set of all symbols of rank m. The rank of Γ is defined as $\mathrm{rk}(\Gamma) = \max\{\mathrm{rk}_\Gamma(a) \mid a \in \Gamma\}$. The set of (*finite, labeled, and ordered*) Γ*-trees*, denoted by T_Γ, is the set of all pairs $t = (\mathrm{pos}(t), \mathrm{label}_t)$, where $\mathrm{pos}(t) \subset \mathbb{N}^*$ is a finite non-empty prefix-closed set of *positions*, $\mathrm{label}_t\colon \mathrm{pos}(t) \to \Gamma$ is a mapping, and for every $w \in \mathrm{pos}(t)$ we have $wi \in \mathrm{pos}(t)$ iff $1 \leq i \leq \mathrm{rk}_\Gamma(\mathrm{label}_t(w))$. We write $t(w)$ for $\mathrm{label}_t(w)$ and $|t|$ for $|\mathrm{pos}(t)|$. We also refer to the elements of $\mathrm{pos}(t)$ as *nodes*, to ε as the *root* of t, and to prefix-maximal nodes as *leaves*. The *height* of t is defined as $\mathrm{height}(t) = \max_{w \in \mathrm{pos}(t)} |w|$. For a leaf $w \in \mathrm{pos}(t)$, the set $\{v \in \mathrm{pos}(t) \mid v \leq_{\mathrm{P}} w\}$ is called a *branch* of t.

Now let $s, t \in T_\Gamma$ and $w \in \mathrm{pos}(t)$. The *subtree of t at w*, denoted by $t{\restriction_w}$, is a Γ-tree defined as follows. We let $\mathrm{pos}(t{\restriction_w}) = \{v \in \mathbb{N}^* \mid wv \in \mathrm{pos}(t)\}$ and for $v \in \mathrm{pos}(t{\restriction_w})$, we let $\mathrm{label}_{t\restriction_w}(v) = t(wv)$. The *substitution of s into w of t*, denoted by $t\langle s \to w\rangle$, is a Γ-tree defined as follows. We let $\mathrm{pos}(t\langle s \to w\rangle) = (\mathrm{pos}(t) \setminus \{v \in \mathrm{pos}(t) \mid w \leq_{\mathrm{P}} v\}) \cup \{wv \mid v \in \mathrm{pos}(s)\}$. For $v \in \mathrm{pos}(t\langle s \to w\rangle)$, we let $\mathrm{label}_{t\langle s\to w\rangle}(v) = s(u)$ if $v = wu$ for some $u \in \mathrm{pos}(s)$, and otherwise $\mathrm{label}_{t\langle s\to w\rangle}(v) = t(v)$. For $a \in \Gamma^{(m)}$ and trees $t_1, \dots, t_m \in T_\Gamma$, we also write $a(t_1, \dots, t_m)$ to denote the tree t with $\mathrm{pos}(t) = \{\varepsilon\} \cup \{iw \mid i \in \{1, \dots, m\}, w \in \mathrm{pos}(t_i)\}$, $\mathrm{label}_t(\varepsilon) = a$, and $\mathrm{label}_t(iw) = t_i(w)$. For $a \in \Gamma^{(0)}$, the tree $a()$ is abbreviated by a.

For a ranked alphabet Γ, a tree over the alphabet $\Gamma_\diamond = (\Gamma \cup \{\diamond\}, \mathrm{rk}_\Gamma \cup \{\diamond \mapsto 0\})$ is called a *Γ-context*. Let $t \in T_{\Gamma_\diamond}$ be a Γ-context and let $w_1, \dots, w_n \in \mathrm{pos}(t)$ be a lexicographically ordered enumeration of all leaves of t labeled $\diamond$. Then we call t an *n-Γ-context* and define $\Diamond_i(t) = w_i$ for $i \in \{1, \dots, n\}$. For an n-Γ-context t and contexts $t_1, \dots, t_n \in T_{\Gamma_\diamond}$, we define $t(t_1, \dots, t_n) = t\langle t_1 \to \Diamond_1(t)\rangle \cdots \langle t_n \to \Diamond_n(t)\rangle$ by substitution of $t_1, \dots, t_n$ into the $\diamond$-leaves of t. A 1-Γ-context is also called a *Γ-word*. For a Γ-word s, we define $s^0 = \diamond$ and $s^{n+1} = s(s^n)$ for $n \geq 0$.

A *commutative semiring* is a tuple $(K, \oplus, \odot, \mathbb{0}, \mathbb{1})$, abbreviated by K, with operations sum $\oplus$ and product $\odot$ and constants $\mathbb{0}$ and $\mathbb{1}$ such that $(K, \oplus, \mathbb{0})$ and $(K, \odot, \mathbb{1})$ are commutative monoids, multiplication distributes over addition, and $\kappa \odot \mathbb{0} = \mathbb{0} \odot \kappa = \mathbb{0}$ for every $\kappa \in K$. In this paper, we mainly consider the following two semirings.

- The *Boolean semiring* $\mathbb{B} = (\{0, 1\}, \vee, \wedge, 0, 1)$ with disjunction $\vee$ and conjunction $\wedge$.
- The *max-plus semiring* $\mathbb{Q}_{\max} = (\mathbb{Q} \cup \{-\infty\}, \max, +, -\infty, 0)$ where the sum and the product operations are max and $+$, respectively, extended to $\mathbb{Q} \cup \{-\infty\}$ in the usual way.

For a commutative semiring $(K, \oplus, \odot, \mathbb{0}, \mathbb{1})$ and an integer $n \geq 1$, the *product semiring* $(K^n, \oplus_n, \odot_n, \mathbb{0}_n, \mathbb{1}_n)$ is defined by componentwise operations and the constants $\mathbb{0}_n = (\mathbb{0}, \dots, \mathbb{0})$ and $\mathbb{1}_n = (\mathbb{1}, \dots, \mathbb{1})$. We will usually denote $\oplus_n$ and $\odot_n$ simply by $\oplus$ and $\odot$.

Let $(K, \oplus, \odot, \mathbb{0}, \mathbb{1})$ be a commutative semiring. A *weighted bottom-up finite state tree automaton (short: WTA) over K and Γ* is a tuple $\mathcal{A} = (Q, \Gamma, \mu, \nu)$ where Q is a finite set (of states), Γ is a ranked alphabet (of input symbols), $\mu\colon \bigcup_{m=0}^{\mathrm{rk}(\Gamma)} Q^m \times \Gamma^{(m)} \times Q \to K$ (the function of transition weights), and $\nu\colon Q \to K$ (the function of final weights). We define $\Delta_{\mathcal{A}} = \mathrm{dom}(\mu)$. A tuple $d \in \Delta_{\mathcal{A}}$ is called a *transition* and d is called *valid* if $\mu(d) \neq \mathbb{0}$. A state $q \in Q$ is called *final* if $\nu(q) \neq \mathbb{0}$.

We call a WTA over the max-plus semiring a *max-plus-WTA* and a WTA over the Boolean semiring a *finite tree automaton* (FTA). We also write a WTA $\mathcal{A} = (Q, \Gamma, \mu, \nu)$ over $\mathbb{B}$ as a tuple $\mathcal{A}' = (Q, \Gamma, \delta, F)$ where $\delta = \{d \in \Delta_{\mathcal{A}} \mid \mu(d) = 1\}$ and $F = \{q \in Q \mid \nu(q) = 1\}$.

For a tree $t \in T_\Gamma$, a mapping $r\colon \mathrm{pos}(t) \to Q$ is called a *quasi-run of $\mathcal{A}$ on t*. For a quasi-run r on t and a position $w \in \mathrm{pos}(t)$ with $t(w) = a \in \Gamma^{(m)}$, the tuple $\mathbb{t}(t, r, w) = (r(w1), \dots, r(wm), a, r(w))$ is called the *transition at w*. The quasi-run r is called a *(valid) run* if for every $w \in \mathrm{pos}(t)$ the transition $\mathbb{t}(t, r, w)$ is valid with respect to $\mathcal{A}$. We call a run r *accepting* if $r(\varepsilon)$ is final. By $\mathrm{Run}_{\mathcal{A}}(t)$ and $\mathrm{Acc}_{\mathcal{A}}(t)$ we denote the sets of all runs and all accepting runs of $\mathcal{A}$ on t, respectively. For a state $q \in Q$, we denote by $\mathrm{Run}_{\mathcal{A}}(t, q)$ the set of all runs $r \in \mathrm{Run}_{\mathcal{A}}(t)$ such that $r(\varepsilon) = q$.

For a run $r \in \mathrm{Run}_{\mathcal{A}}(t)$, the *weight of r* is defined by $\mathrm{wt}_{\mathcal{A}}(t, r) = \bigodot_{w \in \mathrm{pos}(t)} \mu(\mathbb{t}(t, r, w))$. The *behavior of $\mathcal{A}$*, denoted by $[\![\mathcal{A}]\!]$, is the mapping defined for every $t \in T_\Gamma$ by $[\![\mathcal{A}]\!](t) = \bigoplus_{r \in \mathrm{Acc}_{\mathcal{A}}(t)} (\mathrm{wt}_{\mathcal{A}}(t, r) \odot \nu(r(\varepsilon)))$, where the sum over the empty set is $\mathbb{0}$ by convention. The *support* of a WTA $\mathcal{A}$ is the set $\mathrm{supp}(\mathcal{A}) = \{t \in T_\Gamma \mid [\![\mathcal{A}]\!](t) \neq \mathbb{0}\}$. The support of an FTA $\mathcal{A}$ is also called the *language accepted by $\mathcal{A}$* and denoted by $\mathcal{L}(\mathcal{A})$. A subset $L \subseteq T_\Gamma$ is called *recognizable* if there exists an FTA $\mathcal{A}$ with $L = \mathcal{L}(\mathcal{A})$.

For a WTA $\mathcal{A} = (Q, \Gamma, \mu, \nu)$, a run of $\mathcal{A}$ on a Γ-context t is a run of the WTA $\mathcal{A}' = (Q, \Gamma_\diamond, \mu', \nu)$ on t, where $\mu'(\diamond, q) = \mathbb{1}$ for all $q \in Q$ and $\mu'(d) = \mu(d)$ for all $d \in \Delta_{\mathcal{A}}$. We denote $\mathrm{Run}^\diamond_{\mathcal{A}}(t) = \mathrm{Run}_{\mathcal{A}'}(t)$ and for $r \in \mathrm{Run}^\diamond_{\mathcal{A}}(t)$ write $\mathrm{wt}^\diamond_{\mathcal{A}}(t, r) = \mathrm{wt}_{\mathcal{A}'}(t, r)$. For an n-Γ-context $t \in T_{\Gamma_\diamond}$ and states $q_0, \ldots, q_n$, we denote by $\mathrm{Run}^\diamond_{\mathcal{A}}(q_1, \ldots, q_n, t, q_0)$ the set of all runs $r \in \mathrm{Run}^\diamond_{\mathcal{A}}(t)$ such that $r(\varepsilon) = q_0$ and $r(\Diamond_i(t)) = q_i$ for every $i \in \{1, \ldots, n\}$.

We consider the set $\Gamma \times Q$ as an alphabet by defining $\mathrm{rk}_{\Gamma\times Q}(a, q) = \mathrm{rk}_\Gamma(a)$ for every pair $(a, q) \in \Gamma \times Q$ and identify every tree $t' \in T_{\Gamma\times Q}$ with the pair (t, r) given by $t = (\mathrm{pos}(t'), \pi_\Gamma \circ \mathrm{label}_{t'}) \in T_\Gamma$ and $r = \pi_Q \circ \mathrm{label}_{t'}$, where $\pi_\Gamma \colon \Gamma \times Q \to \Gamma$ and $\pi_Q \colon \Gamma \times Q \to Q$ are the projections. For a Γ-word $s \in T_{\Gamma_\diamond}$, a state $q \in Q$, and a run $r_s \in \mathrm{Run}^\diamond_{\mathcal{A}}(q, s, q)$, we define $(s, r_s)^0 = (\diamond, q)$ and $(s, r_s)^{n+1} = (s, r_s)\langle (s, r_s)^n \to \Diamond_1(s)\rangle$ for $n \geq 0$.

We call a WTA $\mathcal{A} = (Q, \Gamma, \mu, \nu)$ over K and Γ *trim* if for every $p \in Q$, there exist $t \in T_\Gamma$, $r \in \mathrm{Acc}_{\mathcal{A}}(t)$, and $w \in \mathrm{pos}(t)$ with $r(w) = p$. The *trim part of* $\mathcal{A}$ is the automaton obtained from $\mathcal{A}$ by removing all states $p \in Q$ for which no such t, r, and w exist. This process obviously has no influence on $[\![\mathcal{A}]\!]$. We call $\mathcal{A}$ *complete* if for every $m \geq 0$, $a \in \Gamma^{(m)}$, and $(q_1, \ldots, q_m) \in Q^m$, there exists at least one $q \in Q$ with $\mu(q_1, \ldots, q_m, a, q) \neq \mathbb{0}$. We call $\mathcal{A}$ *deterministic* or *sequential* if for every $m \geq 0$, $a \in \Gamma^{(m)}$, and $(q_1, \ldots, q_m) \in Q^m$, there exists at most one $q \in Q$ with $\mu(q_1, \ldots, q_m, a, q) \neq \mathbb{0}$. If there exists an integer $M \geq 1$ such that $|\mathrm{Acc}_{\mathcal{A}}(t)| \leq M$ for every $t \in T_\Gamma$, we call $\mathcal{A}$ *M-ambiguous*. We call $\mathcal{A}$ *finitely ambiguous* if it is M-ambiguous for some $M \geq 1$ and *unambiguous* if it is 1-ambiguous. We call the behavior $[\![\mathcal{A}]\!]$ of $\mathcal{A}$ *finitely sequential* if there exist finitely many deterministic WTA $\mathcal{A}_1, \ldots, \mathcal{A}_n$ over K and Γ such that $[\![\mathcal{A}]\!] = \bigoplus_{i=1}^n [\![\mathcal{A}_i]\!]$, where the sum is taken pointwise.

3 The Criterion for Finite Sequentiality

We will show that for a finitely ambiguous max-plus-WTA $\mathcal{A}$, it is decidable whether its behavior $[\![\mathcal{A}]\!]$ is finitely sequential. For this, we first formulate the *separation property*, a generalization of Bala's *A-Fork property* [3]. Then we show that it is decidable whether the separation property is satisfied and that the behavior of a max-plus-WTA is finitely sequential if and only if the separation property is not satisfied. In the following, let Γ be a ranked alphabet. We begin by recalling the tree fork property and the related concepts of *rivals*, *reachers*, *distinguishers*, and *forks*. Intuitively, two states of a finitely ambiguous max-plus-WTA $\mathcal{A}$ are called *rivals* if they can be reached by the same tree u and they can loop in the same Γ-word s but the weights of these loops differ. The tree u is then called a *reacher* of p and q and the Γ-word s a *distinguisher* for p and q. For two rivals p and q, a Γ-word f is called a *p-q-fork* if f can both loop in p and also go from p to q, in a bottom-up sense. We say that $\mathcal{A}$ satisfies the tree fork property if there exist two rivals p and q such that either there exists a p-q-fork or p and q can occur at prefix-independent positions in some run of $\mathcal{A}$. Formally, these definitions are as follows.

▶ **Definition 1.** *Let $\mathcal{A} = (Q, \Gamma, \mu, \nu)$ be a finitely ambiguous max-plus-WTA. Two states $p, q \in Q$ are called* rivals *if there exists a tree $u \in T_\Gamma$ with $\mathrm{Run}_{\mathcal{A}}(u, p) \neq \emptyset$ and $\mathrm{Run}_{\mathcal{A}}(u, q) \neq \emptyset$ and a Γ-word s with runs $r_p \in \mathrm{Run}^\diamond_{\mathcal{A}}(p, s, p)$ and $r_q \in \mathrm{Run}^\diamond_{\mathcal{A}}(q, s, q)$ such that $\mathrm{wt}^\diamond_{\mathcal{A}}(s, r_p) \neq \mathrm{wt}^\diamond_{\mathcal{A}}(s, r_q)$. In this case, we call u a p-q-*reacher *and s a p-q-*distinguisher*. We say that $\mathcal{A}$ satisfies the *tree fork property *if at least one of the following two conditions is satisfied.*

(i) *There exist rivals $p, q \in Q$ and a Γ-word f with $\mathrm{Run}_{\mathcal{A}}(p, f, p) \neq \emptyset$ and $\mathrm{Run}_{\mathcal{A}}(p, f, q) \neq \emptyset$. In this case, we call f a p-q-*fork*.*

(ii) *There exist rivals $p, q \in Q$, a 2-Γ-context $t \in T_{\Gamma_\diamond}$, and a run $r \in \mathrm{Run}^\diamond_{\mathcal{A}}(t)$ with $r(\Diamond_1(t)) = p$ and $r(\Diamond_2(t)) = q$. In this case, we call t a p-q-*split*.*

We have the following theorem relating the tree fork property to finite sequentiality of unambiguous max-plus-WTA.

▶ **Theorem 2** ([34]). *The behavior of a trim unambiguous max-plus-WTA $\mathcal{A}$ is finitely sequential if and only if $\mathcal{A}$ does not satisfy the tree fork property. If $[\![\mathcal{A}]\!]$ is finitely sequential, a finitely sequential representation of $\mathcal{A}$ can be effectively constructed.*

For finitely ambiguous max-plus-WTA, however, the tree fork property does not capture finite sequentiality. To see why, consider an unambiguous max-plus-WTA $\mathcal{A}$ satisfying the tree fork property [34, 4, 22] and let L be the largest weight used in $\mathcal{A}$. Then construct a one-state max-plus-WTA $\mathcal{B}$ whose every transition weight and every final weight is L. Clearly, $\mathcal{B}$ is deterministic and we have $[\![\mathcal{B}]\!] \geq [\![\mathcal{A}]\!]$. By taking the disjoint union $\mathcal{A} \cup \mathcal{B}$ of $\mathcal{A}$ and $\mathcal{B}$, we obtain a 2-ambiguous max-plus-WTA which satisfies the tree fork property but whose behavior coincides with that of the deterministic automaton $\mathcal{B}$. In this particular example, the states relevant for the tree fork property to be satisfied are not relevant at all for the behavior of the automaton.

For the rest of this paper, let $\mathcal{A}$ be a trim finitely ambiguous max-plus-WTA over the ranked alphabet Γ. In order to reduce the finite sequentiality problem of finitely ambiguous max-plus-WTA to that of unambiguous max-plus-WTA, we decompose $\mathcal{A}$ into a maximum of finitely many unambiguous max-plus-WTA $\mathcal{A}_1, \ldots, \mathcal{A}_N$ and then analyze the interplay of these latter automata. We can do so as in fact, every finitely ambiguous WTA can be decomposed into finitely many unambiguous WTA [22, 32]. This is a common approach when dealing with finite ambiguity [22, 25, 33] and is also used by Bala in the corresponding proof for words [3]. In the simplest case, if $\mathcal{A}_1, \ldots, \mathcal{A}_N$ all do not satisfy the tree fork property, we find a finitely sequential representation of $\mathcal{A}$ by constructing such a representation for each $\mathcal{A}_n$ and then combining all of these. However, if some $\mathcal{A}_n$ does satisfy the tree fork property, we have to analyze whether this automaton contributes enough to the behavior of $\mathcal{A}$ for there not to exist a finitely sequential representation of $\mathcal{A}$. For an easier analysis of the automata $\mathcal{A}_1, \ldots, \mathcal{A}_N$, we normalize their final weights and consider their product as follows.

▶ **Lemma 3** ([41, 22, 32, 10]). *We can effectively find an integer $M \in \mathbb{N}$ and construct a trim WTA $\mathcal{U} = (Q, \Gamma, \mu, \nu)$ over $\mathbb{Q}^M_{\max}$ and Γ such that*

- *$\mathcal{U}$ is unambiguous,*
- *$\mu(\Delta_\mathcal{U}) \subseteq \mathbb{Q}^M \cup \{(-\infty, \ldots, -\infty)\}$ and $\nu(Q) \subseteq \{(0, \ldots, 0), (-\infty, \ldots, -\infty)\}$, and*
- *for every $t \in T_\Gamma$ we have $[\![\mathcal{A}]\!](t) = \max_{i=1}^{M} \pi_i([\![\mathcal{U}]\!](t))$,*

where $\pi_i \colon \mathbb{Q}^M_{\max} \to \mathbb{Q}_{\max}$ is the projection to the i-th coordinate for every $i \in \{1, \ldots, M\}$.

Let $\mathcal{U}$ be the automaton we obtain for $\mathcal{A}$ from Lemma 3. For a tree $t \in T_\Gamma$, a Γ-word $s \in T_{\Gamma_\diamond}$, runs $r_t \in \mathrm{Run}_\mathcal{U}(t)$, $r_s \in \mathrm{Run}^\diamond_\mathcal{U}(s)$, states $p, q \in Q$, and a coordinate $i \in \{1, \ldots, M\}$, we let $\mathrm{wt}_i(t, r_t) = \pi_i(\mathrm{wt}_\mathcal{U}(t, r_t))$, $\mathrm{wt}^\diamond_i(s, r_s) = \pi_i(\mathrm{wt}^\diamond_\mathcal{U}(s, r_s))$, and $\mathrm{wt}^\diamond_i(p, s, q) = \mathrm{wt}^\diamond_i(s, r^q_p)$ for the unique run $r^q_p \in \mathrm{Run}^\diamond_\mathcal{U}(p, s, q)$. We define the concepts of rivals, reachers, distinguishers, and forks for $\mathcal{U}$ as follows.

▶ **Definition 4.** *Let $i \in \{1, \ldots, M\}$, $p, q \in Q$, $t \in T_\Gamma$, and $r \in \mathrm{Acc}_\mathcal{U}(t)$.*

- *We call p and q i-*rivals *if there exists a tree $u \in T_\Gamma$ such that $\mathrm{Run}_\mathcal{U}(u, p) \neq \emptyset$ and $\mathrm{Run}_\mathcal{U}(u, q) \neq \emptyset$ and a Γ-word s such that $\mathrm{Run}^\diamond_\mathcal{U}(p, s, p) \neq \emptyset$, $\mathrm{Run}^\diamond_\mathcal{U}(q, s, q) \neq \emptyset$, and $\mathrm{wt}^\diamond_i(p, s, p) \neq \mathrm{wt}^\diamond_i(q, s, q)$. In this case, we also call u a p-q-*reacher *and s an i-p-q-*distinguisher.
- *We call a Γ-word f an i-p-q-*fork *if p and q are i-rivals, $\mathrm{Run}^\diamond_\mathcal{U}(p, f, p) \neq \emptyset$, and $\mathrm{Run}^\diamond_\mathcal{U}(p, f, q) \neq \emptyset$.*

- *We say that (t,r) is i-p-q-*fork-broken *if there exist positions $w_p, w_q \in \operatorname{pos}(t)$ such that $w_q <_P w_p$, $r(w_p) = p$, $r(w_q) = q$, and $(t\langle \diamond \to w_p \rangle)\restriction_{w_q}$ is an i-p-q-fork.*
- *We say that (t,r) is i-p-q-*split-broken *if p and q are i-rivals and there exist two prefix-independent positions $w_p, w_q \in \operatorname{pos}(t)$ with $r(w_p) = p$ and $r(w_q) = q$.*

*When appropriate, we may drop some of the hyphenated modifiers from the terms above; for example, we will refer to (t,r) as i-*fork-broken *if there exist states $p, q \in Q$ such that (t,r) is i-p-q-fork-broken and as i-*split-broken *if there exist states $p, q \in Q$ such that (t,r) is i-p-q-split-broken. We call (t,r) i-*broken *if it is i-fork-broken or i-split-broken.*

Our concept of brokenness, is inspired by Bala's notion of "broken paths" [3]. Of course, as his proof is concerned with words, the concept of split-brokenness does not exist. His notion of brokenness corresponds to our notion of fork-brokenness. Employing the notion of brokenness, Bala characterizes finite sequentiality of finitely ambiguous max-plus word automata using the *A-Fork property*. Translated to tree automata, the A-Fork property is defined as follows. We say that $\mathcal{U}$ satisfies the A-Fork property if for every constant $C > 0$, there exists a tree $t \in T_\Gamma$ and an accepting run $r \in \operatorname{Acc}_{\mathcal{U}}(t)$ such that for some weight-maximal coordinate i, i.e., with $\operatorname{wt}_i(t,r) = \max_{j=1}^{M} \operatorname{wt}_j(t,r)$, we have that (t,r) is i-broken and for every coordinate j such that (t,r) is not j-broken, we have $\operatorname{wt}_j(t,r) < \operatorname{wt}_i(t,r) - C$. In other words, the A-Fork property is satisfied if broken coordinates are able to dominate non-broken coordinates by an arbitrarily large margin. Bala shows that a finitely ambiguous max-plus word automaton is finitely sequential if and only if the corresponding automaton $\mathcal{U}$ does not satisfy the A-Fork property. For tree automata, however, this criterion does not capture finite sequentiality. More precisely, if we know that there do not exist a tree t and a run r on t such that (t,r) is split-broken, then the A-Fork property does capture finite sequentiality also for tree automata. However, if $\mathcal{U}$ satisfies the A-Fork property due to split-broken coordinates dominating non-broken coordinates, the behavior of $\mathcal{A}$ may still be finitely sequential. This is evidenced by the following example.

▶ **Example 5.** Consider the scenario for $\mathcal{U}$ as defined in Figure 1. The support of $\mathcal{U}$ consists of all trees of the form $c(b^k(a_i^m(d_i)), b^l(a_j^n(d_j)))$ with $i, j \in \{1,2\}$, $k, l > 0$, and $m, n \geq 0$. A valid run on such a tree necessarily assigns states from $\{p_1, p_2, p\}$ to the left branch of the tree and states from $\{q_1, q_2, q\}$ to the right branch of the tree. Moreover, if a branch begins with a letter d_i, this branch is assigned states from $\{p_i, q_i, p, q\}$. In particular, we see that $\mathcal{U}$ is unambiguous. The states p and q are 2-rivals as we see from the p-q-reacher $u = b(a_1(d_1))$ and the 2-p-q-distinguisher $s = b(\diamond)$. By considering the trees $t_n = c(b(a_1^n(d_1)), b(a_2^n(d_2)))$, we see that runs exist where p and q occur prefix-independently and the weight of coordinate 2 is arbitrarily larger than the weights of coordinates 1 and 3 since we have $[\![\mathcal{U}]\!](t_n) = (-n, 0, -n)$. However, in t_n the subtrees below p and q are distinct, thus a deterministic automaton can distinguish between them.

In fact, if $\mathcal{U}$ is given this way, we can construct a finitely sequential representation of $[\![\mathcal{A}]\!]$ as follows. All trees of the form $c(b^k(a_1^m(d_1)), b^l(a_1^n(d_1)))$ are assigned the weight $(-m-n+k+l-2, k-l, k+l-2)$, so coordinate 3 is always dominant. Similarly, coordinate 1 is dominant for trees of the form $c(b^k(a_2^m(d_2)), b^l(a_2^n(d_2)))$. These trees can be handled by the deterministic max-plus-WTA obtained from $\mathcal{U}$ by removing the states q_1 and q_2, letting $\mu(p, p, c, \top) = (0,0,0)$, and replacing μ with $\pi_1 \circ \mu$ and $\pi_3 \circ \mu$, respectively. For trees of the form $c(b^k(a_1^m(d_1)), b^l(a_2^n(d_2)))$, we remove the states p_2 and q_1 from $\mathcal{U}$ and then construct three deterministic max-plus-WTA by replacing μ by $\pi_1 \circ \mu, \pi_2 \circ \mu$, and $\pi_3 \circ \mu$, respectively. For the trees $c(b^k(a_2^m(d_2)), b^l(a_1^n(d_1)))$ we can proceed similarly. The pointwise maximum of the automata constructed this way is then equivalent to $[\![\mathcal{A}]\!]$. This example shows in particular that if $\mathcal{U}$ satisfies the A-Fork property, $[\![\mathcal{A}]\!]$ can still be finitely sequential.

$$\nu(\top) = (0,0,0) \qquad \mu(p,b,p) = (1,1,1) \qquad \mu(p_1,a_1,p_1) = \mu(q_1,a_1,q_1) = (-1,0,0)$$

$$\mu(p,q,c,\top) = (0,0,0) \qquad \mu(q,b,q) = (1,-1,1) \qquad \mu(p_2,a_2,p_2) = \mu(q_2,a_2,q_2) = (0,0,-1)$$

$$\mu(p_1,b,p) = \mu(p_2,b,p) = \mu(q_1,b,q) = \mu(q_2,b,q) = (0,0,0)$$

$$\mu(d_1,p_1) = \mu(d_1,q_1) = \mu(d_2,p_2) = \mu(d_2,q_2) = (0,0,0)$$

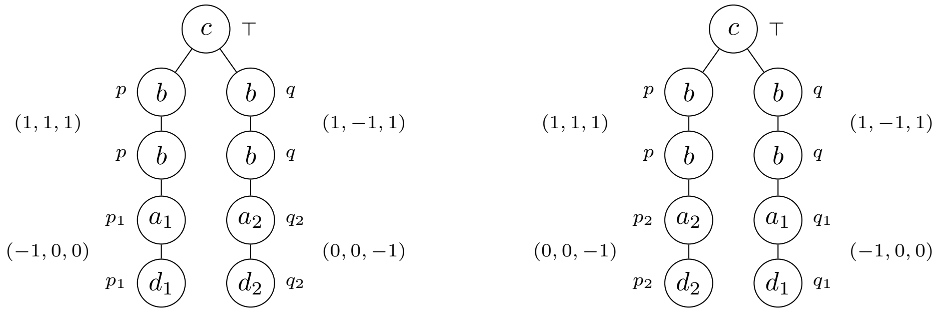


Figure 1 A scenario for the automaton $\mathcal{U}$: The automaton $(\{p_1,p_2,q_1,q_2,p,q,\top\},\Gamma,\mu,\nu)$ over the ranked alphabet $\Gamma = \{a_1,a_2,b,c,d_1,d_2\}$ where $c \in \Gamma^{(2)}$, $a_1,a_2,b \in \Gamma^{(1)}$, and $d_1,d_2 \in \Gamma^{(0)}$. All unspecified weights are assumed to be $-\infty$. The states p and q are 2-rivals.

Our fundamental idea to adapt the A-Fork property to tree automata is to formulate our version not for $\mathcal{U}$ but for a *covering* of $\mathcal{U}$. Oversimplifying, a *covering* of an automaton is a new automaton obtained by enhancing the states of the original automaton with additional capacities to store information. A prominent example of a covering construction is the *Schützenberger covering* of an automaton. The Schützenberger covering in particular has already been employed in a number of decidability results for max-plus automata [22, 4, 3, 33, 34]. For more background on coverings, see [38].

Here, we construct from $\mathcal{U}$ an unambiguous automaton $\boldsymbol{\mathcal{U}}$ with the same behavior as $\mathcal{U}$ and whose states are tuples from $Q \times \mathcal{P}(Q) \times \mathcal{P}(Q^4 \times \mathcal{P}(Q^2))$. Every run $\mathbf{r}$ of $\boldsymbol{\mathcal{U}}$ on a tree $t \in T_\Gamma$ will correspond uniquely to a run of $\mathcal{U}$ on t, given by projecting to the first entry. For a position w, the second entry of $\mathbf{r}(w)$ will be the set of all states $q \in Q$ which can be reached by $t\!\upharpoonright_w$, i.e., for which $\mathrm{Run}_{\mathcal{U}}(t\!\upharpoonright_w, q)$ is non-empty. The third entry of $\mathbf{r}(w)$ will contain a tuple (p,q,p',q',Y) if and only if $t\!\upharpoonright_w$ can reach p and q with two runs r_p and r_q, these runs visited p' and q' simultaneously at some position v in the past, and Y consists of all pairs of states which these runs visited simultaneously up to v. More precisely, the third entry of $\mathbf{r}(w)$ will consist of all tuples (p,q,p',q',Y) such that (1) there exist runs $r_p \in \mathrm{Run}_{\mathcal{U}}(t\!\upharpoonright_w, p)$ and $r_q \in \mathrm{Run}_{\mathcal{U}}(t\!\upharpoonright_w, q)$, (2) for some position below w, i.e., some position $v \in \mathrm{pos}(t\!\upharpoonright_w)$, we have $r_p(v) = p'$ and $r_q(v) = q'$, and (3) Y is the set of all pairs of states $(r_p(vu), r_q(vu))$ with $u \in \mathrm{pos}(t\!\upharpoonright_{wv})$. Our intention of considering the covering $\boldsymbol{\mathcal{U}}$ is to increase the knowledge we have about each pair of rivals. For two rivals of $\mathcal{U}$, all we know is what the definition of rivals specifies. For two rivals of $\boldsymbol{\mathcal{U}}$ on the other hand, we can show that they are necessarily of the form (p,P,V) and (q,P,V) where p and q are rivals of $\mathcal{U}$. This allows us to infer statements about the rivals of $\boldsymbol{\mathcal{U}}$ which are not necessarily true for the rivals of $\mathcal{U}$. The precise construction of $\boldsymbol{\mathcal{U}}$ is as follows.

▶ **Construction 6.** We define $\boldsymbol{\mathcal{U}} = (\mathbf{Q},\Gamma,\boldsymbol{\mu},\boldsymbol{\nu})$ as the trim part of the automaton $\boldsymbol{\mathcal{U}}' = (\mathbf{Q}',\Gamma,\boldsymbol{\mu}',\boldsymbol{\nu}')$ defined as follows. We let $\mathbf{Q}' = Q \times \mathcal{P}(Q) \times \mathcal{P}(Q^4 \times \mathcal{P}(Q^2))$ and for subsets $P_1,\ldots,P_m \subseteq Q$ and a letter $a \in \Gamma$ with $\mathrm{rk}_\Gamma(a) = m$, we let $\mathrm{succ}(P_1,\ldots,P_m,a) = \{q_0 \mid \exists (q_1,\ldots,q_m) \in P_1 \times \ldots \times P_m \text{ with } \mu(q_1,\ldots,q_m,a,q_0) \in \mathbb{Q}^M\}$. For $i \in \{1,\ldots,\mathrm{rk}_\Gamma(a)\}$ and states $(p,q,p',q',Y) \in Q^4 \times \mathcal{P}(Q^2)$, we let $\mathrm{succ}(P_1,\ldots,P_m,(p,q,p',q',Y),i,a) =$

$\mathrm{succ}(P_1, \ldots, P_{i-1}, \{p\}, P_{i+1}, \ldots, P_m, a) \times \mathrm{succ}(P_1, \ldots, P_{i-1}, \{q\}, P_{i+1}, \ldots, P_m, a) \times \{p'\} \times \{q'\} \times \{Y\}$. For $V \subseteq Q^4 \times \mathcal{P}(Q^2)$ and $p, q \in Q$, we let $\mathrm{visited}(p, q, V) = \{(p', q') \mid (p, q, p', q', Y) \in V \text{ for some } Y \subseteq Q^2\}$. Then for $a \in \Gamma$ with $\mathrm{rk}_\Gamma(a) = m$ and $(p_0, P_0, V_0), \ldots, (p_m, P_m, V_m) \in \mathbf{Q}'$, we define $\boldsymbol{\nu}'(p_0, P_0, V_0) = \nu(p_0)$ and $\boldsymbol{\mu}'((p_1, P_1, V_1), \ldots, (p_m, P_m, V_m), a, (p_0, P_0, V_0)) =$

$$\begin{cases} \mu(p_1, \ldots, p_m, a, p_0) & \text{if } P_0 = \mathrm{succ}(P_1, \ldots, P_m, a) \text{ and with} \\ & V = \bigcup_{i=1}^m \bigcup_{(p,q,p',q',Y) \in V_i} \mathrm{succ}(P_1, \ldots, P_m, (p, q, p', q', Y), i, a) \\ & \text{we have } V_0 = V \cup \{(p, q, p, q, Y) \mid p, q \in P_0 \text{ and} \\ & \qquad\qquad Y = \mathrm{visited}(p, q, V) \cup \{(p, q)\}\} \\ (-\infty, \ldots, -\infty) & \text{otherwise.} \end{cases}$$

Then $\boldsymbol{\mathcal{U}}$ satisfies the properties described above. We let $\pi_1 \colon \mathbf{Q} \to Q$, $\pi_2 \colon \mathbf{Q} \to \mathcal{P}(Q)$, and $\pi_3 \colon \mathbf{Q} \to \mathcal{P}(Q^4 \times \mathcal{P}(Q^2))$ be the projections, and let $\mathbf{wt}_i$ and $\mathbf{wt}_i^\diamond$ be defined for $\boldsymbol{\mathcal{U}}$ in the same way we defined wt_i and $\mathrm{wt}_i^\diamond$ for $\mathcal{U}$. Furthermore, we note that the concepts of rivals, reachers, distinguishers, and forks as defined for $\mathcal{U}$ in Definition 4 apply to $\boldsymbol{\mathcal{U}}$ in a similar fashion.

Finally, we introduce our version of the A-Fork property. To allow for easier proofs, we use a different formulation and consequently a different name. But in fact, $\boldsymbol{\mathcal{U}}$ satisfies the separation property if and only if it satisfies the A-Fork property in the way we translated it to trees earlier.

▶ **Definition 7.** *Let $C \in \mathbb{N}$. We call a set $I \subseteq \{1, \ldots, M\}$ C-*separable *if there exists a tree $t \in T_\Gamma$ and a run $\mathbf{r} \in \mathrm{Acc}_{\boldsymbol{\mathcal{U}}}(t)$ such that*

(i) *if $i \in I$, then $(t, \mathbf{r})$ is i-broken and*

(ii) *if $j \in \{1, \ldots, M\} \setminus I$, then $\mathbf{wt}_j(t, \mathbf{r}) \leq \mathbf{wt}_i(t, \mathbf{r}) - C$ for all $i \in I$.*

*In this case, we also say that $(t, \mathbf{r})$ is I-C-*separated. *We call I* separable *if it is C-separable for every $C \in \mathbb{N}$ and define $\mathcal{I}$ as the set of all separable subsets $I \subseteq \{1, \ldots, M\}$. If $\mathcal{I}$ is non-empty, we say that $\boldsymbol{\mathcal{U}}$ satisfies the* separation property *or, for short, that $\boldsymbol{\mathcal{U}}$ is* broken.

Our main result is to prove the following theorem relating the separation property to the finite sequentiality problem of finitely ambiguous max-plus-WTA.

▶ **Theorem 8.** *The behavior $[\![\mathcal{A}]\!]$ of $\mathcal{A}$ is finitely sequential if and only if $\boldsymbol{\mathcal{U}}$ is not broken. Moreover, it is decidable whether $\boldsymbol{\mathcal{U}}$ is broken. In particular, it is decidable whether $[\![\mathcal{A}]\!]$ is finitely sequential.*

We separate the proof of Theorem 8 into three parts. Due to lack of space, we have to restrict ourselves to brief descriptions of our methods. In Section 3.1, we outline that it is decidable whether $\boldsymbol{\mathcal{U}}$ is broken. This part of the proof does not follow any idea from [3] as in his proof, Bala reduces the decidability of the A-Fork property to the decidability of a decidable fragment of Presburger arithmetic. In Section 3.2, we show that if $\boldsymbol{\mathcal{U}}$ is broken, then $[\![\mathcal{A}]\!]$ is not finitely sequential. This part of the proof employs ideas from [4, 3, 34], but extends these non-trivially. This particular proof is the most challenging and technical aspect of our result. Finally, in Section 3.3, we outline how to construct finitely many deterministic max-plus-WTA whose pointwise maximum is equivalent to $[\![\mathcal{A}]\!]$ in case that $\boldsymbol{\mathcal{U}}$ is not broken. Although this part is inspired by an idea in [3], we are not sure whether we employ this idea in the same way.

For all of our proofs, it is crucial that for every two states of $\boldsymbol{\mathcal{U}}$, we can decide whether they are rivals [2, Section 4], [11, Section 5.4]. For two rivals of an unambiguous automaton, it is in fact quite easy to give an upper bound on the size of their smallest distinguisher and smallest reacher. Thus, deciding whether two states are rivals reduces to checking for finitely many trees whether they can reach both states and checking for finitely many Γ-words whether they are a distinguisher for these two states. For Section 3.3, we require an even more precise statement, namely that if s is a distinguisher for two rivals $\mathbf{p}$ and $\mathbf{q}$, then we can obtain a $\mathbf{p}$-$\mathbf{q}$-distinguisher of height at most $4|\mathbf{Q}|^2$ by removing loops from the unique runs looping in $\mathbf{p}$ and $\mathbf{q}$. For this, we employ the notion of a *truncation*. Simply put, for a Γ-word s and a run $\mathbf{r}$ on s, a truncation of $(s, \mathbf{r})$ is any pair $(s', \mathbf{r}')$ of a Γ-word s' and a run $\mathbf{r}'$ on s' which can be obtained by repeatedly cutting loops from $(s, \mathbf{r})$.

▶ **Definition 9.** *Let $s, s' \in T_{\Gamma_\diamond}$ be Γ-words, $\mathbf{r} \in \mathrm{Run}^{\diamond}_{\boldsymbol{\mathcal{U}}}(s)$, and $\mathbf{r}' \in \mathrm{Run}^{\diamond}_{\boldsymbol{\mathcal{U}}}(s')$. We say that $(s', \mathbf{r}')$ is a* truncation *of $(s, \mathbf{r})$, denoted by $(s, \mathbf{r}) \mathrel{\text{✄}} (s', \mathbf{r}')$, if there exists a mapping $g\colon \mathrm{pos}(s') \to \mathrm{pos}(s)$ such that $g(\varepsilon) = \varepsilon$, $g(\Diamond_1(s')) = \Diamond_1(s)$, for all $w \in \mathrm{pos}(s')$ we have $\mathbb{t}(s', \mathbf{r}', w) = \mathbb{t}(s, \mathbf{r}, g(w))$, and for all $w_1, w_2 \in \mathrm{pos}(s')$ we have $g(w_1) \leq_L g(w_2)$ if and only if $w_1 \leq_L w_2$ and $g(w_1) \leq_P g(w_2)$ if and only if $w_1 \leq_P w_2$.*

We can use truncations to bound the size of distinguishers as follows.

▶ **Lemma 10** ([11, Lemma 5.10],[34])**.** *Let $\mathbf{p}, \mathbf{q} \in \mathbf{Q}$ be i-rivals for some $i \in \{1, \ldots, M\}$, let $u \in T_\Gamma$ be a $\mathbf{p}$-$\mathbf{q}$-reacher, let $f \in T_{\Gamma_\diamond}$ be a $\mathbf{p}$-$\mathbf{q}$-fork, let $s \in T_{\Gamma_\diamond}$ be an i-$\mathbf{p}$-$\mathbf{q}$-distinguisher, and let $\mathbf{r_p} \in \mathrm{Run}^{\diamond}_{\boldsymbol{\mathcal{U}}}(\mathbf{p}, s, \mathbf{p})$ and $\mathbf{r_q} \in \mathrm{Run}^{\diamond}_{\boldsymbol{\mathcal{U}}}(\mathbf{q}, s, \mathbf{q})$. Then there exists a $\mathbf{p}$-$\mathbf{q}$-reacher u' with* $\mathrm{height}(u') \leq |\mathbf{Q}|^2$*, a $\mathbf{p}$-$\mathbf{q}$-fork f' with* $\mathrm{height}(f') \leq 2|\mathbf{Q}|^2$*, and an i-$\mathbf{p}$-$\mathbf{q}$-distinguisher s' with* $\mathrm{height}(s') \leq 4|\mathbf{Q}|^2$ *such that for the runs $\mathbf{r}'_\mathbf{p} \in \mathrm{Run}^{\diamond}_{\boldsymbol{\mathcal{U}}}(\mathbf{p}, s', \mathbf{p})$ and $\mathbf{r}'_\mathbf{q} \in \mathrm{Run}^{\diamond}_{\boldsymbol{\mathcal{U}}}(\mathbf{q}, s', \mathbf{q})$, $(s', \mathbf{r}'_\mathbf{p})$ is a truncation of $(s, \mathbf{r_p})$ and $(s', \mathbf{r}'_\mathbf{q})$ is a truncation of $(s, \mathbf{r_q})$. In particular, for every two states $\mathbf{p}, \mathbf{q} \in \mathbf{Q}$, it is decidable whether $\mathbf{p}$ and $\mathbf{q}$ are rivals.*

3.1 Decidability

In order to show that it is decidable whether $\boldsymbol{\mathcal{U}}$ is broken, we construct a covering $\bar{\mathcal{U}}$ of $\boldsymbol{\mathcal{U}}$ with the capability to detect the broken coordinates of a run of $\boldsymbol{\mathcal{U}}$. The covering $\bar{\mathcal{U}}$ possesses the same behavior as $\boldsymbol{\mathcal{U}}$ and each run of $\bar{\mathcal{U}}$ corresponds to exactly one run of $\boldsymbol{\mathcal{U}}$. We construct $\bar{\mathcal{U}}$ by adding to each state of $\mathbf{Q}$ one entry containing all states reachable on the current subtree, one entry containing all states visited on the current run, one entry containing all pairs $(\mathbf{p}, \mathbf{q})$ of states such that $\mathbf{q}$ is reachable by a run which visited $\mathbf{p}$ at a position where our current run also visited $\mathbf{p}$, and one entry containing a record of all broken coordinates. This allows us to infer the brokenness of a run directly from the state at its root. More precisely, we construct $\bar{\mathcal{U}} = (\bar{Q}, \Gamma, \bar{\mu}, \bar{\nu})$ with states from $\bar{Q} = \mathbf{Q} \times \mathcal{P}(\mathbf{Q}) \times \mathcal{P}(\mathbf{Q}) \times \mathcal{P}(\mathbf{Q}^2) \times \{0,1\}^M$ such that if $t \in T_\Gamma$, $\bar{r} \in \mathrm{Run}_{\bar{\mathcal{U}}}(t)$, $w \in \mathrm{pos}(t)$, and $\bar{r}(w) = (\mathbf{q}, P, V, W, \bar{a})$, then the following statements hold. We have (1) the projection $\pi_\mathbf{Q}\colon \bar{Q} \to \mathbf{Q}$ to the first coordinate induces a bijection $\pi_\mathbf{Q}\colon \mathrm{Acc}_{\bar{\mathcal{U}}}(t) \to \mathrm{Acc}_{\boldsymbol{\mathcal{U}}}(t)$ preserving weights of runs, (2) $P = \{\mathbf{p} \in \mathbf{Q} \mid \mathrm{Run}_{\boldsymbol{\mathcal{U}}}(t{\restriction}_w, \mathbf{p}) \neq \emptyset\}$, (3) $V = \{\pi_\mathbf{Q} \circ \bar{r}(wv) \mid v \in \mathrm{pos}(t{\restriction}_w)\}$, (4) $W = \{(\mathbf{p}_1, \mathbf{p}_2) \in \mathbf{Q}^2 \mid \text{for some } v \in \mathrm{pos}(t{\restriction}_w) \text{ we have } \pi_\mathbf{Q} \circ \bar{r}(wv) = \mathbf{p}_1 \text{ and } \mathrm{Run}^{\diamond}_{\boldsymbol{\mathcal{U}}}(\mathbf{p}_1, t\langle \diamond \to wv \rangle{\restriction}_w, \mathbf{p}_2) \neq \emptyset\}$, and (5) $\bar{a}[i] = 1$ if and only if $(t, \pi_\mathbf{Q} \circ \bar{r}){\restriction}_w$ is i-broken.

We let $d_1, \ldots, d_D$ be an enumeration of $\Delta_{\bar{\mathcal{U}}}$ and for a tree $t \in T_\Gamma$ and a run $\bar{r} \in \mathrm{Run}_{\bar{\mathcal{U}}}(t)$, define the *transition Parikh vector* of $(t, \bar{r})$ by $\mathbb{p}(t, \bar{r}) = (|\{w \in \mathrm{pos}(t) \mid \mathbb{t}(t, \bar{r}, w) = d_1\}|, \ldots, |\{w \in \mathrm{pos}(t) \mid \mathbb{t}(t, \bar{r}, w) = d_D\}|)$, i.e., we count the number of occurrences of each transition in $(t, \bar{r})$. We can employ Parikh's Theorem [31, 15] to show that the set $\mathbb{p}(\bar{\mathcal{U}}) = \{\mathbb{p}(t, r) \mid t \in T_\Gamma, r \in \mathrm{Acc}_{\bar{\mathcal{U}}}(t)\}$ is *semilinear*, i.e., that there exist integers $k, k_1, \ldots, k_k$, vectors $\alpha^{(l)} \in \mathbb{N}^D$, and matrices $\beta^{(l)} \in \mathbb{N}^{D \times k_l}$ $(l \in \{1, \ldots, k\})$ with $\mathbb{p}(\bar{\mathcal{U}}) = \bigcup_{l=1}^{k} \{\alpha^{(l)} + \beta^{(l)} \bar{X} \mid \bar{X} = (X_1, \ldots, X_{k_l}) \in \mathbb{N}^{k_l}\}$. We note that Parikh's Theorem is effective, so we can effectively compute all of these integers, vectors, and matrices.

Next, we let $\Omega = (\bar{\mu}(d_1), \ldots, \bar{\mu}(d_D)) \in \mathbb{Q}^{M \times D}$ be a matrix containing the transition weight vectors of $d_1, \ldots, d_D$, let $\omega_1, \ldots, \omega_M$ be the rows of Ω, and let $\tilde{C} = 1 + \max\{|\omega_i \alpha^{(l)} - \omega_j \alpha^{(l)}| \mid i, j \in \{1, \ldots, M\}, l \in \{1, \ldots, k\}\}$. For $I \subseteq \{1, \ldots, M\}$, we let $D_I = \{l \in \{1, \ldots, D\} \mid d_l =$

$(\bar{q}_1, \dots, \bar{q}_m, \bar{a}, (\mathbf{q}, P, V, W, \bar{a})) \in \Delta_{\bar{\mathcal{U}}}$ such that $\bar{a}[i] = 1 \leftrightarrow i \in I\}$ be the set of all indices of transitions whose root state indicates brokenness of exactly the coordinates in I. Furthermore, for every $l \in \{1, \dots, D\}$ we let $\alpha^{(l)}[1], \dots, \alpha^{(l)}[D]$ be the entries of $\alpha^{(l)}$ and $\beta^{(l)}[1], \dots, \beta^{(l)}[D]$ be the rows of $\beta^{(l)}$. It is then elementary to show that every set $I \subseteq \{1, \dots, M\}$ is separable iff it is $\tilde{C}$-separable iff for some $l \in \{1, \dots, M\}$, the system of linear inequalities below possesses an integer solution. The satisfiability of systems of linear inequalities over the rationals with integer solutions is decidable [30][9, Theorem 3.4]. There are only finitely many such systems to consider, so it is decidable whether I is separable. To decide whether $\boldsymbol{\mathcal{U}}$ is broken, it suffices to check whether there exists a separable subset $I \subseteq \{1, \dots, M\}$.

$$(\omega_i \beta^{(l)} - \omega_j \beta^{(l)})\bar{X} \geq \omega_j \alpha^{(l)} - \omega_i \alpha^{(l)} + \tilde{C} \qquad \bar{X} \geq 0 \qquad (i \in I, j \in I^c)$$

$$\sum_{d \in D_I} \beta^{(l)}[d]\bar{X} \geq 1 - \sum_{d \in D_I} \alpha^{(l)}[d] \qquad - \sum_{I \subsetneq I' \subseteq M} \sum_{d \in D_{I'}} \beta^{(l)}[d]\bar{X} \geq \sum_{I \subsetneq I' \subseteq M} \sum_{d \in D_{I'}} \alpha^{(l)}[d]$$

3.2 Necessity

To show that $[\![\mathcal{A}]\!]$ is not finitely sequential if $\boldsymbol{\mathcal{U}}$ is broken, we assume that $[\![\mathcal{A}]\!]$ can be represented as a finite maximum of deterministic max-plus-WTA $\mathcal{A}_1, \dots, \mathcal{A}_N$ and employ Ramsey's Theorem [37] to obtain a contradiction. Although not stated explicitly, Bala's proof for words [3] most likely also involves some form of Ramsey's Theorem as his proof of $\mathcal{U}$ being broken implying $[\![\mathcal{A}]\!]$ to not be finitely sequential "deals with colorings of finite hypercubes". In our proof for tree automata, we are able to handle fork-brokenness without employing Ramsey's Theorem. This suggest that applying our approach to word automata yields a proof which is simpler than the corresponding one used in [3]. The reason for this is that our separation property considers sets of coordinates instead of the single coordinates which the A-Fork property considers. For the separable sets $I \in \mathcal{I}$ which are minimal with respect to set inclusion, we are able to prove a statement for I-C-separated pairs $(t, \mathbf{r})$ which greatly facilitates dealing with fork-brokenness and enables us to deal with split-brokenness in the first place. Namely, if $(t, \mathbf{r})$ is I-C-separated for a sufficiently large C and no subset of I is separable, then for every $\mathbf{p} \in \mathbf{r}(\mathrm{pos}(t))$, every Γ-word s with $\mathrm{height}(s) \leq 4|\mathbf{Q}|^2$ and $\mathrm{Run}^{\diamond}_{\boldsymbol{\mathcal{U}}}(\mathbf{p}, s, \mathbf{p}) \neq \emptyset$, and every two coordinates $i, j \in I$ we have $\mathbf{wt}^{\diamond}_i(\mathbf{p}, s, \mathbf{p}) = \mathbf{wt}^{\diamond}_j(\mathbf{p}, s, \mathbf{p})$.

For the proof, we choose a sufficiently large C, a set $I \in \mathcal{I}$ which is minimal with respect to set inclusion, and an I-C-separated pair $(t, \mathbf{r})$. Then we construct from $(t, \mathbf{r})$ new trees which require more than N deterministic max-plus-WTA in order to be assigned the correct weight. If $(t, \mathbf{r})$ is fork-broken, we construct trees and runs of $\boldsymbol{\mathcal{U}}$ with increasingly more alterations of forks and distinguishers. This approach is similar to the method used in [4] and [34] to deal with fork-brokenness. The challenge we face in adapting the proof from [34] to our situation is that we have to ensure that in the runs we construct, the coordinates from I dominate the other coordinates. In our constructions we may therefore only make "small" modifications to $(t, \mathbf{r})$. Our solution relies on the minimality of I and involves constructing more than the $N + 1$ trees sufficient for the proofs in [4, 34]. The case where $(t, \mathbf{r})$ is split-broken is much more complicated and is in fact the only reason we have to use the covering automaton $\boldsymbol{\mathcal{U}}$ instead of $\mathcal{U}$. As split-brokenness does not apply to words, this was not an issue in [3]. Our approach vastly extends the idea used to deal with split-brokenness in [34]. In [34], we relied on replacing the subtrees below the prefix-independent rivals of a split-broken run by other suitable trees. However, Example 5 shows that this is not possible in our scenario as these subtrees may be indispensable to ensure that broken coordinates dominate all non-broken coordinates. Here, we instead modify the subtrees present and construct trees and runs of $\mathcal{U}$. For this, we employ the properties of $\boldsymbol{\mathcal{U}}$, pumping techniques, and Ramsey's Theorem.

3.3 Sufficiency

To show that $[\![\mathcal{A}]\!]$ is finitely sequential if $\boldsymbol{\mathcal{U}}$ is not broken, we show that in this case we can construct M unambiguous max-plus-WTA which all do not satisfy the tree fork property and whose pointwise maximum is equivalent to $[\![\mathcal{A}]\!]$. By Theorem 2, we obtain a finitely sequential representation of $\mathcal{A}$ by constructing one for each of the unambiguous max-plus-WTA. We essentially construct the unambiguous automata by removing problematic runs from $\boldsymbol{\mathcal{U}}$ and then projecting to the coordinates $1, \ldots, M$. Our fundamental idea is the following. First, let ξ be the smallest difference between the weights of two coordinates of a loop in a Γ-word of height at most $4|\mathbf{Q}|^2$, i.e., for $X = \{|\mathbf{wt}^{\diamond}_i(\mathbf{p}, s, \mathbf{p}) - \mathbf{wt}^{\diamond}_j(\mathbf{q}, s, \mathbf{q})| \mid \mathbf{p}, \mathbf{q} \in \mathbf{Q}, i, j \in \{1, \ldots, M\}, s$ is a Γ-word with $\mathrm{height}(s) \leq 4|\mathbf{Q}|^2, \mathrm{Run}^{\diamond}_{\boldsymbol{\mathcal{U}}}(\mathbf{p}, s, \mathbf{p}) \neq \emptyset, \mathrm{Run}^{\diamond}_{\boldsymbol{\mathcal{U}}}(\mathbf{q}, s, \mathbf{q}) \neq \emptyset\}$, let $\xi = \min X \setminus \{0\}$. Then assume that $(t, \mathbf{r})$ is i-**p**-**q**-broken and that the maximum of $\mathrm{wt}_{\boldsymbol{\mathcal{U}}}(t, \mathbf{r})$ is in coordinate i. Furthermore, assume that in $\mathbf{r}$, some i-**p**-**q**-distinguisher s of height at most $4|\mathbf{Q}|^2$ loops N times in $\mathbf{p}$, where $N \in \mathbb{N}$ is some integer, and that $\mathbf{wt}_i(\mathbf{p}, s, \mathbf{p}) < \mathbf{wt}_i(\mathbf{q}, s, \mathbf{q})$. By removing the loops of s in $\mathbf{p}$ from $(t, \mathbf{r})$ and inserting them back as loops in $\mathbf{q}$, we increase the weight of coordinate i by $N\xi$, but leave the weights of all non-broken coordinates unchanged. Thus, coordinate i then dominates all non-broken coordinates by a margin of at least $N\xi$. As i cannot dominate all non-broken coordinates by an arbitrarily large margin, N cannot be arbitrarily large. In turn, this means that if N is sufficiently large, then $\mathrm{wt}_{\boldsymbol{\mathcal{U}}}(t, \mathbf{r})$ cannot take its maximum weight in coordinate i. This implies that the weight of coordinate i can be discarded if some distinguisher loops in both of its rivals too many times.

We employ this idea in the following way. We define the set $R = \{(i, \mathbf{p}, \mathbf{q}, s) \in \{1, \ldots, M\} \times \mathbf{Q}^2 \times T_{\Gamma_\diamond} \mid i \in \{1, \ldots, M\}$, s is an i-**p**-**q**-distinguisher, $\mathrm{height}(s) \leq 4|\mathbf{Q}|^2\}$ and define the constant $N = \lceil M\tilde{C}\xi^{-1} \rceil$, where $\tilde{C}$ is as in Section 3.1. We note that R is computable, as by Lemma 10, we can decide for every two states $\mathbf{p}, \mathbf{q} \in \mathbf{Q}$ whether they are rivals or not. Let $t \in T_\Gamma$, $\mathbf{r} \in \mathrm{Run}_{\boldsymbol{\mathcal{U}}}(t)$, $(i, \mathbf{p}, \mathbf{q}, s) \in R$, $\mathbf{r_p} \in \mathrm{Run}^{\diamond}_{\boldsymbol{\mathcal{U}}}(\mathbf{p}, s, \mathbf{p})$, and $\mathbf{r_q} \in \mathrm{Run}^{\diamond}_{\boldsymbol{\mathcal{U}}}(\mathbf{q}, s, \mathbf{q})$. We call $(t, \mathbf{r})$

- $(i, \mathbf{p}, \mathbf{q}, s)$*-fork-broken* if there exist positions $u_\mathbf{p}, v_\mathbf{p}, w_\mathbf{p}, w_\mathbf{q}, u_\mathbf{q}, v_\mathbf{q} \in \mathrm{pos}(t)$ with $v_\mathbf{q} <_{\mathrm{P}} u_\mathbf{q} \leq_{\mathrm{P}} w_\mathbf{q} <_{\mathrm{P}} w_\mathbf{p} \leq_{\mathrm{P}} v_\mathbf{p} <_{\mathrm{P}} u_\mathbf{p}$ such that $(t, \mathbf{r})\langle \diamond \to u_\mathbf{p} \rangle\!\upharpoonleft_{v_\mathbf{p}} \text{✂} (s, \mathbf{r_p})^{N+1}$, $(t, \mathbf{r})\langle \diamond \to u_\mathbf{q} \rangle\!\upharpoonleft_{v_\mathbf{q}} \text{✂} (s, \mathbf{r_q})^{N+1}$, $\mathbf{r}(w_\mathbf{p}) = \mathbf{p}$, $\mathbf{r}(w_\mathbf{q}) = \mathbf{q}$, and $t\langle \diamond \to w_\mathbf{p} \rangle\!\upharpoonleft_{w_\mathbf{q}}$ is a **p**-**q**-fork.
- $(i, \mathbf{p}, \mathbf{q}, s)$*-split-broken* if there exist positions $u_\mathbf{p}, v_\mathbf{p}, u_\mathbf{q}, v_\mathbf{q} \in \mathrm{pos}(t)$ such that $v_\mathbf{p} <_{\mathrm{P}} u_\mathbf{p}$, $v_\mathbf{q} <_{\mathrm{P}} u_\mathbf{q}$, $v_\mathbf{p}$ and $v_\mathbf{q}$ are prefix-independent, $(t, \mathbf{r})\langle \diamond \to u_\mathbf{p} \rangle\!\upharpoonleft_{v_\mathbf{p}} \text{✂} (s, \mathbf{r_p})^{N+1}$, and $(t, \mathbf{r})\langle \diamond \to u_\mathbf{q} \rangle\!\upharpoonleft_{v_\mathbf{q}} \text{✂} (s, \mathbf{r_q})^{N+1}$.

Following the idea above, we can show that if $(t, \mathbf{r})$ is $(i, \mathbf{p}, \mathbf{q}, s)$-broken, then $\mathbf{wt}_i(t, \mathbf{r}) < [\![\mathcal{A}]\!](t)$. Furthermore, we can show that for every $i \in \{1, \ldots, M\}$, we can construct a complete and deterministic FTA $\mathcal{B}_i$ over the alphabet $\Gamma \times \mathbf{Q}$ which accepts a tree $(t, \mathbf{r}) \in T_{\Gamma \times \mathbf{Q}}$ if and only if there does not exist $(i, \mathbf{p}, \mathbf{q}, s) \in R$ such that $(t, \mathbf{r})$ is $(i, \mathbf{p}, \mathbf{q}, s)$-broken. Via a product-like construction, we can employ $\mathcal{B}_i$ to construct a WTA $\mathcal{C}_i$ which "filters" the runs of $\boldsymbol{\mathcal{U}}$ so that the runs of $\mathcal{C}_i$ correspond exactly to those runs of $\boldsymbol{\mathcal{U}}$ which are not $(i, \mathbf{p}, \mathbf{q}, s)$-broken for any $(i, \mathbf{p}, \mathbf{q}, s) \in R$. Then $\mathcal{C}_i$ is not i-broken as otherwise, we can construct an accepting run r of $\mathcal{C}_i$ on a tree t whose projection to $\mathbf{Q}$ yields an i-broken pair $(t, \mathbf{r})$ which loops an i-**p**-**q**-distinguisher s for $N + 1$ times in both $\mathbf{p}$ and $\mathbf{q}$. As by Lemma 10, s can be truncated to a distinguisher of height at most $4|\mathbf{Q}|^2$, $(t, \mathbf{r})$ is thus $(i, \mathbf{p}, \mathbf{q}, s)$-broken. We obtain the contradiction that r is not accepting. Consequently, projecting the weights of each automaton $\mathcal{C}_i$ to the i-th coordinate yields M unambiguous max-plus-WTA whose pointwise-maximum coincides with $[\![\mathcal{A}]\!]$ and which all do not satisfy the tree fork property.

References

1 Athanasios Alexandrakis and Symeon Bozapalidis. Weighted grammars and Kleene's theorem. *Information Processing Letters*, 24(1):1–4, 1987.

2 Cyril Allauzen and Mehryar Mohri. Efficient algorithms for testing the twins property. *Journal of Automata, Languages and Combinatorics*, 8(2):117–144, 2003.

3 Sebastian Bala. Which finitely ambiguous automata recognize finitely sequential functions? (extended abstract). In Krishnendu Chatterjee and Jirí Sgall, editors, *38th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 8087 of *Lecture Notes in Computer Science*, pages 86–97. Springer, 2013.

4 Sebastian Bala and Artur Koniński. Unambiguous automata denoting finitely sequential functions. In Adrian-Horia Dediu, Carlos Martín-Vide, and Bianca Truthe, editors, *7th International Conference on Language and Automata Theory and Applications (LATA)*, volume 7810 of *Lecture Notes in Computer Science*, pages 104–115. Springer, 2013.

5 Jean Berstel and Christophe Reutenauer. Recognizable formal power series on trees. *Theoretical Computer Science*, 18:115–148, 1982.

6 Jean Berstel and Christophe Reutenauer. *Rational Series and Their Languages*, volume 12 of *EATCS Monographs in Theoretical Computer Science*. Springer, 1988.

7 Johanna Björklund, Frank Drewes, and Niklas Zechner. An efficient best-trees algorithm for weighted tree automata over the tropical semiring. In Adrian-Horia Dediu, Enrico Formenti, Carlos Martín-Vide, and Bianca Truthe, editors, *9th International Conference on Language and Automata Theory and Applications (LATA)*, volume 8977 of *Lecture Notes in Computer Science*, pages 97–108. Springer, 2015.

8 Meera Blattner and Tom Head. Automata that recognize intersections of free submonoids. *Information and Control*, 35(3):173–176, 1977.

9 Alexander Bockmayr, Volker Weispfenning, and Michael Maher. Solving numerical constraints. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume 1, chapter 12, pages 751–842. Elsevier and MIT Press, 2001.

10 Björn Borchardt. A pumping lemma and decidability problems for recognizable tree series. *Acta Cybernetica*, 16(4):509–544, 2004.

11 Matthias Büchse, Jonathan May, and Heiko Vogler. Determinization of weighted tree automata using factorizations. *Journal of Automata, Languages and Combinatorics*, 15(3/4):229–254, 2010.

12 Laure Daviaud, Pierre Guillon, and Glenn Merlet. Comparison of max-plus automata and joint spectral radius of tropical matrices. In Kim G. Larsen, Hans L. Bodlaender, and Jean-François Raskin, editors, *42nd International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 83 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 19:1–19:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017.

13 Manfred Droste, Werner Kuich, and Heiko Vogler, editors. *Handbook of Weighted Automata*. EATCS Monographs in Theoretical Computer Science. Springer, 2009.

14 Zoltán Ésik and Werner Kuich. Formal tree series. *Journal of Automata, Languages and Combinatorics*, 8(2):219–285, 2003.

15 Javier Esparza, Pierre Ganty, Stefan Kiefer, and Michael Luttenberger. Parikh's theorem: A simple and direct automaton construction. *Information Processing Letters*, 111(12):614–619, 2011.

16 Emmanuel Filiot, Ismaël Jecker, Nathan Lhote, Guillermo A. Pérez, and Jean-François Raskin. On delay and regret determinization of max-plus automata. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–12. IEEE Computer Society, 2017.

17 Zoltán Fülöp and Heiko Vogler. Weighted tree automata and tree transducers. In Manfred Droste, Werner Kuich, and Heiko Vogler, editors, *Handbook of Weighted Automata*, EATCS Monographs in Theoretical Computer Science, chapter 9, pages 313–403. Springer, 2009.

18 Kōsaburō Hashiguchi. Algorithms for determining relative star height and star height. *Information and Computation*, 78(2):124–169, 1988.

19 Kōsaburō Hashiguchi, Kenichi Ishiguro, and Shūji Jimbo. Decidability of the equivalence problem for finitely ambiguous finance automata. *International Journal of Algebra and Computation*, 12(3):445–461, 2002.

20 Daniel Kirsten. A Burnside approach to the termination of Mohri's algorithm for polynomially ambiguous min-plus-automata. *Informatique Théorique et Applications*, 42(3):553–581, 2008.

21 Daniel Kirsten and Sylvain Lombardy. Deciding unambiguity and sequentiality of polynomially ambiguous min-plus automata. In Susanne Albers and Jean-Yves Marion, editors, *26th International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 3 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 589–600. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2009.

22 Ines Klimann, Sylvain Lombardy, Jean Mairesse, and Christophe Prieur. Deciding unambiguity and sequentiality from a finitely ambiguous max-plus automaton. *Theoretical Computer Science*, 327(3):349–373, 2004.

23 Jan Komenda, Sébastien Lahaye, Jean-Louis Boimond, and Ton van den Boom. Max-plus algebra in the history of discrete event systems. *Annual Reviews in Control*, 45:240–249, 2018.

24 Adam Koprowski and Johannes Waldmann. Max/plus tree automata for termination of term rewriting. *Acta Cybernetica*, 19(2):357–392, 2009.

25 Stephan Kreutzer and Cristian Riveros. Quantitative monadic second-order logic. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 113–122. IEEE Computer Society, 2013.

26 Daniel Krob. The equality problem for rational series with multiplicities in the tropical semiring is undecidable. *International Journal of Algebra and Computation*, 4(3):405–426, 1994.

27 Werner Kuich and Arto Salomaa. *Semirings, Automata, Languages*, volume 5 of *EATCS Monographs in Theoretical Computer Science*. Springer, 1986.

28 Filip Mazowiecki and Cristian Riveros. Pumping lemmas for weighted automata. In Rolf Niedermeier and Brigitte Vallée, editors, *35th Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 96 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 50:1–50:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018.

29 Mehryar Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–311, 1997.

30 George L. Nemhauser and Laurence A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, 1988.

31 Rohit Jivanlal Parikh. On context-free languages. *Journal of the ACM*, 13(4):570–581, 1966.

32 Erik Paul. On finite and polynomial ambiguity of weighted tree automata. In Srečko Brlek and Christophe Reutenauer, editors, *20th International Conference on Developments in Language Theory (DLT)*, volume 9840 of *Lecture Notes in Computer Science*, pages 368–379. Springer, 2016.

33 Erik Paul. The equivalence, unambiguity and sequentiality problems of finitely ambiguous max-plus tree automata are decidable. In Kim G. Larsen, Hans L. Bodlaender, and Jean-François Raskin, editors, *42nd International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 83 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 53:1–53:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017.

34 Erik Paul. Finite sequentiality of unambiguous max-plus tree automata. In Rolf Niedermeier and Christophe Paul, editors, *36th International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 126 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 55:1–55:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019.

35 Slav Petrov. Latent variable grammars for natural language parsing. In *Coarse-to-Fine Natural Language Processing*, Theory and Applications of Natural Language Processing, chapter 2, pages 7–46. Springer, 2012.

36 Michael O. Rabin and Dana S. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3(2):114–125, 1959.

37 Frank P. Ramsey. On a problem of formal logic. *Proceedings of the London Mathematical Society*, series 2, 30:264–286, 1930.

38 Jacques Sakarovitch. *Elements of Automata Theory*. Cambridge University Press, 2009.

39 Arto Salomaa and Matti Soittola. *Automata-Theoretic Aspects of Formal Power Series*. Texts and Monographs in Computer Science. Springer, 1978.

40 Marcel-Paul Schützenberger. On the definition of a family of automata. *Information and Control*, 4(2-3):245–270, 1961.

41 Helmut Seidl. On the finite degree of ambiguity of finite tree automata. *Acta Informatica*, 26(6):527–542, 1989.

42 Imre Simon. Limited subsets of a free monoid. In *19th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 143–150. IEEE Computer Society, 1978.

43 Imre Simon. Recognizable sets with multiplicities in the tropical semiring. In Michal P. Chytil, Ladislav Janiga, and Václav Koubek, editors, *13th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 324 of *Lecture Notes in Computer Science*, pages 107–120. Springer, 1988.

44 Johannes Waldmann. Weighted automata for proving termination of string rewriting. *Journal of Automata, Languages and Combinatorics*, 12(4):545–570, 2007.

45 Andreas Weber and Helmut Seidl. On the degree of ambiguity of finite automata. *Theoretical Computer Science*, 88(2):325–349, 1991.

On Skolem-Hardness and Saturation Points in Markov Decision Processes

Jakob Piribauer
Technische Universität Dresden, Germany
jakob.piribauer@tu-dresden.de

Christel Baier
Technische Universität Dresden, Germany
christel.baier@tu-dresden.de

Abstract

The Skolem problem and the related Positivity problem for linear recurrence sequences are outstanding number-theoretic problems whose decidability has been open for many decades. In this paper, the inherent mathematical difficulty of a series of optimization problems on Markov decision processes (MDPs) is shown by a reduction from the Positivity problem to the associated decision problems which establishes that the problems are also at least as hard as the Skolem problem as an immediate consequence. The optimization problems under consideration are two non-classical variants of the stochastic shortest path problem (SSPP) in terms of expected partial or conditional accumulated weights, the optimization of the conditional value-at-risk for accumulated weights, and two problems addressing the long-run satisfaction of path properties, namely the optimization of long-run probabilities of regular co-safety properties and the model-checking problem of the logic frequency-LTL. To prove the Positivity- and hence Skolem-hardness for the latter two problems, a new auxiliary path measure, called weighted long-run frequency, is introduced and the Positivity-hardness of the corresponding decision problem is shown as an intermediate step. For the partial and conditional SSPP on MDPs with non-negative weights and for the optimization of long-run probabilities of constrained reachability properties ($a \,\mathrm{U}\, b$), solutions are known that rely on the identification of a bound on the accumulated weight or the number of consecutive visits to certain sates, called a saturation point, from which on optimal schedulers behave memorylessly. In this paper, it is shown that also the optimization of the conditional value-at-risk for the classical SSPP and of weighted long-run frequencies on MDPs with non-negative weights can be solved in pseudo-polynomial time exploiting the existence of a saturation point. As a consequence, one obtains the decidability of the qualitative model-checking problem of a frequency-LTL formula that is not included in the fragments with known solutions.

2012 ACM Subject Classification Theory of computation → Probabilistic computation; Theory of computation → Logic and verification

Keywords and phrases Markov decision process, Skolem problem, stochastic shortest path, conditional expectation, conditional value-at-risk, model checking, frequency-LTL

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.138

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

Related Version An extended version is available at `https://arxiv.org/abs/2004.11441` ([39]).

Funding The authors are supported by the DFG through the DFG grant 389792660 as part of TRR 248, the Cluster of Excellence EXC 2050/1 (CeTI, project ID 390696704, as part of Germany's Excellence Strategy), the Research Training Group QuantLA (GRK 1763), the Collaborative Research Centers CRC 912 (HAEC), the DFG-project BA-1679/11-1, and the DFG-project BA-1679/12-1.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 138; pp. 138:1–138:17
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Markov decision processes (MDPs) (see, e.g., [40]) constitute one of the most prominent classes of operational models combining randomization and non-determinism and are widely used in verification, articifical intelligence, robotics and operations research. Consequently, a vast landscape of optimization problems on MDPs has been studied. The task usually is to find a strategy resolving the non-deterministic choices, called a *scheduler*, such that a certain objective quantity is optimized or to decide whether the optimal value exceeds a given rational threshold (*threshold problem*).

Stochastic shortest path problems (SSPPs) are one important type of such optimization problems on MDPs equipped with weights. These problems ask for a scheduler maximizing or minimizing the expected accumulated weight before reaching a designated goal state. In the classical setting, only schedulers reaching the goal almost surely are taken into consideration. This *classical SSPP* is known to be solvable in polynomial time using graph-based algorithms and linear-programming techniques [10, 20, 3]. For various purposes, the requirement that the goal has to be reached almost surely, however, is not appropriate. This applies, e.g., to work on the semantics of probabilistic programs when no guarantee on almost sure termination can be given [25, 30, 9, 15, 36], to the analysis of the behavior of fault-tolerant systems in error scenarios which occur with low probability, or to the trade-off analysis when combinations of utility and cost constraints can be achieved with positive probability, but not almost surely (see, e.g., [5]). This motivates a switch to non-classical variants of the SSPP: The *conditional SSPP* [8] asks for a scheduler optimizing the conditional expected accumulated weight before reaching the goal under the condition that the goal will indeed be reached and the *partial SSPP* [16, 38] assigns weight 0 to all executions not reaching the goal. Both variants increase the algorithmic difficulties. In the special case of MDPs with non-negative weights, exponential-time algorithms for the partial and conditional SSPP exploit the monotonicity of accumulated weights and rely on the existence of a *saturation point* (a bound for the accumulated weight) from which on optimal schedulers behave memorylessly. Apart from a PSPACE lower bound and approximation algorithms [38], no algorithms are known for solving the partial or conditional SSPP in integer-weighted MDPs.

Conditional expectations also play a crucial role in risk management: The *conditional value-at-risk* is an established risk measure quanitfying the expected loss in bad cases [45, 1]. Given a probability value p, the *value-at-risk* of a random variable X is defined as the worst p-quantile. Quantile queries on the distribution of path lengths have been studied in [44]. The conditional value-at-risk is the expectation of X under the condition that the outcome is worse than the value-at-risk. For MDPs, the conditional value-at-risk has been studied for mean-payoffs and for weighted reachability where on each run only once a terminal weight is collected when a target state is reached [31]. In this paper, we consider the conditional value-at-risk for the more general accumulated weight before reaching the goal, i.e. for the classical SSPP. To the best of our knowledge, this problem has not been studied.

Other typical optimization problems arise in the context of verification, asking for worst-case schedulers that minimize or maximize the probability of a given path property. While such problems are well-understood, e.g., for properties given by linear temporal logic (LTL)-formulas or non-deterministic Büchi-automata [19], there has been increasing interest in ways to quantify the degree to which a property is satisfied not only by the probability (see [28]). Approaches in this direction include the work on robust satisfaction of temporal specifications [32, 43], coverage semantics [17], robustness distances [13], and the more general model-measurement semantics [29] among others. Furthermore, this has lead to different notions quantifying to which degree a property is satisfied in the long-run: *Frequency-LTL* has been introduced in [23, 24] as an extension of LTL by a frequency modality that makes assertions on the portion of time (or relative frequency of positions in paths) where a given event holds. While [23, 24] presents model-checking algorithms for Markov chains and arbitrary frequency-LTL formulas, the presented model checking algorithms for MDPs are restricted to fragments of frequency-LTL. We address the model checking problem for frequency-LTL formulas not contained in these fragments. Further, the concept of *long-run probabilities* [4] has been introduced for reasoning about the probabilities of path properties when the system is in equilibrium and can, e.g., be useful to formalize refined notions of long-run availability. In [4], a pseudo-polynomial time algorithm that exploits the existence of a saturation point for the computation of optimal long-run probabilities of constrained reachability properties ($a \,\mathrm{U}\, b$) is provided. Here, we study long-run probabilities of general regular co-safety properties.

Contributions. The main contribution of the paper is to provide evidence for the mathematical difficulty of the series of decision problems described above in terms of a reduction from the *Positivity problem* of linear recurrence sequences. The Positivity problem is closely related to the *Skolem problem*, a prominent number-theoretic decision problem for linear recurrence sequences, and the decidability of both problems has been open for many decades (see, e.g., [27]). As it is well-known that the Skolem problem is reducible to the Positivity problem, the provided reductions establish that the investigated decision problems are also at least as hard as the Skolem problem. In the middle column of Table 1, these Skolem-hardness results are listed:

Table 1 Overview of the results.

optimization problem on MDPs	threshold problem Positivity- and hence Skolem-hard for	exponential-time algorithm using a saturation point for
partial SSPP (1)	weights in $\mathbb{Z}$, *Thm. 3*	weights in $\mathbb{N}$ [16] (PSPACE-hard, *Prop. 15*)
conditional SSPP (2)	weights in $\mathbb{Z}$, *Thm. 5*	weights in $\mathbb{N}$ [8] (PSPACE-hard [8])
conditional value-at-risk for the classical SSPP (3)	weights in $\mathbb{Z}$, *Thm. 6*	weights in $\mathbb{N}$, *Thm. 12*
long-run probability (4)	regular co-safety properties, *Thm. 9*	constrained reachability $a \,\mathrm{U}\, b$ [4] (NP-hard [4])
model checking of frequency-LTL (5)	$\Pr^{\max}_{\mathcal{M}}(G^{>\vartheta}_{\inf}(\varphi)) = 1?$ for an LTL-formula φ, *Thm. 11*	$\Pr^{\max}_{\mathcal{M}}(G^{>\vartheta}_{\inf}(a \,\mathrm{U}\, b)) = 1?$ *Cor. 14*

To obtain these results, we construct an MDP-gadget in which a linear recurrence relation can be encoded. Together with different gadgets encoding initial values of a linear recurrence sequence, we use this gadget to establish Positivity-hardness for problems (1)-(3). Afterwards, we introduce a notion of *weighted long-run frequency* for constrained reachability properties that can be seen as a generalization of classical limit-average weights and serves here as a technical vehicle to provide a connective link to long-run probabilities and the model-checking problem of frequency-LTL. The Positivity-hardness for problems (4) and (5) is obtained via the Positivity-hardness of the threshold problem for weighted long-run frequencies by showing how to encode integer weights in terms of the satisfaction of a fixed co-safety property. The

Positivity-hardness of (4) and (5) is somehow surprising: The non-probabilistic variant (4) is shown to be decidable in [4], while our results show that Positivity-hardness of (4) holds even for a simple fixed co-safety property given by a very small counter-free non-deterministic finite automaton. Likewise, Positivity-hardness of (5) is established already for the restriction to the almost-sure satisfaction problem of a simple fixed frequency-LTL formula.

For special cases of some of the problems studied here it is known that optimal values can be computed in exponential time exploiting a saturation point. We extend this picture by showing analogous results for problems (3) and (5) (see Table 1). In particular, we provide a simple exponential time algorithm for the computation of the optimal conditional value-at-risk for the classical SSPP. Further, we pinpoint where the Positivity-hardness of the model checking problem of frequency-LTL arises: We observe that the techniques of [4] allow to solve the qualitative model-checking problem for a frequency-LTL formula with only one constrained reachability ($a\,\mathrm{U}\,b$) property under a frequency-globally modality. Our Positivity-hardness result for model checking frequency-LTL uses an only slightly more complicated fixed formula where a Boolean combination of atomic propositions and constrained reachability properties occurs in the scope of the frequency-globally modality. In particular, the Positivity-hardness does not require deeper nesting of temporal operators.

Related work. Besides the above cited work that presents algorithms for special cases of the investigated problems, closest to our work is [2] where Skolem-hardness for decision problems for Markov chains have been established. The problems are to decide whether for given states s, t and rational number p, there is a positive integer n such that the probability to reach t from s in n steps equals p and the model checking problem for a probabilistic variant of monadic logic and a variant of LTL that treats Markov chains as linear transformers of probability distributions. These decision problems are of quite different nature than the problems studied here, and so are the reductions from the Skolem problem. In this context also the results of [18] and [34] are remarkable as they show the decidability (subject to Schanuel's conjecture) of reachability problems in continuous linear dynamical systems and continuous-time MDPs, respectively, as instances of the continuous Skolem problem.

A class of problems related to SSPPs concerns the optimization of probabilities for weight-bounded reachability properties and also exhibits increasing algorithmic difficulty (for an overview see [41]): For non-negative weights, schedulers optimizing the probability for reaching a target while the accumulated weight stays below a given bound are computable in pseudo-polynomial time and the corresponding probability-threshold problem is in P for qualitative probability thresholds (">0" or "$=1$") and PSPACE-hard in the general case [44, 26]. For integer weights even in finite-state Markov chains, the probabilities for a weight-bounded reachability property can be irrational. Still, decidability for analogous problems for integer-weighted MDPs have been established for certain cases. Examples are pseudo-polynomial algorithms for qualitative threshold problems in integer-weighted MDPs [14, 12, 35, 3] or an exponential-time algorithm and a PSPACE lower bound for the almost-sure termination problem in one-counter MDPs [11].

Switching to more expressive models typically leads to the undecidability of infinite-horizon verification problems. This applies, e.g., to recursive MDPs [21], MDPs with two or more weight functions [7, 42] or partially observable MDPs [33, 6]. However, we are not aware of natural decision problems for standard (finite-state) MDPs with a single weight function and single objective that are known to be undecidable.

2 Preliminaries

We give basic definitions and present our notation (for more details see, e.g., [40]). We then formally define the quantitative objectives studied in this paper.

Notations for Markov decision processes. A *Markov decision process* (MDP) is a tuple $\mathcal{M} = (S, Act, P, s_{init}, wgt, \mathsf{AP}, L)$ where S is a finite set of states, Act a finite set of actions, $s_{init} \in S$ the initial state, $P\colon S \times Act \times S \to [0,1] \cap \mathbb{Q}$ is the transition probability function, $wgt\colon S \times Act \to \mathbb{Z}$ the weight function, AP a finite set of atomic propositions, and $L\colon S \to 2^{\mathsf{AP}}$ a labeling function. If not needed, we might drop the weight function or the labeling. We require that $\sum_{t\in S} P(s,\alpha,t) \in \{0,1\}$ for all $(s,\alpha) \in S \times Act$. We say that action α is enabled in state s iff $\sum_{t\in S} P(s,\alpha,t) = 1$. We assume that for all states s there is an enabled action and that all states are reachable from s_{init}. We call a state absorbing if there is only one enabled action, returning to the state with probability 1 and weight 0. The paths of $\mathcal{M}$ are finite or infinite sequences $s_0\,\alpha_0\,s_1\,\alpha_1 \dots$ where states and actions alternate such that $P(s_i, \alpha_i, s_{i+1}) > 0$ for all $i \geq 0$. For $\pi = s_0\,\alpha_0\,s_1\,\alpha_1\,\dots\alpha_{k-1}\,s_k$, $wgt(\pi) = wgt(s_0, \alpha_0) + \dots + wgt(s_{k-1}, \alpha_{k-1})$ denotes the accumulated weight of π, $P(\pi) = P(s_0, \alpha_0, s_1) \cdot \ldots \cdot P(s_{k-1}, \alpha_{k-1}, s_k)$ its probability, and $last(\pi) = s_k$ its last state. Further, we also write π to denote the word $L(s_0), L(s_1), \dots$. The *size* of $\mathcal{M}$ is the sum of the number of states plus the total sum of the logarithmic lengths of the non-zero probability values $P(s,\alpha,s')$ as fractions of co-prime integers and the weight values $wgt(s,\alpha)$. An end component of $\mathcal{M}$ is a strongly connected sub-MDP.

Scheduler. A *scheduler* for $\mathcal{M}$ is a function $\mathfrak{S}$ that assigns to each finite path π a probability distribution over $Act(last(\pi))$. If there is a finite set X of memory modes and a memory update function $U : S \times Act \times S \times X \to X$ such that the choice of $\mathfrak{S}$ only depends on the current state after a finite path and the memory mode obtained from updating the memory mode according to U in each step, we say that $\mathfrak{S}$ is a finite-memory scheduler. If the choice depends only on the current state, we say that $\mathfrak{S}$ is memoryless. A scheduler $\mathfrak{S}$ is called deterministic if $\mathfrak{S}(\pi)$ is a Dirac distribution for each path π. Given a scheduler $\mathfrak{S}$, $\zeta = s_0\,\alpha_0\,s_1\,\alpha_1 \dots$ is a $\mathfrak{S}$-path iff ζ is a path and $\mathfrak{S}(s_0\,\alpha_0 \dots \alpha_{k-1}\,s_k)(\alpha_k) > 0$ for all $k \geq 0$.

Probability measure. We write $\Pr^{\mathfrak{S}}_{\mathcal{M},s}$ or briefly $\Pr^{\mathfrak{S}}_{s}$ to denote the probability measure induced by $\mathfrak{S}$ and s. For details, see [40]. We will use LTL-like formulas to denote measurable sets of paths. Given a measurable set ψ of infinite paths, we define $\Pr^{\min}_{\mathcal{M},s}(\psi) = \inf_{\mathfrak{S}} \Pr^{\mathfrak{S}}_{\mathcal{M},s}(\psi)$ and $\Pr^{\max}_{\mathcal{M},s}(\psi) = \sup_{\mathfrak{S}} \Pr^{\mathfrak{S}}_{\mathcal{M},s}(\psi)$ where $\mathfrak{S}$ ranges over all schedulers for $\mathcal{M}$. For a random variable X defined on infinte paths in $\mathcal{M}$, we denote the expected value of X under the probability measure induced by a scheduler $\mathfrak{S}$ and state s by $\mathbb{E}^{\mathfrak{S}}_{\mathcal{M},s}(X)$. Furthermore, for a measurable set of paths ψ with positive probability, $\mathbb{E}^{\mathfrak{S}}_{\mathcal{M},s}(X|\psi)$ denotes the conditional expectation of X under ψ. If $s = s_{init}$, we sometimes drop the subscript s.

Partial and conditional SSPP. Let $\mathcal{M}$ be an MDP with an absorbing state *goal*. On infinite paths ζ, we define the random variable $\oplus goal(\zeta)$ to be $wgt(\zeta)$ if $\zeta \vDash \Diamond goal$, and to be 0 otherwise. The *partial expectation* $PE^{\mathfrak{S}}_{\mathcal{M},s}$ of a scheduler $\mathfrak{S}$ is defined as $\mathbb{E}^{\mathfrak{S}}_{\mathcal{M},s}(\oplus goal)$. The maximal partial expectation is $PE^{\max}_{\mathcal{M},s} = \sup_{\mathfrak{S}} PE^{\mathfrak{S}}_{\mathcal{M},s}$. The *conditional expectation* $CE^{\mathfrak{S}}_{\mathcal{M},s}$ is defined as the conditional expected value $\mathbb{E}^{\mathfrak{S}}_{\mathcal{M},s}(\oplus goal|\Diamond goal)$ for all schedulers reaching *goal* with positive probability, and the maximal conditional expectations is $CE^{\max}_{\mathcal{M},s} = \sup_{\mathfrak{S}} CE^{\mathfrak{S}}_{\mathcal{M},s}$ where $\mathfrak{S}$ ranges over all schedulers $\mathfrak{S}$ with $\Pr^{\mathfrak{S}}_{\mathcal{M},s}(\Diamond goal) > 0$. The *partial SSPP* asks for the maximal partial expectations and the *conditional SSPP* for the maximal conditional expectation. These problems were first considered in [16] and [8]. For more details see [8, 38].

Conditional value-at-risk. Given an MDP $\mathcal{M}$ with a scheduler $\mathfrak{S}$, a random variable X defined on runs of the MDP with values in $\mathbb{R}$ and a value $p \in [0,1]$, we define the *value-at-risk* as $\mathit{VaR}^{\mathfrak{S}}_{p}(X) = \sup\{r \in \mathbb{R} | \Pr^{\mathfrak{S}}_{\mathcal{M}}(X \leq r) \leq p\}$. So, the value-at-risk is the point at which the cumulative distribution function of X reaches or exceeds p. Denote $\mathit{VaR}^{\mathfrak{S}}_{p}(X)$ by v. The *conditional value-at-risk* is now the expectation of X under the condition that the outcome belongs to the p worst outcomes. Following the treatment of random variables that are not continuous in general in [31], we define the conditional value-at-risk as follows:

$$\mathit{CVaR}^{\mathfrak{S}}_{p}(X) = 1/p(\Pr^{\mathfrak{S}}_{\mathcal{M}}(X < v) \cdot \mathbb{E}^{\mathfrak{S}}_{\mathcal{M}}(X | X < v) + (p - \Pr^{\mathfrak{S}}_{\mathcal{M}}(X < v)) \cdot v).$$

Outcomes of X which are less than v are treated differently to outcomes equal to v as it is possible that the outcome v has positive probability and we only want to account exactly for the p worst outcomes. Hence, we take only $p - \Pr^{\mathfrak{S}}_{\mathcal{M}}(X < v)$ of the outcomes which are exactly v into account as well.

Threshold problems for the conditional value-at-risk in weighted MDPs have been studied in [31] for two random variables: the mean-payoff and weighted reachability where a set of final states is equipped with terminal weights obtained when reaching these states while all other transitions have weight 0. In this paper, we will address the conditional value-at-risk for the accumulated weight before reaching *goal* in MDPs with an absorbing state *goal*: Define $\Diamond\!\!\!\!+\,\mathit{goal}(\zeta)$ to be $\mathit{wgt}(\zeta)$ if $\zeta \vDash \Diamond \mathit{goal}$ and leave it undefined otherwise. The optimization of the expectation of $\Diamond\!\!\!\!+\,\mathit{goal}$ is known as the *classical SSPP*. Note that the expectation of this random variable is only defined under schedulers reaching *goal* with probability 1.

Long-run probability. Let $\mathcal{M}$ be an MDP with states labeled by atomic propositions from AP. Let φ be a path property, i.e., a measurable set of paths. The *long-run probability* for φ of a path ζ under a scheduler $\mathfrak{S}$ is $\mathit{lrp}^{\mathfrak{S}}_{\varphi}(\zeta) = \liminf_{n\to\infty} \frac{1}{n+1} \cdot \sum_{i=0}^{n} \Pr^{\mathfrak{S}\uparrow\zeta[0\ldots i]}_{\mathcal{M},\zeta[i]}(\varphi)$. Here, $\zeta[0\ldots i]$ denotes the prefix from position 0 to i of ζ, $\zeta[i]$ denotes the state after i steps, and $\mathfrak{S}\uparrow\zeta[0\ldots i]$ denotes the residual scheduler defined by $\mathfrak{S}\uparrow\zeta[0\ldots i](\pi) = \mathfrak{S}(\zeta[0\ldots i] \circ \pi)$ for all finite paths π starting in $\zeta[i]$. The long-run probability of φ under scheduler $\mathfrak{S}$ is $\mathbb{LP}^{\mathfrak{S}}_{\mathcal{M}}(\varphi) = \mathbb{E}^{\mathfrak{S}}_{\mathcal{M}}(\mathit{lrp}^{\mathfrak{S}}_{\varphi})$. The maximal long-run probability for φ is $\mathbb{LP}^{\max}_{\mathcal{M}}(\varphi) = \sup_{\mathfrak{S}} \mathbb{E}^{\mathfrak{S}}_{\mathcal{M}}(\mathit{lrp}^{\mathfrak{S}}_{\varphi})$. This notion was introduced in [4]. In this paper, we are interested in two kinds of path properties: Constrained reachability, $a \, \mathrm{U} \, b$, where a and b are atomic propositions and the more general regular co-safety properties given by a finite non-deterministic automaton (NFA) $\mathcal{A}$ accepting "good" prefixes of a run. For a co-safety property given by an NFA $\mathcal{A}$, we also write $\mathbb{LP}^{\max}_{\mathcal{M}}(\mathcal{A})$.

3 Skolem-hardness

The *Skolem problem* and the closely related *Positivity problem* are outstanding problems in the fields of number theory and theoretical computer science (see, e.g., [27, 37]). Their decidability has been open for many decades. We call a problem to which the Skolem problem is reducible *Skolem-hard*. This is a hardness result in the sense that a decision procedure would imply a major breakthrough by settling the decidability of the Skolem problem and it shows that a problem possesses an inherent mathematical difficulty.

Skolem problem. Given a natural number $k \geq 2$, and rationals α_i and β_j with $1 \leq i \leq k$ and $0 \leq j \leq k-1$, let $(u_n)_{n\geq 0}$ be defined by the initial values $u_0 = \beta_0, \ldots, u_{k-1} = \beta_{k-1}$ and the linear recurrence relation $u_{n+k} = \alpha_1 u_{n+k-1} + \cdots + \alpha_k u_n$ for all $n \geq 0$. The Skolem problem is to decide whether there is an $n \in \mathbb{N}$ with $u_n = 0$.

A closely related problem is the *Positivity problem.* It asks whether $u_n \geq 0$ for all n. It is folklore that the Skolem problem is polynomial-time reducible to the positivity problem (see, e.g., [22]). We will use the Positivity problem for our reductions leading to the main result:

▶ **Main result** (Theorems 3, 5, 6, 9, 11). *The Positivity problem and hence the Skolem problem are polynomial-time reducible to the threshold problems for the partial and conditional SSPP, the conditional value-at-risk in the classical SSPP, and long-run probabilities of regular co-safety properties, as well as to the qualitative model checking problem of frequency-LTL.*

For this purpose, we will construct an MDP gadget depicted in Figure 1a that encodes a linear recurrence relation in terms of the optimal values of different quantitative objectives. For the different problems, we then provide gadgets encoding the initial values of a linear recurrence sequence. We can plug these gadgets together to obtain an MDP and a scheduler $\mathfrak{S}$ such that $\mathfrak{S}$ maximizes the respective objective iff the linear recurrence sequence has no negative member. By computing the optimal values under $\mathfrak{S}$ in the MDPs – which turn out to be rational – we provide reductions from the positivity problem to the respective threshold problems with strict inequality (see also Remark 4).

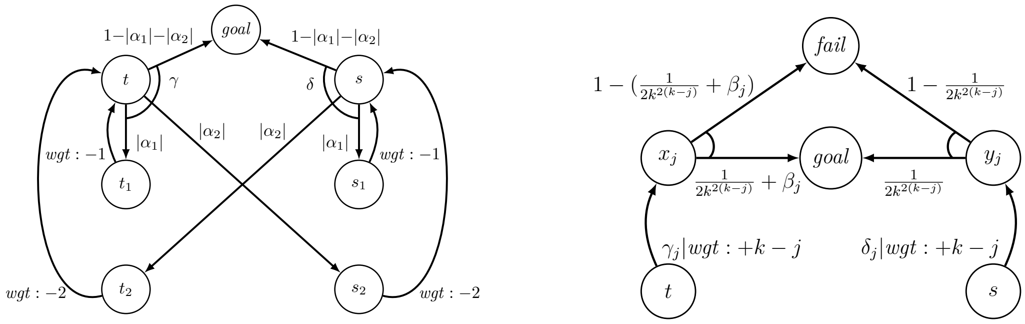
3.1 Partial and Conditional SSPP

Given a linear recurrence sequence, we construct an MDP in which the sequence is encoded in terms of optimal partial expectations. So let k be a natural number and let $(u_n)_{n\geq 0}$ be the linear recurrence sequence given by rationals α_i for $1 \leq i \leq k$ and β_j for $0 \leq j \leq k-1$ as above. As $u_{n+k} = \alpha_1 u_{n+k-1} + \cdots + \alpha_k u_n$ for all n, we see that for any positive $\lambda \in \mathbb{Q}$ the sequence $(v_n)_{n\geq 0}$ defined by $v_n = \lambda^{n+1} u_n$ satisfies $v_{n+k} = \lambda^1 \alpha_1 v_{n+k-1} + \cdots + \lambda^k \alpha_k v_n$ for all n. Furthermore, v_n is non-negative if and only if u_n is. W.l.o.g., we hence can assume that $\sum_i |\alpha_i| < \frac{1}{4}$ and that $0 \leq \beta_j < \frac{1}{4k^{2k+2}}$ for all j (for details, see the extended version [39]).

Now, we construct an MDP-gadget with an example depicted in Figure 1a. This gadget contains states *goal*, s, and t, as well as $s_1, \dots, s_k$ and $t_1, \dots, t_k$. In state t, an action γ is enabled which has weight 0 and leads to state t_i with probability α_i if $\alpha_i > 0$ and to state s_i with probability $|\alpha_i|$ if $\alpha_i < 0$ for all i. The remaining probability leads to *goal*. From each state t_i, there is an action leading to t with weight $-i$. The action δ enabled in s as well as the actions leading from states s_i to s are constructed in the same way. This gadget will be integrated into a larger MDP where there are no other outgoing edges from states $s_1, \dots, s_k, t_1, \dots, t_k$. Now, for each state q and each integer w, let $e(q, w)$ be the optimal partial expectation when starting in state q with accumulated weight w. Further, let $d(w) = e(t, w) - e(s, w)$. The simple proof of the following lemma can be found in [39] and uses that optimal partial expectations satisfy that $e(q, w) = \sum_r P(q, \alpha, r) e(r, w+wgt(q, \alpha))$ if an optimal scheduler chooses action α in state q when the accumulated weight is w.

▶ **Lemma 1.** *Let $w \in \mathbb{Z}$. If an optimal scheduler chooses action γ in t and δ in s if the accumulated weight is w, then $d(w) = \alpha_1 d(w-1) + \cdots + \alpha_k d(w-k)$.*

Now we construct a gadget that encodes the initial values $\beta_0, \dots, \beta_{k-1}$. The gadget is depicted in Figure 1b and contains states t, s, *goal*, and *fail*. For each $0 \leq j \leq k-1$, it additionally contains states x_j and y_j. In state x_j, there is one action enabled that leads to *goal* with probability $\frac{1}{2k^{2(k-j)}} + \beta_j$ and to *fail* otherwise. From state y_j, *goal* is reached with probability $\frac{1}{2k^{2(k-j)}}$ and *fail* otherwise. In state t, there is an action γ_j leading to x_j with weight $+k-j$ for each $0 \leq j \leq k-1$. Likewise, in state s there is an action δ_j leading to y_j with weight $k-j$ for each $0 \leq j \leq k-1$. We now glue together the two gadgets at states s, t, and *goal*. The cumbersome choices of probability values lead to the following lemma via straight-forward computations presented in [39].



(a) In the depicted example, the recurrence depth is 2, $\alpha_1 > 0$, and $\alpha_2 < 0$.

(b) The gadget contains the depicted states and actions for each $0 \le j \le k-1$.

Figure 1 The gadget (a) encoding the linear recurrence relation in all reductions and (b) encoding the intial values in the reduction to the partial SSPP.

▶ **Lemma 2.** *Let $0 \le j \le k-1$. Starting with weight $-(k-1)+j$ in state t or s, action γ_j and δ_j maximize the partial expectation. For positive starting weight, γ and δ are optimal.*

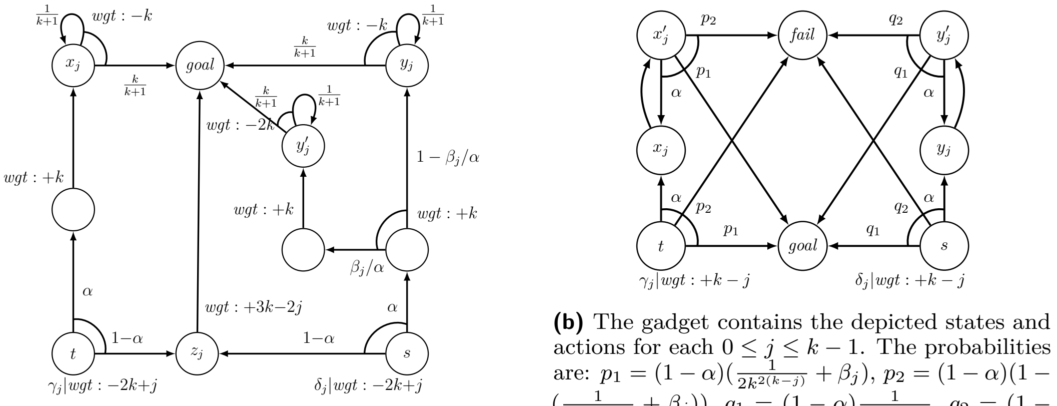
Comparing action γ_j and δ_j for starting weight $-(k-1)+j$, we conclude that the difference between optimal values $d(-(k-1)+j)$ is equal to β_j, for $0 \le j \le k-1$, and hence $d(-(k-1)+n) = u_n$ for all n. Finally, we equip the MDP with a simple initial gadget (see [39]): From the initial state s_{init}, one action with weight $+1$ is enabled. This action leads to a state c with probability $\frac{1}{2}$ and loops back to s_{init} with probability $\frac{1}{2}$. In c, the decision between action τ leading to state t and action σ leading to state s has to be made. So for any $n > 0$, state c is reached with accumulated weight n with positive probability. An optimal scheduler now has to decide whether the partial expectation when starting with weight n is better in state s or t: Action τ is optimal in c for accumulated weight w if and only if $d(w) \ge 0$. Further, the scheduler $\mathfrak{S}$ always choosing τ in c and actions $\gamma, \gamma_0, \dots, \gamma_{k-1}, \delta, \dots$ as described in Lemma 2 is optimal iff the given linear recurrence sequence is non-negative. We can compute the partial expectation of scheduler $\mathfrak{S}$ in the constructed MDP. The partial expectation turns out to be a rational. Hence, using this partial expectation as the threshold ϑ, we obtain the first main result. The technical proof computing the value of $\mathfrak{S}$ in the constructed MDP is given in the extended version [39].

▶ **Theorem 3.** *The Positivity problem is polynomial-time reducible to the following problem: Given an MDP $\mathcal{M}$ and a rational ϑ, decide whether $PE^{\max}_{\mathcal{M}} > \vartheta$.*

▶ **Remark 4.** There is no obvious way to adjust the construction such that the Skolem-hardness of the question whether $PE^{\max}_{\mathcal{M}} \ge \vartheta$ would follow. One attempt would be to provide an ε such that $PE^{\max}_{\mathcal{M}} > \vartheta$ iff $PE^{\max}_{\mathcal{M}} \ge \vartheta + \varepsilon$. This, however, probably requires a bound on the position at which the given linear recurrence sequence first becomes negative. But this question lies at the core of the positivity and the Skolem problem. All Skolem-hardness results in this paper hence concern only threshold problems with strict inequality.

The Skolem-hardness of the threshold problem for the conditional SSPP is obtained by a simple reduction showing that the threshold problems of the partial SSPP is polynomial-time reducible to the threshold problem of the conditional SSPP (see [39]).

▶ **Theorem 5.** *The Positivity problem is reducible in polynomial time to the following problem: Given an MDP $\mathcal{M}$ and a rational ϑ, decide whether $CE^{\max}_{\mathcal{M}} > \vartheta$.*



(a) The gadget contains the depicted states and actions for each $0 \le j \le k-1$. $\alpha = \sum_{1=i}^{k} |\alpha_i|$.

(b) The gadget contains the depicted states and actions for each $0 \le j \le k-1$. The probabilities are: $p_1 = (1-\alpha)(\frac{1}{2k^{2(k-j)}} + \beta_j)$, $p_2 = (1-\alpha)(1-(\frac{1}{2k^{2(k-j)}} + \beta_j))$, $q_1 = (1-\alpha)\frac{1}{2k^{2(k-j)}}$, $q_2 = (1-\alpha)(1-\frac{1}{2k^{2(k-j)}})$. All actions except for γ_j and δ_j have weight 0.

Figure 2 The gadgets encoding initial values for (a) the conditional value-at-risk for the classical SSPP and (b) weighted long-run frequencies.

3.2 Conditional value-at-risk for the classical SSPP

We reuse the gadget depicted in Figure 1a to prove the following result:

▶ **Theorem 6.** *The Positivity problem is polynomial-time reducible to the following problem: Given an MDP $\mathcal{M}$ and rationals ϑ and $p \in (0,1)$, decide whether $CVaR_p^{\max}(\diamondplus goal) > \vartheta$.*

We begin by the following consideration: Given an MDP $\mathcal{M}$ with initial state s_{init}, we construct a new MDP $\mathcal{N}$. We add a new initial state s'_{init}. In s'_{init}, there is only one action with weight 0 enabled leading to s_{init} with probability $\frac{1}{3}$ and to $goal$ with probability $\frac{2}{3}$. So, at least two thirds of the paths accumulate weight 0 before reaching the goal. Hence, we can already say that $VaR^{\mathfrak{S}}_{1/2}(\diamondplus goal) = 0$ in $\mathcal{N}$ under any scheduler $\mathfrak{S}$. Note that schedulers for $\mathcal{M}$ can be seen as schedulers for $\mathcal{N}$ and vice versa. This considerably simplifies the computation of the conditional value-at-risk in $\mathcal{N}$. Define the random variable $\diamondminus goal(\zeta)$ to be $\diamondplus goal(\zeta)$ if $\diamondplus goal \le 0$ and to be 0 otherwise. Now, the conditional value-at-risk for the probability value 1/2 under a scheduler $\mathfrak{S}$ in $\mathcal{N}$ is given by $CVaR^{\mathfrak{S}}_{1/2}(\diamondplus goal) = 2 \cdot \mathbb{E}^{\mathfrak{S}}_{\mathcal{N},s_{init}}(\diamondminus goal) = \frac{2}{3} \cdot \mathbb{E}^{\mathfrak{S}}_{\mathcal{M},s_{init}}(\diamondminus goal)$. So, the result follows from the following lemma:

▶ **Lemma 7.** *The Positivity problem is polynomial-time reducible to the following problem: Given an MDP $\mathcal{M}$ and a rational ϑ, decide whether $\mathbb{E}^{\max}_{\mathcal{M},s_{init}}(\diamondminus goal) > \vartheta$.*

We adjust the MDP used for the Skolem-hardness proof for the partial SSPP. So, let k be a natural number, $\alpha_1, \dots, \alpha_k$ be rational coefficients of a linear recurrence sequence, and $\beta_0, \dots, \beta_{k-1} \ge 0$ the rational initial values. W.l.o.g. we again assume these values to be small, namely: $\sum_{1\le i\le k} |\alpha_i| \le \frac{1}{5(k+1)}$ and for all j, $\beta_j \le \frac{1}{3}\alpha$ where $\alpha = \sum_{1\le i\le k} |\alpha_i|$.

The first important observation is that the optimal expectation of $\diamondminus goal$ for different starting states and starting weights behaves very similar to optimal partial expectations: For each state q and each integer w, let $e(q,w)$ be the optimal expectation of $\diamondminus goal$ when starting in state q with accumulated weight w. If an optimal scheduler chooses α when in q with accumulated weight w, then $e(q,w) = \sum_{r\in S} P(q,\alpha,r) \cdot e(r, w{+}wgt(q,\alpha))$. Reusing the MDP-gadget depicted in 1a, we observe that if we again let $d(w) = e(t,w) - e(s,w)$, the following holds as before: For any $w \in \mathbb{Z}$, if an optimal scheduler chooses action γ in t and δ in s if the accumulated weight is w, then $d(w) = \alpha_1 d(w-1) + \dots + \alpha_k d(w-k)$.

Now, we construct a new gadget that encodes the initial values of a linear recurrence sequence. The new gadget is depicted in Figure 2a. Besides the actions γ_j and δ_j for $0 \leq j \leq k-1$ there are no non-deterministic choices. Again, we glue together the two gadgets in states s, t, and *goal*. The main idea is that for non-negative starting weights in state s or t actions γ_j and δ_j lead to a larger expected tail loss than actions γ and δ. For $0 \leq j \leq k{-}1$ and an accumulated weight $-k{+}j$ in state t or s, the actions γ_j and δ_j are, however, optimal for maximizing the expectation of ⟠*goal* sinve the goal is reached with non-negative weights with high probability under these actions (details in [39]). The difference of optimal values satisfies $e(t, -k+j) - e(s, -k+j) = \beta_j$ for $0 \leq j \leq k{-}1$ again. Finally, we add the same initial component as in the previous section and see that the scheduler $\mathfrak{S}$ always choosing τ in state c is optimal iff the linear recurrence sequence stays non-negative. As the expectation of ⟠*goal* under $\mathfrak{S}$ is again a rational (see [39]), this finishes the proof analogously to the previous section.

3.3 Long-run probability and frequency-LTL

In order to transfer the Skolem-hardness results to long-run probabilities and frequency-LTL, we introduce the auxiliary notion of *weighted long-run frequency*. Let $\mathcal{M}$ be an MDP with a weight function $\mathit{wgt} : S \times \mathit{Act} \to \mathbb{Z}$ and two disjoint sets of states $\mathit{Goal}, \mathit{Fail} \subseteq S$. On an infinite paths $\pi = s_0, \alpha_0, s_1, \dots$, we define the random variable wlf as follows:

$$\mathit{wlf}(\pi) = \liminf_{n \to \infty} \frac{1}{n+1} \sum\nolimits_{i=0}^{n} \mathit{wgt}(s_i, \alpha_i) \cdot \mathbb{1}_{\pi[i\dots] \models \neg \mathit{Fail} \,\mathrm{U}\, \mathit{Goal}}$$

where $\mathbb{1}_{\pi[i\dots] \models \neg \mathit{Fail} \,\mathrm{U}\, \mathit{Goal}}$ is 1 if the suffix $\pi[i\dots] = s_i, \alpha_i, s_{i+1}, \dots$ satisfies $\neg \mathit{Fail} \,\mathrm{U}\, \mathit{Goal}$, and 0 otherwise. Given a scheduler $\mathfrak{S}$, we define the weighted long-run frequency $\mathit{WLF}^{\mathfrak{S}}_{\mathcal{M}} = \mathbb{E}^{\mathfrak{S}}_{\mathcal{M}}(\mathit{wlf})$ and $\mathit{WLF}^{\max}_{\mathcal{M}} = \sup_{\mathfrak{S}} \mathit{WLF}^{\mathfrak{S}}_{\mathcal{M}}$. This can be seen as a long-run average version of partial expectations. Weights are only received if afterwards *Goal* is visited before *Fail* and we measure the average weight received per step according to this rule. Note that we only consider the path property $\neg \mathit{Fail} \,\mathrm{U}\, \mathit{Goal}$ in this paper and hence do not include this property in our notation and terminology. An illustrating example can be found in [39].

We modifiy the MDP that was constructed in Section 3.1 for the Skolem-hardness of the partial SSPP. We replace the gadget encoding the initial values with the gadget depicted in Figure 2b. This gadget differs from the gadget used for partial expectations only in the expected time it takes to reach *goal* or *fail* under γ_j or δ_j. It is constructed in a way such that the expected time to reach *goal* or *fail* from s_{init} does not depend on the scheduler. Finally, we add a transition leading back to the initial state from *goal* and *fail*. An optimal scheduler for weighted long-run frequencies in the constructed MDP $\mathcal{K}$ now just has to maximize the partial expectation leading to the Skolem-hardness result (for more details see [39]).

▶ **Theorem 8.** *The Positivity problem is polynomial-time reducible to the following problem: Given an MDP $\mathcal{M}$ and a rational ϑ, decide whether $\mathit{WLF}^{\max}_{\mathcal{M}} > \vartheta$.*

This result now serves as a tool to establish analogous results for long-run probabilities. The key idea is to encode integer weights via a labelling of states and to use a simple regular co-safety property to mimic the reception of weights in weighted long-run frequencies.

▶ **Theorem 9.** *The Positivity problem is polynomial-time reducible to the following problem: Given an MDP $\mathcal{M}$, an NFA $\mathcal{A}$, and a rational ϑ, decide whether $\mathbb{LP}^{\max}_{\mathcal{M}}(\mathcal{A}) > \vartheta$.*

In the sequel, we consider weighted states instead of weighted state-action pairs. Further, we assume that the weights are only -1, 0, and $+1$. This assumption leads to a pseudo-polynomial blow-up in the general case. The weights in the MDP $\mathcal{K}$ constructed for Theorem

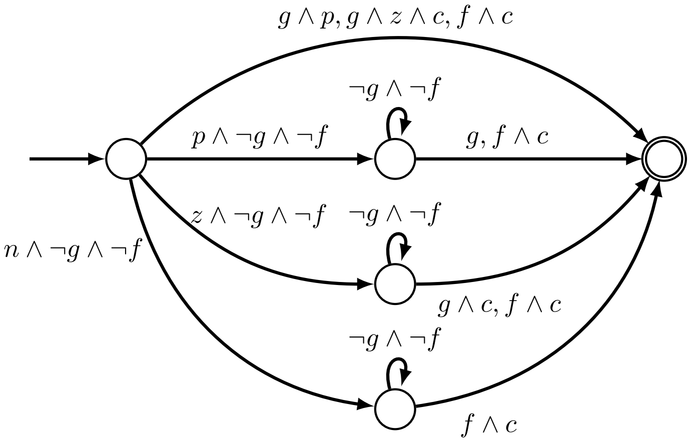

$g \wedge p, g \wedge z \wedge c, f \wedge c$
$\neg g \wedge \neg f$
$p \wedge \neg g \wedge \neg f$
$g, f \wedge c$
$z \wedge \neg g \wedge \neg f$
$\neg g \wedge \neg f$
$n \wedge \neg g \wedge \neg f$
$g \wedge c, f \wedge c$
$\neg g \wedge \neg f$
$f \wedge c$

Figure 3 The NFA $\mathcal{A}$ expressing a property of the form $d \vee \bigvee_{i=1}^{3}(c_i \wedge (a \,\mathrm{U}\, b_i))$.

8 above are, however, at most k. As the MDP has more than $2k$ states, transforming $\mathcal{K}$ to weights -1, 0, and $+1$ only leads to a polynomial blow-up. As this MDP has no non-trivial end-components, $\{goal, fail\}$ is visited infinitely often with probability 1 under any scheduler. Let $\mathsf{AP} = \{n, z, p, c, g, f\}$ be a set of atomic propositions representing *negative* (-1), *zero* (0), and *positive* $(+1)$ weight, *coin flip*, *goal*, and *fail*, respectively. We construct an MDP $\mathcal{L}$: The states *goal* and *fail* are duplicated while one copy of each is labeled with c and whenever *goal* or *fail* are entered in the MDP $\mathcal{K}$, both of the two copies in $\mathcal{L}$ are equally likely. For a formal definition see [39]. In Figure 3, we depict the NFA $\mathcal{A}$ used for the encoding. The NFA $\mathcal{A}$ is constructed such that in $\mathcal{L}$ any run starting in a state labeled *zero* or reaching *fail* before *goal* is accepted with probability 1/2 due to the *coin flips*. A run starting in a state labeled *positive* and reaching *goal* before *fail* is accepted while such a path starting in a state labeled *negative* is not. This leads to the following lemma that proves Theorem 9.

▶ **Lemma 10.** *For the MDPs $\mathcal{K}$ and $\mathcal{L}$ constructed above, we have $WLF_{\mathcal{K}}^{\max} = \frac{1}{2} + \frac{1}{2}\mathbb{LP}_{\mathcal{L}}^{\max}(\mathcal{A})$.*

Proof sketch. It is quite easy to see that the claim holds for finite-memory schedulers as we can rely on steady state probabilities in the resulting Markov chain. That the supremum over all schedulers agrees with the supremum over finite-memory schedulers on both sides follows from Fatou's lemma. Details can be found in [39]. ◀

A consequence of this result is that model checking of frequency-LTL in MDPs is at least as hard as the Skolem problem. The decidability of the model-checking problem for the full logic frequency-LTL has been left open, but set as a goal in [23, 24]. Obtaining this goal by proving the decidability of the model-checking problem hence would settle the decidability of the Skolem problem. The frequency-globally modality $G_{\inf}^{>\vartheta}(\varphi)$ is defined to hold on a path π iff $\liminf_{n\to\infty} \frac{1}{n+1} \sum_{i=0}^{n} \mathbb{1}_{\pi[i\ldots]\models\varphi} > \vartheta$, i.e. iff the long-run average number of positions at which a suffix satisfying φ starts exceeds ϑ.

▶ **Theorem 11.** *There is a polynomial-time reduction from the Positivity problem to the following qualitative model checking problem for frequency LTL for a fixed LTL-formula φ: Given an MDP $\mathcal{M}$ and a rational ϑ, is $\Pr_{\mathcal{M}}^{\max}(G_{\inf}^{>\vartheta}(\varphi)) = 1$?*

Proof sketch. The proof uses the reduction to the threshold problem for the long-run probability of the co-safety property expressed by $\mathcal{A}$. This property is captured by a simple LTL-formula φ (see Figure 3). For finite-memory schedulers $\mathfrak{S}$ inducing a single bottom strongly connected component, we see that $G^{>\vartheta}_{\inf}(\varphi)$ holds with probability 1 iff the expected long-run probability of φ is greater than ϑ. That it is enough to consider such schedulers follows from the argument using Fatou's lemma again. For more details see [39]. ◀

4 Saturation points

Despite the inherent mathematical difficulty shown by the Skolem-hardness results so far, all of the problems studied here are solvable in exponential time under a natural restriction. For the problems on weighted MDPs, this restriction only allows non-negative weights while for the long-run notions the restriction to constrained reachability properties ($a\,\mathrm{U}\,b$) leads to solvability. For the partial and the conditional SSPP [8, 16] and for long-run probabilities [4], the computability of optimal values under these restrictions has been shown. The algorithms exploit the existence of *saturation points*, a bound on the accumulated weight or the consecutive visits to certain states before optimal schedulers can behave memorylessly. We will extend this picture by providing a simple saturation point for the computation of the optimal conditional value-at-risk for the classical SSPP in MDPs with non-negative weights. Afterwards, we transfer the saturation-point algorithm from [4] to weighted long-run frequencies in the setting of non-negative weights. As a consequence, we obtain an exponential-time algorithm for the qualitative model-checking problem of a frequency-LTL formula for which no solutions were known. To conclude the section, we provide accompanying PSPACE lower bounds for the partial SSPP and weighted long-run frequencies with non-negative weights.

4.1 Conditional value-at-risk for the classical SSPP

Let $\mathcal{M}$ be an MDP with non-negative weights. In the classical SSPP, it is decidable in polynomial time whether the optimal expected accumulated weight before reaching the goal is bounded. If this is the case, the usual preprocessing step removes end components [20, 3] and transforms the MDP such that exactly the schedulers reaching the goal with probability 1 can be mimicked in the transformed MDP. So in the sequel, we assume that the absorbing state *goal* forms the only end component. Given a rational probability value $p \in (0,1)$, we are interested in the value $CVaR^{\max}_p(\oplus goal)$. Note that in our formulation the worst outcomes are the paths with the lowest accumulated weight before reaching the goal. Below we will sketch how to treat the case where high outcomes are considered bad.

▶ **Theorem 12.** *Given an MDP $\mathcal{M} = (S, s_{init}, Act, P, wgt, goal)$ with non-negative weights and no end-components except for one absorbing state goal as well as a rational probability value $p \in (0,1)$, the value $CVaR^{\max}_p(\oplus goal)$ is computable in pseudo-polynomial time.*

Proof sketch. As there are no end components, we can provide a *saturation point* $K \in \mathbb{N}$ such that paths accumulate a weight of more than K with probability less than $1-p$. Then, paths reaching an accumulated weight of K do not belong to the worst p outcomes. We construct an MDP with the state space $S \times \{0, \ldots, K\}$ that encodes the accumulated weight of a path up to K. Letting states of the form $(goal, i)$ be terminal with weight i and of the form (s, K) be terminal with weight K, we can then rely on the algorithm computing the conditional value-at-risk for weighted reachability in [31]. As K can be chosen of pseudo-polynomial size and this algorithm runs in time polynomial in the size of the constructed MDP, this leads to a pseudo-polynomial time algorithm. For details see [39]. ◀

Note that the behavior of a scheduler on paths with accumulated weight above K does not matter at all for the conditional value-at-risk. If we want to consider the case where long paths are considered as bad, we can multiply all weights by -1 and use the definitions as before. The idea here now is to compute a saturation point $-K$ such that the probability for a path to accumulate weight less than $-K$ is smaller than p. So, we know that a path with weight less than $-K$ belongs to the p worst paths. On these paths, the best thing to do in order to maximize the conditional value-at-risk is to maximize the expected accumulated weight before reaching the goal. This can be done by a memoryless deterministic scheduler simultaneously for all states and the values are computable in polynomial time [20]. Then we construct the MDP $\mathcal{N}$ as above but change the terminal weights as follows: states of the form $(goal, i)$ get weight $-i$ and states of the form (s, K) get weight $-K + \mathbb{E}^{\max}_{\mathcal{M},s}(\oplus goal)$ where $\mathcal{M}$ is the MDP in which all weights are already multiplied by -1. Afterwards the problem can be solved by the techniques for weighted reachability from [31] again.

4.2 Weighted long-run frequencies and frequency-LTL

The existence of a saturation point for long-run probabilities of constrained reachability properties was shown in [4]. This result can easily be adapted to weighted long-run frequencies following the same arguments. First, it is shown by an application of Fatou's lemma that optimal weighted long-run frequency can be approximated by finite-memory schedulers. Afterwards, it is shown that the memory needed for the optimization can be restricted further: A saturation point $K \in \mathbb{N}$ is provided such that only scheduler keeping track of the accumulated weight up to K have to be considered. The adaptions necessary to the proof in [4] are worked out in the extended version [39] and lead to the following result:

▶ **Theorem 13.** *The maximal value $WLF^{\max}_{\mathcal{M}}$ in an MDP $\mathcal{M}$ with non-negative weights is computable in pseudo-polynomial time.*

▶ **Corollary 14.** *Given an MDP $\mathcal{M}$ and a rational ϑ, it can be checked in pseudo-polynomial time whether $\Pr^{\max}_{\mathcal{M}}(G^{>\vartheta}_{\inf}(a \,\mathrm{U}\, b)) = 1$.*

Proof. The semantics of $G^{>\vartheta}_{\inf}(\neg Fail \,\mathrm{U}\, Goal)$ on a path π agree with the semantics of $wlf(\pi) > \vartheta$ if all weights are set to $+1$. Now, we can check for each end component $\mathcal{E}$ of $\mathcal{M}$ whether $WLF^{\max}_{\mathcal{E}} > \vartheta$. If that is the case, there is a finite memory scheduler for $\mathcal{E}$ inducing only one BSCC achieving a weighted long-run frequency greater than ϑ. Under this scheduler almost all paths π satisfy $wlf(\pi) > \vartheta$. Afterwards, it remains to check whether end components with such a scheduler can be reached with probability 1 in $\mathcal{M}$. ◀

In [24], the fragment of frequency-LTL in which no until-operators occur in the scope of a globally operator has been studied. The formula in the corollary is hence of the simplest form of frequency-LTL formulas for which no solution to the qualitative model-checking problem has been known. Remarkably, the formula used in the Skolem-hardness proof (Theorem 11) is only slightly more complicated as it contains a Boolean combination of constrained reachability properties and atomic propositions under the frequency-globally operator.

4.3 PSPACE lower bounds

For the conditional SSPP with non-negative weights [8] and the long-run probability of constrained reachability properties [4], PSPACE and NP lower bounds, respectively, are known indicating that the pseudo-polynomial time algorithms for the computation can probably not be significantly improved. The threshold problem of the conditional SSPP is already PSPACE-hard in acyclic MDPs with non-negative weights as shown in [8]. In [38], it has been shown that the threshold problem of the conditional SSPP is polynomial-time reducible to the threshold problem for the partial SSPP. This reduction generates an MDP with negative weights, even when all weights in the original MDP are non-negative. Here, we provide a new polynomial reduction for acyclic MDPs from the threshold problem for the conditional SSPP to the threshold problem of the partial SSPP that preserves the non-negativity of weights (see [39]).

▶ **Proposition 15.** *The threshold problem of the partial SSPP is PSPACE-hard in acyclic MDPs with non-negative weights. It is contained in PSPACE for acyclic MDPs with arbitrary integer weights.*

In an acyclic MDP, we can add intermediate states on transitions such that all paths have the same length ℓ. If we additionally add transitions form *goal* and *fail* back to the initial state, the maximal weighted long-run frequency is just the maximal partial expectation divided by ℓ. This allows us to conclude:

▶ **Proposition 16.** *The threshold problem for weighted long-run frequencies, "Does $WLF^{\max}_{\mathcal{M}} \bowtie \vartheta$ hold?", in MDPs with non-negative weights is PSPACE-hard.*

5 Conclusion

We identified a variety of optimization problems – some of which seemed rather unrelated on first sight – with a Skolem-hard threshold problem on MDPs. The results show that an algorithm for the exact solution to these optimization problems would imply a major breakthrough. For the partial and conditional SSPP, however, approximation algorithms were provided in [38]. Investigating the possibility to approximate optimal values might lead to algorithms useful in practice for the other objectives studied here. Further, the problems have a pseudo-polynomial solution under natural restrictions. The key result, the existence of a saturation point, has been established in the setting of stochastic multiplayer games for partial expectations [16]. This raises the question to which extend the saturation point results for the other problems can be transferred to stochastic multiplayer games.

To the best of our knowledge, the conditional value-at-risk for accumulated weights has not been addressed before. While we showed Skolem-hardness in the general setting, the computation of the optimal value is possible in exponential time in the setting of non-negative weights. Studying lower bounds for the complexity of the threshold problem and the combination of constraints on the expected accumulated weight before reaching the goal, the value-at-risk, and the conditional value-at-risk in this setting are left as future work.

References

1 Carlo Acerbi and Dirk Tasche. Expected shortfall: A natural coherent alternative to value at risk. *Economic Notes*, 31(2):379–388, 2002. doi:10.1111/1468-0300.00091.

2 S Akshay, Timos Antonopoulos, Joël Ouaknine, and James Worrell. Reachability problems for Markov chains. *Information Processing Letters*, 115(2):155–158, 2015.

3 Christel Baier, Nathalie Bertrand, Clemens Dubslaff, Daniel Gburek, and Ocan Sankur. Stochastic shortest paths and weight-bounded properties in Markov decision processes. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS'18)*, pages 86–94. ACM, 2018. doi:10.1145/3209108.3209184.

4 Christel Baier, Nathalie Bertrand, Jakob Piribauer, and Ocan Sankur. Long-run satisfaction of path properties. In *Proceedings of the Thirty-Fourth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS'19)*, pages 1–14. IEEE, 2019.

5 Christel Baier, Clemens Dubslaff, Joachim Klein, Sascha Klüppelholz, and Sascha Wunderlich. Probabilistic model checking for energy-utility analysis. In Franck van Breugel, Elham Kashefi, Catuscia Palamidessi, and Jan Rutten, editors, *Horizons of the Mind. A Tribute to Prakash Panangaden*, volume 8464 of *LNCS*, pages 96–123. Springer, 2014.

6 Christel Baier, Marcus Größer, and Nathalie Bertrand. Probabilistic ω-automata. *Journal of the ACM*, 59(1):1:1–1:52, 2012. doi:10.1145/2108242.2108243.

7 Christel Baier, Joachim Klein, Sascha Klüppelholz, and Sascha Wunderlich. Weight monitoring with linear temporal logic: Complexity and decidability. In *Proceedings of the 23rd Conference on Computer Science Logic and the 29th Symposium on Logic In Computer Science (CSL-LICS'14)*, pages 11:1–11:10. ACM, 2014. doi:10.1145/2603088.2603162.

8 Christel Baier, Joachim Klein, Sascha Klüppelholz, and Sascha Wunderlich. Maximizing the conditional expected reward for reaching the goal. In Axel Legay and Tiziana Margaria, editors, *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'17)*, volume 10206 of *Lecture Notes in Computer Science*, pages 269–285. Springer, 2017.

9 Gilles Barthe, Thomas Espitau, Luis María Ferrer Fioriti, and Justin Hsu. Synthesizing probabilistic invariants via Doob's decomposition. In Swarat Chaudhuri and Azadeh Farzan, editors, *Proceedings of the 28th International Conference on Computer Aided Verification (CAV'16), Part I*, volume 9779 of *LNCS*, pages 43–61. Springer, 2016.

10 Dimitri P Bertsekas and John N Tsitsiklis. An analysis of stochastic shortest path problems. *Mathematics of Operations Research*, 16(3):580–595, 1991.

11 Tomás Brázdil, Václav Brožek, Kousha Etessami, Antonín Kučera, and Dominik Wojtczak. One-counter Markov decision processes. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms (SODA'10)*, pages 863–874. SIAM, 2010.

12 Tomás Brázdil, Antonín Kucera, and Petr Novotný. Optimizing the expected mean payoff in energy Markov decision processes. In *14th International Symposium on Automated Technology for Verification and Analysis (ATVA'16)*, volume 9938 of *Lecture Notes in Computer Science*, pages 32–49, 2016.

13 Pavol Cerný, Thomas A. Henzinger, and Arjun Radhakrishna. Simulation distances. *Theoretical Computer Science*, 413(1):21–35, 2012.

14 Krishnendu Chatterjee and Laurent Doyen. Energy and mean-payoff parity Markov decision processes. In *Proceedings of the 36th International Symposium on Mathematical Foundations of Computer Science (MFCS'11)*, volume 6907 of *Lecture Notes in Computer Science*, pages 206–218. Springer, 2011.

15 Krishnendu Chatterjee, Hongfei Fu, and Amir Kafshdar Goharshady. Termination analysis of probabilistic programs through Positivstellensatz's. In Swarat Chaudhuri and Azadeh Farzan, editors, *Proceedings of the 28th International Conference on Computer Aided Verification (CAV'16), Part I*, volume 9779 of *LNCS*, pages 3–22. Springer, 2016.

16 Taolue Chen, Vojtěch Forejt, Marta Kwiatkowska, David Parker, and Aistis Simaitis. Automatic verification of competitive stochastic systems. *Formal Methods in System Design*, 43(1):61–92, 2013.

17 Hana Chockler, Orna Kupferman, and Moshe Y. Vardi. Coverage metrics for temporal logic model checking. *Formal Methods in System Design*, 28(3):189–212, 2006.

18 Ventsislav Chonev, Joël Ouaknine, and James Worrell. On the Skolem problem for continuous linear dynamical systems. In *43rd International Colloquium on Automata, Languages, and Programming, (ICALP'16)*, volume 55 of *LIPIcs*, pages 100:1–100:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

19 Costas Courcoubetis and Mihalis Yannakakis. The complexity of probabilistic verification. *Journal of the ACM*, 42(4):857–907, 1995.

20 Luca de Alfaro. Computing minimum and maximum reachability times in probabilistic systems. In Jos C. M. Baeten and Sjouke Mauw, editors, *Proceedings of the 10th International Conference on Concurrency Theory (CONCUR'99)*, volume 1664 of *Lecture Notes in Computer Science*, pages 66–81. Springer, 1999.

21 Kousha Etessami and Mihalis Yannakakis. Recursive Markov decision processes and recursive stochastic games. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP'05)*, volume 3580 of *Lecture Notes in Computer Science*, pages 891–903. Springer, 2005.

22 Graham Everest, Alfred Jacobus Van Der Poorten, Igor Shparlinski, Thomas Ward, et al. *Recurrence sequences*, volume 104. American Mathematical Society, 2003.

23 Vojtech Forejt and Jan Krcál. On frequency LTL in probabilistic systems. In Luca Aceto and David de Frutos-Escrig, editors, *26th International Conference on Concurrency Theory (CONCUR'15)*, volume 42 of *LIPIcs*, pages 184–197. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.CONCUR.2015.184.

24 Vojtech Forejt, Jan Krcál, and Jan Kretínský. Controller synthesis for MDPs and frequency $\mathrm{LTL}_{\setminus\mathrm{GU}}$. In Martin Davis, Ansgar Fehnker, Annabelle McIver, and Andrei Voronkov, editors, *20th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'15)*, volume 9450 of *Lecture Notes in Computer Science*, pages 162–177. Springer, 2015. doi:10.1007/978-3-662-48899-7_12.

25 Friedrich Gretz, Joost-Pieter Katoen, and Annabelle McIver. Operational versus weakest pre-expectation semantics for the probabilistic guarded command language. *Performance Evaluation*, 73:110–132, 2014.

26 Christoph Haase and Stefan Kiefer. The odds of staying on budget. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Proceedings of the 42nd International Colloquium on Automata, Languages, and Programming (ICALP'15)*, volume 9134, pages 234–246. Springer, 2015.

27 Vesa Halava, Tero Harju, Mika Hirvensalo, and Juhani Karhumäki. Skolem's problem–on the border between decidability and undecidability. Technical report, Technical Report 683, Turku Centre for Computer Science, 2005.

28 Thomas A. Henzinger. Quantitative reactive modeling and verification. *Computer Science - Research and Development*, 28(4):331–344, November 2013. doi:10.1007/s00450-013-0251-7.

29 Thomas A. Henzinger and Jan Otop. From model checking to model measuring. In *24th International Conference on Concurrency Theory (CONCUR'13)*, volume 8052 of *Lecture Notes in Computer Science*, pages 273–287. Springer, 2013.

30 Joost-Pieter Katoen, Friedrich Gretz, Nils Jansen, Benjamin Lucien Kaminski, and Federico Olmedo. Understanding probabilistic programs. In Roland Meyer, André Platzer, and Heike Wehrheim, editors, *Correct System Design - Proceedings of the Symposium in Honor of Ernst-Rüdiger Olderog on the Occasion of His 60th Birthday*, volume 9360 of *LNCS*, pages 15–32. Springer, 2015.

31 Jan Kretínský and Tobias Meggendorfer. Conditional value-at-risk for reachability and mean payoff in Markov decision processes. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS'18)*, pages 609–618. ACM, 2018. doi:10.1145/3209108.3209176.

32 Orna Kupferman and Moshe Y. Vardi. Robust satisfaction. In *10th International Conference on Concurrency Theory (CONCUR'99)*, volume 1664 of *Lecture Notes in Computer Science*, pages 383–398. Springer, 1999.

33 Omid Madani, Steve Hanks, and Anne Condon. On the undecidability of probabilistic planning and infinite-horizon partially observable Markov decision problems. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI'99)*, pages 541–548. MIT Press, 1999.

34 Rupak Majumdar, Mahmoud Salamati, and Sadegh Soudjani. On decidability of time-bounded reachability in CTMDPs. In *47th International Colloquium on Automata, Languages, and Programming, (ICALP'20)*, to appear.

35 Richard Mayr, Sven Schewe, Patrick Totzke, and Dominik Wojtczak. MDPs with energy-parity objectives. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS'17)*, IEEE Computer Society, pages 1–12, 2017.

36 Federico Olmedo, Friedrich Gretz, Nils Jansen, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Annabelle Mciver. Conditioning in probabilistic programming. *ACM Transactions on Programming Languages and Systems*, 40(1):4:1–4:50, 2018.

37 Joël Ouaknine and James Worrell. Positivity problems for low-order linear recurrence sequences. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms (SODA'14)*, pages 366–379. SIAM, 2014.

38 Jakob Piribauer and Christel Baier. Partial and conditional expectations in Markov decision processes with integer weights. In Mikolaj Bojanczyk and Alex Simpson, editors, *Proceedings of the 22nd International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'19)*, volume 11425 of *Lecture Notes in Computer Science*, pages 436–452. Springer, 2019.

39 Jakob Piribauer and Christel Baier. On Skolem-hardness and saturation points in Markov decision processes, 2020. available at `arXiv:2004.11441 [cs.LO]`.

40 Martin L Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* John Wiley & Sons, Inc., 1994.

41 Mickael Randour, Jean-François Raskin, and Ocan Sankur. Variations on the stochastic shortest path problem. In *International Workshop on Verification, Model Checking, and Abstract Interpretation*, pages 1–18. Springer, 2015.

42 Mickael Randour, Jean-François Raskin, and Ocan Sankur. Percentile queries in multi-dimensional Markov decision processes. *Formal methods in system design*, 50(2-3):207–248, 2017.

43 Paulo Tabuada and Daniel Neider. Robust linear temporal logic. In *25th EACSL Annual Conference on Computer Science Logic (CSL)*, volume 62 of *LIPIcs*, pages 10:1–10:21. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

44 Michael Ummels and Christel Baier. Computing quantiles in Markov reward models. In Frank Pfenning, editor, *Proc. of the 16th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'13)*, volume 7794 of *Lecture Notes in Computer Science*, pages 353–368. Springer, 2013. `doi:10.1007/978-3-642-37075-5_23`.

45 Stanislav Uryasev. Conditional value-at-risk: optimization algorithms and applications. In *Proc. Computational Intelligence and Financial Engineering (CIFEr'00)*, pages 49–57. IEEE, 2000.

The Power of a Single Qubit: Two-Way Quantum Finite Automata and the Word Problem

Zachary Remscrim
Department of Mathematics, MIT, Cambridge, MA, USA
remscrim@mit.edu

Abstract

The two-way finite automaton with quantum and classical states (2QCFA), defined by Ambainis and Watrous, is a model of quantum computation whose quantum part is extremely limited; however, as they showed, 2QCFA are surprisingly powerful: a 2QCFA, with a single qubit, can recognize, with bounded error, the language $L_{eq} = \{a^m b^m : m \in \mathbb{N}\}$ in expected polynomial time and the language $L_{pal} = \{w \in \{a,b\}^* : w \text{ is a palindrome}\}$ in expected exponential time.

We further demonstrate the power of 2QCFA by showing that they can recognize the word problems of many groups. In particular 2QCFA, with a single qubit and algebraic number transition amplitudes, can recognize, with bounded error, the word problem of any finitely generated virtually abelian group in expected polynomial time, as well as the word problems of a large class of linear groups in expected exponential time. This latter class (properly) includes all groups with context-free word problem. We also exhibit results for 2QCFA with any constant number of qubits.

As a corollary, we obtain a direct improvement on the original Ambainis and Watrous result by showing that L_{eq} can be recognized by a 2QCFA with better parameters. As a further corollary, we show that 2QCFA can recognize certain non-context-free languages in expected polynomial time.

In a companion paper, we prove matching lower bounds, thereby showing that the class of languages recognizable with bounded error by a 2QCFA in expected *subexponential* time is properly contained in the class of languages recognizable with bounded error by a 2QCFA in expected *exponential* time.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory; Theory of computation → Quantum computation theory

Keywords and phrases finite automata, quantum, word problem of a group

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.139

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

Related Version Full version of the paper: `https://eccc.weizmann.ac.il/report/2019/107/`.

Acknowledgements The author would like to express his sincere gratitude to Professor Michael Sipser for many years of mentorship and support, without which this work would not have been possible, to Professors Richard Lipton and David Vogan for very helpful conversations, and to the anonymous reviewers for many suggestions.

1 Introduction

The theory of quantum computation has made amazing strides in the last several decades. Landmark results, like Shor's polynomial time quantum algorithm for integer factorization [31], Grover's algorithm for unstructured search [14], and the linear system solver of Harrow, Hassidim, and Lloyd [15], have provided remarkable examples of natural problems for which quantum computers seem to have an advantage over their classical counterparts. These theoretical breakthroughs have provided strong motivation to construct quantum computers. However, while significant advancements have been made, the experimental

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 139; pp. 139:1–139:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

quantum computers that exist today are still quite limited, and are certainly not capable of implementing, on a large scale, algorithms designed for general quantum Turing machines. This naturally motivates the study of more restricted models of quantum computation.

In this paper, our goal is to understand the computational power of a small number of qubits, especially the power of a single qubit. To that end, we study two-way finite automata with quantum and classical states (2QCFA), introduced by Ambainis and Watrous [1]. Informally, a 2QCFA is a two-way deterministic finite automaton (2DFA) that has been augmented with a quantum register of constant size, i.e., a constant number of qubits. The quantum part of the machine is extremely limited; however, the model is surprisingly powerful. In particular, Ambainis and Watrous [1] showed that a 2QCFA, using only one qubit, can recognize, with bounded error, the language $L_{eq} = \{a^m b^m : m \in \mathbb{N}\}$ in expected polynomial time and the language $L_{pal} = \{w \in \{a, b\}^* : w \text{ is a palindrome}\}$ in expected exponential time. This clearly demonstrated that 2QCFA are more powerful than 2DFA, which recognize precisely the regular languages [26]. Moreover, as it is known that two-way probabilistic finite automata (2PFA) can recognize L_{eq} with bounded error in exponential time [11], but not in subexponential time [13], and cannot recognize L_{pal} with bounded error in any time bound [10], this result also demonstrated the superiority of 2QCFA over 2PFA.

We investigate the ability of 2QCFA to recognize the word problem of a group. Informally, the word problem for a group G involves determining if the product of a finite sequence of group elements $g_1, \ldots, g_k \in G$ is equal to the identity element of G. Word problems for various classes of groups have a rich and well-studied history in computational complexity theory, as there are many striking relationships between certain algebraic properties of a group G and the computational complexity of its word problem W_G. For example, $W_G \in \mathsf{REG} \Leftrightarrow G$ is finite [3], $W_G \in \mathsf{CFL} \Leftrightarrow W_G \in \mathsf{DCFL} \Leftrightarrow G$ is a finitely generated virtually free group [23], and $W_G \in \mathsf{NP} \Leftrightarrow G$ is a finitely generated subgroup of a finitely presented group with polynomial Dehn function [5].

For a quantum model, such as the 2QCFA, word problems are a particularly natural class of languages to study. There are several results [6, 37, 36] which show that certain (generally significantly more powerful) QFA variants can recognize the word problems of particular classes of groups (see the excellent survey [2] for a full discussion of the many QFA variants). Moreover, there are also results concerning the ability of QFA to recognize certain languages that are extremely closely related to word problems; in fact, the languages L_{eq} and L_{pal} considered by Ambainis and Watrous [1] are each closely related to a word problem.

Fundamentally, the laws of quantum mechanics sharply constrain the manner in which the state of the quantum register of a 2QCFA may evolve, thereby forcing the computation of a 2QCFA to have a certain algebraic structure. Similarly, the algebraic properties of a particular group G impose a corresponding algebraic structure on its word problem W_G. For certain classes of groups, the algebraic structure of W_G is extremely compatible with the algebraic structure of the computation of a 2QCFA; for other classes of groups, these two algebraic structures are in extreme opposition.

In this paper, we show that there is a broad class of groups for which these algebraic structures are quite compatible, which enables us to produce 2QCFA that recognize these word problems. As a corollary, we show that L_{eq} can be recognized by a 2QCFA with better parameters than in the original Ambainis and Watrous result [1].

In a separate paper [27], we establish matching lower bounds on the running time of a 2QCFA (and, more generally, a *quantum Turing machine* that uses sublogarithmic space) that recognizes these word problems, thereby demonstrating the optimality of these results; this allows us to prove that the class of languages recognizable with bounded error by 2QCFA in expected *subexponential* time is properly contained in the class of languages recognizable with bounded error by 2QCFA in expected *exponential* time.

1.1 Statement of the Main Results

We begin by formally defining the word problem of a group; for more extensive background, see, for instance, [21]. Let $F(S)$ denote the free group on the set S. For sets S and R, where $R \subseteq F(S)$, let $\langle R^{F(S)} \rangle$ denote the normal closure of R in $F(S)$; we say that a group G has *presentation* $\langle S|R \rangle$ if $G \cong F(S)/\langle R^{F(S)} \rangle$, in which case we write $G = \langle S|R \rangle$. For a set S, we define the set of formal inverses S^{-1}, such that for each $s \in S$, there is a unique corresponding $s^{-1} \in S^{-1}$, and $S \cap S^{-1} = \emptyset$.

▶ **Definition 1.** Suppose $G = \langle S|R \rangle$, where S is finite. Let $\Sigma = S \sqcup S^{-1}$, let Σ^* denote the free monoid over Σ, let $\phi : \Sigma^* \to G$ denote the natural monoid homomorphism that takes each string in Σ^* to the element of G that it represents, and let 1_G denote the identity element of G. The word problem of G with respect to the presentation $\langle S|R \rangle$ is the language $W_{G=\langle S|R \rangle} = \{w \in \Sigma^* : \phi(w) = 1_G\}$ consisting of all strings that represent 1_G.

Note that if $G = \langle S|R \rangle$, then S (or more precisely the image of S in G under ϕ) is a generating set for G. We say that G is *finitely generated* if it has a generating set S that is finite. If G also has presentation $\langle S'|R' \rangle$, where S' is also finite, then for any complexity class $\mathcal{C}$ closed under inverse homomorphism, $W_{G=\langle S|R \rangle} \in \mathcal{C} \Leftrightarrow W_{G=\langle S'|R' \rangle} \in \mathcal{C}$ [16]. As each complexity class $\mathcal{C}$ considered in this paper is closed under inverse homomorphism, we will use W_G to denote the word problem of a finitely generated group G, and we will write $W_G \in \mathcal{C}$ if $W_{G=\langle S|R \rangle} \in \mathcal{C}$ for some (equivalently, every) presentation $\langle S|R \rangle$ of G with S finite.

We show that, for many groups G, the corresponding word problem W_G is recognized by a 2QCFA with "good" parameters. In order to state these results, we must make use of some terminology and notation concerning 2QCFA and various classes of groups whose word problems are of complexity theoretic interest. We define the 2QCFA model in Section 2. For other definitions and additional background, we refer the reader to the full version of this paper [28]. We use $\mathbb{R}_{>0}$ to denote the positive real numbers.

▶ **Definition 2.** For $T : \mathbb{N} \to \mathbb{N}$, $\epsilon \in \mathbb{R}_{>0}$, $d \in \mathbb{N}$, and $\mathbb{A} \subseteq \mathbb{C}$, let the complexity class $\mathsf{coR2QCFA}(T, \epsilon, d, \mathbb{A})$ consist of all languages $L \subseteq \Sigma^*$ for which there is a 2QCFA M, which has d quantum basis states and transition amplitudes in $\mathbb{A}$, such that, $\forall w \in \Sigma^*$, the following holds: M runs in expected time $O(T(|w|))$, $\Pr[M \text{ accepts } w] + \Pr[M \text{ rejects } w] = 1$, $w \in L \Rightarrow \Pr[M \text{ accepts } w] = 1$, and $w \notin L \Rightarrow \Pr[M \text{ rejects } w] \geq 1 - \epsilon$.

The focus on the transition amplitudes of a 2QCFA warrants a bit of additional justification, as while it is standard to limit the transition amplitudes of a Turing machine in this way, it is common for finite automata to be defined without any such limitation. For many finite automata models, applying such a constraint would be superfluous; for example, the class of languages recognized with bounded error and in expected time $2^{n^{o(1)}}$ by a 2PFA with no restriction at all on its transition amplitudes is precisely the regular languages [9]. However, the power of the 2QCFA model is quite sensitive to the choice of transition amplitudes. A 2QCFA with non-computable transition amplitudes can recognize undecidable languages, with bounded error and in expected polynomial time [29]; whereas, 2QCFA with transition amplitudes restricted to the algebraic numbers $\overline{\mathbb{Q}}$ can only recognize languages in $\mathsf{P} \cap \mathsf{L}^2$, even if permitted unbounded error and exponential time [34]. In particular, the algebraic numbers are arguably the "standard" choice for the permitted transition amplitudes of a quantum Turing machine (QTM). It is desirable for the definition of 2QCFA to be consistent with that of QTMs as such consistency makes it more likely that techniques developed for 2QCFA could be applied to QTMs. Therefore, $\overline{\mathbb{Q}}$ is the the natural choice for the permitted transition amplitudes of a 2QCFA, though we do also consider the impact of allowing transition amplitudes in the slightly broader class $\widetilde{\mathbb{C}} = \overline{\mathbb{Q}} \cup \{e^{\pi i r} : r \in (\overline{\mathbb{Q}} \cap \mathbb{R})\}$.

We begin with a simple motivating example. For a finite alphabet Σ, a symbol $\sigma \in \Sigma$, and a word $w \in \Sigma^*$, let $\#(w, \sigma)$ denote the number of appearances of σ in w. Then the word problem for the group $\mathbb{Z}$ (the integers, where the group operation is addition) is the language $W_{\mathbb{Z}} = \{w \in \{a, b\}^* : \#(w, a) = \#(w, b)\}$. This language is closely related to the language $L_{eq} = \{a^m b^m : m \in \mathbb{N}\}$; in particular, $L_{eq} = (a^* b^*) \cap W_{\mathbb{Z}}$. More generally, the word problem for the group $\mathbb{Z}^k$ (the direct product of k copies of $\mathbb{Z}$) is the language $W_{\mathbb{Z}^k} = \{w \in \{a_1, b_1, \ldots, a_k, b_k\}^* : \#(w, a_i) = \#(w, b_i), \forall i\}$.

Ambainis and Watrous [1] showed that $L_{eq} \in \mathsf{coR2QCFA}(n^4, \epsilon, 2, \widetilde{\mathbb{C}})$, $\forall \epsilon \in \mathbb{R}_{>0}$. We note that the same method would easily imply the same result for $W_{\mathbb{Z}}$, and could be further adapted to produce a similar result for $W_{\mathbb{Z}^k}$. Our first main theorem generalizes and improves upon these results in several ways. Let $\widehat{\Pi}_1$ denote the collections of all finitely generated virtually abelian groups (i.e., all groups that have a finite-index subgroup isomorphic to $\mathbb{Z}^k$, for some $k \in \mathbb{N}$, where $\mathbb{Z}^0$ is the trivial group); we will explain this choice of notation shortly.

► **Theorem 3.** $\exists C \in \mathbb{R}_{>0}$ *such that* $\forall G \in \widehat{\Pi}_1, \forall \epsilon \in \mathbb{R}_{>0}$, $W_G \in \mathsf{coR2QCFA}(n^3, \epsilon, 2, \widetilde{\mathbb{C}}) \cap \mathsf{coR2QCFA}(n^C, \epsilon, 2, \overline{\mathbb{Q}})$.

By the above observation that $L_{eq} = (a^* b^*) \cap W_{\mathbb{Z}}$, the following corollary is immediate.

► **Corollary 4.** $\exists C \in \mathbb{R}_{>0}, \forall \epsilon \in \mathbb{R}_{>0}$, $L_{eq} \in \mathsf{coR2QCFA}(n^3, \epsilon, 2, \widetilde{\mathbb{C}}) \cap \mathsf{coR2QCFA}(n^C, \epsilon, 2, \overline{\mathbb{Q}})$.

The above corollary improves upon the result of Ambainis and Watrous [1] in two distinct senses. Firstly, using the same set of permissible transition amplitudes, our result has a better expected running time. Secondly, our result shows that L_{eq} can be recognized by a 2QCFA with transition amplitudes in $\overline{\mathbb{Q}}$, which still runs in expected polynomial time.

Let CFL denote the context-free languages (languages recognized by non-deterministic pushdown automata), OCL denote the one-counter languages (languages recognized by non-deterministic pushdown automata with single-symbol stack alphabet) and $\mathsf{poly{-}CFL}$ (resp. $\mathsf{poly{-}OCL}$) denote the intersection of finitely many context-free (resp. one-counter) languages. As $W_G \in \mathsf{poly{-}OCL} \Leftrightarrow G \in \widehat{\Pi}_1$ [17], the following corollary is also immediate.

► **Corollary 5.** $\exists C \in \mathbb{R}_{>0}, \forall W_G \in \mathsf{poly{-}OCL}, \forall \epsilon \in \mathbb{R}_{>0}$, $W_G \in \mathsf{coR2QCFA}(n^3, \epsilon, 2, \widetilde{\mathbb{C}}) \cap \mathsf{coR2QCFA}(n^C, \epsilon, 2, \overline{\mathbb{Q}})$.

Moreover, as $W_G \in \mathsf{poly{-}OCL} \cap \mathsf{CFL} \Leftrightarrow G$ is a finitely generated virtually cyclic group [17], the above corollary exhibits a wide class of non-context-free languages that are recognizable by a 2QCFA in polynomial time: the word problem of any group that is virtually $\mathbb{Z}^k$, $k \geq 2$.

Next, let F_k denote the free group of rank k, for $k \in \mathbb{N}$; in particular, F_0 is the trivial group, F_1 is the group $\mathbb{Z}$, and, for any $k \geq 2$, F_k is non-abelian. Notice that W_{F_2} is closely related to the language L_{pal}. Ambainis and Watrous [1] showed that, $\forall \epsilon \in \mathbb{R}_{>0}$, $\exists D \in \mathbb{R}_{\geq 1}$, such that $L_{pal} \in \mathsf{coR2QCFA}(D^n, \epsilon, 2, \overline{\mathbb{Q}})$, and the same method would show the same result for W_{F_2}. We show that the same result holds for any group built from free groups, using certain operations. Let $\widehat{\Pi}_2$ denote the collection of all finitely generated groups that are virtually a subgroup of a direct product of finitely many finite-rank free groups.

► **Theorem 6.** $\forall G \in \widehat{\Pi}_2, \forall \epsilon \in \mathbb{R}_{>0}, \exists D \in \mathbb{R}_{\geq 1}$, *such that* $W_G \in \mathsf{coR2QCFA}(D^n, \epsilon, 2, \overline{\mathbb{Q}})$.

As $W_G \in \mathsf{CFL} \Leftrightarrow G$ is a finitely generated virtually free group [23], we obtain the following.

► **Corollary 7.** $\forall W_G \in \mathsf{CFL}, \forall \epsilon \in \mathbb{R}_{>0}, \exists D \in \mathbb{R}_{\geq 1}$, *such that* $W_G \in \mathsf{coR2QCFA}(D^n, \epsilon, 2, \overline{\mathbb{Q}})$.

Consider the homomorphism $\pi : F_2 \times F_2 \to \mathbb{Z}$, where π takes each free generator of each copy of F_2 to a single generator of $\mathbb{Z}$; then $K = \ker \pi$ is finitely generated, but not finitely presented [32]. All groups G for which $W_G \in \mathsf{CFL} \cup \mathsf{poly{-}OCL}$ are finitely presented [16]. As $K \in \widehat{\Pi}_2$, we have the following corollary.

▶ **Corollary 8.** *There is a finitely generated group K, which is not finitely presented (hence, $W_K \notin$ CFL∪poly−OCL), where $\forall \epsilon \in \mathbb{R}_{>0}, \exists D \in \mathbb{R}_{\geq 1}$, such that $W_K \in$ coR2QCFA$(D^n, \epsilon, 2, \overline{\mathbb{Q}})$.*

▶ Remark 9. It is known that, if $G \in \widehat{\Pi}_2$, then $W_G \in$ poly−CFL [7]. Moreover, it is conjectured that $\widehat{\Pi}_2$ is precisely the class of groups whose word problem is in poly−CFL [7] (cf. [8]).

We next consider a broader class of groups. Let $Z(H)$ denote the center of a group H, let $\mathrm{U}(d, \overline{\mathbb{Q}})$ denote the group of $d \times d$ unitary matrices with entries in $\overline{\mathbb{Q}}$, let $\mathrm{PU}(d, \overline{\mathbb{Q}}) = \mathrm{U}(d, \overline{\mathbb{Q}})/Z(\mathrm{U}(d, \overline{\mathbb{Q}}))$, and let $(\mathrm{PU}(d, \overline{\mathbb{Q}}))^k$ denote the direct product of k copies of $\mathrm{PU}(d, \overline{\mathbb{Q}})$.

▶ **Theorem 10.** *If G is a finitely generated group that is virtually a subgroup of $(\mathrm{PU}(d, \overline{\mathbb{Q}}))^k$, for some $d, k \in \mathbb{N}_{\geq 1}$, then $\forall \epsilon \in \mathbb{R}_{>0}, \exists D \in \mathbb{R}_{\geq 1}$, such that $W_G \in$ coR2QCFA$(D^n, \epsilon, d, \overline{\mathbb{Q}})$.*

In order to state our final main result, as well as to provide appropriate context for the results listed above, we define the classes of groups Σ_j and Π_j, for $j \in \mathbb{N}$, inductively. First $\Sigma_0 = \Pi_0 = \{\mathbb{Z}, \{1\}\}$ (i.e., both classes consist of the two groups $\mathbb{Z}$ and the trivial group $\{1\}$). We use $\times$ to denote the direct product and $*$ to denote the free product. For $j > 1$, we define $\Pi_j = \{H_1 \times \cdots \times H_t : t \in \mathbb{N}_{\geq 1}, H_1, \ldots, H_t \in \Sigma_{j-1}\}$ and $\Sigma_j = \{H_1 * \cdots * H_t : t \in \mathbb{N}_{\geq 1}, H_1, \ldots, H_t \in \Pi_{j-1}\}$. These groups comprise an important subclass of a particularly important class of groups: the right-angled Artin groups. Note that every $G \in \bigcup_j (\Pi_j \cup \Sigma_j)$ is finitely generated. Also note that the Π_j and Σ_j form a hierarchy in the obvious way. We further define $\widehat{\Pi}_j$ (resp. $\widehat{\Sigma}_j$) as the set of all finitely generated groups that are virtually a subgroup of some group in Π_j (resp. Σ_j), which also form a hierarchy in the obvious way.

In particular, $\widehat{\Pi}_1$ (resp. $\widehat{\Pi}_2$) is precisely the class of groups for which Theorem 3 (resp. Theorem 6) demonstrates the existence of a 2QCFA that recognizes the corresponding word problem with bounded error in expected polynomial (resp. exponential) time. We next consider the class $\widehat{\Pi}_3$. While the relationship of this class to the class of groups to which Theorem 10 applies is unclear to us, we can show that the word problem of any group in this class can be recognized by a 2QCFA with negative one-sided *unbounded* error. Let coN2QCFA$(T, d, \mathbb{A})$ be defined as in Definition 2, except we now only require that $\Pr[N \text{ rejects } w] > 0, \forall w \notin L$.

▶ **Theorem 11.** *If $G \in \widehat{\Pi}_3$, then $W_G \in$ coN2QCFA$(n, 2, \widetilde{\mathbb{C}})$.*

▶ Remark 12. $\mathbb{Z} * \mathbb{Z}^2 \in \Sigma_2 \subseteq \widehat{\Pi}_3$. It is conjectured [7, 18] that $W_{\mathbb{Z}*\mathbb{Z}^2} \notin$ poly−CFL ∪ coCFL.

Lastly, we consider 2QCFA with no restrictions on their transition amplitudes, as well as the measure-once one-way quantum finite automaton (MO-1QFA) defined by Moore and Crutchfield [22]. Let coN1QFA denote the class of languages recognizable with negative one-sided unbounded error by a MO-1QFA (with any constant number of states).

▶ **Theorem 13.** *If G is a finitely generated group that is virtually a subgroup of $(\mathrm{PU}(d))^k$, for some $d, k \in \mathbb{N}_{\geq 1}$, then $W_G \in$ coN2QCFA$(n, d, \mathbb{C}) \cap$ coN1QFA.*

Let $\mathcal{D}$ denote the class of groups to which the preceding theorem applies (which includes all groups to which all earlier theorems apply). Let S denote the stochastic languages (the class of languages recognizable by PFA with strict cut-points). By [6, Theorem 3.6], coN1QFA ⊆ coS, which implies the following corollary.

▶ **Corollary 14.** *If $G \in \mathcal{D}$, then $W_G \in$ coS.*

▶ Remark 15. For many $G \in \mathcal{D}$, the fact that $W_G \in$ coS was already known: $W_{F_k} \in$ coS, $\forall k$ [6], which implies (by standard arguments from computational group theory, see for instance [23]) that $\forall G \in \widehat{\Pi}_2$, $W_G \in$ coS. However, for $G \in \mathcal{D} \setminus \widehat{\Pi}_2$, this result appears to be new.

2 Quantum Computation and the 2QCFA

In this section, we briefly recall the fundamentals of quantum computation and the definition of 2QCFA. For further background on quantum computation, see, for instance, [24, 35].

A natural way of understanding quantum computation is as a generalization of probabilistic computation. One may consider a probabilistic system defined over some finite set of states $C = \{c_1, \dots, c_k\}$, where the state of that system, at any particular point in time, is given by a probability distribution over C. Such a probability distribution may be described by a vector $v = (v_{c_1}, \dots, v_{c_k})$, where $v_c \in \mathbb{R}_{\geq 0}$ denotes the probability that the system is in state $c \in C$, and $\sum_c v_c = 1$, i.e., v is simply an element of $\mathbb{R}^k_{\geq 0}$ with L^1-norm 1.

Similarly, consider some finite set of *quantum basis states* $Q = \{q_1, \dots, q_k\}$, which correspond to an orthonormal basis $|q_1\rangle, \dots, |q_k\rangle$ of $\mathbb{C}^k$ (here and throughout the paper we use the standard bra-ket notation). The state of a quantum system over Q, at any particular time, is given by some *superposition* $|\psi\rangle = \sum_q \alpha_q |q\rangle$ of the basis states, where each $\alpha_q \in \mathbb{C}$ and $\sum_q |\alpha_q|^2 = 1$; i.e., a superposition $|\psi\rangle$ is simply an element of $\mathbb{C}^k$ with L^2-norm 1.

Let $\mathrm{U}(k)$ denote the group of $k \times k$ unitary matrices. Given a quantum system currently in the superposition $|\psi\rangle$, one may apply a transformation $t \in \mathrm{U}(k)$ to the system, after which the system is in the superposition $t\,|\psi\rangle$. One may also perform a *projective measurement in the computational basis*, which is specified by some partition $B = \{B_0, \dots, B_l\}$ of Q. Measuring a system that is in the superposition $|\psi\rangle = \sum_q \alpha_q |q\rangle$ with respect to B gives the result $B_r \in B$ with probability $p_r := \sum_{q \in B_r} |\alpha_q|^2$; additionally, if the result of the measurement is B_r, then the state of the system *collapses* to the superposition $\frac{1}{\sqrt{p_r}} \sum_{q \in B_r} \alpha_q |q\rangle$. We emphasize that measuring a quantum system changes the state of that system.

We now define a 2QCFA, essentially following the original definition in [1]. Informally, a 2QCFA is a two-way deterministic finite automaton that has been augmented with a finite size quantum register. Formally, a 2QCFA M is given by an 8-tuple, $M = \{Q, C, \Sigma, \delta, q_{start}, c_{start}, c_{acc}, c_{rej}\}$, where Q (resp. C) is the finite set of quantum (resp. classical) states, Σ is a finite alphabet, δ is the transition function, $q_{start} \in Q$ (resp. $c_{start} \in C$) is the quantum (resp. classical) start state, and $c_{acc}, c_{rej} \in C$, where $c_{acc} \neq c_{rej}$, are the accepting and rejecting states. The *quantum register* of M is given by the quantum system with basis states Q. We define the tape alphabet $\Gamma := \Sigma \sqcup \{\#_L, \#_R\}$ where the two distinct symbols $\#_L, \#_R \notin \Sigma$ will be used to denote, respectively, a left and right end-marker.

Each step of the computation of the 2QCFA M involves either performing a unitary transformation or a projective measurement on its quantum register, updating the classical state, and moving the tape head. This behavior is encoded in the transition function δ. For each $(c, \gamma) \in (C \setminus \{c_{acc}, c_{rej}\}) \times \Gamma$, $\delta(c, \gamma)$ specifies the behavior of M when it is in the classical state c and the tape head currently points to a tape alphabet symbol γ. There are two forms that $\delta(c, \gamma)$ may take, depending on whether it encodes a unitary transformation or a projective measurement. In the first case, $\delta(c, \gamma)$ is a triple (t, c', h) where $t \in \mathrm{U}(|Q|)$ is a unitary transformation to be performed on the quantum register, $c' \in C$ is the new classical state, and $h \in \{-1, 0, 1\}$ specifies whether the tape head is to move left, stay put, or move right, respectively. In the second case, $\delta(c, \gamma)$ is a pair (B, f), where B is a partition of Q specifying a projective measurement, and $f : B \to C \times \{-1, 0, 1\}$ specifies the mapping from the result of that measurement to the evolution of the classical part of the machine, where, if the result of the measurement is B_r, and $f(B_r) = (c', h)$, then $c' \in C$ is the new classical state and $h \in \{-1, 0, 1\}$ specifies the movement of the tape head.

The computation of M on an input $w \in \Sigma^*$ is then defined as follows. If w has length n, then the tape will be of size $n + 2$ and contain the string $\#_L w \#_R$. Initially, the classical state is c_{start}, the quantum register is in the superposition $|q_{start}\rangle$, and the tape head points to the leftmost tape cell. At each step of the computation, if the classical state is currently c and the tape head is pointing to symbol γ, the machine behaves as specified by $\delta(c, \gamma)$. If, at some point in the computation, M enters the state c_{acc} (resp. c_{rej}) then it immediately halts and accepts (resp. rejects) the input w. As quantum measurement is a probabilistic process, the computation of M is probabilistic. For any $w \in \Sigma^*$, we write $\Pr[M \text{ accepts } w]$ (resp. $\Pr[M \text{ rejects } w]$) for the probability that M will accept (resp. reject) the input w.

Let $\mathcal{T} = \{t \in \mathrm{U}(|Q|) : \exists (c, \gamma) \in ((C \setminus \{c_{acc}, c_{rej}\}) \times \Gamma) \text{ such that } \delta(c, \gamma) = (t, \cdot, \cdot)\}$ denote the set of all unitary transformations that M may perform. The *transition amplitudes* of M are the set of numbers $\mathbb{A}$ that appear as entries of some $t \in \mathcal{T}$.

3 Distinguishing Families of Representations

The landmark result of Lipton and Zalcstein [20] showed that, if G is a finitely generated linear group over a field of characteristic zero, then $W_G \in \mathsf{L}$. The key idea behind their logspace algorithm was to make use of a carefully chosen *representation* of the group G in order to recognize W_G (see, for instance, [19] or the full version of our paper [28] for the notation and terminology from representation theory used in this section). Our 2QCFA algorithm will operate in a similar manner; however, the constraints of quantum mechanics will require us to make many modifications to their approach.

A (unitary) representation of a (topological) group G is a continuous homomorphism $\rho : G \to \mathrm{U}(\mathcal{H})$, where $\mathcal{H}$ is a Hilbert space, and $\mathrm{U}(\mathcal{H})$ is the group of unitary operators on $\mathcal{H}$. The Gel'fand-Raikov theorem states that the elements of any locally compact group G are separated by its unitary representations; i.e., $\forall g \in G$ with $g \neq 1_G$, there is some $\mathcal{H}$ and some $\rho : G \to \mathrm{U}(\mathcal{H})$ such that $\rho(g) \neq \rho(1_G)$. For certain groups, stronger statements can be made; in particular, one calls a group maximally almost periodic if the previous condition still holds when $\mathcal{H}$ is restricted to be finite-dimensional.

The core idea of our approach to recognizing the word problem W_G of a particular group G is to construct what we have chosen to call a *distinguishing family of representations* (DFR) for G, which is a refinement of the above notion. Informally, a DFR is a collection of a small number of unitary representations of G, all of which are over a Hilbert space of small dimension, such that, for any $g \in G$ other than 1_G, there is some representation ρ in the collection for which $\rho(g)$ is "far from" $\rho(1_G)$, relative to the "size" of g. The following definition formalizes this, by introducing parameters to quantify the above fuzzy notions. In this definition, and in the remainder of the paper, let $\mathrm{U}(d)$ denote the group of $d \times d$ unitary matrices, let $M(d, \mathbb{A})$ denote the set of $d \times d$ matrices with entries in some set $\mathbb{A}$, let $\mathrm{U}(d, \mathbb{A}) = \mathrm{U}(d) \cap M(d, \mathbb{A})$, and let $G_{\neq 1} = G \setminus \{1_G\}$. For a group $G = \langle S | R \rangle$, let $l(g)$ denote the length of any $g \in G$ relative to the generating set S (i.e., $l(g)$ is the minimum value of m for which $\exists g_1, \ldots, g_m \in S \cup S^{-1}$ such that $g = \phi(g_1 \cdots g_m)$). For a representation $\rho : G \to \mathrm{U}(d)$, let $\chi_\rho : G \to \mathbb{C}$ denote the *character* of ρ (i.e., $\chi_\rho(g) = \mathrm{Tr}(\rho(g))$).

▶ **Definition 16.** Consider a group $G = \langle S | R \rangle$, with S finite. For $k \in \mathbb{N}_{\geq 1}, d \in \mathbb{N}_{\geq 2}$, $\tau : \mathbb{R}_{>0} \to \mathbb{R}_{>0}$ a monotone non-increasing function, and $\mathbb{A} \subseteq \mathbb{C}$, we define a $[k, d, \tau, \mathbb{A}]$-*distinguishing family of representations* (DFR) for G to be a set $\mathcal{F} = \{\rho_1, \ldots, \rho_k\}$ where the following conditions hold.

(a) $\forall j \in \{1, \ldots, k\}$, $\rho_j : G \to \mathrm{U}(d)$ is a representation of G.

(b) $\forall g \in G_{\neq 1}$, $\exists j \in \{1, \ldots, k\}$ such that $|\chi_{\rho_j}(g)| \leq d - \tau(l(g))$.

(c) $\forall \sigma \in S \cup S^{-1}, \forall j \in \{1, \ldots, k\}$, $\exists Y_1, \ldots, Y_t \in \mathrm{U}(d, \mathbb{A})$, such that $\rho_j(\sigma) = \prod_i Y_i$.

Suppose $\mathcal{F} = \{\rho_1, \ldots, \rho_k\}$ is a $[k, d, \tau, \mathbb{A}]$-DFR for $G = \langle S|R\rangle$. We write $I_d = 1_{\mathrm{U}(d)} \in \mathrm{U}(d)$ for the $d \times d$ identity matrix, $\ker(\rho_j) = \{g \in G : \rho_j(g) = I_d\}$ for the kernel of ρ_j, $Z(\mathrm{U}(d)) = \{e^{ir} I_d : r \in \mathbb{R}\}$ for the center of $\mathrm{U}(d)$, and $\mathrm{Pker}(\rho_j) = \{g \in G : \rho_j(g) = Z(\mathrm{U}(d))\}$ for the quasikernel of ρ_j. Clearly, $1_G \in \mathrm{Pker}(\rho_j), \forall j$, but, as ρ_j is not assumed to be P-faithful or even faithful, there may be $g \in G_{\neq 1}$ for which, for certain j, we have $g \in \mathrm{Pker}(\rho_j)$. However, due to the fact that $g \in \mathrm{Pker}(\rho_j)$ exactly when $|\chi_{\rho_j}(g)| = d$, the second defining property of a DFR guarantees not only that $\bigcap_j \mathrm{Pker}(\rho_j) = \{1_G\}$, but, much more strongly, that all $g \in G_{\neq 1}$ are "far from" being in $\bigcap_j \mathrm{Pker}(\rho_j)$. That is to say, $\forall g \in G_{\neq 1}, \exists j$ such that $|\chi_{\rho_j}(g)|$ is at distance at least $\tau(l(g))$ from having value d. The following proposition is then immediate, but we explicitly state it as it is the central notion in our quantum approach to the word problem.

▶ **Proposition 17.** *Suppose $G = \langle S|R\rangle$ has a $[k, d, \tau, \mathbb{A}]$-DFR $\mathcal{F} = \{\rho_1, \ldots, \rho_k\}$. Then, $\forall g \in G$, $g = 1_G \Leftrightarrow \forall j,\ |\chi_{\rho_j}(g)| = d$ and $g \in G_{\neq 1} \Leftrightarrow \exists j$ such that $|\chi_{\rho_j}(g)| \leq d - \tau(l(g))$.*

Note that, in the preceding proposition, $\rho_1 \oplus \cdots \oplus \rho_k : G \to \mathrm{U}(kd)$ is simply a faithful representation of G, decomposed into subrepresentations in a convenient way. Next, we establish some terminology that will better allow us to describe particular types of DFR.

▶ **Definition 18.** Suppose $\mathcal{F} = \{\rho_1, \ldots, \rho_k\}$ is a $[k, d, \tau, \mathbb{A}]$-DFR for a group G.

(a) If $\mathbb{A} = \overline{\mathbb{Q}}$ (equivalently, if $\rho_j(G) \subseteq \mathrm{U}(d, \overline{\mathbb{Q}}), \forall j$), we say $\mathcal{F}$ is an *algebraic* DFR.

(b) If $\rho_j(g)$ is a diagonal matrix $\forall j, \forall g$, then we say $\mathcal{F}$ is a *diagonal* DFR.

(c) If H is a finite-index overgroup of G, we say that H *virtually* has a $[k, d, \tau, \mathbb{A}]$-DFR.

When $\mathcal{F}$ is an algebraic DFR, we will often only write $[k, d, \tau]$ to denote its parameters. Note that only abelian groups have diagonal DFRs, and any DFR of an abelian group can be converted to a diagonal DFR; we define diagonal DFRs for convenience.

Using a $[k, d, \tau, \mathbb{A}]$-DFR for a group G, it will be possible to construct a 2QCFA that recognizes the word problem W_H of any finite-index overgroup H of G, where the parameters of the DFR will strongly impact the parameters of the resulting 2QCFA. In particular, in Section 4, we produce a 2QCFA with d quantum states and transition amplitudes in $\mathbb{A}$ that recognizes W_H, with expected running time approximately $O(\tau(n)^{-1})$. The goal is then to show that a wide collection of groups virtually have DFRs with good parameters.

3.1 Diophantine Approximation

Our constructions of DFRs rely crucially on certain results concerning Diophantine approximation. Most fundamentally, the Diophantine approximation question asks how well a particular real number α can be approximated by rational numbers. Of course, as $\mathbb{Q}$ is dense in $\mathbb{R}$, one can choose $\frac{p}{q} \in \mathbb{Q}$ so as to make the quantity $|\alpha - \frac{p}{q}|$ arbitrarily small; for this reason, one considers $\frac{p}{q}$ to be a "good" approximation to α only when $|\alpha - \frac{p}{q}|$ is small compared to a suitable function of q. One then considers α to be poorly approximated by rationals if, for some "small" constant $d \in \mathbb{R}_{\geq 2}$, $\exists C \in \mathbb{R}_{>0}$ such that, $\forall (p, q) \in \mathbb{Z} \times \mathbb{Z}_{\neq 0}$, we have $|\alpha - \frac{p}{q}| \geq C|q|^{-d}$, where the smallness of d determines just how poorly approximable α is. For $\alpha \in \mathbb{R}$, let $\|\alpha\| = \min_{m \in \mathbb{Z}} |\alpha - m|$ denote the distance between α and its nearest integer. Notice that $\left|\alpha - \frac{p}{q}\right| \geq C|q|^{-d}$, $\forall (p, q) \in \mathbb{Z} \times \mathbb{Z}_{\neq 0} \Leftrightarrow \|q\alpha\| \geq C|q|^{-(d-1)}$, $\forall q \in \mathbb{Z}_{\neq 0}$. Of particular relevance to us is the following result, due to Schmidt [30], that real irrational algebraic numbers are poorly approximated by rationals.

▶ **Proposition 19** ([30])**.** *$\forall \alpha_1, \ldots, \alpha_k \in (\mathbb{R} \cap \overline{\mathbb{Q}})$ where $1, \alpha_1, \ldots, \alpha_k$ are linearly independent over $\mathbb{Q}$, $\forall \epsilon \in \mathbb{R}_{>0}$, $\exists C \in \mathbb{R}_{>0}$ such that $\forall q \in \mathbb{Z}_{\neq 0}$, $\exists j$ such that $\|q\alpha_j\| \geq C|q|^{-(\frac{1}{k}+\epsilon)}$.*

We also require the following result concerning the Diophantine properties of linear forms in logarithms of algebraic numbers, due to Baker [4].

▶ **Proposition 20** ([4]). *Let $L = \{\beta \in \mathbb{C}_{\neq 0} : e^{\beta} \in \overline{\mathbb{Q}}\}$. $\forall \beta_1, \dots, \beta_k \in L$ that are linearly independent over $\mathbb{Q}$, $\exists C \in \mathbb{R}_{>0}$ such that, $\forall (q_1, \dots, q_k) \in \mathbb{Z}^k$ with $q_{max} := \max_j |q_j| > 0$, we have $|q_1\beta_1 + \cdots + q_k\beta_k| \geq (eq_{max})^{-C}$.*

Gamburd, Jakobson, and Sarnak [12] established a particular result concerning the Diophantine properties of $\mathrm{SU}(2, \overline{\mathbb{Q}})$. The following lemma generalizes their result to $\mathrm{U}(d, \overline{\mathbb{Q}})$; a proof of this lemma appears in the full version [28].

▶ **Lemma 21.** *Consider a group $G = \langle S|R \rangle$, with S finite, and a representation $\rho : G \to \mathrm{U}(d, \overline{\mathbb{Q}})$. Then $\exists C \in \mathbb{R}_{\geq 1}$ such that $|\chi_\rho(g)| \leq d - C^{-l(g)}$, $\forall g \in (G \setminus \mathrm{Pker}(\rho))$.*

3.2 Constructions of DFRs

We now show that a wide collection of groups virtually have DFRs with good parameters. We accomplish this by first constructing DFRs for only a small family of special groups. We then present several constructions in which a DFR for a group, or more generally a family of DFRs for a family of groups, is used to produce a DFR for a related group.

We begin with a straightforward lemma expressing a useful character bound. In this lemma, and throughout this section, we continue to write group operations multiplicatively, and so, for $g \in G$ and $h \in \mathbb{Z}$, if $h > 0$ (resp. $h < 0$) then g^h denotes the element of G obtained by combining h copies of g (resp. g^{-1}) with the group operation, and if $h = 0$ then $g^h = 1_G$. Let $S_1 = \{e^{ir} : r \in \mathbb{R}\} \subseteq \mathbb{C}^*$ denote the circle group and let $\mathrm{T}(d) \subseteq \mathrm{U}(d)$ denote the group of all $d \times d$ diagonal matrices where each diagonal entry lies in S_1. For $\mathbb{A} \subseteq \mathbb{C}$, let $S_1(\mathbb{A}) = S_1 \cap \mathbb{A}$ and $\mathrm{T}(d, \mathbb{A}) = \mathrm{T}(d) \cap M(d, \mathbb{A})$. Let $\mathbf{1}_d : G \to \mathrm{U}(d)$ denote the trivial representation of dimension d (i.e., $\mathbf{1}_d(g) = I_d = 1_{\mathrm{U}(d)}$, $\forall g \in G$). For a cyclic group $G = \langle a|R_G \rangle$ and for some $r \in \mathbb{R}$, define the representation $\widehat{\gamma}_r : G \to S_1 \cong \mathrm{U}(1)$ such that $a \mapsto e^{2\pi i r}$; furthermore, define the representation $\gamma_r : G \to \mathrm{T}(2)$ by $\gamma_r = \widehat{\gamma}_r \oplus \mathbf{1}_1$.

▶ **Lemma 22.** *Consider the cyclic group $G = \langle a|R_G \rangle$. Fix $r \in \mathbb{R}$ and define $\gamma_r : G \to \mathrm{T}(2)$ as above. Suppose that $h \in \mathbb{Z}$ and $\epsilon \in \mathbb{R}_{>0}$ satisfy $\|hr\| \geq \epsilon$. Then $\chi_{\gamma_r}(a^h) \leq 2 - \frac{19\pi^2}{24}\epsilon^2$.*

Proof. We have $\chi_{\gamma_r}(a^h) = e^{2\pi i h r} + 1 = 2e^{\pi i h r}\cos(\pi h r)$. Clearly, $\epsilon \leq \frac{1}{2}$. Therefore,

$$|\chi_{\gamma_r}(a^h)| = 2|\cos(\pi h r)| \leq 2\cos(\pi\epsilon) \leq 2\left(1 - \frac{(\pi\epsilon)^2}{2} + \frac{(\pi\epsilon)^4}{24}\right) \leq 2 - \frac{19\pi^2}{24}\epsilon^2.$$ ◀

We first construct DFRs for a very narrow class of special groups: (i) $\mathbb{Z}_m = \langle a|a^m \rangle$, the integers modulo m, where the group operation is addition, (ii) $\mathbb{Z} = \langle a| \rangle$, the integers, where the group operations is addition, and (iii) $F_2 = \langle a, b| \rangle$ the (non-abelian) free group of rank 2.

▶ **Lemma 23.** *$\mathbb{Z}_m = \langle a|a^m \rangle$ has a diagonal algebraic $\left[1, 2, \frac{19\pi^2}{24m^2}\right]$-DFR, $\forall m \in \mathbb{N}_{\geq 2}$.*

Proof. Fix $m \in \mathbb{N}_{\geq 2}$ and let $r = \frac{1}{m}$. Define $\gamma_r : \mathbb{Z}_m \to \mathrm{T}(2)$ as above, and notice that $\gamma_r(\mathbb{Z}_m) \subseteq \mathrm{T}(2, \overline{\mathbb{Q}})$. Consider any $q \in \mathbb{Z}_m$, where $q \not\equiv 0 \mod m$. Then q can be expressed as $q = a^h$, for $h \in \mathbb{Z}$, $h \not\equiv 0 \mod m$. As $\|hr\| \geq \frac{1}{m}$, Lemma 22 implies $|\chi_{\gamma_r}(q)| \leq 2 - \frac{19\pi^2}{24m^2}$. Therefore, $\{\gamma_r\}$ is a diagonal algebraic DFR for $\mathbb{Z}_m$, with the desired parameters. ◀

▶ **Lemma 24.** *$\forall \delta \in \mathbb{R}_{>0}, \exists C \in \mathbb{R}_{>0}$, $\mathbb{Z} = \langle a| \rangle$ has a diagonal $[1 + \lfloor \frac{2}{\delta} \rfloor, 2, Cn^{-\delta}, \widetilde{\mathbb{C}}]$-DFR.*

Proof. Let $k = 1+\lfloor\frac{2}{\delta}\rfloor$ and $\eta = \frac{\delta}{2}-\frac{1}{k} > 0$. Fix $\alpha_1, \ldots, \alpha_k \in (\overline{\mathbb{Q}}\cap\mathbb{R})$ such that $1, \alpha_1, \ldots, \alpha_k$ are linearly independent over $\mathbb{Q}$. For each $j \in \{1, \ldots, k\}$ define the representation $\gamma_{\alpha_j} : \mathbb{Z} \to \mathrm{T}(2)$ as above, and notice that $\gamma_{\alpha_j}(\mathbb{Z}) \subseteq \mathrm{T}(2, \widetilde{\mathbb{C}})$. By Proposition 19, $\exists D \in \mathbb{R}_{>0}$, such that $\forall q \in \mathbb{Z}_{\neq 0}$ (i.e., $\forall q \in \mathbb{Z}$ where $q \neq 0 = 1_{\mathbb{Z}}$), $\exists j$ such that $\|q\alpha_j\| \geq D|q|^{-(\frac{1}{k}+\eta)} = D|q|^{-\frac{\delta}{2}}$. Therefore, for any $q \in \mathbb{Z}_{\neq 0}$, if we take j as above, then by Lemma 22 (with $r = \alpha_j$, $\epsilon = D|q|^{-\frac{\delta}{2}}$, and $h = q$) we have $|\chi_{\gamma_{\alpha_j}}(q)| \leq 2 - \frac{19\pi^2}{24}D^2|q|^{-\delta}$. Therefore, $\{\gamma_{\alpha_1}, \ldots, \gamma_{\alpha_k}\}$ is a diagonal $[1 + \lfloor\frac{2}{\delta}\rfloor, 2, \frac{19\pi^2}{24}D^2 n^{-\delta}, \widetilde{\mathbb{C}}]$-DFR for $\mathbb{Z}$. ◀

▶ **Lemma 25.** *$\exists C_1, C_2 \in \mathbb{R}_{>0}$ such that $\mathbb{Z} = \langle a|\rangle$ has a diagonal algebraic $[1, 2, C_2 n^{-C_1}]$-DFR.*

Proof. As in Proposition 20, let $L = \{\beta \in \mathbb{C}_{\neq 0} : e^{\beta} \in \overline{\mathbb{Q}}\}$ and notice that $\pi i \in L$. Let $R = \{r \in ((\mathbb{R} \setminus \mathbb{Q}) \cap (0, 1)) : 2\pi i r \in L\}$ (e.g., $\hat{r} = \frac{1}{2\pi}\cos^{-1}\left(\frac{3}{5}\right)$ is irrational and has $e^{2\pi i\hat{r}} = \frac{3+4i}{5}$, and so $\hat{r} \in R$). Fix $r \in R$. By definition, $2\pi i r \in L$, which immediately implies $\pi i r \in L$. Also by definition, $r \notin \mathbb{Q}$, which implies $\pi i r$ and πi are linearly independent over $\mathbb{Q}$. Therefore, by Proposition 20, $\exists D \in \mathbb{R}_{>0}$ such that $\forall (q, m) \in \mathbb{Z}^2$ where $q_{max} := \max(|q|, |m|) > 0$, we have $|q\pi i r - m\pi i| \geq (eq_{max})^{-D}$.

For fixed $q \in \mathbb{Z}_{\neq 0}$ and varying $m \in \mathbb{Z}$, $|q\pi i r - m\pi i|$ attains its minimum when $m = \mathrm{round}(qr)$, the closest integer to qr. Notice that $|\mathrm{round}(qr)| \leq |q|$, as $r \in (0, 1)$ and $q \in \mathbb{Z}$. Therefore, for any $q \in \mathbb{Z}_{\neq 0}$, we have

$$\|qr\| = \min_{m\in\mathbb{Z}}|qr - m| = \frac{1}{\pi}\min_{m\in\mathbb{Z}}|q\pi i r - m\pi i| = \frac{1}{\pi}|q\pi i r - \mathrm{round}(qr)\pi i| \geq \frac{1}{\pi}|eq|^{-D}.$$

Define $\gamma_r : \mathbb{Z} \to \mathrm{T}(2)$ as above. By Lemma 22, $|\chi_{\gamma_r}(q)| \leq 2 - \frac{19}{24}|eq|^{-2D}$. Clearly, $\gamma_r(\mathbb{Z}) \subseteq \mathrm{T}(2, \overline{\mathbb{Q}})$. Therefore, $\{\gamma_r\}$ is a diagonal algebraic $[1, 2, \frac{19}{24}e^{-2D}n^{-2D}]$-DFR for $\mathbb{Z}$. ◀

▶ Remark 26. We note that the above constructions of DFRs for $\mathbb{Z}$ are quite similar to the technique used by Ambainis and Watrous [1] to produce a 2QCFA that recognizes L_{eq} (cf. [6, 25]). In particular, their approach relied on the fact that the number $\sqrt{2} \in \overline{\mathbb{Q}}$ is poorly approximated by rationals; our constructions make use of more general Diophantine approximation results. This allows us to produce 2QCFA with improved parameters.

▶ **Lemma 27.** *$\exists C \in \mathbb{R}_{\geq 1}$, such that $F_2 = \langle a, b|\rangle$ has an algebraic $[1, 2, C^{-n}]$-DFR.*

Proof. First, define the representation $\pi : F_2 \to SO(3, \mathbb{Q})$ by

$$a \mapsto \frac{1}{5}\begin{pmatrix} 3 & -4 & 0 \\ 4 & 3 & 0 \\ 0 & 0 & 5 \end{pmatrix} \text{ and } b \mapsto \frac{1}{5}\begin{pmatrix} 5 & 0 & 0 \\ 0 & 3 & -4 \\ 0 & 4 & 3 \end{pmatrix}.$$

This is the “standard” faithful representation of F_2 into SO(3) used in many treatments of the Banach-Tarski paradox. Recall that SU(2) is the double cover of SO(3), i.e., $\mathrm{SU}(2)/Z(\mathrm{SU}(2)) \cong \mathrm{SO}(3)$. Then π induces a homomorphism $\widehat{\pi} : F_2 \to \mathrm{SU}(2)/Z(\mathrm{SU}(2))$ in the obvious way, which, by the universal property of the free group, can be lifted to the representation $\rho : F_2 \to \mathrm{SU}(2, \overline{\mathbb{Q}})$ given by

$$a \mapsto \frac{1}{\sqrt{5}}\begin{pmatrix} 2+i & 0 \\ 0 & 2-i \end{pmatrix} \text{ and } b \mapsto \frac{1}{\sqrt{5}}\begin{pmatrix} 2 & i \\ i & 2 \end{pmatrix}.$$

As π is faithful, we conclude that $\rho(g) \notin Z(\mathrm{SU}(2))$, $\forall g \in (F_2 \setminus 1_{F_2})$. Therefore, by Lemma 21, $\{\rho\}$ is an algebraic $[1, 2, C^{-n}]$-DFR for F_2. ◀

▶ Remark 28. Note that the proof of the preceding lemma uses, fundamentally, the same construction used by Ambainis and Watrous [1] to produce a 2QCFA for L_{pal} (which is closely related to F_2). The algebraic structure of F_2 allows a substantially simpler argument.

We now present several constructions of new DFRs from existing DFRs. We emphasize that all results in the following lemmas are constructive in the sense that, given the supposed DFR or collection of DFRs, each corresponding proof provides an explicit construction of the new DFR. Due to space restrictions, all proofs are omitted and may be found in the full version [28]. We begin by considering conversions of a DFR of a group G to a DFR with different parameters of the same group G. For $C \in \mathbb{R}_{>0}$, let $\eta_C : \mathbb{R}_{>0} \to \mathbb{R}_{>0}$ be given by $\eta_C(n) = Cn$.

▶ **Lemma 29.** *Suppose $\mathcal{F}$ is a $[k, d, \tau, \mathbb{A}]$-DFR for a group $G = \langle S|R\rangle$, with S finite. The following statements hold.*

(i) *G has a $[1, kd, \tau, \mathbb{A}]$-DFR.*

(ii) *If $d' \in \mathbb{N}$ and $d' > d$, then G has a $[k, d', \tau, \mathbb{A}]$-DFR.*

(iii) *Suppose G also has presentation $\langle S'|R'\rangle$, with S' finite. Then $\exists C \in \mathbb{R}_{>0}$ such that $\mathcal{F}$ is also a $[k, d, \tau \circ \eta_C, \mathbb{A}]$-DFR for $G = \langle S'|R'\rangle$.*

Moreover, if $\mathcal{F}$ is a diagonal DFR, then each newly constructed DFR is also diagonal.

Next, we show that a DFR of G and a DFR of H can be used to produce a DFR of $G \times H$, the direct product of G and H. In the following, for a group Q, let $[q_1, q_2] = q_1^{-1}q_2^{-1}q_1q_2$ denote the commutator of elements $q_1, q_2 \in Q$. For functions $\tau, \tau' : \mathbb{R}_{>0} \to \mathbb{R}_{>0}$, we define the function $\tau^{\min}_{\tau,\tau'} : \mathbb{R}_{>0} \to \mathbb{R}_{>0}$ by $\tau^{\min}_{\tau,\tau'}(n) := \min(\tau(n), \tau'(n))$, $\forall n \in \mathbb{R}_{>0}$.

▶ **Lemma 30.** *Consider groups $G = \langle S_G|R_G\rangle$ and $H = \langle S_H|R_H\rangle$, with S_G and S_H finite, and $S_G \cap S_H = \emptyset$. Let $R_{com} = \{[g, h] : g \in S_G, h \in S_H\}$. If G has a $[k, d, \tau, \mathbb{A}]$-DFR and H has a $[k', d', \tau', \mathbb{A}]$-DFR, then $G\times H = \langle S_G \sqcup S_H|R_G \cup R_H \cup R_{com}\rangle$ has a $[k+k', \max(d, d'), \tau^{\min}_{\tau,\tau'}, \mathbb{A}]$-DFR. Moreover, if G and H have diagonal DFRs with the above parameters, then $G \times H$ has a diagonal DFR with the above parameters.*

Now, we show that a DFR of a group G can be used to produce a DFR of a finitely generated subgroup of G, or of a finite-index overgroup of G.

▶ **Lemma 31.** *Suppose $\mathcal{F}_G$ is a $[k, d, \tau, \mathbb{A}]$-DFR for a group $G = \langle S_G|R_G\rangle$, with S_G finite. The following statements hold.*

(i) *Suppose $H \leq G$, where $H = \langle S_H|R_H\rangle$, with S_H finite. Then $\exists C \in \mathbb{R}_{>0}$ such that H has a $[k, d, \tau \circ \eta_C, \mathbb{A}]$-DFR. If, moreover, $\mathcal{F}_G$ is a diagonal DFR, then H will also have a diagonal DFR with the claimed parameters.*

(ii) *Suppose $G \leq Q$, where $Q = \langle S_Q|R_Q\rangle$, with S_Q finite, $S_G \subseteq S_Q$, and $r := [Q : G]$ finite. Then $\exists C \in \mathbb{R}_{>0}$ such that Q has a $[k, dr, \tau \circ \eta_C, \mathbb{A}]$-DFR.*

▶ Remark 32. By the preceding lemma, any group G that virtually has a DFR also has a DFR, but with worse parameters. As will be shown, it is possible to recognize W_G using a DFR for a finite-index subgroup of G, thereby avoiding this worsening of parameters.

We now construct DFRs, with good parameters, for a wide class of groups. Recall that any finitely generated abelian group G admits a unique decomposition $G \cong \mathbb{Z}^r \times \mathbb{Z}_{m_1} \times \cdots \times \mathbb{Z}_{m_t}$, where m_i divides m_{i+1}, $\forall i \in \{1, \ldots, t-1\}$, and each $m_i \in \mathbb{N}_{\geq 2}$. Let $R(r, m_1, \ldots, m_t) = \{a_i^{m_i} : i \in \{1, \ldots, t\}\} \cup \{[a_i, a_j] : i, j \in \{1, \ldots, r+t\}\}$.

▶ **Lemma 33.** *Consider the finite (hence finitely generated) abelian group $G = \mathbb{Z}_{m_1} \times \cdots \times \mathbb{Z}_{m_t} = \langle a_1, \ldots, a_t|R(0, m_1, \ldots, m_t)\rangle$. If $t = 0$ (i.e., G is the trivial group), then G has a diagonal algebraic $[1, 2, 2]$-DFR. Otherwise, G has a diagonal algebraic $\left[t, 2, \frac{19\pi^2}{24m_t^2}\right]$-DFR.*

Proof. If $t = 0$, the claim is obvious. Suppose $t > 0$. By Lemma 23, each factor $\mathbb{Z}_{m_i} = \langle a | a^{m_i} \rangle$ has a diagonal algebraic $\left[1, 2, \frac{19\pi^2}{24m_i^2}\right]$-DFR. Notice that $m_1 \leq \cdots \leq m_t$, as each m_i divides m_{i+1}. The existence of the desired DFR follows from Lemma 30. ◀

▶ **Theorem 34.** *$\exists C_1 \in \mathbb{R}_{>0}$ such that, for any finitely generated abelian group $G = \mathbb{Z}^r \times \mathbb{Z}_{m_1} \times \cdots \times \mathbb{Z}_{m_t} = \langle a_1, \ldots, a_{r+t} | R(r, m_1, \ldots, m_t) \rangle$, the following statements hold.*

(i) *$\exists C_2 \in \mathbb{R}_{>0}$ such that G has a diagonal algebraic $\left[r + t, 2, C_2 n^{-C_1}\right]$-DFR.*

(ii) *$\forall \delta \in \mathbb{R}_{>0}$, $\exists C_3 \in \mathbb{R}_{>0}$, such that G has a diagonal $\left[r\left(1 + \lfloor \frac{2}{\delta} \rfloor\right) + t, 2, C_3 n^{-\delta}, \widetilde{\mathbb{C}}\right]$-DFR.*

Proof. By Lemma 25, $\exists D_1, D_2 \in \mathbb{R}_{>0}$ such that $\mathbb{Z}$ has a diagonal algebraic $[1, 2, D_2 n^{-D_1}]$-DFR, which we call $\mathcal{F}$. We set $C_1 = D_1$. Let $H_1 = \mathbb{Z}^r$ and $H_2 = \mathbb{Z}_{m_1} \times \cdots \times \mathbb{Z}_{m_t}$. If $r = 0$, both claims follow trivially from Lemma 33. Suppose $r > 0$.

(i) Using the DFR $\mathcal{F}$ of $\mathbb{Z}$, Lemma 30 implies H_1 has a diagonal algebraic $[r, 2, D_2 n^{-C_1}]$-DFR $\mathcal{H}_1$. If $t = 0$, then $G = H_1$; therefore, $\mathcal{H}_1$ is the desired DFR for G, with $C_2 = D_2$, and we are done. If $t > 0$, Lemma 33 implies H_2 has a diagonal algebraic $\left[1, 2, \frac{19\pi^2}{24m_t^2}\right]$-DFR $\mathcal{H}_2$. Set $C_2 = \min(D_2, \frac{19\pi^2}{24m_t^2})$. By Lemma 30, we conclude $G = H_1 \times H_2$ has a DFR with the claimed parameters.

(ii) By Lemma 24, $\exists D \in \mathbb{R}_{>0}$ such that $\mathbb{Z}$ has a diagonal $\left[1 + \lfloor \frac{2}{\delta} \rfloor, 2, D n^{-\delta}, \widetilde{\mathbb{C}}\right]$-DFR, $\mathcal{F}'$. The remainder of the proof is analogous to that of part (i), using $\mathcal{F}'$ in place of $\mathcal{F}$. ◀

As in Section 1.1, $\widehat{\Pi}_1$ denotes the set of all finitely generated virtually abelian groups. For $G \in \widehat{\Pi}_1$, there is a unique $r \in \mathbb{N}$ such that G is virtually $\mathbb{Z}^r$. We have the following corollary.

▶ **Corollary 35.** *$\exists C \in \mathbb{R}_{>0}$ such that, $\forall G \in \widehat{\Pi}_1$, the following holds.*

(i) *$\exists D \in \mathbb{R}_{>0}, \exists K \in \mathbb{N}_{\geq 1}$, such that G virtually has a diagonal algebraic $[K, 2, D n^{-C}]$-DFR.*

(ii) *$\forall \delta \in \mathbb{R}_{>0}$, $\exists D \in \mathbb{R}_{>0}, \exists K \in \mathbb{N}_{\geq 1}$, G virtually has a diagonal $\left[K, 2, D n^{-\delta}, \widetilde{\mathbb{C}}\right]$-DFR.*

Next, we consider groups that can be built from finitely generated free groups.

▶ **Lemma 36.** *$\forall r \in \mathbb{N}$, $\exists C \in \mathbb{R}_{\geq 1}$, $F_r = \langle a_1, \ldots, a_r | \rangle$ has an algebraic $[1, 2, C^{-n}]$-DFR.*

Proof. As $F_0 = \{1\}$ and $F_1 = \mathbb{Z}$, Theorem 34 immediately implies the claim when $r \in \{0, 1\}$. Next, consider the case in which $r = 2$. By Lemma 27, $\exists C \in \mathbb{R}_{\geq 1}$ such that $F_2 = \langle a_1, a_2 | \rangle$ has an algebraic $[1, 2, C^{-n}]$-DFR. Finally, suppose $r > 2$. By the Nielsen-Schreier theorem, F_2 has a finite-index subgroup isomorphic to F_r; the claim immediately follows from Lemma 31(i). ◀

▶ **Theorem 37.** *Suppose $G = \langle S | R \rangle$, with S finite, such that $G \leq F_{r_1} \times \cdots \times F_{r_t}$, for some $r_1, \ldots, r_t \in \mathbb{N}$. Then $\exists C \in \mathbb{R}_{\geq 1}$ such that G has an algebraic $[t, 2, C^{-n}]$-DFR.*

Proof. By Lemma 36, each F_{r_i} has an algebraic $[1, 2, C_i^{-n}]$-DFR, for some $C_i \in \mathbb{R}_{\geq 1}$. Lemma 30 implies that $F_{r_1} \times \cdots \times F_{r_t}$ has an algebraic $[t, 2, C^{-n}]$-DFR, where $C = \max_i C_i$, and Lemma 31(i) then implies G has a DFR with the claimed parameters. ◀

As in Section 1.1, $\widehat{\Pi}_2$ denotes the class of finitely generated groups that are virtually a subgroup of a direct product of finitely-many finite-rank free groups.

▶ **Corollary 38.** *$\forall G \in \widehat{\Pi}_2, \exists K \in \mathbb{N}_{\geq 1}, \exists C \in \mathbb{R}_{\geq 1}$, such that G virtually has an algebraic $[K, 2, C^{-n}]$-DFR.*

We conclude with a "generic" construction that covers all groups that have algebraic DFRs. We remark that while this does partially subsume all other results in this section, it does not do so completely, as the earlier constructions of DFRs, for certain particular groups, yield better parameters.

▶ **Theorem 39.** *Consider a group $G = \langle S|R \rangle$, with S finite, where G is not the trivial group. Suppose G has a faithful representation $\pi : G \to \mathrm{U}(l, \overline{\mathbb{Q}})$. Then π has a (unique, up to isomorphism) set of irreducible subrepresentations $\{\pi_j : G \to \mathrm{U}(d_j, \overline{\mathbb{Q}})\}_{j=1}^{m}$ such that $\pi \cong \pi_1 \oplus \cdots \oplus \pi_m$. Let $d_{\max} = \max_j d_j$. Define the value d as follows: if $\bigcap_j \mathrm{Pker}(\pi_j) = \{1_G\}$, let $d = d_{\max}$, otherwise, let $d = d_{\max} + 1$. Partition the non-trivial π_j into isomorphism classes (i.e., only consider those π_j which are not the trivial representation; π_{j_1} and π_{j_2} belong to the same isomorphism class if $\pi_{j_1} \cong \pi_{j_2}$) and let k denote the number of isomorphism classes that appear. Then $\exists C \in \mathbb{R}_{\geq 1}$ such that G has an algebraic $[k, d, C^{-n}]$-DFR.*

Proof. Notice that, as G is not the trivial group, $d \geq 2$. Assume that the π_j are ordered such that $\pi_1, \ldots, \pi_k$ are representatives of the k distinct isomorphism classes of the non-trivial representations that appear among the π_j. For each $j \in \{1, \ldots, k\}$, define the representation $\rho_j = \pi_j \oplus \mathbf{1}_{d-d_j} : G \to \mathrm{U}(d, \overline{\mathbb{Q}})$. By Lemma 21, $\forall j \in \{1, \ldots, k\}, \exists C_j \in \mathbb{R}_{\geq 1}$ such that, $\forall g \notin \mathrm{Pker}(\rho_j)$, $|\chi_{\rho_j}(g)| \leq d - C_j^{-l(g)}$. Set $C = \max_j C_j$.

Next, notice that $\bigcap_j \mathrm{Pker}(\rho_j) = \{1_G\}$. If $\bigcap_j \mathrm{Pker}(\pi_j) = \{1_G\}$, then this is obvious. Suppose $\bigcap_j \mathrm{Pker}(\pi_j) \neq \{1_G\}$. Then $d = d_{\max} + 1 > d_j$, $\forall j$, which implies $\rho_j = \pi_j \oplus \mathbf{1}_{t_j}$, where $t_j := d - d_j \geq 1$. Therefore, for each j, $\rho_j(G) \cap Z(\mathrm{U}(d, \overline{\mathbb{Q}})) = I_d$, and so, by definition, $\mathrm{Pker}(\rho_j) = \ker(\rho_j)$. As π is faithful, $\{1_G\} = \bigcap_{j=1}^{m} \ker(\pi_j) = \bigcap_{j=1}^{k} \ker(\rho_j) = \bigcap_{j=1}^{k} \mathrm{Pker}(\rho_j)$.

Thus, $\forall g \in G_{\neq 1}$, $\exists j$ such that $g \notin \mathrm{Pker}(\rho_j)$, which implies $|\chi_{\rho_j}(g)| \leq d - C_j^{-l(g)} \leq d - C^{-l(g)}$. Therefore, $\{\rho_1, \ldots, \rho_k\}$ is an algebraic $[k, d, C^{-n}]$-DFR for G. ◀

3.3 Projective DFRs

A DFR $\mathcal{F} = \{\rho_1, \ldots, \rho_j\}$ of a group G is a set of unitary representations of G, i.e., group homomorphisms $\rho_j : G \to \mathrm{U}(d)$. We next consider a slight generalization. A *projective* unitary representation of G is a group homomorphism $\rho : G \to \mathrm{PU}(d) = \mathrm{U}(d)/Z(\mathrm{U}(d))$. We may (non-uniquely) lift any such ρ to a function $\widehat{\rho} : G \to \mathrm{U}(d)$ (i.e., $\gamma \circ \widehat{\rho} = \rho$, where $\gamma : \mathrm{U}(d) \to \mathrm{PU}(d)$ is the canonical projection). Note that $\widehat{\rho}$ is not necessarily a group homomorphism and that certain projective representations ρ cannot be lifted to an ordinary representation. However, for any two lifts, $\widehat{\rho}_1$ and $\widehat{\rho}_2$, of ρ, we have $|\chi_{\widehat{\rho}_1}(g)| = |\chi_{\widehat{\rho}_2}(g)|$, $\forall g \in G$. Therefore, the function $|\chi_\rho(\cdot)| : G \to \mathbb{R}$ given by $|\chi_\rho(g)| = |\chi_{\widehat{\rho}}(g)|$ is well-defined.

We then define a $[k, d, \tau, \mathbb{A}]$-PDFR as a set of projective representations $\mathcal{F} = \{\rho_1, \ldots, \rho_j\}$ that satisfies Definition 16 where "representation" is replaced by "projective representation" in that definition. As we will observe in the following section, the same process that allows a DFR for a group G to be used to produce a 2QCFA for the word problem W_G, can also be applied to a PDFR. If a PDFR consists entirely of representations into $\mathrm{PU}(d, \overline{\mathbb{Q}}) = \mathrm{U}(d, \overline{\mathbb{Q}})/Z(\mathrm{U}(d, \overline{\mathbb{Q}}))$, we say it is an *algebraic* PDFR. The following variant of Theorem 39 follows by a precisely analogous proof.

▶ **Theorem 40.** *Suppose the group $G = \langle S|R \rangle$, with S finite, has a family $\mathcal{F} = \{\rho_1, \ldots, \rho_k\}$ of projective representations $\rho_j : G \to \mathrm{PU}(d, \overline{\mathbb{Q}})$, such that $\bigcap_j \ker(\rho_j) = \{1_G\}$. Then $\exists C \in \mathbb{R}_{\geq 1}$ such that $\mathcal{F}$ is an algebraic $[k, d, C^{-n}]$-PDFR for G.*

3.4 Unbounded-Error DFRs

If $\mathcal{F} = \{\rho_1, \ldots, \rho_k\}$ is a DFR for a group G, then $\bigcap_j \mathrm{Pker}(\rho_j) = \{1_G\}$. However, a crucial element in the definition of a DFR is the requirement that, much more strongly, all $g \in G_{\neq 1}$ are "far" from being in $\bigcap_j \mathrm{Pker}(\rho_j)$; in particular, if $\mathcal{F}$ is a $[k, d, \tau, \mathbb{A}]$-DFR, then $\forall g \in G_{\neq 1}, \exists j$ such that $|\chi_{\rho_j}(g)| \leq d - \tau(l(g))$. This requirement is essential in order for our construction of a 2QCFA, that recognizes W_G using a DFR for G, to operate with *bounded* error. We next consider a generalization of a DFR, where this requirement is removed, which will then yield a 2QCFA that recognizes W_G with *unbounded* error.

We say $\mathcal{F} = \{\rho_1, \dots, \rho_k\}$ is an *unbounded-error* $[k, d, \mathbb{A}]$-DFR for a group $G = \langle S|R \rangle$ if the conditions of Definition 16 hold, where Definition 16(b) is replaced by Definition 16(b)': $\forall g \in G_{\neq 1}$, $\exists j$ such that $|\chi_{\rho_j}(g)| < d$. This condition is equivalent to $\bigcap_j \operatorname{Pker}(\rho_j) = \{1_G\}$.

Note that, by Lemma 21, any algebraic unbounded-error $[k, d]$-DFR is also an algebraic $[k, d, C^{-n}]$-DFR, for some $C \in \mathbb{R}_{\geq 1}$; furthermore, as noted in the discussion following Definition 18, only a finitely generated abelian group could have a diagonal unbounded-error $[k, d]$-DFR, and all finitely generated abelian groups were shown to have DFRs in Theorem 34. Therefore, in order to obtain something new, we must consider unbounded-error DFRs that are neither algebraic nor diagonal. Due to space restrictions, we omit the proof of the following theorem, which may be found in the full version [28].

▶ **Theorem 41.** $\forall G \in \widehat{\Pi}_3, \exists k \in \mathbb{N}$ *such that* G *virtually has an unbounded-error* $[k, 2, \widetilde{\mathbb{C}}]$*-DFR.*

4 Recognizing the Word Problem of a Group with a 2QCFA

Consider a group $G = \langle S|R \rangle$, with S finite. As before, let $\Sigma = S \sqcup S^{-1}$, let $\phi : \Sigma^* \to G$ denote the natural map that takes each string in Σ^* to the element of G that it represents, and let $W_G := W_{G=\langle S|R \rangle} = \{w \in \Sigma^* : \phi(w) = 1_G\}$ denote the word problem of G with respect to the given presentation. Suppose $\mathcal{F} = \{\rho_1, \dots, \rho_k\}$ is a $[k, d, \tau, \mathbb{A}]$-DFR (or PDFR) for G. By Proposition 17, if $w \in W_G$, then $|\chi_{\rho_j}(\phi(w))| = d$, $\forall j$, and if $w \notin W_G$, then $\exists j$ where $|\chi_{\rho_j}(\phi(w))| \leq d - \tau(l(\phi(w)))$. Let $G_j = \{g \in G : |\chi_{\rho_j}(g)| \leq d - \tau(l(g))\}$. A 2QCFA can recognize W_G by checking if $\phi(w) \in \bigcup_j G_j = G_{\neq 1}$. The well-known Hadamard test may be used to estimate $\chi_{\rho_j}(\phi(w)) = \operatorname{Tr}(\rho_j(\phi(w)))$; however, as we wish to produce a 2QCFA that has as few quantum states as possible, we wish to avoid the use of ancilla, and so we follow a slightly different approach. We begin by defining several useful 2QCFA subroutines.

▶ **Definition 42.** Suppose M is a 2QCFA with $d \geq 2$ quantum basis states $Q = \{q_1, \dots, q_d\}$, quantum start state $q_1 \in Q$, and alphabet Σ.

(a) Suppose $|\psi_1\rangle = \sum_q \alpha_q |q\rangle$ and $|\psi_2\rangle = \sum_q \beta_q |q\rangle$, where $\alpha_q, \beta_q \in \overline{\mathbb{Q}}, \forall q \in Q$. There are (many) $t \in \operatorname{U}(d, \overline{\mathbb{Q}})$ such that $t|\psi_1\rangle = |\psi_2\rangle$. Let $\mathcal{T}_{|\psi_1\rangle \to |\psi_2\rangle}$ denote an arbitrary such t.

(b) Let $\pi : G \to \operatorname{U}(d)$ be a representation of G and let $|\psi\rangle = \sum_q \beta_q |q\rangle$, where $\beta_q \in \overline{\mathbb{Q}}$, $\forall q \in Q$. Then the *unitary round* $\mathcal{U}(\pi, |\psi\rangle)$ is a particular sub-computation of M on w, defined as follows. The round begins with the quantum register in the superposition $|q_1\rangle$ and the tape head at the right end of the tape. On reading $\#_R$, M performs the unitary transformation $\mathcal{T}_{|q_1\rangle \to |\psi\rangle}$ to its quantum register, and moves its head to the left. On reading a symbol $\sigma \in \Sigma$, M performs the unitary transformation $\pi(\phi(\sigma))$ to the quantum register and moves its head left. When the tape head first reaches the left end of the tape (i.e., the first time the symbol $\#_L$ is read), M performs the identity transformation to its quantum register, and does not move its head, at which point the round ends. As ϕ is a (monoid) homomorphism and π is a (group) homomorphism, we immediately conclude that, at the end of the round, the quantum register is in the superposition $\pi(\phi(w))|\psi\rangle$.

(c) For $t \in \operatorname{U}(d)$, a *measurement round* $\mathcal{M}(\pi, |\psi\rangle, t)$ is a sub-computation of M that begins with the unitary round $\mathcal{U}(\pi, |\psi\rangle)$. Then M performs the unitary transformation t, and does not move its head. After which M performs the quantum measurement specified by the partition $B = \{B_0, B_1\}$ of Q given by $B_0 = \{q_2, \dots, q_d\}$ and $B_1 = \{q_1\}$, producing some *result* $r \in \{0, 1\}$; then M records r in its classical state, and does not move its head, at which point the round is over.

▶ **Lemma 43.** *Consider a group $G = \langle S|R\rangle$, with S finite, and let $W_G = W_{G=\langle S|R\rangle}$. The following statements hold.*

(i) *If G has a diagonal $[k, d, C_1 n^{-C_2}, \mathbb{A}]$-DFR (or PDFR), for some $C_1, C_2 \in \mathbb{R}_{>0}$, then $\forall \epsilon \in \mathbb{R}_{>0}$, $W_G \in \mathsf{coR2QCFA}(n^{\lceil C_2 \rceil + 2}, \epsilon, d, \overline{\mathbb{Q}} \cup \mathbb{A})$.*

(ii) *If G has a $[k, d, C_1^{-n}, \mathbb{A}]$-DFR (or PDFR), for some $C_1 \in \mathbb{R}_{\geq 1}$, then $\forall \epsilon \in \mathbb{R}_{>0}$, $\exists C_2 \in \mathbb{R}_{\geq 1}$ such that $W_G \in \mathsf{coR2QCFA}(C_2^n, \epsilon, d, \overline{\mathbb{Q}} \cup \mathbb{A})$.*

(iii) *If G has an unbounded-error $[k, d, \mathbb{A}]$-DFR (or PDFR), then $W_G \in \mathsf{coN1QFA} \cap \mathsf{coN2QCFA}(n, d, \overline{\mathbb{Q}} \cup \mathbb{A})$.*

Proof Sketch. Suppose $\mathcal{F} = \{\rho_1, \dots, \rho_k\}$ is a $[k, d, \tau, \mathbb{A}]$-DFR of G. Consider any $w \in \Sigma^*$. A 2QCFA M can perform a constant number of measurement rounds (i.e., the number of rounds only depends on k and d, not on $|w|$) using any representation ρ_j such that the following holds: (1) if $\phi(w) \in G_j \subseteq G_{\neq 1}$, then, with probability $\Omega(\tau(|w|))$, the results of those measurement rounds will allow M to be able to conclude *with certainty* that $w \notin W_G$, (2) if $\phi(w) = 1_G \notin G_j$, then the results of those measurement rounds will *never* cause M to incorrectly conclude that $w \notin W_G$. After running this procedure approximately $\tau(n)$ times, for each j, the following holds: (1) if $\phi(w) \in G_{\neq 1} = \bigcup_j G_j$, then $\phi(w) \in G_j$ for at least some j, and so, with probability $\Omega(1)$, M is able to conclude (with certainty) that $w \notin W_G$, (2) if $\phi(w) = 1_G$, then M will never incorrectly conclude that $w \notin W_G$. As soon as M performs a measurement round whose result allows it to conclude that $w \notin W_G$, M immediately rejects. In order to correctly accept all $w \in W_G$, M will run a procedure between measurement rounds that will cause it to accept *any* input w with some small probability, and otherwise continue; by setting this acceptance probability small enough, we assure that any $w \notin W_G$ is not (incorrectly) accepted with high probability. A formal proof can be found in the full version [28]. ◀

Moreover, if H is a finite-index subgroup of G, a 2QCFA that recognizes W_G can be constructed from a 2QCFA that recognizes W_H.

▶ **Lemma 44.** *Consider a group $H = \langle S_H|R_H\rangle$, with S_H finite, and suppose that A_H is a 2QCFA that recognizes W_H, which operates in the manner of our proof of Lemma 43. Further suppose G is a group such that $H \leq G$ and $[G : H]$ is finite. Then G admits a presentation $G = \langle S_G|R_G\rangle$, with S_G finite, such that there is a 2QCFA A_G that recognizes W_G. Moreover, A_G has the same acceptance criteria, asymptotic expected running time, number of quantum basis states, and class of transition amplitudes as A_H.*

Using the above results, and the constructions of DFR from Section 3, the theorems stated in Section 1.1 concerning the recognizability of word problems by 2QCFA easily follow; proofs of the above results and of these theorems appear in the full version [28].

5 Discussion

In this paper, we have shown that 2QCFA can recognize the word problems of many groups. In particular, let $\widehat{\Pi}_1$ (resp. $\widehat{\Pi}_2$) denote the collection of all finitely generated groups that are virtually abelian (resp. virtually a subgroup of a direct product of finitely-many finite-rank free groups), and let $\mathcal{Q}$ denotes the class of groups for which Theorem 10 applies. Then a 2QCFA, with a single-qubit quantum register and algebraic number transition amplitudes, can recognize, with one-sided bounded error, the word problem W_G of any $G \in \widehat{\Pi}_1$ (resp. $G \in \widehat{\Pi}_2$) in expected polynomial (resp. exponential) time. Moreover, if allowed a quantum register of any constant size, such a 2QCFA may recognize the word problem of any group $G \in \mathcal{Q}$ with one-sided bounded error in expected exponential time.

In a companion paper [27], we establish a lower bound on the running time of any 2QCFA (with any size quantum register and no restrictions placed on its transition amplitudes) that recognizes a word problem W_G with bounded error (even under the more generous notion of two-sided bounded error); more strongly, we establish a lower bound on the running time of *any quantum Turing machine* that uses *sublogarithmic* space, though we will not discuss that here. In particular, we show that, $\forall G \in \mathcal{Q} \setminus \widehat{\Pi}_1$, W_G *cannot* be recognized by such a 2QCFA is expected time $2^{o(n)}$. Therefore, the algorithm exhibited in this paper for recognizing the word problem of any group $G \in \mathcal{Q} \setminus \widehat{\Pi}_1$ has (essentially) optimal expected running time; moreover, we have obtained the first provable separation between the classes of languages recognizable with bounded error by 2QCFA in expected exponential time and in expected subexponential time. In that same paper, we also show that if a 2QCFA of this most general type recognizes a word problem W_G in expected polynomial time, then $G \in \mathcal{G}_{vNilp}$, where $\mathcal{G}_{vNilp}$ denotes the finitely generated virtually nilpotent groups, and $\widehat{\Pi}_1 \subsetneq \mathcal{G}_{vNilp}$. This naturally raises the following question.

▶ Open Problem 1. Is there a group $G \in (\mathcal{G}_{vNilp} \setminus \widehat{\Pi}_1)$ such that W_G can be recognized by a 2QCFA with bounded error in expected polynomial time?

We have shown that the (three-dimensional discrete) Heisenberg group $H \in (\mathcal{G}_{vNilp} \setminus \widehat{\Pi}_1)$ is "complete" for this question, in the sense that if W_H *cannot* be recognized with bounded error by a 2QCFA in expected polynomial time, then no such G can [27].

Let $\mathcal{G}_{vSolvLin}$ denote the finitely generated virtually solvable linear groups over a field of characteristic zero, and note that $\mathcal{G}_{vNilp} \subsetneq \mathcal{G}_{vSolvLin}$. Furthermore, note that $W_G \in \mathsf{L}$, $\forall G \in \mathcal{G}_{vSolvLin}$ [20]. However, every $G \in \mathcal{G}_{vSolvLin} \setminus \widehat{\Pi}_1$ *does not* have a faithful finite-dimensional unitary representation (see, for instance, [33, Proposition 2.2]) and, therefore, does not have a DFR (even an unbounded-error DFR); this prevents the techniques of this paper from producing a 2QCFA that recognizes the corresponding W_G.

▶ Open Problem 2. Is there a finitely generated group G that does not have a faithful finite-dimensional unitary representation (for example, any $G \in \mathcal{G}_{vSolvLin} \setminus \widehat{\Pi}_1$ or any finitely generated infinite Kazhdan group) such that W_G can be recognized with bounded error by a 2QCFA at all (i.e., in any time bound)?

Consider the group $\mathbb{Z} * \mathbb{Z}^2 \in \Sigma_2 \subsetneq \widehat{\Pi}_3$, and note that $\mathbb{Z} * \mathbb{Z}^2 \notin \widehat{\Pi}_2$. The complexity of $W_{\mathbb{Z}*\mathbb{Z}^2}$ has been considered by many authors and it is conjectured that $W_{\mathbb{Z}*\mathbb{Z}^2} \notin \mathsf{poly{-}CFL}$ [7] (cf. [8]) and that $W_{\mathbb{Z}*\mathbb{Z}^2} \notin \mathsf{coCFL}$ [18]. By Theorem 11, $W_{\mathbb{Z}*\mathbb{Z}^2}$ is recognizable with one-sided *unbounded* error by a 2QCFA. We ask the following questions.

▶ Open Problem 3. Can $W_{\mathbb{Z}*\mathbb{Z}^2}$ be recognized by a 2QCFA with bounded error? More generally, is $W_{\mathbb{Z}*\mathbb{Z}^r}$ recognizable by a 2QCFA with bounded error, $\forall r \in \mathbb{N}$?

▶ Open Problem 4. Does $\mathbb{Z} * \mathbb{Z}^2$ have an algebraic DFR. More generally, does $\mathbb{Z} * \mathbb{Z}^r$ have an algebraic DFR, $\forall r \in \mathbb{N}$? Even more generally, is the class of groups which have algebraic DFRs closed under free product?

References

1 Andris Ambainis and John Watrous. Two-way finite automata with quantum and classical states. *Theoretical Computer Science*, 287(1):299–311, 2002.

2 Andris Ambainis and Abuzer Yakaryılmaz. Automata and quantum computing. *arXiv preprint*, 2015. arXiv:1507.01988.

3 Ao V Anisimov. Group languages. *Cybernetics and Systems Analysis*, 7(4):594–601, 1971.

4 Alan Baker. *Transcendental number theory.* Cambridge university press, 1990.

5 J-C Birget, A Yu Ol'shanskii, Eliyahu Rips, and Mark V Sapir. Isoperimetric functions of groups and computational complexity of the word problem. *Annals of Mathematics*, pages 467–518, 2002.

6 Alex Brodsky and Nicholas Pippenger. Characterizations of 1-way quantum finite automata. *SIAM Journal on Computing*, 31(5):1456–1478, 2002.

7 Tara Brough. Groups with poly-context-free word problem. *Groups Complexity Cryptology*, 6(1):9–29, 2014.

8 Tullio Ceccherini-Silberstein, Michel Coornaert, Francesca Fiorenzi, Paul E Schupp, and Nicholas WM Touikan. Multipass automata and group word problems. *Theoretical Computer Science*, 600:19–33, 2015.

9 Cynthia Dwork and Larry Stockmeyer. A time complexity gap for two-way probabilistic finite-state automata. *SIAM Journal on Computing*, 19(6):1011–1023, 1990.

10 Cynthia Dwork and Larry Stockmeyer. Finite state verifiers i: The power of interaction. *Journal of the ACM (JACM)*, 39(4):800–828, 1992.

11 Rūsiņš Freivalds. Probabilistic two-way machines. In *International Symposium on Mathematical Foundations of Computer Science*, pages 33–45. Springer, 1981.

12 Alex Gamburd, Dmitry Jakobson, and Peter Sarnak. Spectra of elements in the group ring of su (2). *Journal of the European Mathematical Society*, 1(1):51–85, 1999.

13 Albert G Greenberg and Alan Weiss. A lower bound for probabilistic algorithms for finite state machines. *Journal of Computer and System Sciences*, 33(1):88–105, 1986.

14 Lov K Grover. A fast quantum mechanical algorithm for database search. *Proceedings of the Twenty-Eighth Annual ACM Symposium of Theory of Computing*, pages 212–219, 1996.

15 Aram W Harrow, Avinatan Hassidim, and Seth Lloyd. Quantum algorithm for linear systems of equations. *Physical review letters*, 103(15):150502, 2009.

16 Thomas Herbst. On a subclass of context-free groups. *RAIRO-Theoretical Informatics and Applications-Informatique Théorique et Applications*, 25(3):255–272, 1991.

17 Derek F Holt, Matthew D Owens, and Richard M Thomas. Groups and semigroups with a one-counter word problem. *Journal of the Australian Mathematical Society*, 85(2):197–209, 2008.

18 Derek F Holt, Sarah Rees, Claas E Röver, and Richard M Thomas. Groups with context-free co-word problem. *Journal of the London Mathematical Society*, 71(3):643–657, 2005.

19 Emmanuel Kowalski. *An introduction to the representation theory of groups*, volume 155. American Mathematical Society, 2014.

20 Richard J Lipton and Yechezkel Zalcstein. Word problems solvable in logspace. *Journal of the ACM (JACM)*, 24(3):522–526, 1977.

21 Clara Löh. *Geometric group theory*. Springer, 2017.

22 Cristopher Moore and James P Crutchfield. Quantum automata and quantum grammars. *Theoretical Computer Science*, 237(1-2):275–306, 2000.

23 David E Muller and Paul E Schupp. Groups, the theory of ends, and context-free languages. *Journal of Computer and System Sciences*, 26(3):295–310, 1983.

24 Michael A Nielsen and Isaac Chuang. Quantum computation and quantum information, 2002.

25 Michael O Rabin. Probabilistic automata. *Information and control*, 6(3):230–245, 1963.

26 Michael O Rabin and Dana Scott. Finite automata and their decision problems. *IBM journal of research and development*, 3(2):114–125, 1959.

27 Zachary Remscrim. Lower bounds on the running time of two-way quantum finite automata and sublogarithmic space quantum turing machines. *Electronic Colloquium on Computational Complexity (ECCC)*, 26:182, 2019. URL: `https://eccc.weizmann.ac.il/report/2019/182`.

28 Zachary Remscrim. The power of a single qubit: Two-way quantum/classical finite automata and the word problem for linear groups. *Electronic Colloquium on Computational Complexity (ECCC)*, 26:107, 2019. URL: `https://eccc.weizmann.ac.il/report/2019/107`.

29 AC Say and Abuzer Yakaryilmaz. Magic coins are useful for small-space quantum machines. *Quantum Information & Computation*, 17(11-12):1027–1043, 2017.

30 Wolfgang M Schmidt. Simultaneous approximation to algebraic numbers by rationals. *Acta Mathematica*, 125(1):189–201, 1970.

31 Peter W Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *Proceedings 35th annual symposium on foundations of computer science*, pages 124–134. Ieee, 1994.

32 John Stallings. A finitely presented group whose 3-dimensional integral homology is not finitely generated. *American Journal of Mathematics*, 85(4):541–543, 1963.

33 Andreas Thom. Convergent sequences in discrete groups. *Canadian Mathematical Bulletin*, 56(2):424–433, 2013.

34 John Watrous. On the complexity of simulating space-bounded quantum computations. *Computational Complexity*, 12(1-2):48–84, 2003.

35 John Watrous. *The theory of quantum information.* Cambridge University Press, 2018.

36 Abuzer Yakaryilmaz and AC Cem Say. Languages recognized by nondeterministic quantum finite automata. *Quantum Information & Computation*, 10(9):747–770, 2010.

37 Abuzer Yakaryilmaz and AC Cem Say. Succinctness of two-way probabilistic and quantum finite automata. *Discrete Mathematics and Theoretical Computer Science*, 12(4):19–40, 2010.

Hardness Results for Constant-Free Pattern Languages and Word Equations

Aleksi Saarela
Department of Mathematics and Statistics, University of Turku, Finland
amsaar@utu.fi

Abstract

We study constant-free versions of the inclusion problem of pattern languages and the satisfiability problem of word equations. The inclusion problem of pattern languages is known to be undecidable for both erasing and nonerasing pattern languages, but decidable for constant-free erasing pattern languages. We prove that it is undecidable for constant-free nonerasing pattern languages. The satisfiability problem of word equations is known to be in PSPACE and NP-hard. We prove that the nonperiodic satisfiability problem of constant-free word equations is NP-hard. Additionally, we prove a polynomial-time reduction from the satisfiability problem of word equations to the problem of deciding whether a given constant-free equation has a solution morphism α such that $\alpha(xy) \neq \alpha(yx)$ for given variables x and y.

2012 ACM Subject Classification Mathematics of computing → Combinatorics on words

Keywords and phrases Combinatorics on words, pattern language, word equation

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.140

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

1 Introduction

The first topic of this article is pattern languages. If we fix an alphabet of variables and an alphabet of constants, we can define a pattern as a word consisting of variables and constant letters (constants are often called terminals). Given a pattern U, the nonerasing pattern language of U is the set of images of U under all morphism that map the variables to nonempty constant words and preserve the constants. The erasing pattern language of U is defined in a similar way, except that the variables can be mapped also to the empty word. Nonerasing pattern languages were introduced by Angluin [1] and erasing pattern languages by Shinohara [33].

There are many interesting algorithmic questions about pattern languages, and they are related to applications such as pattern matching and inductive inference. The membership problem of pattern languages, which can also be called the matching problem, is NP-complete in both the nonerasing and erasing case, and so are many of its variations, see, e.g., [10] and [24]. Checking the emptiness of the intersection of two pattern languages is essentially a special case of the satisfiability problem of word equations (discussed later in the introduction), and can therefore be done in polynomial space.

The equivalence problem of two pattern languages is almost trivially decidable in the nonerasing case: If the alphabet of constants is not unary, the nonerasing pattern languages of two patterns are the same if and only if the patterns are identical up to a renaming of the variables [1] (if the alphabet of constants is unary, the problem is a bit more complicated but still easily decidable). In the erasing case, however, the decidability of the equivalence problem is an open question.

The inclusion problem of pattern languages was mentioned as an open question in [1]. It was proved to be undecidable in both the nonerasing and erasing case by Jiang, Salomaa, Salomaa and Yu [18]. They reduced the undecidable problem of determining whether a

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 140; pp. 140:1–140:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

nondeterministic two-counter automaton without input has an accepting computation to the inclusion problem of erasing pattern languages, which they then reduced to the inclusion problem of nonerasing pattern languages. Both proofs are very complicated. They also proved that the inclusion problem of constant-free erasing pattern languages is decidable. This proof is simpler but not trivial.

Analyzing the equivalence and inclusion of pattern languages naturally leads to eight decision problems depending on whether we consider the nonerasing or erasing case and whether we consider patterns with or without constants. By the results mentioned above, the decidability status of six of these eight problems was known after [18], but two questions were left open: Is the equivalence problem decidable for erasing pattern languages, and is the inclusion problem decidable for constant-free nonerasing pattern languages? In [18], a positive answer was conjectured for the first question. The second one was only stated as an open problem with no conjecture. The questions have remained open since then.

The results in [18] were very interesting for many reasons. First, they solved a famous problem that had been open for many years. Second, the inclusion problem is important for inductive inference of pattern languages, see, e.g., the articles of Ng and Shinohara [25] and Reidenbach [28]. Third, nonerasing pattern languages became perhaps the first example of a family of formal languages with a trivially decidable equivalence problem but undecidable inclusion problem. Some other families where the equivalence problem is decidable but the inclusion problem is undecidable are the family of languages accepted by finite deterministic multitape automata (decidability of equivalence proved by Harju and Karhumäki [12]) and the family of deterministic context-free languages (decidability of equivalence proved later by Sénizergues [32]), but in these cases the undecidability result is the easier one. It is also interesting that for the equivalence problem of patterns, the nonerasing case is easier, while for the inclusion problem of constant-free patterns, the erasing case is easier.

A lot of further research on the inclusion problem has been done. For example, Freydenberger and Reidenbach [11] proved that the inclusion problem remains undecidable if the size of the alphabet of constants is fixed to be a positive integer k, as long as $k \geq 2$ in the erasing case and $k \geq 4$ in the nonerasing case. Both variants of the inclusion problem are decidable if the alphabet of constants is unary or infinite. Bremer and Freydenberger [4] proved stronger results: Both the erasing and the nonerasing inclusion problem are undecidable even for a fixed number of variables, as long as this number is large enough, and even if the size of the alphabet of constants is two. This result holds also if the second pattern is required to be constant-free.

In this article, we answer one of the open questions by proving that the inclusion problem of constant-free nonerasing pattern languages is undecidable. The result holds even for a fixed number of variables, as long as this number is large enough, and even if the size of the alphabet of constants is two. Among the problems we have discussed, our new result provides the first example where the constant-free version of a problem is undecidable, and the first example where the decidability status of the nonerasing and erasing version has been proved to be different. See Table 1 for a summary.

Let us now move to the topic of equations. A word equation can be defined as a pair (U, V) of patterns, and a solution can be defined as a constant-preserving morphism α such that $\alpha(U) = \alpha(V)$. Like in the case of pattern languages, there are a couple of variations of word equations. First, we can study either the general case of equations with constants, or the restricted case of constant-free equations. Algorithmic questions are usually studied for equations with constants. Some other questions, such as independence [13, 26] and parameterizability [14, 30], are more often studied for constant-free equations. Second, we

Table 1 The decidability of the equivalence (=) and inclusion (⊆) problem of nonerasing (NE) and erasing (E) languages of constant-free patterns (CF) and patterns with constants (C). The decidable problems have been marked with a plus and the undecidable ones with a minus. The new result proved in this article has been circled.

=	NE	E
CF	+	+
C	+	?

⊆	NE	E
CF	$\ominus$	+
C	−	−

can either allow a solution α to be erasing or require it to be nonerasing. This does not usually make as big of a difference as in the case of the equivalence and inclusion of pattern languages, but it can have an effect on some things, for example, the size of largest known independent systems of equations [20]. We allow solutions to be erasing in this article.

The satisfiability problem of word equations, that is, the problem of deciding whether a given word equation (or a system of equations) has a solution, is one of the major algorithmic problems on words. The satisfiability problem was proved to be decidable by Makanin [23]. A survey of Makanin's algorithm can be found in [8]. The first PSPACE algorithm was given by Plandowski [27]. Jeż gave a simpler PSPACE algorithm [16] and proved that the satisfiability problem is in NSPACE(n) [17]. Linear integer programming and the membership problem of pattern languages can both be easily reduced to the satisfiability problem, so it is NP-hard. The NP-completeness of the satisfiability problem is a big open question.

Many special cases have been analyzed. For one-variable equations, the satisfiability problem can be solved in linear time, as proved by Jeż [15], for two-variable equations, in time $O(n^5)$, as proved by Dąbrowski and Plandowski [7], and for quadratic equations, it is NP-hard, as proved by Robson and Diekert [29]. Some other results can be found in the article of Day, Manea and Nowotka [6]. Some quite powerful generalizations of word equations were proved to be solvable in polynomial space by Diekert and Elder [9].

There is not much research about the satisfiability of constant-free word equations. Constant-free equations always have solutions (at least the one mapping all variables to the empty word, and usually infinitely many other trivial ones), so the natural decision problem for them is to ask whether an equation has a nontrivial solution, for some definition of "nontrivial". It is known that deciding whether a constant-free three-variable equation has a nonperiodic solution is in NP [30]. Nontrivial constant-free equations on one or two variables have only periodic solutions.

Constant-free equations might seem much simpler than general ones, but we prove that deciding whether a given constant-free equation has a nonperiodic solution is NP-hard, and for a given constant-free equation and given variables x, y, deciding whether there exists a solution α such that $\alpha(xy) \neq \alpha(yx)$ is as hard as the general version of the satisfiability problem.

Our proofs are based on the idea of simulating constants by variables in a certain way: We replace the constants by words consisting of new variables, and then we make sure that these words behave sufficiently much like constants by adding prefixes and suffixes to the patterns or new equations to a system of equations. The details differ quite a bit depending on the problem.

2 Preliminaries

First, we recall some standard notation, definitions, and results related to combinatorics on words and free monoids. For more, see [5, 21, 3].

The symbols Σ, Γ, Ξ are always used to denote alphabets. Alphabets are usually finite, but at one point, we use an infinite alphabet. For $k \geq 1$, let $\Sigma_k = \{0, \ldots, k-1\}$. When we need an alphabet of size k, we often use specifically the alphabet Σ_k. The empty word is denoted by ε.

A word U is a *factor* of a word V if there exist words X, Y such that $V = XUY$. If we can choose $X = \varepsilon$, then U is a *prefix*, if we can choose $Y = \varepsilon$, then U is a *suffix*, and if we can choose $X \neq \varepsilon \neq Y$, then U is an *internal factor* of V.

A nonempty word is *primitive* if it is not a power of a shorter word. If $U = V^n$ and V is primitive, then V is a *primitive root* of U.

A language $M \subseteq \Sigma^*$ is a *submonoid* of Σ^* if it is closed under concatenation and contains ε. Let M_1 and M_2 be submonoids of Σ^* and Γ^*. A mapping $\alpha : M_1 \to M_2$ is a *morphism* if $\alpha(UV) = \alpha(U)\alpha(V)$ for all $U, V \in M_1$. The most common case is $M_1 = \Sigma^*$ and $M_2 = \Gamma^*$.

Let $L \subseteq M_1$. A morphism $\alpha : M_1 \to M_2$ is *nonerasing* if $\alpha(x) \neq \varepsilon$ for all $x \neq \varepsilon$, *L-preserving* if $\alpha(x) = x$ for all $x \in L$, *L-periodic* if $\alpha(xy) = \alpha(yx)$ for all $x, y \in L$, and *L-nonperiodic* if $\alpha(xy) \neq \alpha(yx)$ for some $x, y \in L$.

In the following theorem, we have collected some folklore results related to these definitions.

▶ **Theorem 1.** *Let U and V be words.*

1. *$UV = VU$ if and only if there exists a word R such that $U, V \in R^*$.*
2. *Every nonempty word has a unique primitive root.*
3. *If U is primitive, then it is not an internal factor of U^2.*
4. *If $|\Sigma| = 2$, then every Σ-nonperiodic morphism $\Sigma^* \to \Gamma^*$ is injective.*

If every element of a submonoid M of Σ^* has a unique representation as a product of elements of some subset $B \subseteq M$, then M is a *free monoid* and B is its *basis.* Of course, Σ^* is a free monoid and Σ is its basis.

If M is a free monoid with a basis B, then every mapping $\alpha : B \to \Gamma^*$ can be extended to a morphism $M \to \Gamma^*$ in a unique way, and every injective mapping $\alpha : B \to \Gamma$ can be extended to an injective morphism $M \to \Gamma^*$. Moreover, if $L \subseteq B \cap \Gamma$, then every mapping $\alpha : B \setminus L \to \Gamma^*$ can be extended to an L-preserving morphism $M \to \Gamma^*$ in a unique way. Therefore, we often define a morphism α by just saying that it is L-preserving and giving the values $\alpha(x)$ for all $x \in B \setminus L$.

We need the following well-known characterization of free monoids.

▶ **Theorem 2.** *Let M be a submonoid of Σ^*. M is a free monoid if and only if there does not exist words $U, V, W \in \Sigma^*$ such that $U, VW, UV, W \in M$ but $V \notin M$.*

Let Ξ be an alphabet of variables and Σ an alphabet of constants. The alphabets Ξ and Σ are assumed to be disjoint. A *pattern over* (Ξ, Σ) is a word $U \in (\Xi \cup \Sigma)^+$. The pattern U is *constant-free* if $U \in \Xi^+$. The *nonerasing pattern language* of U, denoted by $L_{\mathrm{NE}}(U)$, is the set of images of U under all nonerasing Σ-preserving morphisms $(\Xi \cup \Sigma)^* \to \Sigma^*$.

In the following definitions, by an alphabet of size $\star$, we mean an alphabet of arbitrary finite size. For $m, n, k \in \mathbb{Z}_+ \cup \{\star\}$, we define the following decision problems related to patterns:

- Inclusion problem of nonerasing pattern languages $\mathrm{PatIncl}_{\mathrm{NE}}(m, n, k)$: Given alphabets Ξ_1, Ξ_2, Σ of sizes m, n, k, respectively, a pattern U over (Ξ_1, Σ), and a pattern V over (Ξ_2, Σ), decide whether $L_{\mathrm{NE}}(U) \subseteq L_{\mathrm{NE}}(V)$.
- Inclusion problem of constant-free nonerasing pattern languages $\mathrm{PatIncl}^{\mathrm{CF}}_{\mathrm{NE}}(m, n, k)$: Given alphabets Ξ_1, Ξ_2, Σ of sizes m, n, k, respectively, a constant-free pattern U over (Ξ_1, Σ), and a constant-free pattern V over (Ξ_2, Σ), decide whether $L_{\mathrm{NE}}(U) \subseteq L_{\mathrm{NE}}(V)$.

As mentioned in the introduction, $\mathrm{PatIncl}_{\mathrm{NE}}(\star, \star, \star)$ was shown to be undecidable already in [18]. We need the following stronger result from [4].

▶ **Theorem 3** ([4], Theorem 3.10). $\mathrm{PatIncl}_{\mathrm{NE}}(3, 2554, 2)$ *is undecidable.*

▶ Remark 4. Often, we use symbols from the end of the English alphabet (e.g., u, v, w, x, y, z in the next example) for ordinary variables, symbols from the beginning of the alphabet (e.g., a, b) for special variables that end up playing the role of constants in some sense, and nonnegative integers (e.g., $0, 1$) for the actual constants.

▶ **Example 5.** It is mentioned in several articles [18, 24] that one reason why the inclusion problem of nonerasing pattern languages is so difficult is that many unavoidability properties can be formulated in terms of pattern languages. In our new undecidability proof, an important role is played by one unavoidability result, although an extremely simple one: Every binary word of length at least 4 has a nonempty square factor, and therefore every binary word of length at least 6 has a nonempty internal square factor. By considering patterns over $(\{u, v, w, x, y, z\}, \Sigma_2)$, this can be expressed as $L_{\mathrm{NE}}(uvwxyz) \subseteq L_{\mathrm{NE}}(xy^2z)$. More specifically, we will need the fact that $L_{\mathrm{NE}}(x^2y^2) \smallsetminus L_{\mathrm{NE}}(xy^2z) = \{0011, 1100\}$.

A *word equation over* (Ξ, Σ) is a pair of patterns over (Ξ, Σ). A *solution* of an equation (U, V) is a Σ-preserving morphism $\alpha : (\Xi \cup \Sigma)^* \to \Sigma^*$ such that $\alpha(U) = \alpha(V)$. A *system of equations* is a set of equations. A *solution* of a system is a morphism that is a solution of every equation in the system.

An equation (U, V) is *constant-free* if $U, V \in \Xi^+$. For constant-free equations, Ξ-periodic solutions are considered trivial. We often call these solutions just *periodic* and the others *nonperiodic*.

▶ **Example 6.** Consider word equations over $(\{x, y, z\}, \Sigma_2)$. The equation $(x^2, y0y)$ has no solutions, because $|\alpha(x^2)|$ is even and $|\alpha(y0y)|$ is odd for all Σ_2-preserving morphisms α. The constant-free equation (x^2, yzy) has nonperiodic solutions α defined by

$$\alpha(x) = (PQ)^{i+1}P, \qquad \alpha(y) = (PQ)^iP, \qquad \alpha(z) = QPPQ$$

for all $P, Q \in \Sigma_2^*$, $PQ \neq QP$, $i \in \mathbb{Z}_{\geq 0}$, and periodic solutions α defined by

$$\alpha(x) = P^{i+j}, \qquad \alpha(y) = P^i, \qquad \alpha(z) = P^{2j}$$

for all $P \in \Sigma_2^*$, $i, j \in \mathbb{Z}_{\geq 0}$.

By the theorem of Lyndon and Schützenberger [22], if $\Xi = \{x, y, z\}$ and $k, m, n \geq 2$, then the word equation $(x^k, y^m z^n)$ has only periodic solutions. In other words, if α is a $\{y, z\}$-nonperiodic morphism, then $\alpha(y^m z^n)$ is primitive. In Theorem 7, we state a generalization of this result that we need later. It was proved by Spehner [34] and by Barbin-Le Rest and Le Rest [2]. A shorter proof can be found in [31].

▶ **Theorem 7.** *Let $W \in \{y, z\}^*$ be a primitive word that has at least two occurrences of both letters y and z. Let α be a $\{y, z\}$-nonperiodic morphism. Then $\alpha(W)$ is primitive.*

For $n, k \in \mathbb{Z}_+ \cup \{\star\}$, we define the following decision problems related to systems of word equations:

- Satisfiability problem of word equations $\mathrm{EqSat}(n, k)$: Given alphabets Ξ, Σ of sizes n, k, respectively, and a system S of equations over (Ξ, Σ), decide whether S has a solution.
- Nonperiodic satisfiability problem of constant-free word equations $\mathrm{EqSat}^{\mathrm{CF}}_{\mathrm{NP}}(n, k)$: Given alphabets Ξ, Σ of sizes n, k, respectively, and a system S of constant-free equations over (Ξ, Σ), decide whether S has a nonperiodic solution.
- Noncommuting satisfiability problem of constant-free word equations $\mathrm{EqSat}^{\mathrm{CF}}_{\mathrm{NC}}(n, k)$: Given alphabets Ξ, Σ of sizes n, k, respectively, a system S of constant-free equations over (Ξ, Σ), and variables $x, y \in \Xi$, decide whether S has a $\{x, y\}$-nonperiodic solution.

As mentioned in the introduction, $\mathrm{EqSat}(\star, \star)$ is known to be in PSPACE and NP-hard. If Σ is unary, then word equations are essentially linear Diophantine equations, so $\mathrm{EqSat}(\star, 1)$ is equivalent to linear integer programming in unary notation, which is known to be NP-complete.

We have defined the above decision problems for systems of equations. Studying single equations instead of systems would not make them much easier, except in the case where Σ is unary. If there are at least two distinct constant letters, then for every finite system of word equations, we can find an equation that has exactly the same solutions as the system, and for every finite system of constant-free word equations, we can find a constant-free equation that has exactly the same nonperiodic solutions as the system, as proved by Hmelevskii [14]. Moreover, these equations can be constructed in polynomial time.

If A and B are decision problems and A is polynomial-time reducible to B, we use the notation $A \leq_p B$. If A and B are polynomially equivalent, that is, $A \leq_p B$ and $B \leq_p A$, then we use the notation $A \equiv_p B$.

3 Inclusion problem of pattern languages

We are going to prove that $\mathrm{PatIncl}_{\mathrm{NE}}(\star, \star, 2) \leq_p \mathrm{PatIncl}_{\mathrm{NE}}^{\mathrm{CF}}(\star, \star, 2)$. Let the alphabet of constants be Σ_2. Let a and b be new variables that are supposed to represent the constants 0 and 1. Nonerasing morphisms that map a to 0 and b to 1 or vice versa can be called *good*, and other nonerasing morphism can be called *bad*. For all patterns U, V, we must construct constant-free patterns U', V' such that $L_{\mathrm{NE}}(U') \subseteq L_{\mathrm{NE}}(V')$ if and only if $L_{\mathrm{NE}}(U) \subseteq L_{\mathrm{NE}}(V)$. In other words, we must show that the following conditions are satisfied:

1. If $L_{\mathrm{NE}}(U) \subseteq L_{\mathrm{NE}}(V)$, then for all good morphisms α', there exists a nonerasing morphism β' such that $\beta'(V') = \alpha'(U')$.
2. If $L_{\mathrm{NE}}(U) \subseteq L_{\mathrm{NE}}(V)$, then for all bad morphisms α', there exists a nonerasing morphism β' such that $\beta'(V') = \alpha'(U')$.
3. If $L_{\mathrm{NE}}(U) \not\subseteq L_{\mathrm{NE}}(V)$, then there exists a nonerasing morphism α' such that for all nonerasing morphisms β', $\beta'(V') \neq \alpha'(U')$.

Before giving the definition of U' and V' and the formal proofs, we explain some ideas behind the construction.

The simplest idea would be to replace 0 and 1 by a and b. Let U_1, V_1 be the constant-free patterns we get from U, V this way. If we use U_1, V_1 as U', V', then the first condition is satisfied. However, the next example shows that the third condition does not hold in general.

▶ **Example 8.** Let $U = 0x$, $V = 1x$, $U' = ax$, $V' = bx$. Then clearly $L_{\mathrm{NE}}(U) \not\subseteq L_{\mathrm{NE}}(V)$ and $L_{\mathrm{NE}}(U') \subseteq L_{\mathrm{NE}}(V')$, so the third condition does not hold.

The problem with the third condition is that β' does not necessarily map a and b in the same way as α'. To solve this, we use an idea that is somewhat similar to one used in [4, Subsection 5.2], where prefixes are added to patterns to ensure that certain variables must be mapped in a certain way. Consider the patterns

$$U_2 = a^2 b^2 c^2 U_1, \qquad V_2 = a^2 b^2 c^2 V_1,$$

where c is a new variable. If α' is good and $\alpha'(c)$ is a third letter that does not appear in $\alpha'(U_1)$, then it is quite easy to see that $\beta'(V_2) = \alpha'(U_2)$ is possible only if $\beta'(x) = \alpha'(x)$ for all $x \in \{a, b, c\}$. Of course, there is no third letter in Σ_2, but we can define $\alpha'(c) \in \Sigma_2^+$ so that it still acts as a separator in the same way a unique letter would. If we use U_2, V_2 as U', V', then the first and the third condition are satisfied.

Satisfying the second condition without interfering with the third condition is the most difficult part. To solve the problems, consider patterns of the form

$$U_4 = W_1 pq^2 r W_2 a^2 b^2 W_3, \qquad V_4 = W_4 pq^2 r W_5,$$

where p, q, r are new variables and $W_1, \dots, W_5$ are constant-free patterns. The factors pq^2r and a^2b^2 act as a "switch". If $\beta'(pq^2r) = \alpha'(pq^2r)$, then we can say that the switch is in the first position. If $\beta'(pq^2r) = \alpha'(a^2b^2)$, then we can say that the switch is in the second position. For any α', we can define β' so that $\beta'(pq^2r) = \alpha'(pq^2r)$, so the first position is always possible. On the other hand, we can define β' so that $\beta'(pq^2r) = \alpha'(a^2b^2)$ if and only if α' is bad, so the second position is possible for the bad morphisms but not for the good. This allows us to handle the second condition without causing problems with the other two.

Putting these ideas together leads to the following construction. Let U be a pattern over (Ξ_1, Σ_2) and V a pattern over (Ξ_2, Σ_2). Let a, b, c, p, q, r, s, t be new variables not in $\Xi_1 \cup \Xi_2$. We define a $(\Xi_1 \cup \Xi_2)$-preserving morphism

$$\sigma : (\Xi_1 \cup \Xi_2 \cup \Sigma_2)^* \to (\Xi_1 \cup \Xi_2 \cup \{a, b\})^*, \ \sigma(0) = a, \ \sigma(1) = b.$$

We can construct the constant-free patterns

$$\begin{aligned} U' &= a^2b^2c^2 \cdot c^2 \cdot \sigma(U) \cdot c^2pq^2rc \cdot \sigma(V) \cdot c^2a^2b^2c \cdot a \\ V' &= a^2b^2c^2 \cdot sc \cdot \sigma(V) \cdot c^2pq^2rc \cdot t \end{aligned} \tag{1}$$

where U' is a pattern over $(\Xi_1 \cup \Xi_2 \cup \{a, b, c, p, q, r\}, \Sigma_2)$ and V' is a pattern over $(\Xi_2 \cup \{a, b, c, p, q, r, s, t\}, \Sigma_2)$.

▶ **Lemma 9.** *If $L_{\mathrm{NE}}(U) \subseteq L_{\mathrm{NE}}(V)$, then $L_{\mathrm{NE}}(U') \subseteq L_{\mathrm{NE}}(V')$,*

Proof. Let $\alpha' : (\Xi_1 \cup \Xi_2 \cup \{a, b, c, p, q, r\})^* \to \Sigma_2^*$ be a nonerasing morphism. We must find a nonerasing morphism $\beta' : (\Xi_2 \cup \{a, b, c, p, q, r, s, t\})^* \to \Sigma_2^*$ such that $\beta'(V') = \alpha'(U')$. There are two cases depending on whether α' is good or bad.

If α' is good, then we can define a nonerasing morphism

$$\alpha : (\Xi_1 \cup \Xi_2 \cup \Sigma_2)^* \to \Sigma_2^*, \ \alpha = \alpha' \circ \sigma \circ \alpha' \circ \sigma.$$

It is easy to check that α is Σ_2-preserving. By the assumption $L_{\mathrm{NE}}(U) \subseteq L_{\mathrm{NE}}(V)$, there exists a nonerasing Σ_2-preserving morphism $\beta : (\Xi_2 \cup \Sigma_2)^* \to \Sigma_2^*$ such that $\beta(V) = \alpha(U)$. We can define a morphism

$$\begin{aligned} \beta' : (\Xi_2 \cup \{a, b, c, p, q, r, s, t\})^* \to \Sigma_2^*, \ &\beta'(x) = \alpha'(\sigma(\beta(x)) \text{ for all } x \in \Xi_2, \\ &\beta'(x) = \alpha'(x) \text{ for all } x \in \{a, b, c, p, q, r\}. \\ &\beta'(s) = \alpha'(c), \\ &\beta'(t) = \alpha'(\sigma(V)c^2a^2b^2ca). \end{aligned}$$

It follows directly from the definition of β' that

$$\begin{aligned} &\beta'(a^2b^2c^2sc) = \alpha'(a^2b^2c^4), \\ &\beta'(c^2pq^2rct) = \alpha'(c^2pq^2rc\sigma(V)c^2a^2b^2ca). \end{aligned} \tag{2}$$

Showing that $\beta'(\sigma(V)) = \alpha'(\sigma(U))$ requires some computations. By using the definition of β' and the fact that σ is Ξ_2-preserving and β is Σ_2-preserving, we get

$$\begin{aligned} &\beta'(\sigma(x)) = \beta'(x) = \alpha'(\sigma(\beta(x))) \text{ for all } x \in \Xi_2, \\ &\beta'(\sigma(x)) = \alpha'(\sigma(x)) = \alpha'(\sigma(\beta(x))) \text{ for all } x \in \Sigma_2. \end{aligned} \tag{3}$$

We have

$$\beta'(\sigma(V)) = \alpha'(\sigma(\beta(V))) = \alpha'(\sigma(\alpha(U))) = \alpha(\alpha'(\sigma(U))) = \alpha'(\sigma(U)), \tag{4}$$

where the first equality follows from $V \in (\Xi_2 \cup \Sigma_2)^+$ and (3), the second from $\beta(V) = \alpha(U)$, the third from $(\alpha' \circ \sigma) \circ \alpha = \alpha \circ (\alpha' \circ \sigma)$, and the fourth from α being Σ_2-preserving. It follows from (2) and (4) that $\beta'(V') = \alpha'(U')$

If α' is bad, then $\alpha'(a^2b^2)$ is either 0^4, 1^4, or a binary word of length at least six. In all cases, it has a nonempty internal factor that is a square, so there exists $P, Q, R \in \Sigma_2^+$ such that $\alpha'(a^2b^2) = PQ^2R$. We can define a morphism

$$\begin{aligned} \beta' : (\Xi_2 \cup \{a, b, c, p, q, r, s, t\})^* \to \Sigma_2^*, \ & \beta'(x) = \alpha'(x) \text{ for all } x \in \Xi_2 \cup \{a, b, c\}, \\ & \beta'(p) = P, \\ & \beta'(q) = Q, \\ & \beta'(r) = R, \\ & \beta'(s) = \alpha'(c^2\sigma(U)c^2pq^2r), \\ & \beta'(t) = \alpha'(a). \end{aligned}$$

It follows directly from the definition of β' that $\beta'(V') = \alpha'(U')$. ◀

▶ **Lemma 10.** *If $L_{\mathrm{NE}}(U) \not\subseteq L_{\mathrm{NE}}(V)$, then $L_{\mathrm{NE}}(U') \not\subseteq L_{\mathrm{NE}}(V')$.*

Proof. By the assumption $L_{\mathrm{NE}}(U) \not\subseteq L_{\mathrm{NE}}(V)$, there exist a nonerasing Σ_2-preserving morphism $\alpha : (\Xi_1 \cup \Sigma_2)^* \to \Sigma_2^*$ such that $\beta(V) \neq \alpha(U)$ for all nonerasing Σ_2-preserving morphisms $\beta : (\Xi_2 \cup \Sigma_2)^* \to \Sigma_2^*$. We can define a morphism

$$\begin{aligned} \alpha' : (\Xi_1 \cup \Xi_2 \cup \{a, b, c, p, q, r\})^* \to \Sigma_2^*, \ & \alpha'(x) = \alpha(x) \text{ for all } x \in \Xi_1, \\ & \alpha'(a) = 0, \\ & \alpha'(b) = 1, \\ & \alpha'(c) = 10^N1, \\ & \alpha'(x) = 1 \text{ for all } x \in \Xi_2 \cup \{p, q, r\}, \end{aligned}$$

where $N = 1 + \max\{2, |\alpha'(\sigma(U))|, |\alpha'(\sigma(V))|\}$.

It is easy to see that $\alpha'(U')$ does not contain any other occurrences of $\alpha'(c)$ than the ten obvious ones. We can show that if A^2 is a nonempty square prefix of $\alpha'(U')$, then $A = \alpha'(a) = 0$. First, if $\alpha'(a^2b^2c^3)$ is a prefix of A, then $A\alpha'(a^2b^2c^3)$ is a prefix of $\alpha'(U')$, which is impossible, because $\alpha'(c^3)$ does not have any occurrences in $\alpha'(U')$ starting after the prefix $\alpha'(a^2b^2c^3)$. Second, if A is a prefix of $\alpha'(a^2b^2c^3)$ and $|A| \geq 5$, then $A00111$ is a prefix of $\alpha'(a^2b^2c^4)$, which is impossible, because 111 does not have any occurrences in $\alpha'(c^4)$. Finally, if $|A| \leq 4$, then clearly the only possibility is $A = 0$. Similarly, we can show that if W is the word such that $U' = a^2b^2W$, then the only nonempty square prefix of $\alpha'(b^2W)$ is $\alpha'(b^2) = 11$, and the only nonempty square prefixes of $\alpha'(W)$ are $\alpha'(c^2)$ and $\alpha'(c^4)$.

To complete the proof of the theorem, we assume that

$$\beta' : (\Xi_2 \cup \{a, b, c, p, q, r, s, t\})^* \to \Sigma_2^*$$

is a nonerasing morphism such that $\beta'(V') = \alpha'(U')$ and derive a contradiction. Because $\beta'(V')$ has the nonempty square prefix $\beta'(a^2)$, and the only nonempty square prefix of $\alpha'(U')$ is 00, it must be $\beta'(a) = 0$. Similarly, we see that $\beta'(b) = 1$ and $\beta'(c) \in \{\alpha'(c), \alpha'(c^2)\}$. If $\beta'(c) = \alpha'(c^2)$, then $\beta'(c^2) = \alpha'(c^4)$ has two occurrences in $\alpha'(U')$, which is not possible, so it must be $\beta'(c) = \alpha'(c)$. It follows that

$$\beta'(sc\sigma(V)c^2pq^2rct) = \alpha'(c^2\sigma(U)c^2pq^2rc\sigma(V)c^2a^2b^2ca).$$

Here on the right-hand side, there are only two occurrences of $\alpha'(c^2)$ as an internal factor, so either

$$\beta'(sc\sigma(V)) = \alpha'(c^2\sigma(U)) \qquad \text{and} \qquad \beta'(pq^2rct) = \alpha'(pq^2rc\sigma(V)c^2a^2b^2ca) \tag{5}$$

or

$$\beta'(sc\sigma(V)) = \alpha'(c^2\sigma(U)c^2pq^2rc\sigma(V)) \qquad \text{and} \qquad \beta'(pq^2rct) = \alpha'(a^2b^2ca). \tag{6}$$

There is only one occurrence of $\alpha'(c)$ as an internal factor in $\alpha'(c^2\sigma(U))$, so (5) implies $\beta'(\sigma(V)) = \alpha'(\sigma(U)) = \alpha(U)$, which is a contradiction because $\beta' \circ \sigma$ is a nonerasing Σ_2-preserving morphism. There is only one occurrence of $\alpha'(c)$ as an internal factor in $\alpha'(a^2b^2ca)$, so (6) implies $\beta'(pq^2r) = \alpha'(a^2b^2) = 0011$, which is a contradiction because 0011 does not have a nonempty internal square factor. These contradictions show that the morphism β' does not exist, and therefore $L_{\mathrm{NE}}(U') \not\subseteq L_{\mathrm{NE}}(V')$. ◀

▶ **Theorem 11.** *For all $m, n \in \mathbb{Z}_+$,*

$$\mathrm{PatIncl}_{\mathrm{NE}}(m, n, 2) \leq_p \mathrm{PatIncl}_{\mathrm{NE}}^{\mathrm{CF}}(\max\{m, n\} + 6, n + 8, 2).$$

Proof. Let U be a pattern over (Ξ_1, Σ_2) and V a pattern over (Ξ_2, Σ_2), where $|\Xi_1| = m$ and $|\Xi_2| = n$. Because renaming the variables in one of U and V does not change the pattern language, we can assume that one of Ξ_1 and Ξ_2 is a subset of the other, and therefore $|\Xi_1 \cup \Xi_2| = \max\{m, n\}$. The constant-free patterns U' and V' defined in (1) can be constructed in polynomial time. By Lemmas 9 and 10, $L_{\mathrm{NE}}(U) \subseteq L_{\mathrm{NE}}(V)$ if and only if $L_{\mathrm{NE}}(U') \subseteq L_{\mathrm{NE}}(V')$. The claim follows. ◀

▶ **Corollary 12.** *The decision problem* $\mathrm{PatIncl}_{\mathrm{NE}}^{\mathrm{CF}}(2560, 2562, 2)$ *is undecidable.*

Proof. By Theorem 3, $\mathrm{PatIncl}_{\mathrm{NE}}(3, 2554, 2)$ is undecidable. It follows from Theorem 11 that $\mathrm{PatIncl}_{\mathrm{NE}}^{\mathrm{CF}}(2560, 2562, 2)$ is undecidable. ◀

4 Nonperiodic satisfiability

We are going to prove that the decision problem $\mathrm{EqSat}_{\mathrm{NP}}^{\mathrm{CF}}(\star, 2)$ is NP-hard. This is based on the NP-hardness of $\mathrm{EqSat}(\star, 1)$. Before the proofs, we give a brief informal explanation of the idea.

We are going to transform a system of word equations over (Ξ, Σ_1) into a similarly-behaving system of constant-free word equations over $(\Xi \cup \{a, b\}, \Sigma_2)$, where a and b are new variables. The letter 0 in the original system has two important properties: It is primitive, and the images of all variables are powers of it. We want to replace 0 by a word consisting of variables that has similar properties. We can use the word a^2b^2. For all $\{a, b\}$-nonperiodic solutions β, the word $\beta(a^2b^2)$ is primitive, and we can force $\beta(x)$ to be a power of $\beta(a^2b^2)$ by adding the equation (xa^2b^2, a^2b^2x) for all $x \in \Xi$. Finally, adding the equation (xy, yx) for all $x, y \in \Xi$ makes sure that every $\{a, b\}$-periodic solution is periodic.

▶ **Theorem 13.** *For all $n \in \mathbb{Z}_+$,*

$$\mathrm{EqSat}(n, 1) \leq_p \mathrm{EqSat}_{\mathrm{NP}}^{\mathrm{CF}}(n + 2, 2) \qquad \textit{and} \qquad \mathrm{EqSat}(\star, 1) \leq_p \mathrm{EqSat}_{\mathrm{NP}}^{\mathrm{CF}}(\star, 2).$$

Proof. Let S be a system of word equations over (Ξ, Σ_1). Let a, b be new variables not in Ξ. Let us define a Ξ-preserving morphism

$$\sigma : (\Xi \cup \Sigma_1)^* \to (\Xi \cup \{a, b\})^*,\ \sigma(0) = a^2b^2,$$

and a morphism

$$\tau : \{a, b\}^* \to \Sigma_2^*,\ \tau(a) = 0,\ \tau(b) = 1.$$

We can construct in polynomial time a system of constant-free word equations

$$S' = \{(\sigma(U), \sigma(V)) \mid (U, V) \in S\} \cup \{(x\sigma(0), \sigma(0)x) \mid x \in \Xi\} \cup \{(xy, yx) \mid x, y \in \Xi\}$$

over $(\Xi \cup \{a, b\}, \Sigma_2)$. To complete the proof of the theorem, we show that S' has a nonperiodic solution if and only if S has a solution.

First, assume that S has a solution α. We can define a nonperiodic Σ_2-preserving morphism

$$\begin{aligned}\beta : (\Xi \cup \{a, b\} \cup \Sigma_2)^* \to \Sigma_2^*,\ \beta(x) &= \tau(x) \text{ for all } x \in \{a, b\},\\ \beta(x) &= \tau(\sigma(\alpha(x))) \text{ for all } x \in \Xi\end{aligned}$$

and show that it is a solution of S'. By using the definition of β and the fact that α is Σ_1-preserving and σ is Ξ-preserving, we get

$$\begin{aligned}&\beta(\sigma(0)) = \tau(\sigma(0)) = \tau(\sigma(\alpha(0))),\\ &\beta(\sigma(x)) = \beta(x) = \tau(\sigma(\alpha(x))) \text{ for all } x \in \Xi.\end{aligned} \tag{7}$$

For all $(U, V) \in S$, from $U, V \in (\Xi \cup \Sigma_1)^*$, (7), and $\alpha(U) = \alpha(V)$, it follows that

$$\beta(\sigma(U)) = \tau(\sigma(\alpha(U))) = \tau(\sigma(\alpha(V))) = \beta(\sigma(V)).$$

Thus β is a solution of $(\sigma(U), \sigma(V))$ for all $(U, V) \in S$. For all $x \in \Xi$, we have $\alpha(x) \in 0^*$ and therefore

$$\beta(x) = \tau(\sigma(\alpha(x))) \in \tau(\sigma(0))^* = \beta(\sigma(0))^*.$$

Thus β is a solution of all the other equations in S' as well.

Second, assume that S' has a nonperiodic solution β. From $\beta(xy) = \beta(yx)$ for all $x, y \in \Xi \cup \{\sigma(0)\}$ it follows that there exists a primitive word R such that $\beta(x) \in R^*$ for all $x \in \Xi \cup \{\sigma(0)\}$. If β is $\{a, b\}$-periodic, then $\beta(\sigma(0)) = \beta(a^2b^2) \in R^*$ implies $\beta(a), \beta(b) \in R^*$, and then β is periodic, a contradiction. Therefore β must be $\{a, b\}$-nonperiodic. It follows from Theorem 7 that $\beta(\sigma(0))$ is primitive and therefore $\beta(\sigma(0)) = R$. We can define a bijective morphism

$$\phi : R^* \to \Sigma_1^*,\ \phi(R) = 0,$$

and a morphism

$$\alpha : (\Xi \cup \Sigma_1)^* \to \Sigma_1^*,\ \alpha = \phi \circ \beta \circ \sigma.$$

Then α is well-defined because the image of $\beta \circ \sigma$ is a subset of R^*, and α is Σ_1-preserving because $\alpha(0) = \phi(R) = 0$, and α is a solution of S because

$$\alpha(U) = \phi(\beta(\sigma(U))) = \phi(\beta(\sigma(V))) = \alpha(V)$$

for all $(U, V) \in S$. ◀

▶ **Corollary 14.** *The decision problem* $\mathrm{EqSat}^{\mathrm{CF}}_{\mathrm{NP}}(\star, 2)$ *is NP-hard.*

Proof. Follows from Theorem 13 because $\mathrm{EqSat}(\star, 1)$ is NP-hard. ◀

5 Noncommuting satisfiability

We have proved that $\mathrm{EqSat}_{\mathrm{NP}}^{\mathrm{CF}}$ is NP-hard, but based on this result alone, it might be possible that, for example, $\mathrm{EqSat}_{\mathrm{NP}}^{\mathrm{CF}}$ is NP-complete but EqSat is not in NP. We would like to prove that constant-free equations are, in some sense, as hard as general word equations. We are going to prove this kind of a result for the decision problem $\mathrm{EqSat}_{\mathrm{NC}}^{\mathrm{CF}}$.

When trying to generalize the ideas of the previous section to the case where the alphabet of constants is Σ_k with $k > 1$, it is quite easy to define words $C_i \in \{a, b\}$ for all $i \in \Sigma_k$ so that for all $\{a, b\}$-nonperiodic morphisms β, the words $\beta(C_i)$ are distinct primitive words and $\{\beta(C_0), \dots, \beta(C_{k-1})\}$ is the basis of a free monoid. The problem is that we cannot make sure that $\beta(x)$ is in this free monoid for all original variables x. (This difficulty is related to the fact that if $X \in \Sigma^*$, $\Gamma \subsetneq \Sigma$, $|\Gamma| \geq 2$, then the property $X \in \Gamma^*$ cannot be expressed by word equations, see [19].) However, we can make sure that $\beta(x)$ is in a certain larger free monoid whose basis contains the words $\beta(C_0), \dots, \beta(C_{k-1})$. This is sufficient to prove the results.

▶ **Lemma 15.** *Let $k \in \mathbb{Z}_+$ and let a, b be variables. Let*

$$\begin{aligned} A_i &= a^i b^2 a^{k-i+1} && \text{for all } i \in \{0, \dots, k+1\}, \\ B &= A_k^2 A_{k+1}^2 A_k^2, \\ C_i &= B A_i B && \text{for all } i \in \{0, \dots, k-1\}. \end{aligned}$$

Let $\beta : \{a, b\}^* \to \Sigma^*$ be an $\{a, b\}$-nonperiodic (and therefore injective) morphism and let

$$M = (\beta(B)\Sigma^* \cap \Sigma^* \beta(B)) \cup \{\varepsilon\}.$$

The following are true:

1. *$\beta(A_0), \dots, \beta(A_{k+1}), \beta(B)$ are primitive.*
2. *M is a free monoid.*
3. *$\beta(C_0), \dots, \beta(C_{k-1})$ are in the basis of M.*
4. *If $U \in M \smallsetminus \{\varepsilon\}$ and $UV = VU$, then $V \in M$.*

Proof.

1. The primitivity of the words $\beta(A_i)$ follows from Theorem 7. The word $\beta(B)$ is the image of 001100 under the morphism defined by $0 \mapsto \beta(A_k)$, $1 \mapsto \beta(A_{k+1})$, and $\beta(A_k A_{k+1}) \neq \beta(A_{k+1} A_k)$ because $A_k A_{k+1} \neq A_{k+1} A_k$, so also the primitivity of $\beta(B)$ follows from Theorem 7.
2. We use Theorem 2. Clearly M is a monoid. We have to show that if $U, V, W \in \Sigma^*$ and $U, VW, UV, W \in M$, then $V \in M$. If $V = \varepsilon$, then $V \in M$. If $|V| \geq |\beta(B)|$, then VW and thus also V begins with $\beta(P)$, and UV and thus also V ends with $\beta(P)$, so $V \in M$. If $0 < |V| < |\beta(B)|$, then we can write $U = X\beta(B)$, $UV = Y\beta(B)$, $UVW = X\beta(B)^2 Z$ for some words X, Y, Z such that $|X| < |Y| < |X\beta(B)|$, so $\beta(B)$ is an internal factor of $\beta(B)^2$, which contradicts the primitivity of $\beta(P)$. Thus M is a free monoid by Theorem 2.
3. Clearly $\beta(C_i) \in M$. If $\beta(C_i)$ is not in the basis for some $i \in \{0, \dots, k-1\}$, then it is a product of two nonempty elements of M, so it has a factor $\beta(B^2)$ and thus also a factor $\beta(A_k^4)$ and we can write
$$\beta(C_i) = \beta(A_k^2 A_{k+1}^2 A_k^2 A_i A_k^2 A_{k+1}^2 A_k^2) = U\beta(A_k^4)V. \tag{8}$$
for some words U, V. Let $l = |\beta(A_k)|$. Note that $l = |\beta(A_j)|$ for all j. If l divides $|U|$, then it follows from (8) that $\beta(A_k) = \beta(A_{k+1})$ or $\beta(A_k) = \beta(A_i)$, which contradicts the injectivity of β. If l does not divide $|U|$, then it follows from (8) that $\beta(A_k)$ is an internal factor of $\beta(A_k^2)$, which contradicts the primitivity of $\beta(A_k)$. This proves the claim.

4. If $V = \varepsilon$, then $V \in M$. If $|V| \geq |\beta(B)|$, then U and thus also V begins and ends with $\beta(B)$, so $V \in M$. If $0 < |V| < |\beta(B)|$, then $\beta(B) = R^k R'$, where R is the common primitive root of U and V, R' is a nonempty prefix of R, and $k \geq 1$. Because $\beta(B)$ is primitive, $0 < |R'| < |R|$. But R is a suffix of U and thus of $\beta(B) = R^k R'$, so R is an internal factor of R^2, which contradicts the primitivity of R. This proves the claim. ◀

▶ **Theorem 16.** *For all $n \in \mathbb{Z}_+$,*

$$\mathrm{EqSat}(n, \star) \leq_p \mathrm{EqSat}^{\mathrm{CF}}_{\mathrm{NC}}(2n+2, 2) \qquad \textit{and} \qquad \mathrm{EqSat}(\star, \star) \leq_p \mathrm{EqSat}^{\mathrm{CF}}_{\mathrm{NC}}(\star, 2).$$

Proof. Let S be a system of word equations over (Ξ, Σ_k), where $\Xi = \{x_1, \dots, x_n\}$. Let $a, b, y_1, \dots, y_n$ be new variables not in Ξ and let $\Xi' = \Xi \cup \{a, b, y_1, \dots, y_n\}$. Let B, C_i, M be as in Lemma 15. Let us define a Ξ-preserving morphism

$$\sigma : (\Xi \cup \Sigma_k)^* \to (\Xi \cup \{a, b\})^*, \ \sigma(i) = C_i \text{ for all } i \in \Sigma_k,$$

and a morphism

$$\tau : \{a, b\}^* \to \Sigma_2^*, \ \tau(a) = 0, \ \tau(b) = 1.$$

We can construct in polynomial time a system of constant-free word equations

$$S' = \{(\sigma(U), \sigma(V)) \mid (U, V) \in S\} \cup \{(x_i B y_i B, B y_i B x_i) \mid i \in \{1, \dots, n\}\}.$$

over (Ξ', Σ_2). To complete the proof of the theorem, we show that S' has an $\{a, b\}$-nonperiodic solution β if and only if S has a solution.

First, assume that S has a solution α. For all i, if $\sigma(\alpha(x_i)) = \varepsilon$, let $Y_i = \varepsilon$, and otherwise let $\sigma(\alpha(x_i)) = B Y_i B$. Such words Y_i exist by the definition of σ. We can define an $\{a, b\}$-nonperiodic Σ_2-preserving morphism

$$\begin{aligned} \beta : (\Xi' \cup \Sigma_2)^* \to \Sigma_2^*, \ & \beta(x) = \tau(x) \text{ for all } x \in \{a, b\}, \\ & \beta(x_i) = \tau(\sigma(\alpha(x_i))) \text{ for all } i, \\ & \beta(y_i) = \tau(Y_i) \text{ for all } i, \end{aligned}$$

and show that it is a solution of S'. By using the definition of β and the fact that α is Σ_k-preserving and σ is Ξ-preserving, we get

$$\begin{aligned} & \beta(\sigma(i)) = \tau(\sigma(i)) = \tau(\sigma(\alpha(i))) \text{ for all } i \in \Sigma_k, \\ & \beta(\sigma(x)) = \beta(x) = \tau(\sigma(\alpha(x))) \text{ for all } x \in \Xi. \end{aligned} \tag{9}$$

For all $(U, V) \in S$, from $U, V \in (\Xi \cup \Sigma_k)^*$, (9), and $\alpha(U) = \alpha(V)$, it follows that

$$\beta(\sigma(U)) = \tau(\sigma(\alpha(U))) = \tau(\sigma(\alpha(V))) = \beta(\sigma(V)).$$

Thus β is a solution of $(\sigma(U), \sigma(V))$ for all $(U, V) \in S$. We have $\beta(x_i) = \varepsilon$ or $\beta(x_i) = \tau(B Y_i B) = \beta(B y_i B)$ for all i, so β is a solution of the other equations $(x_i B y_i B, B y_i B x_i)$ in S' as well.

Second, assume that S' has an $\{a, b\}$-nonperiodic solution β. By Lemma 15, from $\beta(x_i B y_i B) = \beta(B y_i B x_i)$ it follows that $\beta(x_i) \in M$ for all i. Again by Lemma 15, M is free and the words $\beta(C_i)$ are in the basis of M, so there exists an infinite alphabet Γ containing Σ_k and an injective morphism

$$\phi : M \to \Gamma^* \text{ such that } \phi(\beta(C_i)) = i \text{ for all } i. \text{ We can define a } \Sigma_k\text{-preserving morphism}$$

$$\psi : \Gamma^* \to \Sigma_k^*, \ \psi(x) = \varepsilon \text{ for all } x \in \Gamma \setminus \Sigma_k$$

and a morphism

$$\alpha : (\Xi \cup \Sigma_k)^* \to \Sigma_k^*, \ \alpha = \psi \circ \phi \circ \beta \circ \sigma.$$

Then α is well-defined because $\beta(\sigma(x)) \in M$ for all $x \in \Xi \cup \Sigma_k$, and α is Σ_k-preserving because

$$\alpha(i) = \psi(\phi(\beta(C_i))) = \psi(i) = i$$

for all $i \in \Sigma_k$, and α is a solution of S because from $\beta(\sigma(U)) = \beta(\sigma(V))$ it follows that

$$\alpha(U) = \psi(\phi(\beta(\sigma(U)))) = \psi(\phi(\beta(\sigma(V)))) = \alpha(V)$$

for all $(U, V) \in S$. ◀

▶ **Corollary 17.** $\mathrm{EqSat}(\star, \star) \equiv_p \mathrm{EqSat}^{\mathrm{CF}}_{\mathrm{NC}}(\star, \star)$.

Proof. We proved in Theorem 16 that $\mathrm{EqSat}(\star, \star) \leq_p \mathrm{EqSat}^{\mathrm{CF}}_{\mathrm{NC}}(\star, \star)$. To prove the other direction, let S be a system of constant-free equations over (Ξ, Σ_k), $k \geq 2$, and $x, y \in \Xi$. Let p, q, r be new variables not in Ξ. It is easy to see that the system

$$S' = S \cup \{(xy, p0q), (yx, p1r)\}$$

over $(\Xi \cup \{p, q, r\}, \Sigma_k)$ has a solution if and only if S has a $\{x, y\}$-nonperiodic solution: If β is a solution of S', then the restriction of β on $(\Xi \cup \Sigma_k)^*$ is a solution of S, and it is $\{x, y\}$-nonperiodic because

$$\beta(xy) = \beta(p)0\beta(q) \neq \beta(p)1\beta(r) = \beta(yx).$$

On the other hand, if α is an $\{x, y\}$-nonperiodic solution of S, then we can write $\alpha(xy) = PaQ$ and $\alpha(yx) = PbR$ for some words P, Q, R and distinct letters a, b. Because S is constant-free, every morphism we get from α by permuting the constant letters in the images of the variables is also an $\{x, y\}$-nonperiodic solution of S, so we can assume that $a = 0$ and $b = 1$. Then we can extend α to a solution α' of S' by defining $\alpha'(p) = P$, $\alpha'(q) = Q$, $\alpha'(r) = R$. This completes the proof. ◀

6 Conclusion

We have proved that the inclusion problem of nonerasing pattern languages is undecidable even in the case of constant-free patterns. We have also proved that the nonperiodic satisfiability problem of constant-free word equations is NP-hard, and the noncommuting satisfiability problem of constant-free word equations is polynomially equivalent to the general satisfiability problem of word equations.

The following questions remain open:

- Is the equivalence problem of erasing pattern languages decidable?
- Is the satisfiability problem of word equations in NP?
- For some fixed $n \geq 3$, can we prove that $\mathrm{EqSat}(n, \star)$ is in P or NP or NP-hard?

There are also several smaller open questions raised by the new results:

- Can the numbers 2560 and 2562 in Corollary 12 be made significantly smaller? This would require either a rather different approach or improving the results in [4].
- Is EqSat$(\star, \star)$ polynomial-time reducible to $\mathrm{EqSat}^{\mathrm{CF}}_{\mathrm{NP}}(\star, \star)$?
- In Theorem 16, we used $n + 2$ new variables. Would a constant number of new variables be sufficient?
- The satisfiability problem remains NP-hard for several restricted subfamilies of word equations. Can we prove NP-hardness results for some interesting subfamilies of constant-free equations?

References

1 Dana Angluin. Finding patterns common to a set of strings. *Journal of Computer and System Sciences*, 21(1):46–62, 1980. doi:10.1016/0022-0000(80)90041-0.

2 Evelyne Barbin-Le Rest and Michel Le Rest. Sur la combinatoire des codes à deux mots. *Theoretical Computer Science*, 41(1):61–80, 1985. doi:10.1016/0304-3975(85)90060-X.

3 Jean Berstel, Dominique Perrin, and Christophe Reutenauer. *Codes and Automata*. Cambridge University Press, 2010.

4 Joachim Bremer and Dominik D. Freydenberger. Inclusion problems for patterns with a bounded number of variables. *Information and Computation*, 220/221:15–43, 2012. doi:10.1016/j.ic.2012.10.003.

5 Christian Choffrut and Juhani Karhumäki. Combinatorics of words. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages*, volume 1, pages 329–438. Springer, 1997. doi:10.1007/978-3-642-59136-5_6.

6 Joel D. Day, Florin Manea, and Dirk Nowotka. The hardness of solving simple word equations. In *Proceedings of the 42nd MFCS*, volume 83 of *LIPIcs*, pages 18:1–14. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.MFCS.2017.18.

7 Robert Dąbrowski and Wojtek Plandowski. Solving two-variable word equations (extended abstract). In *Proceedings of the 31st ICALP*, volume 3142 of *LNCS*, pages 408–419. Springer, 2004. doi:10.1007/978-3-540-27836-8_36.

8 Volker Diekert. Makanin's algorithm. In M. Lothaire, editor, *Algebraic Combinatorics on Words*, pages 387–442. Cambridge University Press, 2002.

9 Volker Diekert and Murray Elder. Solutions of twisted word equations, EDT0L languages, and context-free groups. In *Proceedings of the 44th ICALP*, volume 80 of *LIPIcs*, pages 96:1–14. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.96.

10 Henning Fernau and Markus L. Schmid. Pattern matching with variables: a multivariate complexity analysis. *Information and Computation*, 242:287–305, 2015. doi:10.1016/j.ic.2015.03.006.

11 Dominik D. Freydenberger and Daniel Reidenbach. Bad news on decision problems for patterns. *Information and Computation*, 208(1):83–96, 2010. doi:10.1016/j.ic.2009.04.002.

12 Tero Harju and Juhani Karhumäki. The equivalence problem of multitape finite automata. *Theoretical Computer Science*, 78(2):347–355, 1991. doi:10.1016/0304-3975(91)90356-7.

13 Tero Harju, Juhani Karhumäki, and Wojciech Plandowski. Independent systems of equations. In M. Lothaire, editor, *Algebraic Combinatorics on Words*, pages 443–472. Cambridge University Press, 2002.

14 Ju. I. Hmelevskiĭ. *Equations in free semigroups*. American Mathematical Society, 1976. Translated by G. A. Kandall from the Russian original: Trudy Mat. Inst. Steklov. 107 (1971).

15 Artur Jeż. One-variable word equations in linear time. *Algorithmica*, 74(1):1–48, 2016. doi:10.1007/s00453-014-9931-3.

16 Artur Jeż. Recompression: a simple and powerful technique for word equations. *Journal of the ACM*, 63(1):Art. 4, 51, 2016. doi:10.1145/2743014.

17 Artur Jeż. Word equations in nondeterministic linear space. In *Proceedings of the 44th ICALP*, volume 80 of *LIPIcs*, pages 95:1–13. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.95.

18 Tao Jiang, Arto Salomaa, Kai Salomaa, and Sheng Yu. Decision problems for patterns. *Journal of Computer and System Sciences*, 50(1):53–63, 1995. doi:10.1006/jcss.1995.1006.

19 Juhani Karhumäki, Filippo Mignosi, and Wojciech Plandowski. The expressibility of languages and relations by word equations. *Journal of the ACM*, 47(3):483–505, 2000. doi:10.1145/337244.337255.

20 Juhani Karhumäki and Wojciech Plandowski. On the defect effect of many identities in free semigroups. In Gheorghe Paun, editor, *Mathematical aspects of natural and formal languages*, pages 225–232. World Scientific, 1994. doi:10.1142/9789814447133_0012.

21 M. Lothaire. *Algebraic Combinatorics on Words*. Cambridge University Press, 2002. URL: http://www-igm.univ-mlv.fr/~berstel/Lothaire/AlgCWContents.html.

22 Roger C. Lyndon and Marcel-Paul Schützenberger. The equation $a^M = b^N c^P$ in a free group. *The Michigan Mathematical Journal*, 9(4):289–298, 1962. doi:10.1307/mmj/1028998766.

23 G. S. Makanin. The problem of the solvability of equations in a free semigroup. *Mat. Sb. (N.S.)*, 103(2):147–236, 1977. English translation in Math. USSR Sb. 32:129–198, 1977.

24 Florin Manea and Markus L. Schmid. Matching patterns with variables. In *Proceedings of the 12th WORDS*, volume 11682 of *LNCS*, pages 1–27. Springer, 2019. doi:10.1007/978-3-030-28796-2_1.

25 Yen Kaow Ng and Takeshi Shinohara. Developments from enquiries into the learnability of the pattern languages from positive data. *Theoretical Computer Science*, 397(1–3):150–165, 2008. doi:10.1016/j.tcs.2008.02.028.

26 Dirk Nowotka and Aleksi Saarela. An optimal bound on the solution sets of one-variable word equations and its consequences. In *Proceedings of the 45th ICALP*, volume 107 of *LIPIcs*, pages 136:1–136:13. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.136.

27 Wojciech Plandowski. Satisfiability of word equations with constants is in PSPACE. *Journal of the ACM*, 51(3):483–496, 2004.

28 Daniel Reidenbach. Discontinuities in pattern inference. *Theoretical Computer Science*, 397(1–3):166–193, 2008. doi:10.1016/j.tcs.2008.02.029.

29 John Michael Robson and Volker Diekert. On quadratic word equations. In *Proceedings of the 16th STACS*, volume 1563 of *LNCS*, pages 217–226. Springer, 1999. doi:10.1007/3-540-49116-3_20.

30 Aleksi Saarela. On the complexity of Hmelevskii's theorem and satisfiability of three unknown equations. In *Proceedings of the 13th DLT*, volume 5583 of *LNCS*, pages 443–453. Springer, 2009. doi:10.1007/978-3-642-02737-6_36.

31 Aleksi Saarela. Studying word equations by a method of weighted frequencies. *Fundamenta Informaticae*, 162(2–3):223–235, 2018. doi:10.3233/FI-2018-1722.

32 Géraud Sénizergues. The equivalence problem for deterministic pushdown automata is decidable. In *Proceedings of the 24th ICALP*, volume 1256 of *LNCS*, pages 671–681. Springer, 1997. doi:10.1007/3-540-63165-8_221.

33 Takeshi Shinohara. Polynomial time inference of extended regular pattern languages. In *RIMS Symposia on Software Science and Engineering*, volume 147 of *LNCS*, pages 115–127. Springer, 1983. doi:doi.org/10.1007/3-540-11980-9_19.

34 Jean-Claude Spehner. *Quelques problémes d'extension, de conjugaison et de présentation des sous-monoïdes d'un monoïde libre*. PhD thesis, Univ. Paris, 1976.

Bisimulation Equivalence of Pushdown Automata Is Ackermann-Complete

Wenbo Zhang
BASICS, Shanghai Jiao Tong University, Shanghai, China
wbzhang@sjtu.edu.cn

Qiang Yin[1]
Alibaba Group, Shanghai, China
qiang.yq@alibaba-inc.com

Huan Long
BASICS, Shanghai Jiao Tong University, Shanghai, China
longhuan@sjtu.edu.cn

Xian Xu
East China University of Science and Technology, Shanghai, China
xuxian@ecust.edu.cn

Abstract

Deciding bisimulation equivalence of two pushdown automata is one of the most fundamental problems in formal verification. Though Sénizergues established decidability of this problem in 1998, it has taken a long time to understand its complexity: the problem was proven to be non-elementary in 2013, and only recently, Jančar and Schmitz showed that it has an Ackermann upper bound. We improve the lower bound to Ackermann-hard, and thus close the complexity gap.

2012 ACM Subject Classification Theory of computation → Logic and verification; Theory of computation → Formal languages and automata theory

Keywords and phrases PDA, Bisimulation, Equivalence checking

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.141

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

Acknowledgements This work is supported by NSF of China (61772336, 61872142, 61572318). The authors are grateful to Yuxi Fu for insightful discussions on this topic and support. We also thank the anonymous reviewers for their helpful comments and suggestions.

1 Introduction

In the area of formal verification, equivalence checking plays a central role in characterizing when two systems should be considered as the same. A classical equivalence is language equivalence, which asks if two processes recognize the same language. To characterize more refined behavioural relations, Milner proposed a fundamental equivalence called bisimulation equivalence (a.k.a. bisimilarity) [15]. Two processes are bisimilar to each other if every transition from one process can be simulated by the other one, and the resulting two processes keep in the same bisimilarity relation. If internal actions are allowed in a bisimulation step, we will get a more complicated equivalence called weak bisimilarity [15]. A seminal result proven in [1] shows that bisimulation equivalence is decidable for processes generated by context-free grammars, while the language equivalence between context-free grammars is well-known to be undecidable [6]. Extensive works followed up ever since, studying different equivalence relations on various infinite-state systems. See [13] for a survey.

[1] corresponding author

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 141; pp. 141:1–141:14
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Table 1 Complexity results of bisimulation equivalence problems for (variants of) PDA.

Model	Lower bound	Upper bound
DPDA	P-hard	TOWER [9, 28]
PDA	**Ackermann-hard**	ACKERMANN [10]
FOG	ACKERMANN-hard [9]	ACKERMANN [10]

Pushdown automata (PDA) extend finite-state automata with a stack memory and can be used to model recursive programs naturally. Since PDA recognize the same language as context-free grammars [6], language equivalence is undecidable. The language equivalence of deterministic PDA (DPDA), raised by Ginsburg and Greibach in [5], was first proved to be decidable for the real-time subclass, i.e., DPDA without internal actions [17, 16]. The decidable result was extended to DPDA in Sénizergues's remarkable work [23]. Observe that on DPDA, language equivalence coincides with weak bisimulation equivalence, which implies the decidability of weak bisimilarity for DPDA. Sénizergues later generalized the decidability result to weak bisimilarity of PDA with deterministic internal actions [22, 24]. An internal action is deterministic if there is no alternative. There are also quite a few works trying to simplify Sénizergues's proof [27, 25, 26, 7, 8]. Stirling revisited the decidability proof for DPDA via a tableau system [27]. He also generalized the tableau system for PDA [26]. Another line of work is conducted by Jančar in the framework of first-order grammars (FOG) [7, 8]. FOG has very close relationship with PDA [4]. It can describe PDA with deterministic internal actions by collapsed graphs where all internal actions are absorbed.

Concerning the complexity issue of bisimilarity of PDA, the best known upper bound for the deterministic case (DPDA) is TOWER [28, 9], while only P-hardness is known. For general PDA, EXPTIME-hardness was proven by Kučera and Mayr [14]. The EXPTIME-hardness even holds for a subclass of PDA, named BPA (Basic Process Algebra), of which the set of control states is a singleton [12]. The EXPTIME-hardness was further improved to non-elementary (TOWER-hard actually) [2]. This non-elementary lower bound also holds for the normed subclass, where every PDA process can empty its stack. As for the upper bound for bisimilarity of PDA, little was known until very recently. Jančar and Schmitz gave an ACKERMANN algorithm in [10]. This upper bound is actually proven in the framework of FOG. It also matches the ACKERMANN-hard lower bound for bisimilarity of FOG [9].

Observe that FOG are equivalent to PDA with deterministic internal actions. Without internal actions, the ACKERMANN-hardness of FOG cannot be applied to PDA. Thus the best known lower-bound for bisimilarity of PDA is still TOWER-hard.

Our Contribution. We show that bisimilarity of PDA is actually ACKERMANN-complete by improving the TOWER-hard lower bound to ACKERMANN-hard. This is done by a reduction from the coverability problem of reset Petri net [19]. Moreover, our reduction also gives rise to a parametric complexity result, i.e., $\mathbf{F}_{d-1}$-hardness if the number of control states $d \geq 4$ is fixed. Our proof extends an early work by Jančar [9], where similar results for first-order grammars are established. We improve the reduction by avoiding ε-rules.

We summarize some mentioned results in Table 1 with our result presented in bold.

Further Comments. According to Table 1, the complexity classes of bisimilarity problems for PDA and first-order grammars happen to be the same. Thus one may wonder whether these two models are actually equal with respect to bisimilarity. The answer was known to be negative [3]. In this paper, we present a new proof which shows that pushdown automata are strictly weaker than first-order grammars as far as bisimilarity is considered. It also demonstrates why the reduction in [9] cannot be applied to real-time PDA directly.

Organization. The rest of the paper is organized as follows. Section 2 introduces the background knowledge and necessary notations. Section 3 establishes the Ackermann hardness for bisimilarity of PDA. Section 4 shows that PDA are strictly weaker than FOG considering bisimilarity. Section 5 concludes this paper.

2 Preliminaries

2.1 Pushdown Automata

▶ **Definition 1** (PDA). *A* pushdown automaton $\mathcal{A} = (\mathcal{Q}, \Gamma, \Sigma, \mathcal{R})$ *consists of*

- *a finite set of control states* $\mathcal{Q}$ *ranged over by* r, p, q*;*
- *a finite set of stack symbols* Γ *ranged over by* X, Y, Z*;*
- *a finite set of actions* Σ *ranged over by* a, b, c, d, f*;*
- *a finite set of rules* $\mathcal{R} \subseteq \mathcal{Q} \times \Gamma \times \Sigma \times \mathcal{Q} \times \Gamma^*$*.*

The set Σ^* of words will be ranged over by u, v, w, and the set Γ^* of finite strings of stack symbols will be ranged over by α, β. We write $\alpha\beta$ (respectively uv) for the concatenation of α and β (respectively u and v). As usual, $|\alpha|$ and $|u|$ represent the length of α and u respectively. For $n \in \mathbb{N}$, we use a^n to denote n consecutive actions a, and similarly for X^n. We write $pX \xrightarrow{a} q\alpha$ to mean $(p, X, a, q, \alpha) \in \mathcal{R}$.

The syntax of a PDA process is $p\alpha$, where $p \in \mathcal{Q}$ and $\alpha \in \Gamma^*$. The size of process $p\alpha$, denoted by $|p\alpha|$, is defined as its stack height $|\alpha|$. The set of PDA processes $\mathcal{P}$ is ranged over by O, P, Q. The semantics of the PDA processes is defined by the following rule:

$$\frac{pX \xrightarrow{a} q\alpha \in \mathcal{R}}{pX\beta \xrightarrow{a} q\alpha\beta} \tag{1}$$

If $w = a_1 a_2 \ldots a_n$ with $a_i \in \Sigma$ $(i \in 1, \ldots, n)$, then $P \xrightarrow{w} Q$ stands for $P \xrightarrow{a_1} P_1 \xrightarrow{a_2} \ldots P_{n-1} \xrightarrow{a_n} Q$ for some $P_1, P_2, \ldots, P_{n-1}$. A process P is *normed* if $P \xrightarrow{w} p$ for some $w \in \Sigma^*$ and $p \in \mathcal{Q}$, i.e., P can empty its stack. A PDA $\mathcal{A}$ is *normed* (denoted as nPDA) if every process defined in $\mathcal{A}$ is normed.

▶ **Definition 2** (Bisimulation). *A binary relation* $R \subseteq \mathcal{P} \times \mathcal{P}$ *is a* bisimulation *if, for all* $a \in \Sigma$*, the following statements are valid:*

1. *whenever* $(P, Q) \in R$ *and* $P \xrightarrow{a} P'$*, then* $Q \xrightarrow{a} Q'$ *and* $(P', Q') \in R$ *for some* Q'*;*
2. *whenever* $(P, Q) \in R$ *and* $Q \xrightarrow{a} Q'$*, then* $P \xrightarrow{a} P'$ *and* $(P', Q') \in R$ *for some* P'*.*

The largest bisimulation relation, denoted by $\sim$, is an equivalence relation called *bisimulation equivalence* or *bisimilarity* [15].

When silent actions are considered, we use a special symbol ε to represent a silent action. Note that in Definition 1 we assume $\varepsilon \notin \Sigma$. PDA without silent actions are called *real-time* PDA. When silent actions are allowed, we will specify action set as $\Sigma_\varepsilon = \Sigma \uplus \{\varepsilon\}$. A rule of the form $pX \xrightarrow{\varepsilon} q\alpha$ is referred to as an ε-rule. For an ε-rule $pX \xrightarrow{\varepsilon} q\alpha$, we say it is popping if $|\alpha| < 1$; it is pushing if $|\alpha| > 1$; it is deterministic if $pX \xrightarrow{a} q'\alpha'$ implies $a = \varepsilon$, $q' = q$, and $\alpha' = \alpha$. We will write $\overset{\varepsilon}{\Longrightarrow}$ for the reflexive and transitive closure of $\xrightarrow{\varepsilon}$; and write $\overset{a}{\Longrightarrow}$ for $\overset{\varepsilon}{\Longrightarrow}\xrightarrow{a}\overset{\varepsilon}{\Longrightarrow}$ if $a \neq \varepsilon$.

▶ **Definition 3** (Weak Bisimulation). *A binary relation* $R \subseteq \mathcal{P} \times \mathcal{P}$ *is a* weak bisimulation *if for all* $a \in \Sigma_\varepsilon$*, the following statements are valid:*

1. *whenever* $(P, Q) \in R$ *and* $P \xrightarrow{a} P'$*, then* $Q \overset{a}{\Longrightarrow} Q'$ *and* $(P', Q') \in R$ *for some* Q'*;*
2. *whenever* $(P, Q) \in R$ *and* $Q \xrightarrow{a} Q'$*, then* $P \overset{a}{\Longrightarrow} P'$ *and* $(P', Q') \in R$ *for some* P'*.*

The largest weak bisimulation, denoted by $\approx$ is called weak bisimilarity [15], and is also an equivalence relation.

Fast-Growing Complexity. We will use an ordinal-indexed hierarchy of “fast-growing” complexity classes defined in [18]. This hierarchy grows as $\mathbf{F}_1, \mathbf{F}_2, \mathbf{F}_3, \ldots, \mathbf{F}_\omega, \mathbf{F}_{\omega+1} \ldots$ and allows the classification of many decision problems with a non-elementary complexity. Here $\mathbf{F}_3 = \textsc{Tower}$ is the lowest non-elementary complexity class; $\bigcup_{k \in \mathbb{N}} \mathbf{F}_k$ is the primitive-recursive complexity class; and $\mathbf{F}_\omega = \textsc{Ackermann}$ is the lowest non-primitive-recursive complexity class. A complexity class is closed under reduction functions from “lower” complexity class. For example, all classes starting from $\mathbf{F}_3$ are closed under elementary reduction.

Bisimilarity Problem for PDA. We are interested in the *bisimilarity problem* for PDA defined as follows. Given a PDA $\mathcal{A} = (\mathcal{Q}, \Gamma, \Sigma, \mathcal{R})$ and two process $p\alpha$ and $q\beta$, where $p, q \in \mathcal{Q}$ and $\alpha, \beta \in \Gamma^*$, the bisimilarity problem asks if $p\alpha \sim q\beta$.

Our main result stated as follows improves the previous Tower-hard lower bound [2].

▶ **Theorem 4.** *The bisimilarity problem for PDA is Ackermann-hard and is $\mathbf{F}_{d-1}$-hard if the number of control states $d \geq 4$ is fixed.*

Combining with Ackermann upper bound from [10], we have the following result.

▶ **Corollary 5.** *The bisimilarity problem for PDA is Ackermann-complete.*

2.2 Bisimulation Game

Bisimulation equivalence has a nice game characterization called the *bisimulation game.*

Bisimulation Game. Given a pair of processes (P_0, Q_0), a bisimulation game for (P_0, Q_0) is played between *Attacker* and *Defender.* The game is played in rounds. In round i, Attacker chooses a transition $P_{i-1} \xrightarrow{a_i} P_i$ (resp. $Q_{i-1} \xrightarrow{a_i} Q_i$), then Defender chooses a transition with a same action $Q_{i-1} \xrightarrow{a_i} Q_i$ (resp. $P_{i-1} \xrightarrow{a_i} P_i$). We use $(P_{i-1}, Q_{i-1}) \xrightarrow{a_i} (P_i, Q_i)$ to denote a round. Defender wins if it never gets stuck; otherwise Attacker wins. We say that one player has a winning strategy if it can always win no matter how the opponent plays. The following result is well known.

▶ **Lemma 6.** *$P \sim Q$ if and only if Defender has a winning strategy in the bisimulation game for (P, Q).*

Macro Rules. Following [2], we introduce two kinds of macro rules to facilitate the design of bisimulation game and make our presentation concise.

(1). A macro rule $(pX, qY) \overset{ATT}{\hookrightarrow} (p_1\alpha_1, q_1\beta_1)$ denotes a pair of transitions:

$$pX \xrightarrow{a} p_1\alpha_1 \qquad qY \xrightarrow{a} q_1\beta_1$$

Here action a is *fresh.* This macro rule favours Attacker. In a bisimulation game for (pX, qY), if Attacker chooses transition $pX \xrightarrow{a} p_1\alpha_1$ (or $qY \xrightarrow{a} q_1\beta_1$), then Defender is forced to choose transition $qY \xrightarrow{a} q_1\beta_1$ (or $pX \xrightarrow{a} p_1\alpha_1$).

(2). A macro rule $(pX, qY) \overset{DEF}{\hookrightarrow} \{(p_1\alpha_1, q_1\beta_1), (p_2\alpha_2, q_2\beta_2)\}$ denotes a set of transitions:

$$\begin{array}{llll}
pX \xrightarrow{a_1} pZ_1 & pX \xrightarrow{a_1} pZ_2 & pX \xrightarrow{a_1} pZ_3 & \\
qY \xrightarrow{a_1} pZ_2 & qY \xrightarrow{a_1} pZ_3 & & \\
pZ_1 \xrightarrow{a_2} p_1\alpha_1 & pZ_1 \xrightarrow{a_3} p_2\alpha_2 & pZ_2 \xrightarrow{a_2} p_1\alpha_1 & pZ_2 \xrightarrow{a_3} q_2\beta_2 \\
pZ_3 \xrightarrow{a_2} q_1\beta_1 & pZ_3 \xrightarrow{a_3} p_2\alpha_2 & &
\end{array}$$

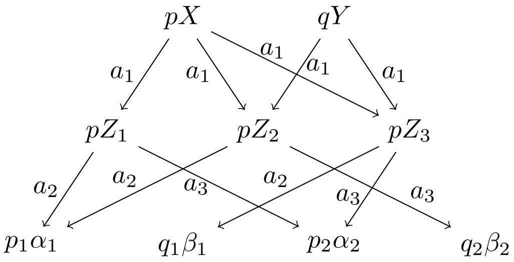


Figure 1 State transition diagram of macro rules $\xhookrightarrow{DEF}$.

Here a_1, a_2, a_3 are fresh actions, and Z_1, Z_2 and Z_3 are fresh stack symbols. This macro rule favours Defender. It is powered by a useful technique called *Defender's forcing* [11]. The state transition diagram of this macro rule is shown in Fig. 1. In a nutshell, with this macro rule, Defender can decide whether the game should continue with $(p_1\alpha_1\alpha, q_1\beta_1\alpha)$ or $(p_2\alpha_2\alpha, q_2\beta_2\alpha)$ in the bisimulation game for $(pX\alpha, qY\alpha)$.

Let us take a look at the development of the game for $(pX\alpha, qY\alpha)$.

1. If the game reaches a configuration with two identical processes, Defender wins immediately. Thus Attacker's optimal choice in the first step is $pX\alpha \xrightarrow{a_1} pZ_1\alpha$.
2. Then Defender can make a choice between $qY\alpha \xrightarrow{a_1} pZ_2\alpha$ and $qY\alpha \xrightarrow{a_1} pZ_3\alpha$. If Defender chooses transition $qY\alpha \xrightarrow{a_1} pZ_2\alpha$, the game continues with $(pZ_1\alpha, pZ_2\alpha)$. If Defender chooses transition $qY\alpha \xrightarrow{a_1} pZ_3\alpha$, the game continues with $(pZ_1\alpha, pZ_3\alpha)$.
3. In the case of $(pZ_1\alpha, pZ_2\alpha)$, Attacker is forced to choose action a_3, the game comes into $(p_2\alpha_2\alpha, q_2\beta_2\alpha)$. Similarly, in the case of $(pZ_1\alpha, pZ_3\alpha)$, Attacker is forced to choose action a_2 and the game reaches $(p_1\alpha_1\alpha, q_1\beta_1\alpha)$.

3 Lower Bound

We prove our main result by a reduction from the coverability problem of reset Petri net. We recall reset Petri net and its ACKERMANN-complete coverability problem in Section 3.1. We then construct an exponential time reduction in Section 3.2. Although the exponential time reduction suffices for our purpose, we revise it to a polynomial one in Section 3.3.

3.1 Reset Petri Net

Reset Petri Net (RPN). A reset Petri net is a tuple $\mathcal{N} = (\mathcal{S}, \mathcal{C}, \delta)$ consists of

- a finite set of control states $\mathcal{S}$ ranged over by s, t;
- a finite set of counters $\mathcal{C} = \{c_1, c_2, \ldots, c_d\}$;
- a finite set of instructions $\delta \subseteq \mathcal{S} \times \mathcal{O} \times \mathcal{S}$, where the set of operations $\mathcal{O}$ consists of $\mathsf{INC}(c_i)$, $\mathsf{DEC}(c_i)$ and $\mathsf{RESET}(c_i)$ for $i = 1, 2, \ldots, d$.

A configuration is a tuple $(s, n_1, \ldots, n_d)$ with $s \in \mathcal{S}$ representing the current state, and $n_1, \ldots, n_d \in \mathbb{N}$ representing the current contents of the counters. If $(s, op, t) \in \delta$ then we have $(s, n_1, \ldots, n_d) \to (t, n'_1, \ldots, n'_d)$ in the following cases:

- $op = \mathsf{INC}(c_i)$, $n'_i = n_i + 1$, and $n'_j = n_j$ for all $j \neq i$; or
- $op = \mathsf{DEC}(c_i)$, $n_i > 0$, $n'_i = n_i - 1$, and $n'_j = n_j$ for all $j \neq i$; or
- $op = \mathsf{RESET}(c_i)$, $n'_i = 0$, and $n'_j = n_j$ for all $j \neq i$.

By $\rightarrow^*$ we denote the reflexive and transitive closure of $\rightarrow$. We define a partial order $\leq$ on the configurations of $\mathcal{N}$:

$$(s, n_1, \ldots, n_d) \leq (t, m_1, \ldots, m_d) \text{ if } s = t \ \wedge \ n_1 \leq m_1 \ \wedge \ \ldots \ \wedge \ n_d \leq m_d.$$

We say σ_2 is *coverable* from σ_1 if there is some σ such that $\sigma_1 \rightarrow^* \sigma$ and $\sigma \geq \sigma_2$.

Coverability Problem of RPN. Given a Reset Petri Net $\mathcal{N} = (\mathcal{S}, \mathcal{C}, \delta)$, an initial configuration σ_1 and a final configuration σ_2, where the counters of σ_1 and σ_2 are given in binary, the *coverability* problem asks if σ_2 is coverable from σ_1.

We recall the following complexity result for coverability problem of RPN [19, 20, 21].

▶ **Theorem 7.** *The coverability problem of RPN is* Ackermann*-complete, and is* $\mathbf{F}_d$*-complete if the number of counters* $d \geq 3$ *is fixed.*

3.2 An Exponential Time Reduction

Given a RPN $\mathcal{N} = (\mathcal{S}, \mathcal{C}, \delta)$, an initial configuration σ_1, and a final configuration σ_2, we will construct a PDA $\mathcal{A} = (\mathcal{Q}, \Gamma, \Sigma, \mathcal{R})$ and two processes P, Q of $\mathcal{A}$ in exponential time such that

$$\sigma_2 \text{ is coverable from } \sigma_1 \text{ if and only if } P \not\sim Q. \qquad (\star)$$

Reduction Overview. Our reduction encodes the run of $\mathcal{N}$ from configuration σ_1 as a bisimulation game for (P, Q). In the bisimulation game, Attacker aims to show that σ_2 is coverable from σ_1, while Defender aims to show the opposite.

In order to complete the reduction, we should pay attention to the following aspects.

- A configuration σ' of $\mathcal{N}$ corresponds to a game for (P', Q') in the sense that the state and counter values of σ' are both encoded on the stack of both P' and Q'.
- To track the counters in the run from σ, the bisimulation game pushes every counter operation from the initial configuration on the stacks. Observe that the value of a counter of $\mathcal{N}$ can never become negative. Our reduction will guarantee that Attacker can never cheat by decreasing a counter with value zero. This is fulfilled by Defender's forcing. More specifically, for every $\mathsf{DEC}(c_i)$ operation, Defender has the power to verify its validity by initiating what we shall call a *zero check* for c_i.
- If a configuration that covers σ_2 is reached, Attacker wins the game. In the reduction, we will introduce a special witness action f. When a configuration $\sigma \geq \sigma_2$ is reached, the game will finally come into some (P', Q') where P' can do action f while Q' cannot.

As mentioned earlier, the basic idea of our reduction follows from [9], where Ackermann-hardness is proven for the bisimilarity problem of first-order grammars. However, as what will become clear in Section 4, first-order grammars are strictly more powerful than PDA w.r.t. bisimilarity. Here we highlight the main differences between our reduction and the one in [9].

- The reduction in [9] records the increasing and decreasing operations of d counters into $2d$ sub-processes (i.e. sub-terms in the terminology of first-order grammars). These sub-processes can work "in parallel" and be accessed without being interfered by each other. Zero check of a specific counter is achieved by skipping irrelevant sub-processes and comparing the relevant ones directly. Due to the sequential nature of stacks, this is beyond the reach of (real-time) PDA (see also Section 4). Instead, the increasing and decreasing operations recorded on the stack are interfered by each other unavoidably. To check zero for a counter, we introduce a novel mechanism by comparing the stack as a whole (Lemma 11).

- Our reduction uses $d+1$ PDA control states to encode a RPN with d counters. This yields a $\mathbf{F}_{d-1}$-hardness result, a first parametric lower-bound for the bisimilarity problem for PDA. In contrast, parametric complexity is not studied in [9]. Moreover, if we transform the reduction for first-order grammars [9] directly to a new reduction for PDA, the resulting PDA would require (i) at least $2d$ control states and (ii) popping ε-rules.

▶ Remark 8. In [9], the Ackermann-hardness for bisimilarity of FOG is established by reduction from the *control state reachability problem* of RPN. When the counters in the configurations of RPN are given in binary, the coverability problem of RPN is equivalent to the control state reachability problem under exponential time reduction. Incorporating the counter encoding and zero check trick into the reduction of [9], we can get the Ackermann-hard lower bound for bisimilarity of PDA as well. The main reason that we choose the coverability problem of RPN instead is to build a parametric lower bound under polynomial time reduction.

The Reduction. We fix a RPN $\mathcal{N} = (\mathcal{S}, \mathcal{C}, \delta)$, an initial configuration $\sigma_1 = (t_s, n_1, \ldots, n_d)$, a final configuration $\sigma_2 = (t_f, m_1, \ldots, m_d)$ in this subsection. The corresponding PDA $\mathcal{A} = (\mathcal{Q}, \Gamma, \Sigma, \mathcal{R})$ and two processes P, Q are defined as follows.

1. We introduce $d+1$ control states:

$$\mathcal{Q} \overset{\text{def}}{=} \{p, q_1, q_2, \ldots, q_d\}$$

Here q_1, q_2, …, q_d correspond to c_1, c_2, …, c_d of $\mathcal{C}$, respectively. The usage of state p will become clear later.

2. To record the operations on counters in $\mathcal{C}$, we introduce the following stack symbols:

$$\Gamma_{\mathcal{C}} \overset{\text{def}}{=} \{X_i^+, X_i^-, X_i^0 : i = 1, 2, \ldots, d\} \subseteq \Gamma$$

Here X_i^+, X_i^- and X_i^0 represent operations $\mathsf{INC}(c_i)$, $\mathsf{DEC}(c_i)$ and $\mathsf{RESET}(c_i)$, respectively.

3. To record the states of $\mathcal{N}$, for every state $s \in \mathcal{S}$, we introduce a pair of stack symbols:

$$\{Y_s, Y_s'\} \subseteq \Gamma$$

4. The two processes P and Q of the initial bisimulation game configuration are defined by

$$P \overset{\text{def}}{=} pY_s(X_1^+)^{n_1} \ldots (X_d^+)^{n_d} \qquad Q \overset{\text{def}}{=} pY_s'(X_1^+)^{n_1} \ldots (X_d^+)^{n_d}$$

In the rest of this subsection, we will complete the definition of $\mathcal{A}$ to validate property ($\star$). We start with encoding counters of $\mathcal{N}$ on the stacks of PDA processes. We then introduce rules for $\mathcal{A}$ to manipulate counters and perform zero checks. Last we show how to verify the coverability property in bisimulation games.

Counter Encoding. For each $i = 1, 2, \ldots, d$, we introduce a function $\mathsf{count}_i : {\Gamma_{\mathcal{C}}}^* \to \mathbb{Z}$. Let $\alpha = X_n X_{n-1} \ldots X_1 \in {\Gamma_{\mathcal{C}}}^*$. Intuitively, $\mathsf{count}_i(\alpha)$ computes the value of c_i after a sequence of operations $X_1, X_2, \ldots, X_n$. More specifically, let K_i be the largest index such that $X_{K_i} = X_i^0$, i.e., the leftmost occurrence of a reset operation of counter c_i. If X_i^0 does not appear in α, then $K_i = 0$. Let I_i and D_i be the respective numbers of occurrences of X_i^+ and X_i^- in the subsequence $X_n X_{n-1} \ldots X_{K_i+1}$ of α. The function count_i is defined by $\mathsf{count}_i(\alpha) = I_i - D_i$.

▶ Remark 9. Since the counters in reset Petri net cannot be negative, some strings, e.g. $X_1^- X_1^- X_1^+$, are invalid. For simplicity, the functions count_i are defined on all strings in ${\Gamma_{\mathcal{C}}}^*$.

▶ **Example 10.** $\mathsf{count}_1(X_2^+ X_1^0 X_2^+ X_1^+) = 0$ and $\mathsf{count}_2(X_2^+ X_1^0 X_2^+ X_1^+) = 2$.

We next introduce rules for counter manipulations. The PDA processes record every operation to the counters on their stacks. The trick here is to avoid invalid operations on counters. For example, at configuration $(pY_s X_1^- X_1^+,\ pY_s' X_1^- X_1^+)$, the counter c_1 is zero and it is not supposed to push another X_1^-. Here we use the Defender's forcing technique to fulfill this goal. Defender has a chance to force Attacker to check whether a decrease operation is valid. If it is not valid, Defender can win the game.

Counter Manipulation. For each instruction $I = (s, op, t)$ of $\mathcal{N}$, we introduce a set of PDA rules to $\mathcal{R}$ for counter manipulations as follows.

- If $op = \mathsf{INC}(c_i)$, we introduce $(pY_s, pY_s') \xhookrightarrow{ATT} (pY_t X_i^+, pY_t' X_i^+)$.
- If $op = \mathsf{RESET}(c_i)$, we introduce $(pY_s, pY_s') \xhookrightarrow{ATT} (pY_t X_i^0, pY_t' X_i^0)$.
- If $op = \mathsf{DEC}(c_i)$, we introduce $(pY_s, pY_s') \xhookrightarrow{DEF} \{(pY_t X_i^-, pY_t' X_i^-), (p, q_i)\}$.

We will make more comments on the decrease operation. The control states q_1, q_2, …, q_d are mainly used to zero check the counters c_1, c_2, …, c_d, respectively. In the sequel, we will introduce some rules to ensure the correctness of zero checks as stated in the following lemma.

▶ **Lemma 11.** *$q_i\alpha \sim p\alpha$ if and only if $\mathsf{count}_i(\alpha) = 0$, where $\alpha \in {\Gamma_{\mathcal{C}}}^*$.*

By Lemma 11, if Attacker intends to do an invalid decrease operation on counter c_i, Defender can force the game to the branch of (p, q_i) and win. If Attacker intends to do a valid decrease operation on counter c_i, Defender would lose in the branch of (p, q_i). In that case, Defender will force the game to the branch of $(pY_t X_i^-, pY_t' X_i^-)$ and the game continues.

Zero Check. We introduce the following rules to ensure the correctness of zero check. For the sake of concision, we introduce another macro rule. Given $w = a_1 a_2 \ldots a_n$, we use $pX \xhookrightarrow{w} q\alpha$ to stand for the sequence of transitions $pX \xrightarrow{a_1} pX_1 \xrightarrow{a_2} \ldots \xrightarrow{a_{n-1}} pX_{n-1} \xrightarrow{a_n} q\alpha$. Note that these macro rules do not introduce any new state. All the stack symbols introduced in these macro rules are fresh.

$$
\begin{array}{rcll}
pX & \xhookrightarrow{b_{pop}b_{pop}} & p & \text{For } X \in \Gamma_{\mathcal{C}} \\
q_i X_i^- & \xhookrightarrow{b_{pop}} & q_i & \text{For } i = 1, 2, \ldots, d \\
q_i X_i^+ & \xhookrightarrow{b_{pop}b_{pop}b_{pop}} & q_i & \text{For } i = 1, 2, \ldots, d \\
q_i X_i^0 & \xhookrightarrow{b_{pop}b_{pop}} & p & \text{For } i = 1, 2, \ldots, d \\
q_i X & \xhookrightarrow{b_{pop}b_{pop}} & q_i & \text{For } i = 1, 2, \ldots, d, \qquad X \in \Gamma_{\mathcal{C}} \setminus \{X_i^-, X_i^+, X_i^0\}
\end{array}
$$

Proof of Lemma 11. Observe that the bisimilarity between $q_i\alpha$ and $p\alpha$ only depends on the total numbers of consecutive actions b_{pop} from $q_i\alpha$ and $p\alpha$. Process $p\alpha$ can do action b_{pop} twice for every stack symbol. Process $q_i\alpha$ can do three b_{pop} actions for each X_i^+ and one b_{pop} action for each X_i^- before it meets the first X_i^0. Let I_i and D_i be the numbers of X_i^+ and X_i^- before it meets the first X_i^0. It follows that $q_i\alpha \sim p\alpha$ if and only if $3I_i + D_i + 2(|\alpha| - I_i - D_i) = 2|\alpha|$. Note that $\mathsf{count}_i(\alpha) = I_i - D_i$. Thus $q_i\alpha \sim p\alpha$ if and only if $\mathsf{count}_i(\alpha) = 0$. ◀

▶ Remark 12. The implementation of counters and their zero check mechanism is the main technical difference of our reduction from the one for first-order grammars [9]. The reduction in [9] compares the increasing and decreasing operations of a counter directly, by skipping non-relevant operations. As will be seen in Section 4, the ability to skip non-relevant operations also makes first-order grammars more powerful than PDA.

Coverability Check. When the game reaches $(pY_{t_f}\alpha, pY'_{t_f}\alpha)$, Attacker can initiate a coverability check. Then Defender will choose a counter c_i and check if $\mathsf{count}_i(\alpha) \geq m_i$. To this end, we introduce $d+1$ pairs of stack symbols $(Z_1, Z'_1), \dots (Z_{d+1}, Z'_{d+1})$ for counter choice. For each $1 \leq i \leq d$, we introduce $m_i + 1$ pairs of stack symbols $(Z_{i,0}, Z'_{i,0}), \dots, (Z_{i,m_i}, Z'_{i,m_i})$ for coverability check of counter c_i.

(1). We first add rules to check coverability of a specific counter c_i $(1 \leq i \leq d)$. For the pair $(pZ_{i,n}, pZ'_{i,n})$, where $0 \leq n \leq m_i$, we introduce the following rules.

- $(pZ_{i,n}, pZ'_{i,n}) \xhookrightarrow{DEF} \{(pZ_{i,n-1}X_i^-, pZ'_{i,n-1}X_i^-), (p, q_i)\}$;
- $pZ_{i,0} \xrightarrow{f} p, \qquad pZ'_{i,0} \xrightarrow{b_{pop}} p.$

If $\mathsf{count}_i(\alpha) < m_i$, then by Defender's forcing, Defender can push $\mathsf{count}_i(\alpha)$ number of X_i^- to the stack and the game reaches $(p\gamma\alpha, p_i\gamma\alpha)$, where $\gamma = (X_i^-)^{\mathsf{count}_i(\alpha)}$. By Lemma 11, Defender wins. If $\mathsf{count}_i(\alpha) \geq m_i$, Defender's best strategy would lead the game into a configuration $(pZ_{i,0}\beta\alpha, pZ'_{i,0}\beta\alpha)$ where $\beta = (X_i^-)^{m_i}$. Attacker wins since $pZ_{i,0}\beta\alpha$ admits a fresh action f, while $pZ'_{i,0}\beta\alpha$ does not. As a result, we have the following lemma.

▶ **Lemma 13.** *$pZ_{i,m_i}\alpha \sim pZ'_{i,m_i}\alpha$ if and only if $\mathsf{count}_i(\alpha) < m_i$, where $\alpha \in \Gamma^*$.*

(2). We next introduce the following rules to help Defender pick a specific counter to perform coverability check.

- $(pY_{t_f}, pY'_{t_f}) \xhookrightarrow{ATT} (pZ_1, pZ'_1)$;
- $(pZ_i, pZ'_i) \xhookrightarrow{DEF} \{(pZ_{i,m_i}, pZ'_{i,m_i}), (pZ_{i+1}, pZ'_{i+1})\}$ for $i = 1, 2, \dots, d$;
- $pZ_{d+1} \xrightarrow{f} p, \qquad pZ'_{d+1} \xrightarrow{b_{pop}} p.$

Suppose that Attacker initiates a coverability check and the game reaches $(pZ_1\alpha, pZ'_1\alpha)$. Defender can choose a specific counter to verify by Defender's forcing. Indeed if $\mathsf{count}_i(\alpha) < m_i$ for some $1 \leq i \leq d$, Defender can force the game to $(pZ_{i,m_i}\alpha, pZ'_{i,m_i}\alpha)$. By Lemma 13, Defender wins. If $\mathsf{count}_i(\alpha) \geq m_i$ for all $1 \leq i \leq d$, then Defender's best strategy is leading the game to configuration $(pZ_{d+1}\alpha, pZ'_{d+1}\alpha)$. Attacker wins via a fresh action f.

▶ **Proposition 14.** *The coverability problem of RPN (with d counters) can be reduced to the complement of the bisimilarity problem for PDA (with $d+1$ states) in exponential time.*

Proof. We construct the reduction as described in the section and prove the property $(\star)$.
($\Rightarrow$). If $(t_f, m_1, \dots, m_d)$ is coverable from $(t_s, n_1, \dots, n_d)$, then there exists a configuration $(t_f, m'_1, \dots, m'_d) \geq (t_f, m_1, \dots, m_d)$, which is reachable from $(t_s, n_1, \dots, n_d)$. We describe a winning strategy for Attacker. Attacker simply follows the path from $(t_s, n_1, \dots, n_d)$ to $(t_f, m'_1, \dots, m'_d)$. Since every decrease operation is valid, Defender will not ask for zero checks. The bisimulation game will reach $(pY_{t_f}\alpha, pY'_{t_f}\alpha)$ such that $\mathsf{count}_i(\alpha) = m'_i$ for each $i \in \{1, 2, \dots, d\}$. Attacker starts a coverability check and wins the game.

($\Leftarrow$). If $(t_f, m_1, \dots, m_d)$ is not coverable from $(t_s, n_1, \dots, n_d)$, we describe a winning strategy for Defender. Defender just follows what Attacker does and asks for zero check if Attacker tries to decrease a zero counter. Whenever the game reaches $(pY_{t_f}\beta, pY'_{t_f}\beta)$ for some $\beta \in \Gamma^*$, there exists some $i \in \{1, 2, \dots, d\}$, $\mathsf{count}_i(\beta) < m_i$. If Attacker asks for a coverability check, then Defender can choose to verify the value of counter c_i. By Lemma 13, Defender wins. If Attacker never initiates a coverability check, Defender also wins as it will never get stuck. ◀

By Theorem 7 and Proposition 14, we have our main result Theorem 4. Observe that it is safe to introduce rules $pY_s \xrightarrow{b_{pop}} p$ and $pY'_s \xrightarrow{b_{pop}} p$ for every $s \in \mathcal{S}$ since these transitions will never be chosen by Attacker (or Attacker loses immediately). As a result, the PDA $\mathcal{A}$ is normed and our lower bound can be generalized to the normed case.

▶ **Corollary 15.** *The bisimilarity problem of normed PDA is* Ackermann*-complete and is* $\mathbf{F}_{d-1}$*-hard if the number of control states is* $d \geq 4$*.*

3.3 A Polynomial Time Reduction

Although an elementary reduction suffices for our purpose, we show that the exponential time reduction can be actually revised to be polynomial. There are two exponential factors in the reduction presented in Section 3.2: (i) the size of the initial configuration of the bisimulation game can be exponentially large; and (ii) the number of rules for coverability check can be exponentially many. We eliminate both by incorporating a binary representation of counters.

Counter Binary Encoding. Let m be the maximal number occurs in the initial and final configuration of $\mathcal{N}$, and k the smallest number such that $2^k \geq m$. For every counter c_i, we introduce two new stack symbols $X^+_{i,j}$ and $X^-_{i,j}$ for each $j \in \{0, 1, \dots, k\}$. Intuitively, $X^+_{i,j}$ (resp. $X^-_{i,j}$) represents 2^j increasing (resp. decreasing) operations on counter c_i. Stack symbol X^0_i is still used to represent operation $\mathsf{RESET}(c_i)$. We then replace the initial game configuration by this binary representation. The function count_i can also be revised easily by taking the binary representation into account. We omit the details.

▶ **Example 16.** An initial RPN configuration $\sigma_1 = (s, 6, 3)$ can be encoded by a game configuration $(pY_sX^+_{1,2}X^+_{1,1}X^+_{2,1}X^+_{2,0}, pY'_sX^+_{1,2}X^+_{1,1}X^+_{2,1}X^+_{2,0})$.

We next revise the rules to make zero check and coverability check work.

Zero Check. We introduce new rules for $X^+_{i,j}$ and $X^-_{i,j}$ for zero check, where $0 \leq j \leq k$.

- $pX^+_{i,j} \xhookrightarrow{b_{pop}b_{pop}} pX^+_{i,j-1} \dots X^+_{i,0}$, $\quad pX^+_{i,0} \xhookrightarrow{b_{pop}b_{pop}} p$;
- $pX^-_{i,j} \xhookrightarrow{b_{pop}b_{pop}} pX^-_{i,j-1} \dots X^-_{i,0}$, $\quad pX^-_{i,0} \xhookrightarrow{b_{pop}b_{pop}} p$;
- $q_iX^+_{i,j} \xhookrightarrow{b_{pop}b_{pop}b_{pop}} q_iX^+_{i,j-1} \dots X^+_{i,0}$, $\quad q_iX^+_{i,0} \xhookrightarrow{b_{pop}b_{pop}b_{pop}} q_i$;
- $q_iX^-_{i,j} \xhookrightarrow{b_{pop}} q_iX^-_{i,j-1} \dots X^-_{i,0}$, $\quad q_iX^-_{i,0} \xhookrightarrow{b_{pop}} q_i$;

Note that besides the rules introduced above, we keep the rule $q_iX^0_i \xhookrightarrow{b_{pop}b_{pop}} p$ for $i \in \{1, \dots, d\}$. Lemma 11 is still valid and can be proved along the same line.

Coverability Check. Let m_i be the value of c_i in the final configuration σ_2 and k_i be the least number such that $2^{k_i} - 1 \geq m_i$. We replace the rules for $(pZ_{i,j}, pZ'_{i,j})$ $(0 \leq j \leq m_i)$ by

- $(pZ_{i,m_i}, pZ'_{i,m_i}) \xhookrightarrow{ATT} (pW_{i,0}\beta, pW'_{i,0}\beta)$, $\quad (pW_{i,k_i}, pW'_{i,k_i}) \xhookrightarrow{ATT} (p, q_i)$;
- $(pW_{i,j}, pW'_{i,j}) \xhookrightarrow{DEF} \{(pW_{i,j+1}X^{-}_{i,j}, pW'_{i,j+1}X^{-}_{i,j}), (pW_{i,j+1}, pW'_{i,j+1})\}$ $\qquad (0 \leq j < k_i)$

Here $W_{i,0}, \ldots, W_{i,k_i}$ are new stack symbols for coverability check and β consists of the binary stack symbols introduced above such that $\mathsf{count}_i(\beta) = 2^{k_i} - m_i$.

We show that Lemma 13 still holds. Consider the bisimulation game for $(pZ_{i,m_i}\alpha, pZ'_{i,m_i}\alpha)$. Intuitively, the coverability check starts with increasing counter c_i with $2^{k_i} - m_i$. After that, Defender decreases c_i by a combination of k_i choices and the game reaches a configuration $(p\gamma\alpha, q_i\gamma\alpha)$. If $\mathsf{count}_i(\alpha) < m_i$, Defender can pick γ such that $\mathsf{count}_i(\gamma\alpha) = 0$. By Lemma 11, Defender wins. If $\mathsf{count}_i(\alpha) \geq m_i$, then $\mathsf{count}_i(\alpha) + (2^{k_i} - m_i) - (2^{k_i} - 1) > 0$. The counter c_i can never be decreased to zero and Attacker wins.

4 Relative Expressiveness of PDA

First-order grammars have close relationship with pushdown automata [7, 9, 10], as they are equivalent to PDA with deterministic and popping ε-rules. More specifically, any process T definable in a first-order grammar can be encoded into a PDA process P with deterministic and popping ε-rules, and vice versa. The processes T and P are weakly bisimilar. In the real-time case, i.e, when no ϵ-rules is allowed, FOG have strictly richer behaviours than PDA. This can be proved by considering the transition graph of real-time PDA and PDA with deterministic ε-rules (see, e.g., [3], Proposition 5.8).

Nevertheless, to build better intuition for this result and also demonstrate why the original reduction in [9] cannot be directly applied to real-time PDA, we give another proof in this section to show that FOG are strictly stronger than PDA, as far as bisimilarity is considered.

To avoid introducing the definition of FOG, we construct a special PDA with deterministic and popping ε-rules and show that it can define a process which cannot be bisimulated by any real-time PDA process. The special PDA $\mathcal{A}$ contains the following rules:

- $pX_0 \xrightarrow{a} pX$, $\quad pX \xrightarrow{a} pXX$, $\quad pX \xrightarrow{b} pYX$, $\quad pY \xrightarrow{b} pYY$;
- $pY \xrightarrow{c} p_1Y$, $\quad pY \xrightarrow{d} p_2Y$;
- $p_1Y \xrightarrow{\varepsilon} p_1$, $\quad p_1X \xrightarrow{a} p_1$, $\quad p_2Y \xrightarrow{b} p_2$, $\quad p_2X \xrightarrow{\varepsilon} p_2$.

Observe that pX_0 can do actions $a^m b^n$ and record m and n by pushing $Y^m X^n$ into stack. The popping ε-rules enable the process to recover the information of m or n immediately (via doing actions a^m after c or actions b^n after d). We will show that such ε-rules are necessary in the sense that no real-time PDA is weakly bisimilar to $\mathcal{A}$. Intuitively, any real-time PDA process, after executing actions $a^m b^n$, cannot access both the information of m and n on the stack without undesired interference. The necessity of ε-rules is also the basic reason why the reduction in the hardness proof for first-order grammars [9] fails for real-time PDA.

▶ **Proposition 17.** *For any real-time PDA process O, $O \not\approx pX_0$.*

A direct consequence is that first-order grammars are strictly stronger that PDA. Indeed, following the transformation in [10], one can construct a process T in a first-order grammar such that (i) $T \approx pX_0$ and (ii) T has no ε transitions (the ε transitions are absorbed in FOG during transformation). As a result T cannot be related to any PDA process by bisimilarity.

▶ **Corollary 18.** *There exists a process T definable by a first-order grammar, such that for any real-time PDA process O, $O \not\sim T$.*

We prove Proposition 17 in the rest of this section. To do that, we need a useful pumping property of pushdown automata. We say a transition sequence $\xi : P_0 \xrightarrow{a_1} P_1 \xrightarrow{a_2} \ldots \xrightarrow{a_k} P_k$ is *non-sinking* if $|P_i| \geq |P_0|$ for $1 \leq i \leq k$. Moreover, ξ is *pumpable* if (i) ξ is *non-sinking* and (ii) $P_0 = pZ\alpha$ and $P_k = pZ\beta\alpha$ for some $p \in \mathcal{Q}$, $Z \in \Gamma$, $\alpha \in \Gamma^*$, and $\beta \in \Gamma^+$.

▶ **Lemma 19.** *Suppose that P_0 is a PDA process defined in $\mathcal{A} = (\mathcal{Q}, \Gamma, \Sigma, \mathcal{R})$ and it holds that $|\alpha| \leq 2$ for each $(p, X, a, q, \alpha) \in \mathcal{R}$. If there exists a transition sequence $\xi : P_0 \xrightarrow{a_1} P_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} P_n$ such that (i) $n \geq |P_0| \cdot (|Q||\Gamma| + 1)^{|Q||\Gamma|+1}$ and (ii) $P_i \not\sim P_j$ $(\forall i \neq j, 0 \leq i, j \leq n)$, then ξ contains a pumpable transition subsequence.*

Proof. Let $\xi' : Q_0 \xrightarrow{b_1} Q_1 \xrightarrow{b_2} \ldots \xrightarrow{b_m} Q_m$ be a *non-sinking* transition sequence such that $Q_i \not\sim Q_j$ $(\forall i \neq j)$. Denote by $\mathsf{type}(\xi')$ the set of different pairs of state and top stack symbol that occur in ξ'. More specifically, $\mathsf{type}(\xi') \overset{\text{def}}{=} \{(q, Z) :$ there is a process $qZ\beta$ in $\xi'\}$. It is sufficient to prove the following property:

$$\text{if } m \geq (|\mathsf{type}(\xi')| + 1)^{|\mathsf{type}(\xi')|+1}, \text{ then } \xi' \text{ contains a pumpable transition subsequence.} \qquad (\star\star)$$

Our lemma is established by observing that ξ contains a non-sinking subsequence ξ' of length at least $(|\mathcal{Q}||\Gamma| + 1)^{|\mathcal{Q}||\Gamma|+1}$. Suppose otherwise, then in less than $|P_0| \cdot (|Q||\Gamma| + 1)^{|Q||\Gamma|+1}$ steps P_0 will reach a process $p'\varepsilon$ for some $p' \in \mathcal{Q}$. This contradicts with our assumption (i).

We next prove $(\star\star)$ by induction on $|\mathsf{type}(\xi')|$.

- $|\mathsf{type}(\xi')| = 1$. It is trivial since ξ' is pumpable.
- $|\mathsf{type}(\xi')| = k+1$. By assumption, ξ' is of length at least $(k+2)^{k+2}$. Since ξ' is non-sinking and any two processes in ξ' are not equal, there are no more than k other processes of the size $|Q_0|$. By removing these (shortest) processes, we get at most $k + 1$ transition subsequences. There must be a sequence (denoted as ξ'') of length at least $(k + 1)^{k+1}$. By assumption of $\mathcal{A}$, the stack height will increase at most by one in a transition. It follows that ξ'' is non-sinking. Let $Q_0 = qZ\alpha$. If $(q, Z) \in \mathsf{type}(\xi'')$, say $Q_j = qZ\beta\alpha$, then $Q_0 \xrightarrow{b_1} \ldots \xrightarrow{b_j} Q_j$ is pumpable. Otherwise $|\mathsf{type}(\xi'')| = k$. By induction hypothesis, ξ'' contains a pumpable transition subsequence. ◀

We are ready to prove Proposition 17.

Proof of Proposition 17. Suppose otherwise and let O be a process of a real-time PDA $\mathcal{B} = (\mathcal{Q}, \Gamma, \Sigma, \mathcal{R})$ and $O \approx pX_0$. W.l.o.g., we assume that $|\alpha| \leq 2$ for each $(p, X, q, \alpha) \in \mathcal{R}$. Let $N = |\mathcal{Q}||\Gamma| + 1$ and $m \geq 1$. Consider the following transition sequence of pX_0

$$pX_0 \xrightarrow{a^m} pX^m \xrightarrow{b} pYX^m \xrightarrow{b} \ldots \xrightarrow{b} pY^nX^m. \qquad (2)$$

Since $O \approx pX_0$ and O is real-time, there exists $O_{m,i}$ $(0 \leq i \leq n)$ such that

$$O \xrightarrow{a^m} O_{m,0} \xrightarrow{b} O_{m,1} \xrightarrow{b} \ldots \xrightarrow{b} O_{m,n} \qquad (3)$$

and $O_{m,i} \approx pY^iX^m$ for $0 \leq i \leq n$. Observe that $pY^iX^m \not\approx pY^jX^m$ $(\forall i \neq j)$. Hence $O_{m,i} \not\sim O_{m,j}$ $(\forall i \neq j)$. Let $n = |O_{m,0}|N^N$. By Lemma 19, we can find $q \in \mathcal{Q}$ and $Z \in \Gamma$ such that

(i) $qZ\alpha = O_{m,s}$ and $qZ\beta\alpha = O_{m,t}$ for some $0 \leq s < t \leq n$, $\alpha \in \Gamma^*$ and $\beta \in \Gamma^+$; and

(ii) the transition subsequence of $O_{m,s} \xrightarrow{b} O_{m,s+1} \xrightarrow{b} \ldots \xrightarrow{b} O_{m,t}$ is pumpable.

Let $k = t - s$, we can repeat the pumpable transition sequence as many times as we want

$$qZ\alpha \xrightarrow{b^k} qZ\beta\alpha \xrightarrow{b^k} qZ\beta\beta\alpha \xrightarrow{b^k} \dots \tag{4}$$

As $qZ\alpha = O_{m,s} \approx pY^sX^m$, the above sequence must be simulated by

$$pY^sX^m \xrightarrow{b^k} pY^{s+k}X^m \xrightarrow{b^k} pY^{s+2k}X^m \xrightarrow{b^k} \dots \tag{5}$$

such that $qZ\beta^i\alpha \approx pY^{s+ki}X^m$ for $i \geq 0$.

For different m, we can have different transition sequences (4) and (5), possibly with different $s > 0$, $k > 0$, $q \in \mathcal{Q}$, $Z \in \Gamma$, $\alpha \in \Gamma^*$ and $\beta \in \Gamma^+$. By the pigeonhole principle, there are two different numbers $1 \leq m_1 < m_2 \leq N$ so that q and Z are the same in transition sequence (4). Assume w.l.o.g. that $qZ\alpha_1 = O_{m_1,s_1} \approx pY^{s_1}X^{m_1}$, $qZ\alpha_2 = O_{m_2,s_2} \approx pY^{s_2}X^{m_2}$ and $qZ \xrightarrow{b^{k_1}} qZ\beta_1$ for some $s_1, s_2, k_1 > 0$, $\alpha_1, \alpha_2 \in \Gamma^*$ and $\beta_1 \in \Gamma^+$. The actions $O_{m_1,s_1} \xrightarrow{b^{k_1 \cdot N}} qZ\beta_1^N\alpha_1$ must be simulated by $pY^{s_1}X^{m_1} \xrightarrow{b^{k_1 \cdot N}} pY^{s_1+k_1\cdot N}X^{m_1}$. Similarly $O_{m_2,s_2} \xrightarrow{b^{k_1 \cdot N}} qZ\beta_1^N\alpha_2$ must be simulated by $pY^{s_2}X^{m_2} \xrightarrow{b^{k_1 \cdot N}} pY^{s_2+k_1\cdot N}X^{m_2}$. We also have that $pY^{s_1+k_1\cdot N}X^{m_1} \approx qZ\beta_1^N\alpha_1$ and $pY^{s_2+k_1\cdot N}X^{m_2} \approx qZ\beta_1^N\alpha_2$.

Consider the following transition sequence of $pY^{s_2+k_1\cdot N}X^{m_2}$

$$pY^{s_2+k_1\cdot N}X^{m_2} \xrightarrow{c} p_1Y^{s_2+k_1\cdot N}X^{m_2} \overset{\varepsilon}{\Longrightarrow} p_1X^{m_2} \xrightarrow{a^{m_1+1}} p_1X^{m_2-m_1-1}. \tag{6}$$

It can be matched by $qZ\beta_1^N\alpha_2 \xrightarrow{ca^{m+1}} P_1$ for some P_1. Since $qZ\beta_1^N\alpha_2$ is real-time and $N \geq m_1+1$, the actions ca^{m_1+1} from $qZ\beta_1^N\alpha_2$ only depend on $qZ\beta_1^N$. Thus $qZ\beta_1^N\alpha_1 \xrightarrow{ca^{m_1+1}} Q$ for some Q. This contradicts with $qZ\beta_1^N\alpha_1 \approx pY^{s_1+k_1\cdot N}X^{m_1}$, since after transition $pY^{s_1+k_1\cdot N}X^{m_1} \xrightarrow{c} p_1Y^{s_1+k_1\cdot N}X^{m_1}$, the longest non-$\varepsilon$ action sequence is a^{m_1}. ◀

5 Conclusion

In this paper, we show that the bisimilarity of PDA is Ackermann-hard. Combining with the result in [10], we can conclude that the bisimilarity of PDA is Ackermann-complete. The result can be generalized to normed PDA. Besides, we give another proof on that the so-called real-time pushdown processes are strictly weaker than first-order grammars. Our work answers the question proposed by Jančar and Schmitz in [10].

When the number of control states of PDA is fixed as $d \geq 4$, bisimilarity is $\mathbf{F}_{d-1}$-hard. An obvious open problem is to close the complexity gap with the $\mathbf{F}_{d+4}$ upper bound in [10].

References

1 Jos CM Baeten, Jan A Bergstra, and Jan Willem Klop. Decidability of bisimulation equivalence for process generating context-free languages. *Journal of the ACM*, 40(3):653–682, 1993. doi:10.1145/174130.174141.

2 Michael Benedikt, Stefan Göller, Stefan Kiefer, and Andrzej S Murawski. Bisimilarity of pushdown automata is nonelementary. In *Proc. LICS '13*, pages 488–498. IEEE, 2013. doi:10.1109/LICS.2013.55.

3 Didier Caucal. Bisimulation of context-free grammars and of pushdown automata. *Modal Logic and Process Algebra*, 53:85–106, 1995.

4 Bruno Courcelle. Recursive applicative program schemes. In *Formal Models and Semantics*, pages 459–492. Elsevier, 1990. doi:10.1016/B978-0-444-88074-1.50014-7.

5 Seymour Ginsburg and Sheila Greibach. Deterministic context free languages. *Information and Control*, 9(6):620–648, 1966. doi:10.1016/S0019-9958(66)80019-0.

6 J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley Publishing Company, 1979.

7 Petr Jančar. Decidability of DPDA Language Equivalence via First-Order Grammars. In *Proc. LICS '12*, pages 415–424. IEEE, 2012. doi:10.1109/LICS.2012.51.

8 Petr Jančar. Bisimulation Equivalence of First-Order Grammars. In *Proc. ICALP '14*, LNCS, pages 232–243. Springer, 2014. doi:10.1007/978-3-662-43951-7_20.

9 Petr Jančar. Equivalences of pushdown systems are hard. In *Proc. FoSSaCS '14*, pages 1–28. Springer, 2014. doi:10.1007/978-3-642-54830-7_1.

10 Petr Jančar and Sylvain Schmitz. Bisimulation equivalence of first-order grammars is ACKERMANN-complete. In *Proc. LICS '19*, pages 1–12. IEEE, 2019. doi:10.1109/LICS.2019.8785848.

11 Petr Jančar and Jivří Srba. Undecidability of bisimilarity by Defender's forcing. *Journal of the ACM (JACM)*, 55(1):5, 2008. doi:10.1145/1326554.1326559.

12 Stefan Kiefer. BPA bisimilarity is EXPTIME-hard. *Information Processing Letters*, 113(4):101–106, 2013. doi:10.1016/j.ipl.2012.12.004.

13 Antonín Kučera and Petr Jančar. Equivalence-checking on infinite-state systems: Techniques and results. *Theory and Practice of Logic Programming*, 6(201), 2006. doi:10.1017/S1471068406002651.

14 Antonín Kučera and Richard Mayr. On the complexity of checking semantic equivalences between pushdown processes and finite-state processes. *Information and Computation*, 208(7):772–796, 2010. doi:10.1016/j.ic.2010.01.003.

15 Robin Milner. *Communication and concurrency*, volume 84. Prentice-Hall, Inc., 1989.

16 Michio Oyamaguchi. The equivalence problem for real-time DPDAs. *Journal of the ACM (JACM)*, 34(3):731–760, 1987. doi:10.1145/28869.28881.

17 V Yu Romanovskii. The equivalence problem for real-time deterministic pushdown automata. *Cybernetics*, 22(2):162–175, 1986. doi:10.1007/BF01074776.

18 Sylvain Schmitz. Complexity hierarchies beyond elementary. *ACM Transactions on Computation Theory (TOCT)*, 8(1):3, 2016. doi:10.1145/2858784.

19 Sylvain Schmitz. *Algorithmic Complexity of Well-Quasi-Orders.* Habilitation thesis, École Normale Supérieure Paris-Saclay, 2017.

20 Sylvain Schmitz. The parametric complexity of lossy counter machines. In *Proc. ICALP '19*, volume 132, pages 129:1–129:15. LZI, 2019. doi:10.4230/LIPIcs.ICALP.2019.129.

21 Philippe Schnoebelen. Revisiting Ackermann-hardness for lossy counter machines and reset Petri nets. In *Proc. MFCS '10*, pages 616–628. Springer, 2010. doi:10.1007/978-3-642-15155-2_54.

22 Géraud Sénizergues. Decidability of bisimulation equivalence for equational graphs of finite out-degree. In *Proc. FOCS '98*, pages 120–129. IEEE, 1998. doi:10.1109/SFCS.1998.743435.

23 Géraud Sénizergues. L(A)=L(B)? Decidability results from complete formal systems. *Theoretical Computer Science*, 251(1-2):1–166, 2001. doi:10.1016/S0304-3975(00)00285-1.

24 Géraud Sénizergues. The bisimulation problem for equational graphs of finite out-degree. *SIAM Journal on Computing*, 34(5):1025–1106, 2005. doi:10.1137/S0097539700377256.

25 Colin Stirling. Decidability of bisimulation equivalence for normed pushdown processes. *Theoretical Computer Science*, 195(2):113–131, 1998. doi:10.1016/S0304-3975(97)00216-8.

26 Colin Stirling. Decidability of bisimulation equivalence for pushdown processes. Technical report, University of Edinburgh, 2000.

27 Colin Stirling. Decidability of DPDA equivalence. *Theor. Comput. Sci.*, 255(1-2):1–31, 2001. doi:10.1016/S0304-3975(00)00389-3.

28 Colin Stirling. Deciding DPDA equivalence is primitive recursive. In *Proc. ICALP '02*, pages 821–832. Springer, 2002. doi:10.1007/3-540-45465-9_70.